Internet-Scale Route Tracing Capture and Analysis

Matthieu Gouel

Sorbonne Université
Laboratoire d'Informatique de Paris 6 (UMR 7606)

Internet-Scale Route Tracing Capture and Analysis

Présentée par Matthieu Gouel

Pour le grade de Docteur de Sorbonne Université

Soutenue le 12 juin 2023 devant le jury composé de :

Mme. Clémence Magnien	Directrice de Recherche, Sorbonne Université	Présidente du jury
M. Chadi Barakat	Directeur de Recherche INRIA, Université Côte d'Azur	Rapporteur
Mme. Cristel Pelsser	Professeure, Université Catholique de Louvain	Rapporteuse
M. Jordan Augé	Chercheur, Cisco Systems France	Examinateur
M. Timur Friedman	Maître de Conférences, Sorbonne Université	Co-encadrant
M. Olivier Fourmaux	Professeur, Sorbonne Université	Directeur de thèse

Acknowledgments

I would like to express my gratitude to the French Ministry of Defense for fully funding my thesis and professional experiences during this period.

Many thanks to Olivier Fourmaux and Timur Friedman for your trust and guidance throughout my research and teaching work. Thanks a lot to Robert Beverly and Justin P. Rohrer for hosting me at the Naval Postgraduate School during an unforgettable two-month period and for their involvement in all our collaborations. Thanks to Chadi Barakat, Cristel Pelsser, Jordan Augé and Clemence Magnien for accepting to be members of my thesis committee and for their thorough review of my dissertation. Thanks to Matthieu Latapy and Kavé Salamatian for agreeing to be on the follow-up committee and for their many pieces of advice. Huge thanks to my colleagues from LIP6 Maxime Mouchet, Hugo Rimlinger, Berat Şenel, Kevin Vermeulen, as well as all my colleagues from LINCS, for their unconditional support, exciting exchanges, and inspiring work.

I am also deeply grateful to my friends who have followed me through my ups and downs since high school with kindness and patience. The same goes for my family and particularly my brothers, parents, and grandparents, who cultivated my passion for science and gave me everything to pursue my dreams.

Lastly, I want to express all my deepest love and gratitude to my partner, Lucie, who supported me every day of this long but exciting adventure.

Merci.

Abstract

English version

The Internet is one of the most remarkable human creations, enabling communication among about two thirds of the global population. This network of networks spans the entire globe and is managed in a highly decentralized way, making it impossible to fully comprehend at IP-level. Nonetheless, for over two decades, researchers have been devising new techniques, developing new tools, and creating new platforms to capture and provide more precise and comprehensive maps of the Internet's topology. These efforts support network operators in the industry and other researchers in improving core features of the Internet such as its connectivity, performance, security, or neutrality.

This thesis presents new contributions that improve the scalability of Internet topology measurement. It introduces a state-of-the-art measurement platform that enables the use of high-speed probing techniques for IP route tracing at Internet scale, as well as a reinforcement learning approach to maximize the discovery of the Internet topology. Because the analysis of the route tracing data collected requires additional metadata, the evolution of IP address geolocation over a 10-year period in a widely used proprietary database is examined, and lessons are provided to avoid biases in studies using this database. Finally, a large-scale analysis framework is developed to effectively utilize the large number of collected data and augmented metadata from other sources, such as IP address geolocation, to produce insightful studies at the Internet scale.

This work aims to considerably improve the study of the Internet topology by providing tools to collect and analyze large amounts of Internet topology data. This will allow researchers to better understand how the Internet is structured and how it evolves over time, leading to a more comprehensive understanding of this complex system.

French version

Le réseau Internet est l'une des réalisations les plus remarquables de notre civilisation, permettant la communication entre environ deux tiers de la population mondiale. Ce réseau de réseaux a une portée internationale et est géré de manière hautement décentralisée, rendant sa représentation globale impossible au niveau IP. Cependant, depuis plus de deux décennies, les chercheurs ont développé de nouvelles techniques, construit de nouveaux outils et créé de nouvelles plateformes pour capturer et fournir des cartes plus précises et complètes de la topologie de l'Internet, soutenant ainsi les opérateurs réseau de l'industrie et les autres chercheurs dans l'amélioration des composantes essentielles du réseau Internet comme sa sécurité, ses performances, sa connectivité ou sa neutralité.

Cette thèse présente de nouvelles contributions visant à améliorer la mesure de la topologie de l'Internet à grande échelle. Elle introduit une plateforme de mesure permettant l'utilisation d'un traçage à grande vitesse des routes IP, ainsi qu'une approche d'apprentissage par renforcement pour optimiser la découverte de la topologie de l'Internet. L'analyse des données des routes de l'Internet collectées nécessitant des métadonnées supplémentaires, cette thèse étudie également l'évolution de la géolocalisation des adresses IP dans une base de données privée couramment utilisée sur une période de 10 ans et fournit des conseils pour éviter les biais dans les études utilisant cette base de données. Enfin, un cadre d'analyse à grande échelle a été développé pour exploiter de manière efficace cette grande quantité de données collectées, ainsi que les métadonnées supplémentaires provenant d'autres sources telles que la géolocalisation des adresses IP, afin de produire des études pertinentes à l'échelle de l'Internet.

Ce travail vise à considérablement améliorer l'étude de la topologie de l'Internet en proposant des méthodes et des outils pour la collecte et l'analyse de quantités massives de données sur la topologie IP de ce réseau. Cette approche permettra une meilleure compréhension de la structure et de l'évolution de l'Internet, facilitant ainsi la compréhension de ce système complexe.

Contents

Chapter 1

Introduction

The Internet has become a ubiquitous communication infrastructure connecting an estimated 5.3 billion people worldwide in 2022, representing about two thirds of the global population [84]. However, due to its highly distributed nature, no single entity has a comprehensive understanding of the entire Internet, and especially at the IP-level granularity. This vast network consists of tens of thousands of networks of varying sizes and degrees of interconnectivity, known as Autonomous Systems (ASes)[106]. For over two decades, the Internet measurement research community has been publishing papers on improving tools and techniques for capturing more accurate and complete information about the Internet topology [14, 16, 18]. To this end, numerous measurement platforms have been developed over the years [26, 148, 110], to enhance the scalability and convenience of Internet measurements.

In this introduction, we aim to motivate the need to develop new platforms and algorithms to facilitate and optimize the use of Internet-scale probing techniques [21, 168, 79]. Additionally, we stress the importance of designing and implementing frameworks to efficiently process the vast amount of data collected. We also present the contributions of our work towards addressing these challenges.

1.1 Motivation

To produce meaningful and comprehensive results, the Internet measurement community must be capable of capturing data at the scale of the Internet and also be able to interpret the data collected. These results support the study of multiple aspects of the Internet such as its connectivity [4, 163], performance [103], security [181] or neutrality [180]. The ability to produce

Internet-scale IP-level maps of the Internet more frequently is crucial in order to understand the dynamics of the Internet topology and potentially uncover short-term changes, and so improve our reactiveness to connectivity and security events.

Recently, a new set of probing tools, such as Yarrp [21], Diamond-Miner [168], and FlashRoute [79], have been developed, which enable high-speed probing of the Internet's IP-level topology. These tools allow for probing the routes from each source at a higher probing rate, enabling one to collect more snapshots of the Internet topology in the same amount of time. But the use of these tools can present challenges in terms of scale, due to the amount of measurement data to collect and to process.

This work revolves around this issue of scale. The first question we aim to address is how to collect comprehensive and accurate route tracing data at the scale of the Internet. The second question is how to effectively analyze the dynamics of Internet data, particularly IP geolocation data, to avoid errors in subsequent analysis. Finally, we explore how to efficiently analyze the vast amounts of collected data, including additional metadata, to generate valuable insights for the Internet measurement community and fully capitalize on this wealth of information.

1.2 Contributions

This thesis presents three major contributions to the capture and the analysis of route tracing data at Internet-scale. It follows the regular flow of data, from data collection (Chap. 3) to data processing (Chap. 4 and Chap. 5).

First, we present a resilient reinforcement learning approach to distributed IP route tracing (Chap. 3), which improves the capture of the IP-level Internet topology thanks to Iris, a new resilient and high-speed probing platform and Zeph, a reinforcement learning approach for choosing the IP routes to probe. With Iris and Zeph, we are able to discover 3 times more nodes and 10 times more links for the same number of prefixes in a shorter period of time than the state of the art. This work has led to two per-reviewed publications: one in the ACM SIGCOMM Computer Communication Review (CCR) journal in the issue of January 2022 [71] and one at the CoRes conference in 2022 [70]. In addition, we have released Iris software, Zeph software, as well as the evaluation analyses of the papers as a free, open-source and liberally licensed code [159]. We also maintain a production system [160] that is available to the scientific community.

Then, we propose a longitudinal study (Chap. 4), which analyze 10-

year data dynamics of a geolocation database popular in research work as well as in industry. This large-scale study allows us to find short-term dynamics of IP geolocation data in this database that could introduce bias in prior research study's results. These lessons found in this study lead us to propose recommendations to appropriately use the data of this database in research. This work has led to a per-reviewed publication at the Network Traffic Measurement and Analysis (TMA) conference in 2021 [68] as well as a technical report [69].

Also, we present a framework for traceroute processing (Chap. 5), which allows performing large-scale analysis of IP route tracing data, augmented with external metadata such as IP-to AS information or IP geolocation. With this framework, we are able to analyze millions of traceroutes in a few seconds to study incomplete AS paths because of stars in traceroutes. Also, MetaTrace enables us to perform a longitudinal study on Internet flattening over 5 years on 6 billion traceroutes, reappraising the results of previous studies on this topic. We are preparing this work for a peer-reviewed submission and we released MetaTrace as a free, open-source and liberally licensed code [159].

In conclusion, we present a perspective section (Chap. 6) that discusses the development of more ethical measurement platforms that can benefit from high-speed probing while avoiding harm to the network. We define a beneficent platform as one that minimizes rate limit violations defined by network operators. We also discuss the challenges in building such a beneficent system and suggest ways to overcome them. This work has been accepted for publication after a per-review process at the CoRes conference in 2023.

Chapter 2

State of the art

This chapter presents the prior works in the Internet measurements necessary to fully comprehend the basics of this thesis and the context in which this work has been conducted. In addition to this state of the art, each chapter of the thesis that presents a research contribution (Chap. 3 to Chap. 5) that contains a related work section specific to the addressed subject.

This chapter follows the same overall approach as the thesis, i.e., going from data collection to data analysis.

We will first develop general knowledge about IP route tracing tools and algorithms in order to collect the most accurate and comprehensive Internet topology data with a high-probing rate. Then, we will review the current state-of-the-art platforms and which tools and algorithms they use. These two sections will help us to understand the motivation behind the creation of the Iris platform and Zeph algorithm in Chap. 3.

The third section will present the various types of metadata that enhance IP route tracing data to provide additional context and information for analysis. One example of such metadata is the physical geolocation of IP addresses, which can be inferred through geolocation databases or various probing techniques. This section is related to Chap. 4, where we examine the data dynamics of a geolocation database that is extensively used in both research papers and the industry.

The fourth section will focus on the analysis of IP route tracing data and metadata, presenting the data format defined by the probing platforms and the standard methods used for analysis. This section is especially relevant for understanding the motivation behind MetaTrace, our data analysis framework presented in Chap. 5.

2.1 IP route tracing tools and algorithms

In this section we present general concepts, tools and techniques for IP route tracing, relevant to understand the challenges in discovering accurate and comprehensive IP-level topology of the Internet.

IP route tracing is the action of measuring the IP route between two Internet hosts. Usually, the two hosts are composed of a *source*, i.e., the host executing the measurement, and a *destination*, which is a routable host in the measured network. In the literature, the *source* host is also named the *vantage point*. The IP route is a succession of router interfaces. We call the *ingress interface* an interface of a router facing the source and the *egress interface* an interface facing the destination.

Figure 2.1: Single-path route between the source VP and the destination DT

In the example of Fig. 2.1, we can see a route between the source VP and a destination DT. This route is defined as *single-path* as there is only one possible ordered list of router interfaces that create a path between the source and the destination.

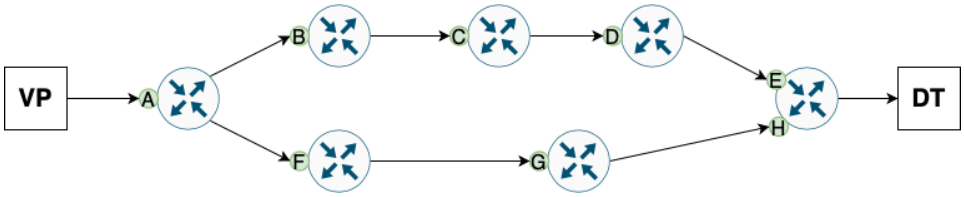


Figure 2.2: Multi-path route between the source VP and the destination DT

Fig. 2.2 shows an example of a *multi-path* route between the source VP and the destination DT. Here, the router with the ingress interface A balances the traffic between two of its egress interfaces. These routers are called *load balancers*. This can be done at the router-level with a manual configuration, or automatically by using the *Equal Cost Multipath Protocol (ECMP)* mechanism. At the end of the route, the router with ingress interfaces E and H is merging the two paths and all the packets towards DT are routed to the same egress interface. This structure between the first router and the last one is called a *diamond* [14]. Vermeulen et al. [168] found that multi-path routes are prevalent in their measurements, where 64% of the traces towards all of the

/24 prefixes contain at least one load balancer. The maximum width of a diamond, which refers to the number of unique interfaces at a given TTL, can be several dozen. Meanwhile, the length of a diamond, which refers to the size of the longest path, can exceed 20 [169].

2.1.1 Introduction of traceroute

In 1988, i.e., 7 years after the publication of the RFC 791 [49] that defined the Internet Protocol, Van Jacobson established the main principles of the traceroute program still implemented today in UNIX operating systems [85].

The idea behind traceroute is to measure the route from a source host to a destination host by sending probes towards the destination. It relies on exploiting the *Time to Live* (TTL) field of the IPv4 header. This 1-byte header field is typically used to prevent routing loops. When an IP packet is routed by a router, the router decrements the TTL by one. When the TTL reaches 0, the router discards the packet and sends a notification to the source. This notification takes the form of an *ICMP time exceeded* message, with, in most cases, the ingress router interface listed as the source IP address and the vantage point IP listed as the destination address.

The IP address used as the source IP address of the reply packet may occasionally be different from the ingress router interface, such as a loopback address or the IP address of another interface, depending on the router configuration. Also, with this methodology, traceroute will not measure the egress interfaces of the route.

Traceroute sends multiple probes to a destination by incrementally setting the TTL value from 1, then 2, and so on, until it reaches the destination. With each probe, traceroute records the interface of the router at each TTL, also known as a *hop*. Finally, when a probe reaches the destination, the destination sends a notification different from an ICMP time exceeded, and the ICMP notification type will vary depending on the protocol type of probe sent by the source. In cases where the destination does not respond, traceroute provides a maximum TTL option to end the measurement.

Van Jacobson's traceroute was initially implemented using UDP probes. In this setup, the destination would send an *ICMP destination unreachable* because the destination port used by traceroute is typically high and not open at the destination host. However, the same approach can be used with ICMP or TCP probes. With ICMP probes, the source sends *ICMP echo request* probes, and the destination responds with an *ICMP echo reply* notification. With TCP, the source sends a TCP segment with the SYN flag raised, and the destination responds with a TCP segment with the RST flag raised if

the destination port is closed or with SYN+ACK flags if the port is open. Luckie et al. [108] studied how these different traceroute probe methods affect destination reachability and IP link discovery.

Although the initial implementation of Van Jacobson was developed for IPv4, the same principles can be easily applied to IPv6. In IPv6, the TTL field is replaced by the *Hop Limit* field, and ICMP notifications are replaced with ICMPv6 notifications.

2.1.2 Improving traceroute reliability

This version of traceroute works as expected with single-path routes (Fig. 2.1) but may yield unreliable results in case of multi-path routes.

Let's illustrate the issue with traceroute measuring the route of Fig. 2.2. For clarity, we will pretend that traceroute sends only one probe per TTL, instead of 3 probes by default, but the issue remains the same.

At hop 1, traceroute will discover A. At hop 2, traceroute will discover either B or F. At hop 3, traceroute will discover either C or G. At hop 4, traceroute will discover either D or H. At hop 5, traceroute will discover either E or the destination DT. Finally, at hop 6, traceroute will only discover the destination DT. Thus, traceroute may discover paths that do not actually exist, such as `VP - A - B - G - D - DT`.

These errors are caused by load balancers. In Fig. 2.2, there is one load balancer, which has the ingress interface A. To resolve the issue, one should ensure that traceroute will trace only one of the possible paths towards the destination, i.e., `VP - A - B - C - D - E - DT` and `VP - A - F - G - H - DT` in this example.

To achieve this, one must be aware of how a load balancer selects the possible egress interface if multiple ones are available. The paper of Augustin et al. [14] shows that there are mainly 3 types of load balancers.

- *Per-packet*, where the packet will be routed based on a policy that does not depend on the packet fields (for instance, a round-robin policy)
- *Per-destination*, where the packet will be routed based on its destination IP address. As a result, all packets with the same destination will take the same path.
- *Per-flow*, where the packet is routed based on the *five-tuple* fields of the packet. These fields include the source and destination IP addresses, the protocol fields in the IPv4/v6 header, and either the source and destination port fields for UDP and TCP or the code and checksum

fields for ICMP. In consequence, all packets belonging to a same session, or a same *flow*, will take the same path.

Almeida et al. [5] confirmed that per-destination and per-flow load balancers are prevalent for all types of transport layer (UDP, TCP and ICMP). The per-packet load balancers represent at most 0.3% of the load balancer classification.

Paris-Traceroute [14] utilizes this information by fixing the header fields that are responsible for per-flow and per-destination load-balancing for all the probes in a single trace. By doing so, Paris-Traceroute can ensure that the measured path is correct when the route only includes this type of load balancer. Since per-packet load balancers are uncommon, this technique is considered safe and widely used in the networking community today.

2.1.3 Improving traceroute completeness

Paris-Traceroute provides accurate traceroute results for routes that contain load-balanced paths. The next logical step is to extend this capability to enable tracing of all paths for a given route.

Augustin et al. developed a technique called the *Multipath Detection Algorithm* (MDA) first presented as a poster and a workshop [15, 17], then presented at IMC in 2007 [16] and finally mathematically strengthened at INFOCOM in 2009 [165].

The MDA works on a hop-by-hop basis. For each interface r at hop $h-1$, the MDA selects flows that are expected to reach r and uses them to probe the successor interfaces or r at hop h (referred to as the *nexthops* of r). The MDA sends enough probes to be certain at a confidence level $1-\epsilon$ that it has discovered all the nexthops of r. Initially, the MDA sends enough probes to ensure with $100 * (1-\epsilon)\%$ confidence that there is at most one nexthop to r. If it discovers one interface after n_1 probes, then r is considered resolved. If it discovers $k > 1$ interfaces, it sends n_k probes and continues to do so until it sends n_k without discovering any new interfaces.

In order to compute the values n_k, the MDA makes multiple assumptions: (1) no routing change; (2) load-balancing is uniform to all nexthops; (3) all probes are answered; and (4) there is no per-packet load-balancing.

The number of probes to send to reach the statistical guarantees n_k are given in a table for $\epsilon = 0.05$. Later, Jacquet et al. [86] gave a general formula to compute the values n_k in function of ϵ.

The MDA varies the flow of the probes by changing the port numbers or the ICMP checksum only. So in this initial implementation, the MDA can

only work with per-flow load balancers and not with per-destination load balancers. But the MDA can be extended by targeting a *destination prefix* instead of a destination address [18] and then create flow by varying the destination address instead of the port numbers. This way the MDA would be compatible with per-flow and per-destination load-balancing.

In 2018, Vermeulen et al. [169] developed MDA-Lite, which saves probes for certain topologies and falls back to the regular MDA when necessary, thus reducing the number of required probes.

In 2020, Almeida et al. [5] generalized the MDA, by varying any of IPv4/v6 and TCP/UDP/ICMP field and thus characterizing the different load balancers in the wild. They determined that the vast majority of load balancers are per-flow or per-destination. They also observed that a small fraction of per-flow load balancers are in fact "per-application" where only the transport port (in TCP/UDP) participates in the load-balancing.

2.1.4 Improving traceroute throughput

In an effort to map the Internet routes more exhaustively and rapidly, one would want to send traceroute probes at a high probing rate. But the implementations presented so far are: (1) *stateful*, because traceroute has to match the probe sent with the reply; and (2) *sequential*, because traceroute implementation operates hop-by-hop with a low degree of parallelism. Thus, the implementations presented so far rarely exceed 100 pps.

Yarrp [21] dramatically increases single-path traceroute throughput thanks to two mechanisms. First, it encodes meaningful information, such as the probe's TTL and sent timestamp, in probe fields to match the ICMP time exceeded reply and the probe without state in the sender. This information is retrieved in the quotation of the ICMP reply. Second, it shuffles the probes to send in the space *destination * TTL*, instead of a sequential hop-by-hop approach. It allows sending at high speed without putting too much pressure on a particular path or interface. Yarrp6 [23] is an extension of Yarrp for IPv6 support.

FlashRoute [79] builds upon Yarrp principles but reintroducing carefully managed states with the aim of removing redundancy in the probing. It leverages ideas from DoubleTree [57] that explore routes in the backward direction and stopping when reaching an already probed interface.

But Yarrp and FlashRoute only probe one path per route and so are not using the MDA. Diamond-Miner [168] leverages Yarrp principles but adapts the MDA to be used with no state during probing. It works by conducting rounds of probing in which probes are sent using the methodology of Yarrp.

But the flows to be sent in each round are computed based on the results of previous rounds, with the goal of meeting the statistical guarantees of the MDA. As a result, it allows performing multi-path traceroutes while reaching high probing rates.

Lessons Decades of research in Internet measurements have led to the development of tools and techniques that improve the reliability, completeness, and speed of route tracing data. The use of these tools repeatedly shows [133, 47, 64] that routes over the Internet are globally stable on a daily scale. However, no tool currently utilizes previous measurement data to speed up or improve the completeness of upcoming measurements. In this thesis, we contribute to this effort by developing Zeph (see Chap. 3), a new scheduling algorithm that uses a reinforcement learning approach to select the next IP routes to probe from each vantage point, maximizing the discovery of the Internet topology at high throughput.

2.2 IP route tracing platforms

Up until now, we have discussed techniques and tools that can improve the correctness, completeness, and throughput of traceroute-style measurements. All of these tools, such as Yarrp, FlashRoute, and Diamond-Miner, can be used as standalone solutions on a single vantage point. However, to perform large-scale, multi-vantage-point measurements of the Internet topology, with the goal of monitoring and studying its dynamics and changes over time, these tools need to be integrated into a probing platform.

A platform is hardware and software infrastructure that enables the execution of one or multiple probing tools from one or more vantage points. Additionally, the platform facilitates the collection and storage of data gathered by the probing tools, and provides a means to share or query the stored measurement data.

Besides these main components of probing execution, data storage, and data sharing, a platform often includes precise monitoring of measurements and vantage points, facilitates system maintenance with software resilience, and stores additional measurement metadata such as the tool used, its parameters, and measurement timestamps.

Many platforms have appeared and disappeared over the years [55]. In this section though we will focus on the platforms that are still in use at the time of writing.

CAIDA's Archipelago (Ark) [26] uses its 110 vantage points to issue single-path Paris-Traceroute measurements at a probing rate of 100 pps. It follows a cycle of measurements that last roughly a day, where all the routed IPv4 /24 prefixes are shared among the vantage points. As a result, one should find inside a cycle one and only one traceroute towards a particular destination. Ark's vantage points are globally located and conduct regular measurements of the Internet topology, which benefit the Internet measurement community. They also allow researchers to perform measurements with their on-demand topology measurement service Vela [29].

RIPE Atlas [148] allows its user to perform pings and single-path Paris-Traceroute at a low probing rate from around 10,0000 vantage points located in many Autonomous Systems. In addition, RIPE Atlas performs regular traceroute mesh measurements between their "anchors" vantage points and towards particular destinations such as Domain Name System (DNS) root IP addresses.

M-Lab [110] issues multi-path Paris-Traceroute measurements towards each client in reaction to a Network Diagnostic Tool (NDT) [111] test towards their platform. They allow the community to use the data collected, after anonymization, from their Google's BigQuery database [66].

These three main probing platforms are all implementing Paris-Traceroute, with or without the MDA capabilities. Ark and M-Lab are using Scamper [102], a prober software developed by CAIDA that supports the regular and the MDA traceroute, whereas RIPE Atlas uses their own implementation of Paris-Traceroute.

Lessons Numerous probing platforms have emerged from various organizations, each with different models. However, it is worth noting that no existing platform uses the latest tools such as Yarrp, Diamond-Miner or FlashRoute, that can improve the throughput of route tracing measurements. This thesis introduces Iris (see Chap. 3), a new probing platform that is compatible with Yarrp and Diamond-Miner, relies on open-source software, and can be audited and replicated by any actor in the Internet measurement community.

2.3 Data augmentation

In the previous sections, we talked about traceroute-style probing techniques, tools that implement these techniques and in-use platforms using some of these tools.

Once the routes traced, one would want to use this data to perform

analysis that increase our understanding of the Internet. Quite often, though, IP-level topological data is not sufficient. Additional data, or *metadata*, will help to complete the view and gives context of route structure and dynamics. We analyzed the papers published in the IMC, PAM and TMA conferences from 2017 and 2021. We automatically extracted the papers that contain the world *traceroute* and then manually investigated these papers. Among them, 74% use at least one of the following metadata sources to analyze traceroute data: reverse DNS resolution (20%), IP geolocation (19%), alias resolution (19%), IP-to-PoP resolution (31%) and IP-to-AS resolution (62%).

In this section, we give an overview of this metadata and how to acquire them.

2.3.1 Reverse DNS resolution

Reverse DNS resolution is the process of resolving a domain name from an IPv4 or IPv6 address. This is achieved by performing a reverse DNS lookup in the Domain Name System and retrieving the PTR records associated with the IP address. Unlike other types of metadata, reverse DNS resolution is publicly accessible, and no particular techniques are required to access this information. However, performing lookups on millions of IP addresses can be challenging if the information is needed for large-scale studies. One possible solution is to use tools such as ZDNS [184], which enable high-speed DNS lookups.

2.3.2 IP geolocation

Numerous techniques have been developed over the last 20 years to physically geolocate IP addresses. Some of these techniques are based on latency measurements [48, 75, 78, 177]. These techniques can be performed on every routable IP addresses but can suffer from lack of precision due to routing detours or routing path asymmetry.

Others combine latency measurements with topology measurements to better constraints the geolocation [89, 171, 58] These techniques are more precise than techniques only relying on delay measurements but require sending more probes.

Others are based on finding hints in the reverse DNS record of each IP address [81, 151, 107]. These techniques do not require performing active measurements but can be used only if a reverse DNS is present for the IP address to geolocate and if geolocation hints are present inside the DNS record.

Finally, multiple commercial services offer geolocation databases [118, 127, 83, 82] that map IP prefixes into geolocation, often along with a confidence level, but do not disclose their methodology to gather geolocation data.

2.3.3 Alias resolution

Alias resolution involves grouping IP addresses that belong to the same router, allowing the production of router-level topology maps. Like IP geolocation, alias resolution techniques must balance precision and coverage by using both active and passive measurement data.

Some techniques rely on signature to group IP addresses together like the source IP address [130, 72], IPv4 identifier [161, 20, 94], IPv4 timestamp option [156, 116], IPv6 source routing [143, 144], IPv6 fragmentation identifier [104], or ICMP rate limiting [167]. Some other techniques rely on IP topology information such as traceroute information [76, 93] or IPv4 record route option [157]. Finally, other techniques uses reverse DNS information [161, 95].

2.3.4 IP-to-PoP resolution

Another approach is to map the Internet topology by *Point of Presence* (PoP). A PoP is a physical location where network operators place their routers to provide connectivity to their customers. Two or more ASes (Autonomous Systems) can also share the same PoP, which is called an *Internet Exchange Point* (IXP), where they exchange peering information and traffic between their networks.

PeeringDB [137] is a non-profit, free and collaborative database that stores networks data, IXP interconnection and geolocation data. It is meant to facilitate interconnection decisions as a network operator.

Apart from PeeringDB database, numerous techniques were developed to cluster IP addresses or aliased routers into PoPs by using active [154, 178] or passive measurements [8] or by using reverse DNS record [161, 113].

2.3.5 IP-to-AS resolution

The most zoomed out level of the Internet topology is the representation of the graph as a collection of Autonomous Systems (AS). In this representation, nodes in the graph represent ASes, while links between them represent the peering relationships between the ASes. An AS is a group of IP prefixes that are under the control of a single organization, which is identified by a unique Autonomous System Number (ASN). The AS-level topology can be obtained

using passive measurements of BGP updates, by analyzing the AS-paths of these updates. Collectors are used to collect these BGP updates, which are transmitted to them via peering links [164, 150]. Each link between two ASes in the AS-level graph represents a pair of successive ASes in the AS-path of a BGP update.

Another technique to obtain the AS-level topology consists in starting from the IP route level topology and perform an IP-to-AS resolution. It can be done by also looking at BGP updates and mapping IP prefixes to its origin AS [13].

More refined techniques allow to better map the IP address at the boundary of the AS making it more accurate [105, 115, 117] using IP topology information and alias resolution techniques.

Lessons We discussed the four main types of metadata that are typically used to analyze route tracing measurement data. Researchers often rely on datasets provided by the private sector or by other researchers who present a new technique for obtaining a certain type of metadata. However, no studies have analyzed the temporal evolution of data in these datasets, or the potential impact that changes in data could have on research that utilizes these datasets. In this thesis, we conduct a study on the most widely used IP geolocation database in both industry and research, and discovered that significant changes occur between snapshots of this database. These changes could potentially introduce bias into studies that rely on this database (see Chap. 4).

2.4 Data analysis

The data structure used to share and analyze IP route tracing data depends on the probing platform. Ark [26] is using the *Warts* format which is the native output file format of Scamper. It is a binary format holding all of the information regarding the measurement itself, and the results associated with it. The warts files of each of Ark's vantage points can be downloaded from their FTP server [27]. CAIDA developped Fantail [1] that uses Apache Spark and Elasticsearch to query their own data. RIPE Atlas [148] uses JSON [88] format, with the possibility to fetch one's measurement data from their public API or aggregated as files from their FTP server [149]. RIPE Atlas also developped a system based on Apache Spark on Hadoop [2] to query their own data and also pushes their data to Google's BigQuery database [66]. M-Lab [110] is using BigQuery to store all its data, which can be fetched

from BigQuery's API. The heterogeneity of these data formats makes the analysis of multiple sources of data difficult because no common standard exists.

Moreover, we saw that the analysis of route tracing data is usually done using additional metadata such as IP geolocation or IP-to-AS resolution. This adds another layer of complexity to analyze the massive number of data produced by probing platforms in an effective way.

The tools used to analyze traceroute data are usually adopted at the discretion of the authors and not advertised in research projects, being considered as a technical detail. As far as we know, based on discussion with peers, evaluation tools are made with custom in-house scripts, that will manipulate the data in memory [64], in a relational database [91, 166] or a graph database [113].

To the best of our knowledge, no research paper has contributed to an open-source framework with a robust, bug-free, and efficient implementation for analyzing the large amount of data produced by the various actors in the field of Internet measurement research.

Lessons Multiple route tracing platforms exist, each with its own data structure for storing traceroute data, which produce massive amounts of data over time. Additionally, there are a large number of metadata datasets necessary for analyzing this route tracing data. Also, with the development of new tools and platforms that allow for probing the Internet topology at a high rate, the amount of data collected is likely to increase in the next few years. In this thesis, we contribute to the analysis of large amounts of traceroute data by proposing MetaTrace (see Chap. 5), an open-source framework that allows for large-scale heterogeneous traceroute data processing, augmented with metadata.

Chapter 3

Efficient distributed IP route tracing

This first chapter presents two contributions to accelerate the capture of the Internet IP-level topology: (1) Zeph, an algorithm based on a reinforcement learning approach coordinates route tracing efforts across agents in order to optimize the topology discovery; and (2) Iris, a new fault tolerant system for orchestrating high-speed Internet measurements across distributed agents of heterogeneous probing capacities.

3.1 Introduction

Mapping the Internet's topology has been a challenge for the research community for more than two decades. Topological knowledge underpins our understanding of issues such as security [181], connectivity [163, 4], and performance [103]. It also leads to better models of the Internet that aid in the design of new protocols [175]. The IP-level Internet topology, in particular, is an essential input for building other Internet datasets [31], such as AS-relationships [65, 87], MPLS tunnel detection [56, 109], alias resolution [123, 104] and IP geolocation [90].

Systems that measure the topology at the IP level, i.e., the addresses and the traceroute [85] style links from one address to the next, face a tension between completeness of discovery and overhead in the number of probe packets sent. Recent high-speed probing tools Yarrp [21], Diamond-Miner [168], and FlashRoute [79] are capable of tracing towards all routable IPv4 prefixes from a single vantage point at rates exceeding 100,000 packets per second (pps). Probing at these rates enables the creation of Internet topology maps in a

reduced time, avoiding staleness of data due to routing changes. However, very few probing agents with such capacity are available to the community; agents deployed on RIPE Atlas [162], PlanetLab Europe [138], or Archipelago (Ark) [35], for instance, are constrained either by machine resources or by operator policy. Nonetheless, probing from multiple vantage points can bring important marginal gains for topology discovery [19]. Therefore, so as to maximize total discovery given a set of vantage points, we are challenged to design an algorithm that intelligently allocates probing directives, i.e., the instructions to an agent to conduct a route trace towards a specific /24 prefix.

This chapter presents Zeph, a reinforcement learning algorithm for allocating probing directives to agents, and Iris, the generic distributed measurement orchestration system that supports Zeph.

Our contributions are:

- The Zeph algorithm, that learns how to ameliorate the allocation to agents at multiple vantage points of the destination prefixes that each should probe (Sec. 3.4). We find that discoveries increase by 57% for the nodes and 90% for the links in 10 cycles of learning, showing the importance of selecting the right prefixes to probe from each vantage point. Moreover, our system discovers 3 times more nodes and 10 times more links for the same number of prefixes in a shorter period of time than the state-of-the-art system Ark [35] (Sec. 5.6).

- The Iris system, a robust and scalable measurement system, built from free open-source software, that supports multi-vantage-point measurement techniques, and that Zeph can ask to perform single-path or multipath route traces (Sec. 3.5).

- Publicly available code and data [159], along with a production deployment of Zeph and Iris to serve the research community with data series and the ability to perform one's own measurements [160]. The results in this chapter are fully reproducible on commercial cloud instances via the Jupyter notebooks that we provide.[1]

3.2 Related Work

Measurement systems that have been deployed to map and characterize the Internet's IPv4 IP-level topology based on traceroute-like [85] measurements

[1] Instructions for reproducing this work: https://github.com/dioptra-io/zeph-evaluation

from distributed vantage points around the globe historically include Rocketfuel [161], DIMES [154], and iPlane [113]. Today's production systems include RIPE Atlas [162], which performs single-path Paris-Traceroutes from its over 10,000 low-power hardware agents and software agents running at end user's homes and work premises; and M-Lab [110], which issues a multipath Paris-Traceroute [165, 169] towards each client that performs an NDT test [111]. Today's longest-running and most used production topology mapping system is CAIDA's Ark [35], which utilizes agents at over 110 vantage points to perform single-path Paris Traceroutes [14] to one random destination per routed /24 prefix.

Two longstanding challenges face any system that aims to maximize coverage of the Internet's paths, i.e., to run at the scale of the entire IPv4 Internet: how to trace more efficiently and more rapidly.

More efficient tracing Doubletree [57] exploited path commonality to terminate measurements when they converged on a previously probed path. iPlane [113] used BGP data to aggregate destinations by common AS path, and Beverly et al. proposed a series of primitives for more efficient mapping (e.g., subnetting inferences) [22]. Our Zeph algorithm aims for a more efficient assignment of probe destinations to probing agents.

High speed topology discovery techniques Yarrp [21] introduced high speed topology mapping: it encodes sufficient state in each probe packet for the returning ICMP replies, which contain the first bytes of the probe packets, to be self-identifying, thereby eliminating the need to probe slowly in order to associate replies with probes. Whereas Yarrp traces single paths from source to destination, Diamond-Miner [168] added multipath probing. FlashRoute [79] reduces the overhead of Yarrp by first estimating the TTL, i.e., the hop count, to the destination. Our production system uses Diamond-Miner high speed probing.

More recent work has focused on the dynamics of Internet routes and the challenge this presents to obtaining up-to-date data. Cunha et al. [47] designed DTrack to predict the stability of network paths, helping to determine when to initiate new path measurements and thereby optimize the probing budget. Giotsas et al. [64] designed techniques to detect traceroute staleness, reducing the number of traceroutes to reissue. By tracing quickly, our system aims to keep stale data to a minimum.

3.3 Overview

Iris is our new measurement platform (Sec. 3.5) that exposes a REST API allowing a client to request that measurements such as Ping, Yarrp [21], or Diamond-Miner [168] be performed from distributed agents, and to obtain the measurement results. We have implemented the Zeph algorithm (Sec. 3.4) as a Python-based client that pilots Iris so as to survey the IPv4 IP-level topology of the Internet, learning to improve the probing directives that it issues over the course of successive cycles of measurements.

3.4 Zeph scheduling algorithm

In designing Zeph, we reasoned that the relatively static nature of much routing in the Internet [64] would allow a topology discovery system to learn to achieve good node and link coverage on the basis of its experience with the results of its own probing directives (Sec. 3.4.1). But experience would not be a completely reliable guide due to the existence of routing changes and their unpredictable nature [47, 46, 64], meaning that continuous exploration of new directives would also be required. In considering learning systems that combine experience with exploration so as to select among many choices that offer uncertain rewards, we turned to reinforcement learning [99]. Zeph proceeds across a series of cycles, with the directives for each cycle being based in large part upon the success of the directives that were used in the previous cycle. New directives are also tried out each cycle, with the overall aim being to improve the completeness of coverage over successive cycles. In reinforcement learning terms, the reuse of directives is *exploitation* and the trying out of new directives is *exploration*.

Zeph's challenge is to best use the probing budgets of its agents, i.e., choose which directives to issue to each agent so as to obtain the overall most complete route trace picture possible within a cycle. There are many ways of conceiving of "completeness", and the one adopted here is to maximize coverage of the traceroute-style directed links that are available to be discovered, a directed link consisting in an ordered pair of IP addresses.

Alg. 1 describes the high-level loop of the Zeph algorithm. The parameter ϵ specifies the minimum portion of each agent's probing budget that is set aside for exploration. Each cycle i of Zeph involves a series of interactions with the Iris API, culminating (line 9) with Zeph sending Iris a collection of probing directives D_i. These directives are prepared on the basis of the results R_{i-1} of the preceding cycle of probing (line 2). Following the first

Algorithm 1: Zeph

Input: ϵ : Fraction of budget per agent reserved for exploration

1 **for** $i = 1$ **to** ∞ **do**
2 $\quad R_{i-1} \leftarrow$ `ResultsFromPreviousCycle(Iris)`
3 $\quad A_i \leftarrow$ `AgentsWithBudgets(Iris)`
$\quad$ `/* Assign exploitation directives to agents` A_i `*/`
4 $\quad$ **if** $i > 1$ **then**
5 $\quad\quad D_i \leftarrow$ `Exploitation`(A_i, R_{i-1})
6 $\quad \mathbb{D}_i \leftarrow$ `PossibleExplorationDirectives(Iris)`
7 $\quad$ **for** a *in* A_i **do**
$\quad\quad$ `/* Assign exploration directives to agent` a `*/`
8 $\quad\quad D_{i,a} \leftarrow$ `Exploration`$(D_i, \mathbb{D}_{i,a}, \epsilon)$
9 $\quad$ `Probe(Iris,` D_i`)`

cycle, which consists in random directives, there are previous results to build upon, and Zeph starts assembling the collection of probing directives on the basis of exploiting those results (line 5). Once the exploitation directives have been assigned, Zeph proceeds agent by agent to round out the assignments with exploration directives (line 8). The remainder of this section describes these steps in detail.

3.4.1 Agents, directives and results

In each of its cycles i, Zeph instructs Iris's available agents A_i to follow a collection of probing directives D_i the size of which depends on the agent's probing budget. It receives from Iris in return a collection of results, which appears in the subsequent cycle, after i has been incremented, as R_{i-1}.

As Zeph operates over days and weeks, we can expect that the set of Iris agents that are available to it will vary over time; some agents will go down, and others will arrive. As it begins each cycle i, Zeph queries Iris for the currently available set of agents, A_i, along with their associated probing budgets (line 3).

The universe $\mathbb{D}_i$ of possible exploration directives that Zeph obtains from Iris for each cycle i (line 6) potentially includes all /24 prefixes extracted from all public unicast IPv4 address blocks. So as to avoid sending unnecessary probes towards non-routed prefixes, we restrict Zeph to blocks obtained from Oregon Route Views [164].

Zeph only considers ICMP Time Exceeded replies as relevant to its results,

as our survey focuses on routing infrastructure, not end-systems. Individual probe replies are assembled into traceroute-style directed links, which are pairs (v_1, v_2) of IPv4 addresses. In the event that one of the two probes did not receive a reply (in common traceroute parlance, a *star*), $(v_1, \texttt{null})$ and $(\texttt{null}, v_2)$ are also considered to be valid links. These links are matched with their initial directives, so that the results that Zeph receives consist of sets of links, each set associated with the agent and the directive that resulted in its being discovered.

3.4.2 Exploitation

If routing and routers' readiness to reply to probes were to remain unchanged from one cycle to the next, having an agent repeating its directives from the previous cycle would cause it to discover precisely the same set of links as before. Even under these ideal conditions, any given link might be discovered by multiple agents and it might be discovered multiple times by the same agent, and this redundancy potentially leaves room for improvement, as is well known from earlier route tracing work [72, 57]. If the same results can be obtained by executing fewer directives, a portion of some agents' probing budgets can be redirected towards trying out new directives. Zeph therefore sorts each agent's directives, giving highest priority to the directives that, in the prior round, are judged by a heuristic to have made the greatest contribution to link discovery. The aim of this sorting is to discard directives that the heuristic judges to make little or no marginal contribution.

From the results of the previous cycle R_{i-1}, Zeph considers only the agents that are available for the current cycle i. For each such agent, it sorts the directives, and the set of sorted directives for all agents in A_i constitutes the collection D_i (line 5). It uses the Cormode et al. Disk-Friendly Greedy algorithm (DFG) [43] as the heuristic means of sorting the directives, which is an approximation of the greedy algorithm to solve the set cover problem for large datasets. At each iteration, instead of selecting the directive that covers the most uncovered results (classic greedy algorithm), DFG groups the directives into buckets of similar size. Starting from the bucket with the directives that provide the biggest number of results, if the number of results of a directive added to the cover set of results is greater than a parametrized threshold, the results are added to C and the directive is added to D_i. DFG terminates when all prior results have been covered, i.e., $C = R_{i-1}$.

3.4.3 Exploration

When the time comes for Zeph to designate the exploration directives, the agents that were present in the previous cycle will each have an ordered set of exploitation directives that were assigned by the heuristic described above. To round out these directives, and to assign a complete set of directives to agents that were not present in the previous round, Zeph calls upon the universe of possible exploration directives $\mathbb{D}_i$ (line 6). As previously described, this consists in all of the routable /24 IPv4 prefixes. As opposed to the heuristic employed for choosing exploitation directives, where knowledge about previous results allows a directive chosen for one agent to preclude the choice of a directive for another agent, the exploration choices are made in relative ignorance of their consequences, and are therefore conducted agent by agent, considering each agent $a \in A_i$ separately (line 7).

The parameter ϵ enters into play here, to ensure that this portion of each agent's directives are reserved for exploration. If, perchance, the portion of directives assigned for exploitation exceeds $1 - \epsilon$, a sufficient number of lowest priority exploitation directives are removed to make room for exploration.

Having rounded out the probing budget of each agent with exploration directives (line 8), the cycle's collection of directives $D_i = \bigcup_{a \in A_i} D_{i,a}$ is ready to be sent to Iris (line 9).

3.5 Iris measurement platform

The Iris system, shown in Fig. 3.1, allows us to run Zeph, but it has been designed to run any sort of Internet measurement algorithm that requires access to geographically distributed probing agents. At the moment, three tools are available in Iris: Diamond-Miner [168], Yarrp [21], and Ping. More tools can easily be added in the future. Moreover, Iris works the same with IPv4 and IPv6 addresses.

Workflow A client such as Zeph submits an HTTPS request to run a measurement via Iris's Rest API. The API informs the message broker Redis [145] of the measurement request. The message broker chooses an inactive worker from an available pool. This worker maintains the state of the measurement throughout its life in the system. The worker registers the measurement parameters in the ClickHouse [37] database and asks the agents to perform the measurement. When the measurement is completed by an agent, it sends the results to an object storage MinIO [120], an open-source alternative to Amazon S3 [7]. Then the worker pulls the results

from the object storage and inserts the results into the database. When the measurement is finished, the worker updates the measurement state to mark it as ready to be pulled by the client. All of the components described above generate logs that are stored with a monitoring stack built from the combination of Prometheus [142] for storing the system's metrics, Loki [74] for storing the system's logs, and Grafana [73] to allow visualizations of these metrics and logs.

Fig. 3.1 shows this workflow visually. Arrows symbolize the connections between the components (e.g., the worker connects to the database). Purple shows dataplane flow (e.g., compressed CSV files) while green shows control plane flow (e.g., measurement parameters to execute a measurement tool on an agent). Yellow shows log and metric collection.

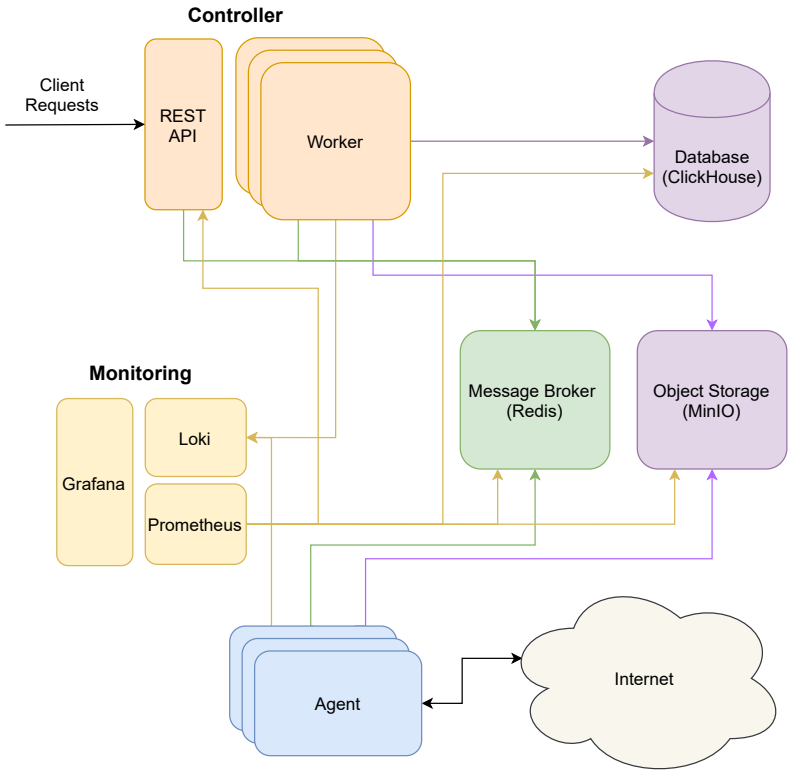


Figure 3.1: Iris architecture with arrows indicating which component initiates each connection; each box is a self-contained Docker container. Colors indicate the type of data flows: purple for dataplane, green for control plane, and yellow for logs.

3.5.1 Design considerations

The Iris was designed to meet the following demands:

1. As Zeph measurements can last for days, Iris needs to be robust to agents crashing, or being unreachable, and provide the ability to restart an agent's measurements if and when it returns (*resilience*).

2. Zeph will work from as many vantage points as it can access, so Iris should scale well with the number of agents (*agent scalability*).

3. Zeph supports Internet scale high speed topology measurement techniques, generating billions of ICMP replies. Iris should scale well with this amount of data (*data scalability*)

4. We continue to improve Zeph, so Iris should support easy deployment of new control algorithms and probing software (*continuous delivery*).

5. To improve affordability and maintainability, Iris should be built as much as possible from free open-source software (*maintainability*).

We chose Docker [54], and Redis to improve the resilience of the system: each of Iris's components runs in its own Docker container. Moreover, a failed container is automatically reintegrated without external intervention and retrieves the measurements states from Redis. Two aspects of Redis improve the message broker's resilience. First, its *persistence* feature [146] regularly saves its own current state to disk, preserving context in case of failures and restarts. Second, it maintains connection state, which Iris uses to alert workers to the departure of any agent. Any worker with an ongoing measurement stops waiting for that agent to send data, and no further measurements can be requested of the agent until it reconnects to the broker. A new worker is automatically created from the worker pool to carry on the measurement algorithm from the point where the failed worker had left it.

We chose Redis and MinIO for agent scalability: Redis can handle millions of simultaneous connections; MinIO has proven capable, with Zeph, of supporting transfer of data files in the hundreds of gigabytes;

We chose the ClickHouse database for data scalability: it is a database optimized for insert and read operations, which are the only operations Iris perform on the result data. ClickHouse has supported tables containing up to 5 billion rows, and the in-base calculation language provided by ClickHouse's *arrays* [36] feature reduces computation times.

Redis, MinIO and ClickHouse scale "horizontally", meaning that it is possible to deploy multiple message brokers, multiple object storage containers,

and multiple replicas of the database to support a larger number of agents. Scaling beyond one instance each was not necessary in order for Iris to support Zeph, but the potential is there.

We chose Docker for continuous delivery: As we move forward, this will ease large-scale deployment by lifting many constraints on the machines and VMs that can host an Iris agent. Also, Iris takes advantage of such containers' ability to run unchanged over a wide variety of operating systems.

Finally, all the components of the system are free and open source, improving maintainability.

3.6 Evaluation

There are three main results: (1) Zeph, requesting Diamond-Miner multipath route traces from Iris, provides the most comprehensive view to date of the IPv4 Internet in terms of nodes and links discovered in a short period of time. Once it has been trained, a single cycle of Zeph discovers more than 3 times as many nodes and 10 times as many links as does a single cycle of the state-of-the-art Ark platform (Sec. 3.6.3); (2) When compared on single-path route traces, as performed by Ark, Zeph's reinforcement learning approach outperforms Ark's random exploration strategy (Sec. 3.6.2); and (3) Zeph saves 50% of the probing budget, dividing the probing time by 2, compared to an exhaustive strategy of tracing multipath routes towards every prefix from every agent (Sec. 3.6.3), while maintaining nearly the same number of discoveries.

3.6.1 Vantage points and setup

All of the measurements are run on the EdgeNet [44] platform with nodes hosted in 5 different Google Compute Engine (GCE) [67] zones: `asia-east2-a` (Hong Kong), `asia-northeast1-a` (Tokyo), `asia-southeast1-a` (Singapore), `europe-west6-a` (Zurich) and `southamerica-east1-a` (São Paulo). The measurements can be fully reproduced on nodes in the same locations by applying our open-source evaluation code. The instances use the *standard* network tier which allows the packets to exit to the Internet as soon as possible, instead of the default *premium* tier which privileges Google's internal network to the public Internet.

3.6.2 Topology discovery

We perform two experiments to evaluate Zeph's performance: (1) a comparison of Zeph's prefix allocation strategy with other approaches, performed from the same agents, with common parameter settings; and (2) a comparison of the maximum raw discoveries by Zeph on Iris against Ark system for the same number of prefixes probed and the same number of probes sent.

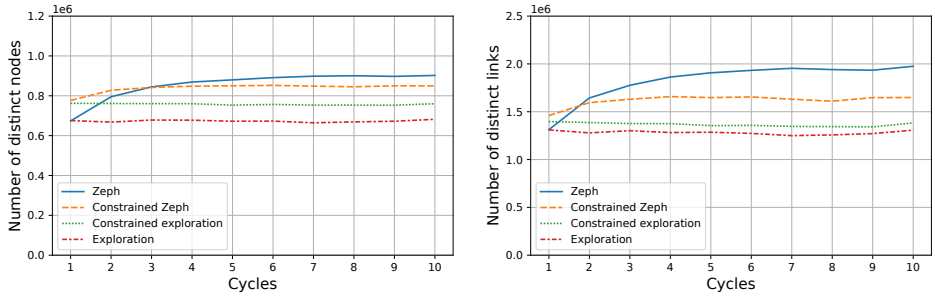
Zeph's reinforcement learning approach outperforms random allocation

We compare Zeph's prefix allocation strategy with that used by the state-of-the-art topology discovery system Ark [35]. Ark applies what we call *constrained random allocation*, which consists in allocating the /24 prefixes of the IPv4 space uniformly at random over the different agents, each prefix being probed by exactly one agent (the constraint). Zeph's reinforcement learning approach splits the probing directives of each agent into two subsets for exploitation (Sec. 3.4.2) and exploration (Sec. 3.4.3). Zeph exploration differs from the Ark's random allocation by allowing a same prefix to be probed by multiple agents. We call Zeph's exploration strategy *unconstrained random allocation*.

Experiment We run 10 cycles of different combinations of probing techniques simultaneously: exploitation and unconstrained random allocation (Zeph with $\epsilon = 0.1$), constrained random allocation (Ark's approach), unconstrained random allocation, and exploitation and constrained random allocation ($\epsilon = 0.1$). Unconstrained random allocation allows us to evaluate Zeph's exploration alone, while exploitation and constrained random allocation allow us to evaluate Zeph's exploration in combination with exploitation.

On August 4th, 2021, we extracted 11.9M /24 prefixes from the routed prefixes provided by Route Views. Dividing these among the 5 agents, at each cycle, and for all four approaches, each agent probes 2.4M prefixes. We run the approaches simultaneously and the agent for each approach probes at 100,000 packets per second using single path tracing [21] with ICMP probes up to TTL 32. Each cycle takes around 15 minutes to complete, running from August 5th to 6th, 2021.

Results Fig. 3.2 shows the results of the different strategies. The main result is that exploitation together with exploration using unconstrained random allocation (Zeph's approach) outperforms all of the other approaches



(a) Nodes discovered

(b) Links discovered

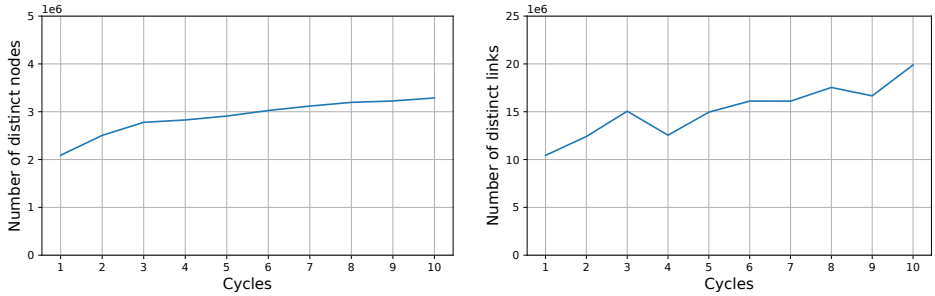
Figure 3.2: Zeph outperforms random exploration strategies.

once Zeph has had ten cycles during which to learn. In particular, it discovers 18% more nodes and 42% more links than constrained random allocation (Ark's approach). The other result is that constrained random allocation outperforms unconstrained random allocation by 12% more nodes and 6% more links over each cycle. This highlights that it is the combination of the exploitation and the unconstrained allocation together that allows Zeph's strategy to perform well.

Zeph/Iris conducting multipath traceroutes perform competitively with respect to the current state-of-the-art Internet scale topology discovery system.

In the previous section, we have shown that Zeph's prefix allocation strategies outperforms others when limited to the same single-path probing budget. But Zeph and Iris are capable of discovering more than what is shown in Fig. 3.2. To use their full capacity, we perform 10 cycles of measurement with multipath route traces (i.e., capturing the load-balanced paths) obtained by Diamond-Miner [168] with ICMP probes at 100,000 pps. We retrieved routed prefixes from Route Views on January 21st, 2022 and broke them down into 12M /24 prefixes, each of the five agents receiving a per-cycle budget of 2.4M /24 prefixes. The measurements were gathered from January 22th to 28th, 2022. Each cycle took between 10 hours, 12 minutes (cycle 1) and 15 hours, 14 minutes (cycle 10) to complete.

Fig. 3.3 shows that Zeph also works with multipath traceroutes: the number of nodes improves by 57% (+1,2M) the number of links improves by 90% (+9.4M) between cycle 1 and cycle 10. Note that an agent crashed at cycle 4, reducing the number of links found in that cycle, but Zeph adapted



(a) Nodes discovered (b) Links discovered

Figure 3.3: Node and link discoveries in multipath probing

and the discoveries resumed increasing in cycle 5.

Table 3.1: Comparison of node and link discoveries between the tenth cycle of Zeph + Iris, and a cycle of the Ark platform for the same number of prefixes probed (12 million, second row) and the same number of probes sent (12 billion, third row).

	Time	Nodes	Links
Zeph + Iris	15h15	3,288,325	19,890,422
Ark (prefixes)	19h12	1,009,738	2,087,903
Ark (probes)	45 days	3,597,042	9,241,146

Finally, we compare raw discoveries of Zeph + Iris and Ark platform. Tab. 3.1 shows the number of nodes and links discovered during the last Zeph cycle and the number and nodes and links discovered by CAIDA's Ark [30] platform (1) when the same number of prefixes is probed, and (2) when the same number of probes is sent. Zeph with Iris takes 20% less time to probe all routed /24 prefixes but discovers more than 3 times more nodes and 10 times more links. Moreover, Ark discovers 10% more nodes and around two times fewer links with the same number of probes sent, but takes 45 days instead of less than 15 hours to do so.

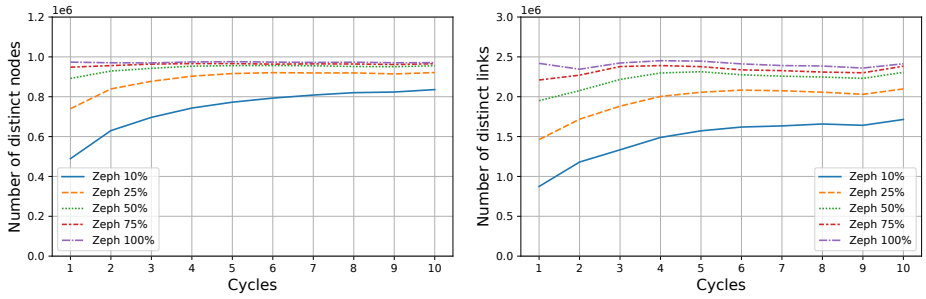
3.6.3 Zeph probe savings

This section describes the tradeoff between reducing the number of prefixes probed by each agent (and therefore reducing the duration of a cycle and the number of probes sent) and discovery.

Experiment

We collect 10 cycles of 4 Zeph runs where each agent probes 10%, 25%, 50% or 75% of the routed /24 prefixes. In addition, we run an *exhaustive* measurement, with all the agents probing 100% of the routed /24 prefixes. The measurements for each budget were performed from August 2nd to 4th, 2021. We use the same Route Views data as in Sec. 3.6.2. Each agent runs Yarrp at 100,000 pps with ICMP probes. Depending on the budget, each cycle lasts between 10 minutes and 1 hour and 30 minutes. In this experiment, we reduce the number of prefixes that are probed by each agent to simulate a more constrained budget but keep a high probing rate. We could also reduce the probing rate, but this would have significantly increased the time of the experiment.

Results



(a) Nodes discovered

(b) Links discovered

Figure 3.4: Number of nodes and links discovered over Zeph cycles for different budgets: Zeph with 50% finds almost the same number of nodes and links as with 100% of the budget.

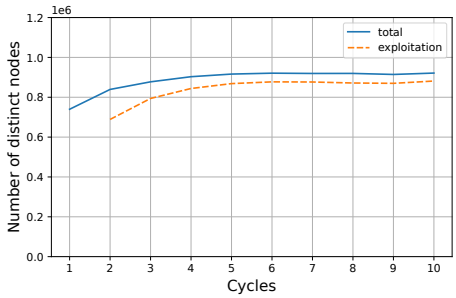
Fig. 3.4 shows the number of nodes and links discovered by all the agents together for the different budgets. The main result is that Zeph discovers 98% of the nodes and 95% of the links that the exhaustive approach discovers when just 50% of the prefixes are probed by the agents. The 25% (10%) curves show that even with a significantly reduced probing budget, Zeph is able to discover 94% (87%) of the nodes and 86% (71%) of the links. Additionally, reducing the number of probes also reduces the time of a cycle. With 50% (25%, 10%) of the prefixes, 10 cycles took 7.4 hours (3.6, 1.5), compared to the 12.5 hours of the exhaustive approach.

3.6.4 Reinforcement learning analysis

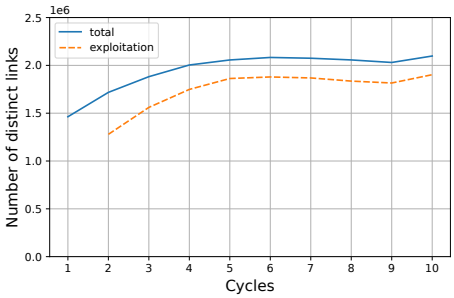
Finally, we dive into one measurement of Sec. 3.6.3 where the budget is 25% of the routed prefixes, to understand more about the contributions of exploitation and exploration.

Exploitation and exploration budgets

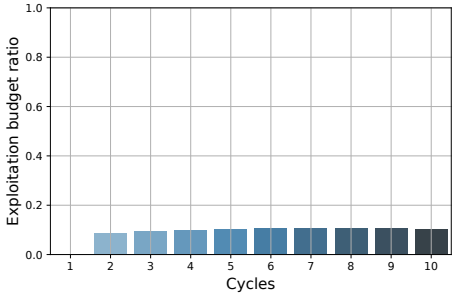
In choosing a reinforcement learning approach for Zeph, we anticipated that many of each agent's directives could be repeated (exploitation) from one cycle to the next, and that complementing these directives with new ones (exploration) would aid in improving overall discovery. We find that the exploitation directives were indeed capable of discovering most of the links previously discovered, and that exploration did indeed lead over time to better overall discovery.



(a) Nodes discovered



(b) Links discovered



(c) Fraction of the budget allocated to exploitation

Figure 3.5: Exploitation counts for 95% of the nodes and 90% of the links discoveries.

Fig. 3.5 shows the number of nodes (Fig. 3.5a) and links (Fig. 3.5b) that are discovered by exploitation and in total. The main result is that reinforcement learning works: exploitation improves over time, going from 739k nodes at cycle 2 to 921k nodes at cycle 10 (+25%) and 1.2M links at cycle 2 to 1.9M links at cycle 10 (+48%). In Fig. 3.5c, exploitation is responsible for most of the discoveries, i.e., 95% of the nodes and 90% of the links. Interestingly, although we allocated 90% of the budget to exploitation, Zeph actually used only 10% for it. We interpret this result as a consequence of the high redundancy of Internet paths, and leave the study of the optimal value of ϵ for future work.

3.7 Conclusion

Zeph is a new algorithm for distributed tracing at the IP level of the routes that packets take through the IPv4 Internet. It learns the probing directives to allocate to the vantage points in order to maximize topology discovery. Zeph is platform agnostic, and independent of the probing tool used and the agent's capacities.

Iris is a distributed Internet measurement system based on a modern resilient architecture that exposes an API that allows various algorithms, including Zeph, to be run. Together, Zeph and Iris discover 3 times more nodes and 10 times more links than the state-of-the-art Ark platform for the same number of prefixes probed.

All of the code of Iris and Zeph and data of the evaluation are publicly accessible and we now offer regular Zeph Internet topology data series to the community and the ability to perform one's own measurements.

In future work, we will extend Zeph to IPv6 and analyze in depth the dynamics of the Internet topology.

Chapter 4

Longitudinal study of a geolocation database

IP geolocation is one of the forms of metadata that can be used in MetaTrace (Chap. 5) to leverage the large amount of data collected by Iris and Zeph (Chap. 3), or other in-use probing platforms. In this chapter, we present a 10-year data study of a widely used IP geolocation database, MaxMind. We find that significant differences can exist even between two successive weekly snapshots, a previously underappreciated source of potential error for analysis that use this source of data.

4.1 Introduction

Determining the physical location of Internet hosts is important for a range of applications including, but not limited to, advertising, content and language customization, security and forensics, and policy enforcement [90, 80, 172]. However, the Internet architecture includes no explicit notion of physical location and hosts may be unable or unwilling to share their location. As a result, the process of third-party IP geolocation – mapping an IP address to a physical location – emerged as a research topic [129] more than two decades ago and has since matured into commercial service offerings [118, 83, 82].

IP addresses represent network attachment points, thus IP geolocation is often inferential. Commercial geolocation providers compete, so the methodologies for creating their databases are proprietary. State-of-the-art techniques include combining latency constraints [75], topology [90], registries [129], public data [59], and privileged feeds [96].

This work takes a fresh look at IP geolocation data from a *temporal*

perspective. Specifically, we examine the longitudinal stability of locations in an IP geolocation database, the characteristics of location changes when they do occur, and the extent to which a particular instance of a geolocation database impacts conclusions that depend on locations. To wit, network and systems researchers frequently utilize available IP geolocation database snapshots. However, the date of the snapshot may only loosely align with the time of the lookup operation, or the lookups may span multiple snapshots, e.g., a long-running measurement campaign. We show that snapshots of the same geolocation database separated even closely in time can have a non-trivial effect on research results and findings.

For example, across database snapshots in a three month window, we find up to 22% of IP addresses move more than 40km, while coverage (the simple presence or absence of an address in the database) varies by as much as 18%. Despite this temporal sensitivity, the *date* of the geolocation database snapshot is rarely reported in the academic literature – an omission that we show confounds scientific reproducibility.

We use 10 years of data from the most popular, publicly available, and frequently used database: MaxMind [118]. We use this large collection of snapshots to examine the longitudinal evolution of its location mappings and address coverage, as well as to conduct a reproducibility case study. Our contributions include:

- A survey of how recent systems and networking literature utilizes and depends on IP geolocation data.
- The first longitudinal study of a widely used IP geolocation database where we find significant short-term dynamics.
- A case study of prior research that depended on geolocation, showing that the results fundamentally differ based on the instance of the geolocation database used.
- Recommendations for the sound use of IP geolocation data in research.

Our findings provide several tangible lessons for the broader network research community:

- IP locations in geolocation databases can be highly dynamic, with non-negligible coverage and movement differences even over short (< 3 month) time scales.

- Despite this variation, published network research frequently omits specific details of the geolocation snapshot. Not only does this hinder reproducibility, research results that depend on IP geolocation can significantly differ depending on the instance of the snapshot used.
- Researchers should ensure they publish details of the IP geolocation database, align lookups with measurements, and investigate the sensitivity of their results to different instances of the database.

4.2 Related work

Mapping IP addresses to the physical world is an important topic that has seen two decades of research. Early efforts used landmarks, hosts with known position, to assign locations to unknown targets at coarse granularity [129]. Landmark-based geolocation was subsequently enhanced to use latency constraints [75], network topology [90], and population densities [59] to improve accuracy. Because the accuracy of latency-based techniques is often proportional to the distance between the target and its nearest landmark, Wang et al. developed techniques to find and utilize additional landmarks [172].

IP geolocation has since matured, with several competing commercial offerings including [118, 83, 82]. While the exact methodology of these commercial services is proprietary, they likely use a combination of databases (e.g., whois and DNS), topology, latency, and privileged data feeds from providers [96].

Even so, the inference-based nature of IP geolocation imparts errors and inaccuracies even in commercial databases [80, 155], demonstrated by several prior analyses. For instance, Poese et al. found 50-90% of ground-truth locations to be geolocated with greater than 50km of error [139]; most recently Komosnỳ et al. studied eight commercial geolocation databases and found mean errors ranging from 50-657km [97]. Geolocation of network infrastructure, including routers, is known to be particularly problematic [81, 61]. However, as shown in Sec. 4.3, MaxMind is still widely used, for the simple reason that there exists no other alternative than geolocation database to get an Internet scale IP geolocation mapping.

Our work looks at IP geolocation through a novel lens by analyzing the longitudinal characteristics of a popular geolocation database. By showing the stability of locations at different granularities and timescales, we offer a first look at the error bounds for particular classes of applications that utilize geolocations, as well as offer practical lessons for consumers of IP geolocation data.

4.3 Motivation

To better understand IP geolocation as used in the network and systems research community, we surveyed the academic literature. We performed full-text queries, over all time, on four popular digital libraries for three common geolocation databases, MaxMind [118], NetAcuity [127], and IP2Loc [82]. Tab. 4.1 shows the number of papers in each library. MaxMind is clearly the most popular by an order of magnitude. Therefore the analysis in the reminder of this work focuses on MaxMind.

4.3.1 MaxMind

Founded in 2002, MaxMind is a commercial entity specializing in IP geolocation and related services. MaxMind offers two IP geolocation databases, one that is free (GeoLite) and one that requires a license (GeoIP). The academic literature uses both GeoLite and GeoIP. GeoLite is available as a complete database "snapshot". Snapshots are currently updated weekly and available for public download. GeoLite snapshots contain variable length IP prefixes, each with an associated geolocation. The geolocation may include country, city, latitude/longitude, and accuracy (in km); however many prefixes only provide a geolocation at the country granularity. This work studies the IPv4 GeoLite databases. Henceforth, we refer to GeoLite (and its successor, GeoLite2) informally as "MaxMind" for simplicity.

4.3.2 Survey Methodology

We characterized the use of MaxMind across nine systems, security and networking conferences during the five year period from 2016-2020. To find papers in the literature using MaxMind, as well understand how it is used, we adopt a semi-automatic method: first, for a given conference venue, we obtain the complete proceedings and perform a case-insensitive search for the string "maxmind." We manually inspect each paper found to contain "maxmind" to determine whether the work utilizes the database or is simply referencing MaxMind. For example, in [98], "maxmind" appears only as a citation to the sentence "Current IP-based geolocation services do not provide city-level accuracy..." Only those papers that used MaxMind's database for their research are included.

Keeping in mind the variety of research questions and geolocation requirements inherent in the various papers, we sought to distinguish what was being geolocated and at what granularity. We manually extract from each paper

Table 4.1: References to geolocation databases

	ACM	IEEE	arXiv.org	Springer
MaxMind	171	373	96	162
NetAcuity	10	10	8	7
IP2Loc	3	3	0	0

Table 4.2: Literature survey of MaxMind use in academic venues (2016-2020). "Affected" column specifies if MaxMind was used in methodology (Y) or for validation (V).

			MaxMind							**Affected**		**Snapshot date specified**		**Free (F) Paid(P) (N/A)**		
			Granularity			**IP type**										
Conference	**Area**	**Papers**	AS	Country	City	All	End user	End host infrastructure	Router	Y	V	Y	N	F	P	N/A
IMC	Meas.	16	1	13	3	2	5	8	1	12	4	1	15	8	3	6
PAM	Meas.	6	0	2	4	1	3	2	0	5	1	0	6	3	1	2
TMA	Meas.	4	1	0	3	3	0	1	0	3	1	2	2	1	2	1
USENIX Sec	Security	10	0	7	3	0	4	7	0	10	0	2	8	1	4	5
CCS	Security	6	2	1	3	0	1	2	3	6	0	0	6	2	3	1
SIGCOMM	Systems	3	0	1	2	0	3	1	0	2	1	0	3	0	3	0
NSDI	Systems	1	0	0	1	0	0	0	1	1	0	0	1	0	0	1
CoNEXT	Systems	2	1	1	0	1	0	1	0	2	0	0	2	1	0	1
WWW	Web	10	0	7	3	0	4	7	0	10	0	2	8	2	2	6
Total	All	58	7	30	22	8	22	28	6	51	7	6	52	18	20	23

the granularity required (country, city, or AS) and the type of IP addresses geolocated (all, end users, end host infrastructure, and router). The "end user" category contains IP addresses belonging to residential users (e.g., [128]), or, more broadly, end users issuing web traffic (e.g., [131]). The "end host infrastructure" category includes addresses belonging to Internet infrastructure, typically web [51], proxies [174], or DNS [136] servers. "Routers" include the IP addresses of network router interfaces. Finally, the "all" category contains papers that geolocate all types of addresses such as [100, 176]. Note that these sets are mutually exclusive, but a paper can use MaxMind on several types of IP addresses. For instance, [9] studies the Mirai botnet where the infected IP addresses can belong to both end users and end host infrastructure.

4.3.3 MaxMind in the Literature

Tab. 4.2 summarizes our findings. We follow the rhetorical structure of Scheitle et al. [152] to classify the impact of MaxMind on the paper's results.

- Affected "Y" are papers that use MaxMind in their methodology to obtain a result. For example, Papadopoulos et al. [131] use MaxMind to build a classifier to infer how much advertisers pay to reach users.

- Affected "V" are papers that do not use MaxMind to obtain results,

but rather to *compare* their results. For example, Weinberg et al. [174] compare their inferred proxy locations to MaxMind's locations.

The "Date" column indicates whether or not the paper explicitly provides the MaxMind snapshot date. The last column indicates which MaxMind version is used: either free, paid or if the info was not available.

From a macro perspective, MaxMind is both used at country (53%) and city (37%) granularity. Second, it is mostly used to geolocate end users (38%) and end host infrastructure (49%) rather than routers (9%). Then, the majority of papers (86%) use MaxMind to obtain results, and few (11%) provide the snapshot date. Finally, we see that free and paid version of MaxMind are equally used by the community. Note that the totals do not sum to the number of papers as, e.g., a paper may use MaxMind for both AS and country information [100], or use both the free and paid version [61].

Lesson MaxMind is the most popular geolocation database to support other research. Further, the results of many papers may be sensitive to geolocation variation, especially given the lack of snapshot dates, large windows of measurement or data, and no explicit alignment between data collected and the geolocation snapshot.

4.4 Metrics

This section defines metrics used to characterize the impact of selecting one geolocation database snapshot rather than a different snapshot in time. We assume that snapshots contain IP prefixes and their associated locations. For all IP prefixes, we expand them to their constituent set of individual addresses.

We define metrics using two concepts for comparing two geodatabase snapshots: *coverage difference* and *distance distribution*. For these definitions:

- Let A be a set of IPv4 addresses (either all address, or a population e.g., addresses known to be router interfaces).
- Let L be the set of locations present in a geodatabase (either latitude-longitude pairs, cities, or countries).
- Let M be a snapshot of this database, defined as a set of pairs (a, l) that map addresses to locations.
- Let $A_M \subset A$ be the set of addresses that appear in snapshot M that belong to population A.

4.4.1 Coverage difference

Intuitively, coverage difference means the portion of IP addresses that appear in one snapshot or another, but not both. Two identical snapshots have a coverage difference of zero while the coverage difference is one for two snapshots with no IP addresses in common.

Formally, coverage difference is an extension of the concept of 'coverage', which for snapshot M with respect to a set of IPv4 addresses A is:

$$\text{coverage}(M) = \frac{|A_M|}{|A|} \tag{4.1}$$

The coverage difference between two snapshots M_i and M_j on A is the Jaccard distance between A_{M_i} and A_{M_j}:

$$\text{covdiff}(M_i, M_j) = \frac{|(A_{M_i} \bigcup A_{M_j}) - (A_{M_i} \bigcap A_{M_j})|}{|A_{M_i} \bigcup A_{M_j}|} \tag{4.2}$$

4.4.2 Distance distribution

An address a can appear in a location in one geodatabase snapshot and different location in a second snapshot. Let $\text{dist}(a, M_i, M_j)$ be the Haversine distance [32] between the locations of a in M_i and M_j, using latitude-longitude values for each location. The distance distribution between the two snapshots is the set of distances, one for each address that appears in both snapshots:

$$\text{D}_{\text{dist}}(a, M_i, M_j) = \{\text{dist}(a, M_i, M_j) \mid a \in A_{M_i} \bigcap A_{M_j}\} \tag{4.3}$$

The definitions above serve to define the differences between two snapshots. Because we are not interested in only comparing two snapshots, but also knowing the average difference between two snapshots or the worst case, we need to compare those differences. For coverage, we can sort the pairs of snapshots by their Jaccard distance. To compare two distance distributions, we define the following metric:

$$\begin{aligned} \text{distdiff}(\text{M}_\text{i}, \text{M}_\text{j}) = (\text{mean}(\{&\log_{10}(\text{dist}(a, M_i, M_j)) \\ &\mid a \in A_{M_i} \bigcap A_{M_j}\}) \end{aligned} \tag{4.4}$$

By taking the mean of the log of the distances, we prevent outliers (e.g., distances potentially up to 20,000 km) from disproportionately outweighing lower, but nonetheless meaningful, distances, e.g., on the order of 100 km.

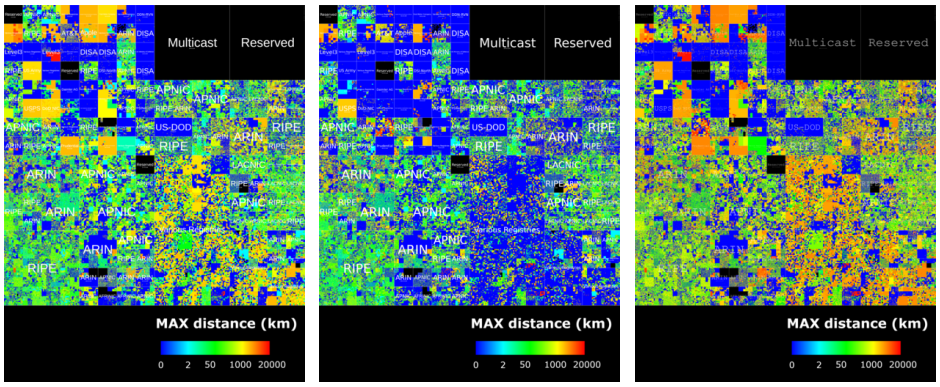


Figure 4.1: Hilbert heatmap of maximum distance (Eq. (4.6)) in 2018 (left) 2019 (center) and absolute difference (right) for the entire IPv4 address space (log scale; each pixel represents a /24). MaxMind exhibits a high degree of global geolocation dynamics and year-to-year variation.

Note that while the median is a more robust statistic, typically more than 50% of the address have zero distance, i.e., did not move between snapshots (Sec. 4.6.3). Thus, the mean provides a meaningful non-zero measure. Other metrics that we define on the distance distributions are the quantiles of a distribution, along with the maximum value.

For both coverage and distance, the higher the metric, the larger the difference is between the two snapshots.

4.4.3 Distance

Our survey looks at the distribution of distance values per address. We define the maximum distance of an address a as being the maximum distance between two of its locations. Formally, the distribution of distances of a is:

$$D(a) = \{\text{dist}(l_i, l_j) : l_i, l_j \in \mathcal{L}_a\}\} \tag{4.5}$$

where $\mathcal{L}_a$ is the list of locations of a in a considered set of snapshots. The maximum distance of a is then:

$$\max(D(a)) \tag{4.6}$$

4.5 Data

4.5.1 MaxMind snapshots

We collect 214 MaxMind snapshots spanning the ten year period from January 2010 to December 2019. There are two primary challenges in the raw data: (1) the snapshots we obtain are not uniformly distributed in time; and (2) IP addresses appear within prefixes of different networks and lengths over time. To utilize this data within the framework of our methodology and metrics, we pre-processed it.

Sampling the snapshots for time uniformity

Sec. 4.4 assumes a uniform distribution of snapshots in time. Our evaluation examines a ten year span from 2010-2019. Within this ten year period, we have at least one snapshot per month, but sometimes as many as one snapshot per week. Therefore to ensure uniformity, we simply down-sample so that the ten year period includes one snapshot per month. Our evaluation is conducted on this subset of snapshots such that they are uniformly distributed in time.

Prefixes of different lengths

A MaxMind snapshot contains a mapping of prefix blocks to geolocation. However, these prefixes may split, be aggregated, or even overlap in time. While our analysis is at the per-IP address granularity, rather than prefix, maintaining the geolocation for all IP addresses over time is inefficient. Our first step then is to find a data structure to efficiently store and query the snapshots. Over all prefixes in all snapshots, we construct the set of covering longest length prefixes and construct a Patricia trie [158]. We build one Patricia trie for each geolocation granularity: country, city, and coordinates. The Patricia trie contains, per prefix, all its locations over the period of time.

To handle prefix variation over time, we insert into the Patricia trie the longest prefixes seen in the snapshots. The resulting fine-grained prefixes will be inserted in a database. As an example, consider the prefix 1.0.0.0/23 located in London in the snapshot s1. On the other hand, in the snapshot s2, the prefix 1.0.0.0/24 is located at London and the prefix 1.0.1.0/24 is located at Paris. We place the two /24 prefixes in the trie for each of the snapshots, London being inserted for the two prefixes of the snapshot s1.

The prefixes considered in our database therefore do not necessarily correspond to BGP prefixes nor are the same as the initial prefixes on

MaxMind snapshots. The resulting prefix lengths vary between /9 and /32, the most common being /29 with 19.2% of the total prefixes.

4.5.2 Different types of IP addresses

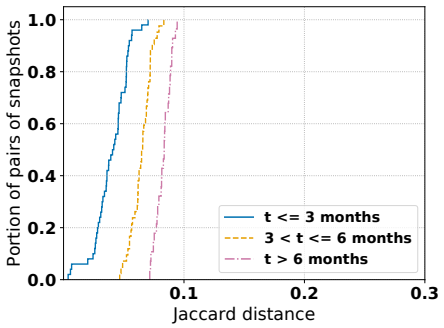
Sec. 4.3 has shown that researchers use MaxMind to locate three classes of IP addresses: end users, end hosts infrastructure and routers. We therefore collect and label three sets of IP addresses corresponding to these three types.

- **End users:** M-Lab [110] performs and records measurements to end users requesting performance tests (i.e., a "speedtest"). From the M-Lab public datasets we extract targets in the year 2019. We randomly sample these targets to obtain 6.7M IPv4 addresses in approximately 2M unique /24 prefixes.
- **End host infrastructure:** For end host infrastructure, we extract the daily top list made available by [152]. We perform an intersection of all 2019 lists in order to minimize the number of IP addresses that could be reassigned for other purpose. Because these top lists are volatile, our filtering for high-confidence end host infrastructure addresses produces 26,231 IP addresses in 16,942 /24 prefixes.
- **Routers:** We leverage both CAIDA ITDK dataset [28] and Diamond-Miner [168] public Internet topology datasets to collect IP addresses belonging to router interfaces. Both datasets are the result of Internet-wide traceroute style probing. We take the intersection of 2019-01, 2019-04 ITDK and 2019-08 Diamond-Miner datasets and obtain 730k IP addresses in more than 177k different /24 prefixes. By taking the intersection over time, the aim is again to ensure the likelihood that the addresses indeed belong to routers.

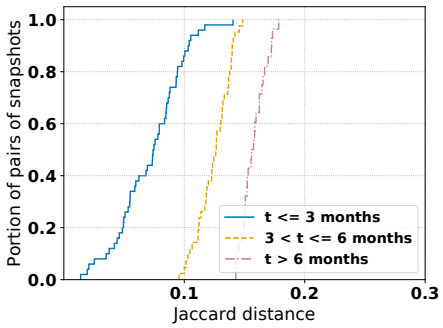
4.5.3 Ethical Considerations

Our work does not involve human subjects, questionnaires, or personally identifiable information, and, hence, does not meet the standards for IRB review. The MaxMind data we analyze is covered by the Creative Commons Attribution-ShareAlike 4.0 International (CC BY-SA 4.0) license which permits adaption of the database: "remix, transform, and build upon the material for any purpose, even commercially." Beginning in January 1, 2020, MaxMind adopted a more restrictive policy in order to comply with GDPR requirements [119]; our research does not analyze any data after 2019.

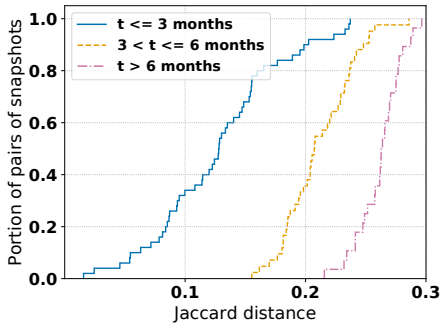
4.6 Evaluation



(a) End users



(b) End host infrastructure



(c) Routers

Figure 4.2: Comparing pairs of database snapshots by city coverage difference (Eq. (4.2)). Across all classes of IP addresses, there are significant coverage differences, even on among snapshots closely separated in time.

This section presents an evaluation of the MaxMind data snapshots from 2018 to 2019 using the metrics defined in the methodology. We examine the extent and impact of both location movement and coverage across several dimensions. Primary results are presented here, while a more exhaustive evaluation, including the full 10 years of longitudinal data, is available in an accompanying technical report [69].

4.6.1 Limitations

Our primary contribution in this work is to define metrics that characterize the dynamics of IP geolocation databases, and understand how these dynamics

can impact network research that depends on geolocation. We keenly recognize that location changes within the database may be genuine or may be artifacts of the geolocation system's methodology. We do not investigate the root causes of the dynamics we observe. Indeed this limitation is fundamental – MaxMind's algorithm is proprietary so that we cannot provide true causal analysis. Rather, we focus on providing lesson for how researchers should view and utilize geo-databases.

4.6.2 Visualizing Internet-wide geo movement

Fig. 4.1 shows an exhaustive representation of the maximum distance change for each /24 of the entire IPv4 space for 2018 and 2019, as well as the absolute difference between the two years. Each pixel represents a /24, and the color represents the maximum distance between two locations; black pixels indicate that the IP address is not present in the database. If the /24 contains more specific entries in the MaxMind database, we take the maximum of the maximum distance of the IP addresses within the /24.

We see that the visualization of 2019 differs from 2018: Many of the various registries in the bottom center right and top left part of the plots have a maximum distance of more than 1000km in 2018, whereas they did not move in 2019. There is also a red square in the prefixes belonging to Level3, that had a maximum distance of 20,000km in 2018 but did not move in 2019. Surprisingly, there are also some IP addresses that were covered in 2018 (i.e., in this case, having lat/long coordinates) which are not covered in 2019. This is the case of some blocks of IP addresses in the bottom center left of the graph belonging to APNIC and AFRINIC.

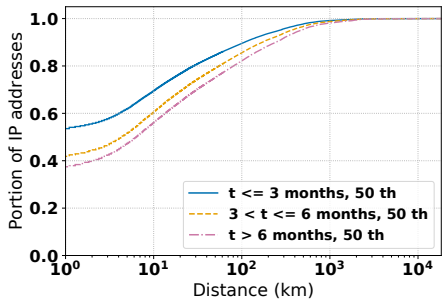
All these differences between the two years are highlighted by the map on the right: we clearly see the center and the bottom left mainly colored in orange and red as well as some big prefixes on the top right. It reveals a significant dynamic change not only along the prefixes but also through time.

Overall, by looking at the Hilbert representations of each year over the 10 years dataset, it is difficult to perceive a trend that could lead us to say that prefixes are experiencing bigger or smaller distance change over years.

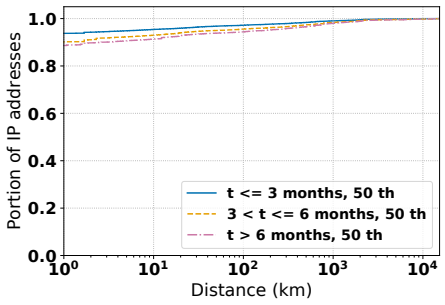
Lesson These visualizations confirm not only the **high degree of global geolocation dynamics**, but also the presence of **year-to-year variation in geolocation movement**. An IP address can experience a maximum distance change of 0km in 2018 and more than 1000km in 2019, and vice versa.

4.6.3 Impact

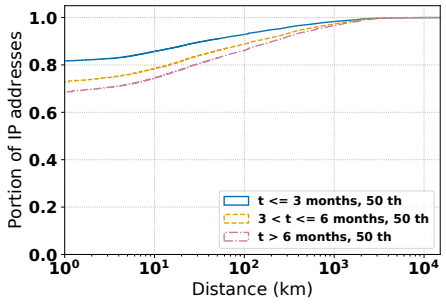
While the preceding analysis demonstrates how our metrics can shed light on the underlying dynamics of a geolocation database, we conclude this section with an analysis of the potential *impact* of selecting a particular snapshot of MaxMind versus a different snapshot, for instance as a researcher seeking to geolocate a population of IP addresses under study. To bound our results, we compare pairs of snapshots from 2019 within three time windows: when the snapshots differ by less than 3 months, between 3 to 6 months, and between 6 to 12 months. We evaluate the impact across the three IP classes: end users, end host infrastructure, and routers.



(a) End users



(b) End host infrastructure



(c) Routers

Figure 4.3: Comparing pairs of database snapshots by IP address distance difference (Eq. (4.3)). Up to 22% of addresses move more than 40km among snapshots in a three month window.

Coverage (Fig. 4.2)

For coverage we show results at the city level as we find no country level coverage differences between snapshots; almost all IP addresses, across all classes of addresses, have a country geolocation present in the database.

Not unexpectedly, for all types of IP addresses, we observe that the coverage difference increases with time between the two snapshots. As shown in our tech report (Fig.5 of [69]), the overall coverage is globally constant, therefore this cannot be imputed to an increase of the total coverage.

We see that even for two snapshots created within less than three months of each other, there is a significant coverage difference, up to 6%, 11% and 20% for end users, end host infrastructure and routers respectively. As seen in Fig. 4.2c, there is a 50% probability of more than 12% coverage difference between two snapshots created less than three months apart. Between two snapshots of more than six months and less than a year, the difference can be even worse, up to 9%, 17% and 30%.

Distance (Fig. 4.3)

We first sort the pairs of snapshots by the metric defined in Eq. (4.4), the mean of the logarithmic distances (MLD). Recall, the higher the MLD, the more the snapshots differ. We compute distance across pairs of snapshots within the same time ranges as for coverage: less than three months, between three and six months, and between six and twelve months. From the MLD distribution, we then show the pair of snapshots corresponding to the median. For example, in Fig. 4.3a, one should read: On the end users dataset, 15% of the IP addresses moved by at least 40km. This corresponds to the median result for a pair of snapshots that are less than three months apart.

Fig. 4.3 shows two trends. First, as one might expect, for all types of IP addresses, the more time between two snapshots, the more IP addresses move. Then, the percentage of IP addresses moving depend on the type of IP addresses. We observe that end users tend to move more than routers and end host infrastructure. In details, for a pair of snapshots that are more than six months apart, we have 28%, 8% and 18% of IP addresses that move more than 40km for respectively end users, end host infrastructure, and routers.

If we consider that 40km corresponds to most metropolitan areas [61], this implies that a non-trivial portion of IP addresses experience a location change out of the metro area – a significant change. However, distances greater than 1000km are rare, accounting for less than 5% of IP addresses across all addresses classes.

Lesson There are non negligible differences in both coverage and distance of movement even for database snapshots created closely in time (< 3 months). Therefore: **we recommend, insofar as possible, aligning geolocation database snapshots with the measurements that produced them**, for instance by programmatically using an API to lookup IP addresses on-demand as they are gathered. Further, one should look at several snapshots closely spaced in time over the measurement period and more deeply **investigate IP addresses that experienced significant changes**.

4.6.4 A longitudinal study

Due to space constraints, we leave the 10 year longitudinal study for our accompanying tech report [69] and associated research [45]. In this chapter, we define new metrics and extend the ones defined in Sec. 4.4 to enable the comparison of an arbitrary number of snapshots and the analysis of the dynamics of the geolocation of an IP address over time. One of the main results is that we find that a majority of IP addresses are mapped to at least two locations far from 40 km (a metropolitan area) within a year, with a high variance depending on the country and the type of IP address. This reinforces our call to be very cautious about the usage of MaxMind when data are collected during a period longer than few weeks.

4.7 Use Case

Previous sections have shown two things. MaxMind is a widely used database (Sec. 4.3), and selecting a particular snapshot in a time period can have a significant impact on the results (Sec. 5.6). In this section, we concretely demonstrate the potential impact on research that depends on MaxMind by reproducing the results from Gharaibeh et al. 's IMC 2017 work [61] with different MaxMind snapshots. Gharaibeh et al. study the accuracy of different databases for router geolocation, including MaxMind. Using the author's publicly available ground truth, we reproduce their accuracy results (see Sec. 5.2, Fig. 2 of [61]).

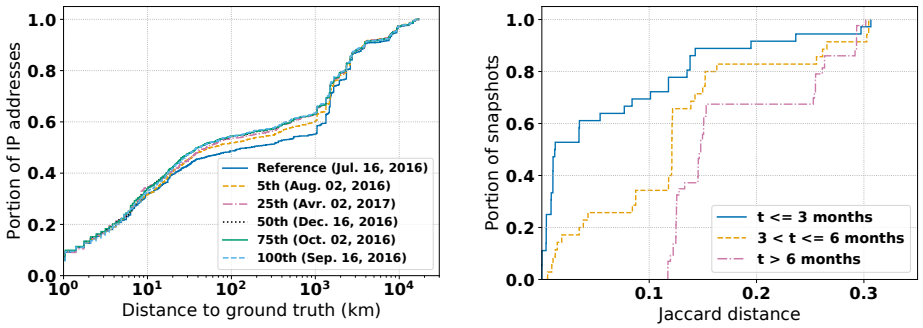
Surprisingly, the MaxMind snapshot that produces the largest impact on the results was created within only two months of the snapshot used by the authors. This snapshot shifts the median of the distance distribution to ground truth from more than 100 km to 40 km, which is close to the results of the paid version. Given this variability, the claim that the free version of MaxMind is worse than the paid version seems to depend on the specific instance studied.

4.7.1 Dataset

The Gharaibeh dataset consists of 16,586 router interface IP addresses with corresponding ground truth locations inferred either with RTT-based measurements or DNS-based techniques. The authors do not mention which specific snapshot of MaxMind they used, however: "The databases are accessed again on early July 2016, to geolocate the ground truth." We inquired with the authors for the exact snapshot date, but unfortunately they could not be more specific. We therefore select the closest snapshot as our reference, from July 8, 2016. Sec. 4.7.2 confirms that the results of this snapshot are very close to those presented in the original paper.

The measurement period for the ground-truth collection and validation, however, spanned a larger time period. As stated in the paper: "Overall, between May 2016 and September 2017, 8,197 (69.1%) [...] have different hostnames, and 6.9% no longer have rDNS records." We therefore restrict our comparison between snapshots belonging to this period of time, on which the authors consider that the ground truth is valid.

4.7.2 Results



(a) Distance to ground truth CDF on different snapshots

(b) Coverage difference on city with the reference snapshot

Figure 4.4: Reproducing result of [61] with different snapshots demonstrates the sensitivity of the results to the instance of the geo database. The median distance of addresses used shifted over 100km and coverage differed by up to 30%

Distance to ground truth

We compute the distance to ground truth distribution of all the snapshots from May 2016 to September 2017. We then compare each of these distributions to the distribution of the reference snapshot, using the Kolmogorov-Smirnov (KS) test [101]. The KS test quantifies the dissimilarity between two distributions, with higher values indicating less similarity. Fig. 4.4a shows the distance to ground truth distribution of the snapshots corresponding to the 5th, 25th, 50th, 75th, and 100th percentiles of the KS distribution. We also show the reference snapshot.

First, we compare Fig.2 of Gharaibeh et al. with our reference snapshot. We infer that on Fig.2 of Gharaibeh et al. ∼8%, 47%, 50%, 55%, and 96%, are located at less than respectively 1, 40, 100, 1,000, and 10,000km from the ground truth, whereas it is 8%, 46%, 49%, 56% and 97% in our reference snapshot. Overall, the qualitative shape of the distribution is identical to the original figure, giving us confidence in our ability to reproduce the author's results.

However, we observe significant differences between results derived from the other snapshots versus the reference. The median shifts from 167km in the reference snapshot to respectively 57, 51, 41, 40 and 40km for the 5th, 25th, 50th, 75th and 100th percentile.

When we consider the snapshots dates with these percentiles, it is surprising to observe that the 100th percentile was created only two months after the reference snapshot, whereas the 5th percentile is a snapshot taken one month later and the 25th percentile corresponds to a snapshot taken nine months later. This implies that MaxMind did not improve over time for these addresses, but also that there are significant differences in the results within a relatively short time.

Finally, we look at the comparison between the free and paid version of MaxMind. On Fig.2 of Gharaibeh et al. we infer that the paid version has a median between 30 and 40km, so that the difference between this distribution and the different snapshots of Fig. 4.4a is less pronounced than the difference between the free and paid version of their graph. Therefore the conclusion that the paid version performs better than the free one should be taken with caution.

Coverage

Finally, we examine coverage variability. In Gharaibeh et al. , the authors only compute the distance to ground truth if the IP address is covered by

MaxMind at the city level.

Fig. 4.4b shows the distribution of the coverage difference (Eq. (4.2)) as we did in Sec. 4.6.3, but only comparing snapshots with the reference snapshot. We observe that even with snapshots taken three months apart from the reference snapshot, 30% of the snapshots have more than 10% of coverage difference. It is even worse for snapshots between 3 and 6 months and snapshots with more than 6 months of difference, with a coverage difference up to 30%.

Lesson Work is being published in the network research community without specifying the MaxMind snapshot dates. Had the authors used a different snapshot, the set of IP addresses over which they would have computed their accuracy measures – and, hence, their results – would have significantly changed.

4.8 Conclusion

Physical mapping of Internet hosts and resources is critical in this day and age. Techniques to perform IP geolocation have matured into commercial offerings. While the accuracy of these geolocation databases has been extensively studied, little attention has been paid to understand the way they have evolved over time. Our work demonstrates that a commonly used geolocation database, MaxMind, exhibits significant changes in address coverage and locations, especially when considering particular subsets of addresses.

These changes can occur even on short timescales, including on the order of a typical measurement study duration. In this way we highlight the importance of geolocation lookups that are contemporaneous with the time an IP address is measured, observed, or gathered. Via a case study, we demonstrate the potential for a large discrepancy in results depending on the particular date of a geolocation snapshot. Similar large variances in auxiliary data sources at short time scales have been demonstrated in the past, e.g., for DNS and Internet top lists [152]. Thus, a take-away of our work is to encourage alignment of geolocation lookups with measurements, publishing the exact date of a geolocation snapshot or lookup methodology, and rigorously investigating addresses that change geolocation significantly over the course of a measurement study. In the spirit of similar measurement best practices [135], we hope to encourage more sound and reproducible measurement research. Because MaxMind does not provide access to historical data, we provide historical snapshots on demand to the community.

In future work, we plan to more deeply investigate the root causes of the geolocation movement we observe, characterize IPv6 geolocation, and work toward integrating our findings into more robust geolocation services.

Chapter 5

Large-scale heterogeneous traceroute data processing

In Chap. 3 we presented contributions to efficiency collect IP-level topology at Internet-scale. However, this new state-of-the-art infrastructure deployed in production collects a large amount of data, and it is difficult to analyze with classical approaches. In addition, the analysis of this kind of data often requires use of additional metadata such as IP-to-AS resolution or IP geolocation. In Chap. 4 we studied one type of metadata, IP geolocation, offered by a widely used commercial database, and give best practices to avoid bias in studies using this geolocation data. In this chapter we study how to use large-scale heterogeneous route tracing data. To this end, we present MetaTrace, a framework to process large-scale route tracing data efficiently. This framework allows us to conduct studies on millions of traceroutes in a few seconds and demonstrate the potential of MetaTrace by analyzing incomplete AS paths due to stars in traceroutes and by examining Internet flattening over 5 years of route tracing data.

5.1 Introduction

Today, measurement platforms such as Ark [26], RIPE Atlas [162] and M-Lab [110] perform millions of traceroutes per day and make them publicly available for researchers and operators. In parallel, recent work has designed high-speed traceroute tools that are able to collect hundreds of millions of traceroutes per day from a single server [21, 168, 79].

These millions of traceroute measurements contain very useful information to analyze and troubleshoot Internet paths, but this information is not easy to

extract: no publicly available framework exists to store and query traceroutes. Indeed, public traceroutes platforms and tools provide traceroutes in different formats, such as JSON, warts, or CSV, leaving the burden to implement specific drivers for each format on the user side. If analyzing thousands of traceroutes can be done with the basic technique of loading all the traceroutes in memory, this approach falls short when the number of traceroutes increases orders of magnitude.

Moreover, traceroute data is often augmented with higher-level information (e.g., AS), or *metadata*, to reveal information from Internet paths. For instance, AS paths inferred from traceroute IP-level paths serves as a basis to answer more complex questions, such as: "Are paths becoming shorter because of Internet flattening?", or "What are the peering relationships between ASes?". As we lack a framework to query traceroutes, we also lack a framework to add the metadata to the traceroutes and perform queries on them.

In this chapter, we go deeper into IP route tracing analysis and propose MetaTrace, a framework that combines traceroute mechanisms and database design choices to optimize traceroute storage and queries. Based on the goals of the system derived from the needs for operators to troubleshoot their paths and the research community to perform large-scale studies (Sec. 5.3), we identify the two types of queries that MetaTrace should serve: queries to filter a subset of traceroutes, called *predicate queries*, and queries to compute aggregated metrics on traceroutes, called *aggregated queries*. We make our database design choices to build MetaTrace to better serve those queries: choosing the right format under which MetaTrace stores the traceroutes, the order in which MetaTrace stores them, and which database mechanisms we set up to optimize query processing (Sec. 5.4). In short, MetaTrace uses a column-oriented format based on traceroute replies, a sorting key based keeping traceroutes with the same (source, destination) pair contiguous, and a general mechanism of auxiliary tables indexed on the columns that serve for filters, where these columns can be any column of the traceroute reply or an arbitrary metadata.

Finally, we demonstrate the usefulness of MetaTrace with two use cases: a study of how stars in traceroutes can hide entire ASes from the AS paths (Sec. 5.7), and a study of Internet flattening over 5 years on the Ark dataset (Sec. 5.8).

Our contributions are:

- The MetaTrace framework and its implementation.
- A study of how stars in traceroutes can hide entire ASes.

- A study of Internet flattening on 6 billion traceroutes over 5 years.
- An open-source release of MetaTrace [159] available for the research community.

Our main results include that MetaTrace is more than 500 times faster than a hand-crafted Rust solution and more than 3 times faster than a raw database solution to filter traceroutes based on a condition; MetaTrace scales linearly and is able to compute aggregate metrics per path on 202 million traceroute in 11 seconds. In our study of stars in traceroute, we find that at least 9.7% and 17.5% of the (source, destination) traceroute pairs in IPv4 and IPv6 can have incomplete AS paths because of stars. Finally, we find surprising results on Internet flattening, showing that there is no statistical evidence that Tier 1 influence is decreasing or paths are shortening between 2016 and 2021.

5.2 Related Work

Today's production systems performing traceroutes include RIPE Atlas [162], M-Lab [110], Iris [71] and CAIDA Ark [26]. Typically, measurement data is shared as JSON, CSV, binary Warts files or BigQuery [66] datasets and must be manually refined by the end user. MetaTrace provides a unified format transparent to the user with drivers transforming JSON, CSV, Warts or BigQuery traceroutes into MetaTrace's format.

More generally, any system using traceroute measurements [113, 112, 46, 64, 91] had to develop its own implementation to process the traceroutes. For instance, some used a custom Python implementation [64] with each traceroute being represented as a JSON object, a raw database [91, 166], or a graph representation of traceroute paths [113].[1] In any case, these systems would benefit from MetaTrace to scale up.

Studies to highlight potential errors returned by traceroute and improve its accuracy and coverage include the inference of false links in presence of load balancing [15], techniques to reveal hidden MPLS tunnels [109], or detecting forwarding loops [170]. To the best of our knowledge, we are the first to propose a methodology to detect and characterize the stars corresponding to missing AS hops.

Internet flattening has been studied for more than a decade [63] with different approaches. Some looked for structural property change of the

[1] Private communication with the authors of the different papers.

Internet topology [53, 173], while some ran specific traceroutes to uncover peering links [34, 12] and better capture the new peering ecosystem [33]. The work closest to ours looked at the properties of the Internet paths aggregated over time [25]. Our methodology differs: instead of computing metrics aggregated per year on a set of paths in arbitrary snapshots of traceroutes not necessarily composed of the same (source, destination) pairs, we compute the trend with an evaluation per path of (source, destination) pairs having been repeatedly measured for 5 years.

5.3 Goals

We expect MetaTrace to be useful both for the operators troubleshooting networks and for the research community conducting studies on the Internet.

Operational troubleshooting We want MetaTrace to be useful for a network operator (e.g., a CDN or a Tier-1) to monitor the performance of its routes towards its clients. As a rough estimate, to have a full view of the paths between the CDN and the different ASes in the Internet, a CDN can monitor the routes to each of the 900,000 announced BGP prefixes [11] at the reasonable frequency of once every 15 minutes [52], so that a performance degradation can be quickly detected. The number of traceroutes to process every 15 minutes depends on the interval of the traceroutes to be kept before deletion: if an operator wants to keep an entire day of traceroute, MetaTrace must be able to process 1.3B traceroutes in a few minutes.

Research studies Unlike operational troubleshooting, there is no hard constraint on the time to process traceroutes when performing a research study. However, our goals are both to save debugging time for researchers and allow researchers to study a large number of traceroutes (e.g., billions) in a reasonable time. For instance, the Ark Prefix-Probing dataset [27] contains daily traceroutes to every announced BGP prefix from a subset of Ark monitors, starting from 2016, and containing 6B traceroutes between 2016 and 2021. We want researchers to be able to run studies on datasets of this scale in a reasonable time (hours) with a single server with 256 GB of RAM, 128 CPU threads and 20 TB of storage, which is what we used to build and evaluate MetaTrace, and what we consider to be resources that are accessible to a research team in our field.

5.4 Design

To be clear, MetaTrace is not a new type of database designed to exclusively process traceroutes. Rather, MetaTrace is the sound application of a set of existing database optimization techniques for our particular use case of analyzing traceroutes. Our methodology to answer the two aforementioned goals is to translate them into different database design questions.

Overview MetaTrace serves two types of queries corresponding to the needs of filtering traceroutes and aggregating statistics on them. The "aggregate" queries compute aggregated metrics from traceroutes where a metric is computed on a set of traceroutes with the same source and destination. The "predicate" queries filter traceroutes based on a predicate (Sec. 5.4.1), e.g., if an AS is crossed in a traceroute. MetaTrace stores the traceroutes in a column based format ordered by the (source, destination, traceroute timestamp) key where each row represents a reply received by a traceroute (Sec. 5.4.2). To optimize queries filtering traceroutes based on a predicate, MetaTrace creates auxiliary tables containing one field plus the columns forming the sorting key, and creates subqueries from the predicates to optimize the usage of the auxiliary tables (Sec. 5.4.4). Finally, MetaTrace uses radix trees to map addresses to metadata, so that the metadata is just considered as another column.

5.4.1 What queries should MetaTrace serve?

Based on our goals, we identified two types of queries that we want to serve: (1) Compute aggregated statistics on traceroutes from the same source and destination over time; and (2) finding all the traceroutes satisfying a predicate. The predicate can either contain a field directly available from the traceroute (e.g., source or destination) or metadata (e.g., the presence of an AS in the path). We call these two types of queries the *aggregate* queries and the *predicate* queries.

Both types of queries are useful for our goals or network monitoring and research studies. For instance, an operator might want to monitor latency degradation over repeated traceroutes between the same source and destination. One would first issue an aggregate query to obtain latency statistics per (source, destination) pairs. Then, one could issue predicate queries to further troubleshoot. For a research study, one might want to extract aggregate statistics such as the evolution of AS path length over time, using an aggregate query, and then characterize the findings by AS type,

using a predicate query.

5.4.2 How to represent a traceroute?

Table 5.1: Fields of MetaTrace flat format.

Name	JSON Type	Database Type
measurement_id	String	String
agent_id	String	String
traceroute_start	ISO8601 String	DateTime
probe_protocol	Integer	UInt8
probe_src_addr	IPv6 String	IPv6
probe_dst_addr	IPv6 String	IPv6
probe_src_port	Integer	UInt16
probe_dst_port	Integer	UInt16
probe_ttl	Integer	UInt8
quoted_ttl	Integer	UInt8
reply_ttl	Integer	UInt8
reply_size	Integer	UInt16
reply_mpls_labels	Array(Integer)	Array(UInt32)
reply_src_addr	IPv6 String	IPv6
reply_icmp_type	Integer	UInt8
reply_icmp_code	Integer	UInt8
rtt	Integer	UInt16

We choose a column-oriented format, where the columns are the fields of the ICMP reply to a traceroute probe, and the fields uniquely identifying the traceroute that issued the probe (source, destination, traceroute timestamp). The full set of columns is shown in Tab. 5.1. A traceroute is then a set of rows. Notice the presence in our set of columns of some port columns, which can serve to distinguish between multiple traceroute paths in case of per-flow load balancing [169, 168]. To that end, the key becomes (flow_information, traceroute timestamp), where the flow typically corresponds to five columns representing each field of the 5-tuple, which depends on the protocol. For simplicity, we consider in the rest of this section that we manipulate single path traceroutes.

Such a format is (1) adapted to all existing traceroute tools as it is the lowest common denominator to all traceroute platforms and tools, as they use the standard traceroute mechanism of sending TTL limited probes to elicit

ICMP TTL exceeded messages from routers on the path; and (2) memory efficient for predicate queries, so one only needs to load the column of the predicate to find the rows matching a predicate [3]. At first sight, this format introduces a lot of redundancy, as some fields (e.g., source and destination) are repeated for each reply of a traceroute. However, we show that this redundancy results only in a small disk usage overhead (Sec. 5.6.2) if we sort the data properly (Sec. 5.4.3).

Ark and RIPE Atlas use row-oriented (each traceroute is a row), variable length formats. These formats are suboptimal to store and analyze traceroutes. First, they have no particular sorting order except the time at which the traceroute was run, for instance RIPE Atlas provides daily dumps of all the traceroutes run during that day. As a result, the compression is suboptimal (Sec. 5.6.2). Second, without parsing traceroutes and inserting them in a database, one cannot benefit from database optimizations, such as indexes, and has to scan all the traceroutes to serve any predicate query. At most, as Ark and RIPE Atlas traceroute file names contain a date, one could hand-craft a solution to only look at the files corresponding to the dates of interest.

5.4.3 How to order the traceroutes?

Now that we have defined our column-oriented format to describe the traceroutes, we have to choose the sorting key, that is, the set of columns that determines in which order the traceroute replies are stored.

The choice of the sorting key is important for two reasons. First, it determines the quality of the compression of columns, and a better compression saves disk usage and reduces query time as it limits I/O calls and speeds up loading the data in RAM during the queries. Second, different sorting keys influence the resources needed to perform the queries, such as the time of the queries, and the memory used.

In a column-oriented database (Sec. 5.4.2), each column is stored in a different file and the compression is performed on each file. A column has a good compression rate if there is a lot of redundancy between values that are close in space. Most compression algorithms used by open-source databases [122, 37] are derived from the LZ77 algorithm [183]. The idea of the LZ77 algorithm is to read the file using a sliding window that keeps the tokens (e.g., a character) and sequences of tokens that have been read in that sliding window in memory. Each token or sequence of tokens is mapped to a symbol taking less space than the token (typically an integer). When a token is read, the algorithm looks into the map for the longest sequence of tokens already appearing in the sliding window, and replaces it with its

corresponding symbol. If no matching is found, the token is added to the map. As a result, the closer redundant data the column has, the better it will compress.

An ideal sorting key would be a key that gives a good compression rate for the columns, reduces the time of aggregate and predicate queries and minimizes their memory usage. Unfortunately, it is clear that no sorting key can work for all predicate queries. Indeed, a sorting key on some metadata (e.g., AS) will optimize the performance for predicate query on this metadata, but not necessarily on another uncorrelated metadata (e.g., geolocation). As the sorting key cannot satisfy the ideal goal of satisfying all the predicate queries, we use other mechanisms to optimize the predicate queries (Sec. 5.4.4).

We are left to choose a sorting key that has at least a good compression rate and limit memory usage and time for aggregate queries. Recall, aggregate queries extract statistics over traceroutes with the same source and destination. These queries influence the choice of our sorting key: aggregate queries strongly benefit from traceroutes with the same source and destination being contiguous. Indeed, it allows us to stream the data, and limit memory usage. One can compute the statistic over the traceroutes of a (source, destination) pair and then release the memory taken by the traceroutes and go to the next pair. Natural choices for the sorting key are therefore (source, destination, traceroute timestamp) or (destination, source, traceroute timestamp). Conversely, the (traceroute timestamp, source, destination) sorting key is not well suited for our aggregate queries per source destination, as one would have to keep all the traceroutes in memory to compute the aggregate statistic per source destination.

As a consequence, we choose the sorting key (source, destination, traceroute timestamp) as it compresses better than the (destination, source, traceroute timestamp) key (Sec. 5.6.2).

5.4.4 How to serve the queries?

This section describes the design choices to serve the aggregate and the predicate queries, and the trade-offs between memory usage, disk usage, and query time. Up to now, we have described how we store the traceroutes, each reply corresponding to a row. This table, ordered by the sorting key (source, destination, traceroute timestamp) is called the main table (as opposed to auxiliary tables, that we describe next).

How to perform aggregate queries?

An aggregate query looks like:

```
SELECT aggregate_statistic
FROM main_table
GROUP BY (source, destination)
```

These queries require a full scan of the main table and we can only optimize for memory usage that is influenced by the choice of the sorting key (Sec. 5.4.3).

How to filter traceroutes on a predicate?

Our sorting key does not optimize for filtering traceroutes on a predicate (Sec. 5.4.3), as if the column to apply the predicate on does not appear in the sorting key, the database will have to perform a full scan to find the database to find the rows matching the predicate. We identify three options to serve a predicate query with (one or more) fields: (1) having no optimization; (2) having an index on each field of the main table; and (3) having an auxiliary table for each field of the main table where the columns are (field, source, destination, traceroute timestamp) and are sorted in this order. These different choices represent a trade-off between query time and disk usage, going from the highest query time and the lowest disk usage in case (1) to the lowest query time and the highest disk usage in case (3). We now explain the advantages and the drawbacks of each approach, and why choice (3) is the most suited for our goals.

No optimization This choice incurs no storage overhead, but it also results in a full scan of the data, which translates into query times at least 3x higher than the solution (3) (Sec. 5.6).

Having an index on each field of the main table Putting an index on each field can appear as the natural solution to reduce the number of rows to scan. Let us take an example to explain why the performance of this solution depends on the traceroute dataset. A common use case is to find traceroutes going through a certain AS or a certain facility.

A first approach would be to use a hash index mapping an AS value to the corresponding rows. However, such an approach does not scale with billions of rows, as the index would consume too much space: assuming that

rows are numbered with 64-bit integers, indexing 7B rows (one day of RIPE Atlas data) would consume 56 GB of memory.

Another approach is to use sparse indexes: one would store the distinct AS values for each block of N lines, and use this index to discard blocks that do not contain the relevant AS when filtering rows. However, the performance of this index is dependent on the traceroute dataset. For instance, in a cycle of Ark traceroutes between Ark's sources and one destination in each BGP prefix, Tier 1 ASes and ASes close to the sources appear in a lot of traceroutes, whereas small stub ASes only appear in traceroutes to them. Conversely, in the traceroute dataset of a study uncovering the peerings of Google [34], only a few traceroutes contain Tier-1 ASes, as Google has built a whole infrastructure to bypass Tier-1 ASes to directly peer with its clients. As we cannot predict the performance of this approach that depends on the traceroute dataset, we do not choose it.

Having auxiliary tables This is the approach selected in MetaTrace. The idea is to have one auxiliary table with the columns (field, source, destination, traceroute timestamp) sorted in that order for each field that can serve in predicates. This table stores only deduplicated values. With these auxiliary tables, predicate queries look like:

```
                SELECT * FROM main_table WHERE sorting_key IN
(SELECT sorting_key FROM auxiliary_table_1 WHERE predicate_1)
                                           AND sorting_key IN
(SELECT sorting_key FROM auxiliary_table_2 WHERE predicate_2)
                                           AND sorting_key IN
                                                        ...
```
(5.1)

This approach allows us to quickly select the sorting keys of the traceroutes matching the predicates since the auxiliary tables are ordered by the predicates fields. It is also fast to retrieve all the replies of these traceroutes since the main table is ordered by the sorting key. Also, the disk usage overhead of the auxiliary tables is acceptable compared to the solution with the indexes given the 3.4x times overhead that we save in terms of query time (Sec. 5.6).

5.4.5 Adding metadata to the traceroutes

Metadata is the result of a mapping from IP addresses to other information, such as ASes or geolocations. To generate an auxiliary table on metadata,

we use a radix tree mapping a prefix to the metadata. This radix tree can be parametrized by a date in order to use the most relevant metadata for each traceroute. For each row of the main table, the column `reply_src_addr` is mapped to its metadata, and we insert the unique (metadata, source, destination, traceroute timestamp) columns in the auxiliary table. The metadata is just another column available for a predicate. This method is generic and works with any mapping between IP addresses to other information. Also, this method only requires one scan of the main table to generate all the auxiliary tables.

5.5 Implementation

Sec. 5.4 describes the database concepts and techniques that we are using to build MetaTrace, which are common to any relational database. MetaTrace can be implemented on top of any relational database, and our implementation of MetaTrace lies on top of the ClickHouse database. We choose the ClickHouse database after a benchmark between different relational databases, including MySQL [122] and PostgreSQL [140], two standard SQL databases. We give some more details of implementation, sometimes specifics to ClickHouse, to make our methodology fully reproducible (Sec. 5.5.1). We also give more details about MetaTrace's currently supported traceroute formats (Sec. 5.5.2) and how a user can interact with MetaTrace (Sec. 5.5.3).

5.5.1 Tables and metadata

Tables The main table is a standard ClickHouse table, while the auxiliary tables are materialized views of the main table [40]. A materialized view is a "view" of the main table in the sense that they are the results of a query on the main table, but add the feature of when data are modified or inserted in the main table, the view automatically updates as well. Materialized views are well suited for predicate queries, and we can create one materialized view per metadata on which we want to be able to run predicate queries. In query 5.1, auxiliary tables are just replaced by materialized views. Materialized views are not specific to ClickHouse.

Metadata Metadata types are columns added to the main table usually mapping IP addresses to some values, such as AS numbers or geolocation information. If one could compute this metadata before the insertion, ClickHouse allows us to compute them on the fly during the insertion by creating

views on these data with the dictionary feature [38]. Dictionaries are special tables allowing mapping keys to values, which is exactly what we need here. For instance, the column `reply_asn` in the main table will have the following type:

```
reply_asn UInt32 MATERIALIZED
dictGetUInt32(dict_asn, 'asn', reply_src_addr)
```

This column type indicates that the `reply_asn` column should be obtained by querying the dict `dict_asn` table, with the key `reply_src_addr`, asking for the value `asn`. Moreover, ClickHouse has an IP trie layout [38], so that `dict_asn` actually maps IP prefixes to values, not IP addresses, avoiding to insert new entries in the `dict_asn` table when traceroutes with new IP addresses are inserted in the main table.

This implementation choice is better than pre-computing the values of the metadata because we can easily modify the values of this metadata, if necessary (e.g., if some IP to AS technique is better than another one and we want to change our IP to AS mapping). We would just have to change the content `dict_asn`, or make the `reply_asn` points to another dictionary table, and rematerialize the view [39]. Without this feature, we would have to recreate the main table with new values for the `reply_asn` column, or perform many updates, which are less performant and more error-prone.

5.5.2 Handling multiple platforms

MetaTrace provides a unified tool, PanTrace to download, parse and transform traceroutes from formats from the most used platforms into our format (Tab. 5.1) and insert them into ClickHouse. Our system currently supports the Warts format offered by Ark, the line-delimited JSON format of RIPE Atlas. We released PanTrace to the community that can convert traceroutes from any (supported) format to another one [159].

5.5.3 Flow of a query

MetaTrace provides a Python API where a user can specify with a builder pattern, for instance:

```
Table("ark_prefix_probing")
  .start("2022-01-01").end("2022-01-02")
  .filter(m.asn == 3356 and m.country == "France")
```

Table 5.2: Query time on one day (June 12, 2022) of IPv4 and IPv6 RIPE Atlas data (202 million traceroutes). Time shown in seconds, peak memory and disk usage shown in MB. Python, the database, and MetaTrace, are limited to 16 threads.

		Insert			Select by reply IP			Select by ASN/Country			Average S-D RTT		
Method	**Disk**	**Time**	**Mem**	**Read**	**Time**	**Mem**	**Read**	**Time**	**Mem**	**Read**	**Time**	**Mem**	**Read**
Python	71,778	0	0	0	945	21.38	71,778	918	206.57	71,778	1204	615.32	71,778
Rust	71,778	0	0	0	602	97.15	71,778	780	676,95	71,778	690	97.228	71,778
BigQuery	-	-	-	-	16	-	58,562	-	-	-	16	-	101,340
Database	28,707	3603	4176	71,778	3.68	59.96	4604	3.38	76.11	1958	11	1588	14,088
Database/index	29,270*	3603	4176	71,778	4.74	79.69	835	5.08	68.94	192	11	1588	14,088
MetaTrace	52,877**	3603	5295	71,778	1.09	55.14	63	1.13	75.64	50	11	1588	14,088

* Including 581 MB for the data skipping index on the reply IP, 25.58 MB for the reply ASN and 3.84 MB for the reply country.
** Including 14,612 MB for the auxiliary table on the reply IP, 5562 MB for the reply ASN and 3307 MB for the reply country.

This query requests all the traceroutes from the Ark table containing traceroutes from January 1st to January 2nd going through AS 3356 (Level 3). If needed, the user can also directly execute SQL queries against the database.

5.6 Evaluation

On a dataset of 202 million traceroutes from RIPE Atlas, MetaTrace takes 25% less disk space than the compressed JSON source data (Sec. 5.6.2). MetaTrace is able to serve predicate queries 552x faster than a naive multiprocessor Rust solution, and is 3.4x faster than a raw database. On aggregate queries, MetaTrace is 62x faster than an optimized Rust solution. Overall, MetaTrace is able to serve both types of queries with a very reasonable 1.6 GB maximum usage of memory. These results on queries are shown in Sec. 5.6.3. MetaTrace scales linearly with the resources available and the number of traceroutes (Sec. 5.6.3). This performance allows us to say that MetaTrace satisfies our two goals (Sec. 5.3).

5.6.1 Dataset and setup

Our dataset is composed of all the IPv4 and IPv6 traceroutes performed on RIPE Atlas on June 12, 2022. This dataset is downloadable from RIPE Atlas's FTP server [149] and is available on RIPE Atlas's BigQuery project. It contains 202 million traceroutes and 7.4 billion replies.

We implement five alternative solutions to evaluate the impact of our design choices:

1. Python: this implementation represents the performance of a naive multiprocessor Python implementation to analyze traceroutes.

2. Rust: the same implementation as Python but (supposedly) more performant.

3. Raw database: this implementation represents the performance of the database that we used to implement MetaTrace (ClickHouse) without any intelligence, just putting the traceroutes in our format and storing them in a database.

4. Database with indexes: it is the raw database plus indexes on columns used in the predicate queries.

5. MetaTrace: this is MetaTrace's implementation as described in Sec. 5.4: traceroutes are inserted in the format in the main table, and auxiliary tables are created accordingly.

We chose to compare MetaTrace to these alternatives because they illustrate well the trade-off that researchers usually face when processing traceroutes: either implement a custom solution allowing fast prototyping but somewhat inefficient (e.g., Python), or a more sound solution that requires a lot of efforts to implement, but it saves a lot of time in future processing, such as MetaTrace.

To be fair with the Python and Rust solution, we split the 202M traceroutes into multiple files such that the computation can be performed in parallel. Moreover, the files are compressed using the LZ4 algorithm that is used by default on column files by our database (reducing disk read overhead). The raw database and the database with indexes solutions use MetaTrace's sorting key (Sec. 5.4.3). The indexes used by the database with indexes solution are ClickHouse implementation of two types of indexes that are good for range and equality queries [41], which are the equivalent of the B-Tree and hash indexes used in other well-known databases [121, 141]. Each column appearing in the predicate queries (Sec. 5.6.3) has one index of each type.

The evaluation is performed on a single server equipped with a 64-core CPU, 256 GB of memory (but only 2 GB is used during the queries (Sec. 5.6.3)) and an SSD delivering $\approx$ 1 GB/s read performance. We restrict the five alternatives to 16 CPU threads in order to evaluate MetaTrace on a server that we consider accessible to any researcher.

5.6.2 MetaTrace storage efficiency

We first evaluate the compression of the main table when we use different sorting keys. MetaTrace's sorting key (source, destination, traceroute timestamp) achieves better compression than (destination, source, traceroute timestamp) and (traceroute timestamp, source, destination): the main table takes respectively 28, 31, and 70 GB on disk.

Then, we look at how MetaTrace compares to the original compressed data. The compressed JSON files with LZ4 take 77 GB, whereas MetaTrace takes 28 GB for the main table, plus 24 for three auxiliary tables (one ordered by reply IP address, one by its ASN and one by its country), for a total of 52 GB disk space. This shows that adding auxiliary tables remains reasonable in terms of disk space and that even with multiple auxiliary tables, MetaTrace remains in the same order of magnitude as the original dataset.

5.6.3 MetaTrace performance on queries

We evaluate MetaTrace's performance of three queries: two predicate queries, one with a single field and one with multiple fields (Sec. 5.6.3), and an aggregate query (Sec. 5.6.3).

We measure the time taken, the memory (RAM) used and the size of the number of bytes read by each solution to serve the queries. The number of bytes read by each solution is particularly important to consider for systems with slow storage, as I/O disk reads can become a bottleneck.

Predicate queries

The query consists in finding the traceroutes going through a particular interface and retrieving all of their content. As shown in Tab. 5.2, MetaTrace outperforms the other solutions in terms of query time and data read, while being very reasonable in memory. MetaTrace takes 1.09 seconds compared to 945 seconds for the Python solution and 602 seconds for the Rust solution, 3.68 seconds for the database and 4.74 for the database with indexes. MetaTrace only reads 63 MB, whereas the Python and Rust solutions read 71.8 GB, the database solution reads 4.604 GB, the database with indexes 835 MB. Finally, MetaTrace memory usage stays under 60 MB, which is negligible on modern computers. The difference in bytes read between the Python and Rust solutions and solutions with the database comes from how a column-oriented database works on queries with a predicate: it only loads the column(s) on which there is a predicate, and then loads the other columns of the rows matching that predicate. In the Python and Rust solutions, each row has to be entirely read in order to filter on one column. We perform the same comparison with a predicate on multiple fields, obtaining similar results (Tab. 5.2).

We also provide as information the performance of BigQuery: we cannot really provide an apple-to-apple comparison with MetaTrace, as we do not know the details of BigQuery's implementation nor its infrastructure.

BigQuery takes 16 seconds, reads 58 GB of data, and more importantly, amounts to $0.29 at a cost of $5 per TB. The information about memory usage is unavailable on BigQuery. Queries on the country and ASN cannot be performed as this metadata are not available on RIPE Atlas's BigQuery project.

Aggregate queries

The query consists in computing the mean RTT for each (source, destination) pair. As expected, a raw database and a database with indexes perform as well as MetaTrace as we gave them the same sorting key (Secs. 5.4.3 and 5.6.1) where the query takes 11 seconds. The most important result comes from the comparison with Python and Rust solutions. MetaTrace is 94x faster than the Python solution and 62x faster than the Rust solution. Again, the memory usage is reasonable with 1.6 GB. It is possible to even further reduce the memory usage, as MetaTrace stores data ordered by (source, destination), it can optionally stream the results per (source, destination) pair instead of processing multiple (source, destination) pairs in parallel.

Insertion overhead

Unlike the Python or Rust solution, database solutions require loading the data, once, in the database. MetaTrace takes 1 hour to insert the results. However, given the number of predicate queries that one usually needs to perform to study traceroutes (e.g., see Sec. 5.7, we ran dozens of predicate queries), the benefit is worth the cost. The addition of auxiliary tables or indexes has no discernible impact on the insertion time. However the use of auxiliary tables requires an additional gigabyte of memory at insertion time.

MetaTrace scaling

Table 5.3: Time in milliseconds to compute the average RTT per origin-destination pair.

	1M	**10M**	**100M**	**1B**	**7.4B rows**
2 threads	21	124	1110	11,263	80,928
4 threads	18	74	577	5771	40,970
8 threads	18	44	302	2951	20,884
16 threads	12	34	167	1548	10,799

We run the same aggregate query as in Sec. 5.6.3 on subsamples of 1M, 10M, 100M, 1B rows and the full table, with 2, 4, 8 and 16 CPU threads. The results are displayed in Tab. 5.3. Above 100M, the scaling is linear in both the number of threads and the number of rows.

5.6.4 MetaTrace vs alternative databases

In this section, we compare MetaTrace to a graph database, which is a completely different approach than a relational database. A graph database might sound like a good option to store traceroutes, as we can represent the results of a traceroute as a directed graph where nodes are IP addresses and edges are links between them. Rather than reinvent the wheel, we use the Neo4j graph database [125], a well-known commercial graph database. We use the community (free) edition in this evaluation, and our comparison is therefore limited to this database.

Data schema

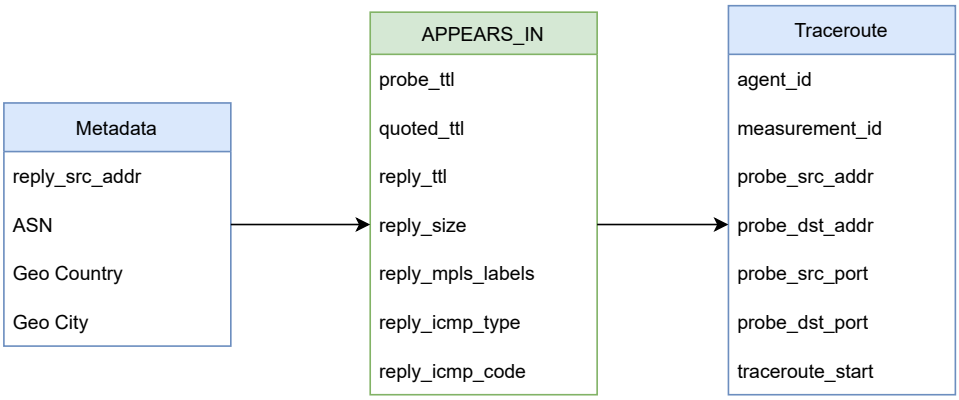


Figure 5.1: Our Neo4j data schema to store the traceroutes.The nodes *Metadata* and *Traceroute* are represented in blue and the relationship *APPEARS_IN* is represented in green. This schema allows us to fully rebuild a traceroute and contains all the information in MetaTrace schema (Tab. 5.1) and is optimized for predicate queries.

In Neo4j, data are stored into nodes and relationships between nodes. In our case, we chose the following implementation, with two types of nodes: `Metadata` and `Traceroute`, and one relationship between them, `APPEARS_IN`. Nodes and relationships can have properties associated to them, and for us,

the `Metadata` node properties are the metadata fields which can be used in MetaTrace predicate queries, e.g., `reply_src_addr` or `ASN`. `Traceroute` node properties are all the information relative to a particular traceroute such as the source, the destination, and the timestamp of the measurement. The `APPEARS_IN` relationship properties are all the information relative to a particular reply in the traceroute, such as the `probe_ttl`. Fig. 5.1 details this schema. With this schema, one can rebuild the original traceroute by querying all the `Metadata` nodes that have an `APPEARS_IN` relationship with this traceroute.

This schema is meant to optimize predicate queries. Indeed, for instance, to find all the traceroutes going through a particular IP address, Neo4j needs to find the Metadata node corresponding to the IP address, and then find all the `Traceroute` nodes having a `APPEARS_IN` relationship with it. To make a fair comparison between MetaTrace and our graph database, we put an index on each field of the `Metadata` node, so the lookup of the `Metadata` node for predicate queries should be faster.

Queries

Queries can be easily done with the Cypher query language of Neo4j [124] and this data schema. For instance, the predicate query counting all the traceroutes passing through a particular IP address, here `88.204.208.2` looks like:

```
MATCH (Metadata {reply_src_addr:
"88.204.208.2"})-[APPEARS_IN]->(tr:Traceroute)
RETURN Count(tr);
```

Similarly, an aggregate query which computes the average RTT for each (source, destination) pairs looks like:

```
MATCH (m:Metadata)-[a:APPEARS_IN]->(tr:Traceroute)
WHERE m.reply_src_addr = tr.probe_dst_addr
RETURN tr, avg(a.rtt);
```

Results

We insert our dataset (Sec. 5.6.1) in our Neo4j database using the Neo4j administration tool. It resulted in creating 204 M nodes (2M `Metadata` and 202 M `Traceroute`) and more than 7 B relationships, corresponding to the number of replies in the traceroutes.

The insertion took almost 7 hours, compared to one hour for MetaTrace (Sec. 5.6.3). The peak memory usage for the insertion was a reasonable 3.2 GB.

As we used the community version of Neo4j (as opposed to the enterprise edition, which is not free), we are limited in terms of the features we can use. In particular, we cannot monitor the memory used [126], as we did for MetaTrace. What we do, though, is to look at the time taken by queries: the predicate query `SELECT by reply IP)` (third column of Tab. 5.2) took 31 seconds, which is an order of magnitude worse than the 1.13 seconds taken by MetaTrace. It's even worse for aggregate queries. The query presented in Sec. 5.6.3 (last column of Tab. 5.2) to get the average RTT group by (source, destination) took multiple hours to complete.

5.7 Stars and missing AS hops in the traceroutes

Stars, or " * ", as they are represented in traceroutes, are one of the plagues of traceroute measurements. Due to an absence of responses matching the TTL limited probes sent by traceroute, it is hard to know the root cause of a star. It could be due to a router unresponsive to traceroute probes, routers performing ICMP rate limiting, or any other reason that could make the reply not come back to the source. In some cases, stars can hide some entire ASes of the paths, which can distort our understanding of the Internet: a missing AS hop in an AS path increases the difficulty to troubleshoot bad performance for operators, as they might miss the AS to contact, and can confuse researchers trying to reason about AS path length, for instance to understand Internet flattening or optimize anycast performance [179, 182].

Our main findings are that 9.7% and 17.5% of IPv4 and IPv6 (source, destination) traceroute pairs can be affected by a missing AS hop, and that for 3.5% of the (source, destination) pairs in IPv6, there is a probability higher than 0.5 to have a missing AS hop when performing a traceroute. The missing AS hops concerned a wide range of networks, from Tier 1s to small providers. We discuss how MetaTrace help us to perform our study at the end of the section.

5.7.1 Goals and challenges

To show that stars in traceroutes can distort our understanding of AS paths, we want to characterize how frequent it is that we can miss entire ASes in AS paths because of stars, and which ASes are concerned by this phenomenon, for both IPv4 and IPv6.

Table 5.4: Traceroutes measured at 15-minute intervals between two RIPE Atlas anchors. Second traceroute contains a missing AS hop (AS 12657).

TTL	T hops	T' hops	T AS hops	T' AS hops
1	2001:67c:6ec:201:145:220:0:1	2001:67c:6ec:201:145:220:0:1	1101	1101
2	2001:67c:6ec:1101:192:12:54:241	2001:67c:6ec:1101:192:12:54:241	1101	1101
3	2001:610:fc7:0:145:145:2:218	2001:610:fc7:0:145:145:2:218	1103	1103
4	*	*	*	*
5	2001:a60::89:303	2001:a60::89:303	8767	8767
6	2001:a60:0:217::1:1	2001:a60:0:217::1:1	8767	8767
7	2001:1578:0:ff::1:2	*	12657	*
8	2001:1578:400:111:1:0:85:1	2001:1578:400:111:1:0:85:1	35003	35003

Tab. 5.4 shows an example of two paths from the same (source, destination) pair measured at 15 minutes intervals with traceroute, with their corresponding AS paths. Looking at hop 4, it is hard to say anything about this star, as it appears in both traceroutes, so we have no hints that it could correspond to a missing AS hop. However, at hop 7, it is probable that the star at hop 7 of the second traceroute corresponds to the AS hop 12657 of the first traceroute, and is a missing AS hop. If we remove this AS from the AS path of the second traceroute, this AS does not appear in the AS path anymore. Our goal is to find a methodology to have a high confidence that hop 7 of the second traceroute is actually the same as hop 7 of the first traceroute, but in the second traceroute appear as a star, as, for instance, router at hop 7 performs rate limiting, so hop 7 sometimes appears as a normal hop, and sometimes as a star.

So, to be clear, our study will give a lower bound of the number of traceroutes with missing AS hops in the traceroutes, as we cannot say anything about routers that are always unresponsive to traceroutes. They could be missing AS hops, or not. Also, there could be invisible MPLS tunnels [109] also hiding entire ASes from the AS path. In this study, we focus on missing AS hops that are likely to be due to rate limiting or other transient states of the router (e.g., , congestion which would deprioritized ICMP(v6) traffic) which can cause it to not respond to traceroute, because we can develop a methodology to have high confidence that these hops are actually missing AS hops, as they sometimes appear as hops, and not always as stars.

5.7.2 Methodology

In our case, we define a missing AS hop as follows: a missing AS hop is an AS that does not appear in the AS path revealed by traceroute but is still crossed

by the traceroute probe, and appears as a star in the path. Again, this does not count all the hidden AS hops, but only those that our methodology is capable of uncovering, which gives a lower bound of the phenomenon.

High level idea

To infer that a star corresponds to a missing AS hop, our idea is to use path similarity between historical traceroutes in conjunction with evidence of no path change. At high levels, for the same (source, destination) pair, if two paths measured by two traceroutes close in time only differ at some TTLs and the hops at those TTLs are replaced by stars, and we have high confidence that there is no path change, the stars might correspond to missing AS hops. This happens when a star causes an AS to disappear from an AS path.

Tab. 5.4 shows an example with two traceroutes between two RIPE Atlas anchors measured at 15-minute intervals. The two paths are the same except at hop 7 where the hop `2001:1578:0:ff::1:2` is replaced by a star. If we have high confidence that there was no path change between the two anchors, hop 7 is likely to correspond to `2001:1578:0:ff::1:2`, which translates to AS 12657. As `2001:1578:0:ff::1:2` is the only IP address in this path belonging to AS 12657, it is a missing AS hop in the second traceroute.

To establish high confidence that there was no path change between the source and the destination, we apply the following reasoning: if the root cause of the apparition of a star is a path change, then the star corresponds to a different IP address than `2001:1578:0:ff::1:2`. So it could mean that either AS 35003 changed its announcement for the destination prefix, or AS 8767 changed its best route to the prefix of the destination. In each case, AS 8767 should send a BGP update to its neighbors (here, AS 1103) to inform them that the route to the destination prefix has changed. According to the BGP protocol [147], for each AS on the path, the AS will propagate the update to its neighbors if the BGP update is different than the last BGP update sent for this prefix. And even if the update is the same, an AS might still send the update to its neighbors, because it does not keep in memory which update it has sent to a neighbor for a particular prefix (called BGP duplicates) [77].

Conversely, if there are no BGP updates sent and received by the ASes that appear before the star (in our example 1101, 1103, and 8767) on the AS path, the root cause of the star is not a path change, and the star is certainly due to a transient state of the router (e.g., rate limiting).

Formalization

We formalize the high-level idea that we described in the previous paragraph. For a given (source, destination) pair:

- Let p_t and $p_{t'}$ two paths measured by two traceroutes at time t and t' with $t < t'$.
- Let $h_1^{p_t}, ..., h_d^{p_t}$ and $h_1^{p_{t'}}, ..., h_{d'}^{p_{t'}}$ the IP hops of p_t and $p_{t'}$.
  Let $\text{AS}_1^{p_t}, ..., \text{AS}_d^{p_t}$ and $\text{AS}_1^{p_{t'}}, ..., \text{AS}_{d'}^{p_{t'}}$ the AS hops of p_t and $p_{t'}$ and A^{p_t} and $A^{p_{t'}}$ the corresponding set of ASes.
- Let B the ensemble of BGP updates between t and $t' + 5$ for the BGP prefix of the destination where t and t' are expressed in minutes.

We say that there is a missing AS hop in $p_{t'}$ if the following conditions are satisfied:

1. $d = d'$
2. $\forall k \in [1..d'], h_k^{p_t} = h_k^{p_{t'}} \vee h_k^{p_t} = \text{“}*\text{”} \vee h_k^{p_{t'}} = \text{“}*\text{”}$
3. $\exists i \in [1..d], h_i^{p_t} \neq \text{“}*\text{”} \wedge h_i^{p_{t'}} = \text{“}*\text{”} \wedge \text{AS}_i^{p_t} \notin A^{p_{t'}}$
4. $B = \emptyset$

Conditions (1), (2) and (3) define what we have explained in the previous paragraph. Each hop of path p_t and $p_{t'}$ must be identical except for stars, and at least one hop in path in $p_{t'}$ must be a star and an AS hop in p_t not appearing in $p_{t'}$ AS path. Note that for condition (3), to not make notations too heavy, we omit to formalize that p_t and $p_{t'}$ are interchangeable, but they actually are. Condition (4) assumes that we have an oracle for all of the BGP updates of the Internet and ensures that no BGP update has been made for the prefix of the destination. In practice, we have partial visibility on BGP updates, so we distinguish between two cases: whether we have a route collector within one of the ASes or not. If we do, as we have explained in the previous paragraph, it is highly likely that if there was a path change, the route collector would have seen a BGP update, and conversely, if we see no update in the route collectors, we are highly confident that there was no path change. If we do not have a route collector in one AS of the AS path with a star, we are less confident that there is no path change, as it could be that there is a path change but our collectors are not seeing the update because there is no path change for them. So we conservatively do not count

these cases as traceroutes with missing AS hops. Finally, in our case, t and t' are separated by a 15-minute interval, as they are taken from the meshed RIPE Atlas anchors measurements (see next section), so it is unlikely that the update arrives at the route collector after $t' + 5$, as the BGP convergence time usually takes up to 2 minutes [114, 60].

Notice that our conditions are conservative in the number of traceroutes with missing AS hops. For instance, it could be that condition (1) is not respected, i.e., paths have different lengths, because of asymmetric load balancing [169, 18] but still have a missing AS hop due to a star. Also, it could be that a BGP update has been sent for the prefix of the destination, but is not related to the star. For instance, if we have a route collector in AS 1101, in Tab. 5.4, AS 1103 could have sent a BGP update to AS 1101 for an unrelated reason with the star, so our results provide a lower bound of how our analysis is impacted by these missing AS hops in traceroutes.

5.7.3 Dataset

We collect 5 days of publicly available traceroutes from the RIPE Atlas platform from their FTP [149] from the 28th of September to the 2nd of October 2022, filtering the built-in traceroute measurements between anchors. These measurements consist of running meshed traceroutes between the RIPE Atlas anchors every 15 minutes. We only keep these recurrent traceroutes as we need multiple traceroutes between the same source and destination to be able to compare the paths close in time. We have a total of 576M traceroutes, 321M for IPv4 and 255M for IPv6, which consists of 548K (source, destination) pairs for IPv4 and 394K pairs in IPv6. We map the IP addresses to their ASes using the state-of-the-art method BDRMAPIT [117].

We collect all the BGP updates from all the RIPE RIS collectors (RRC00 to RRC26) during the same days as the traceroutes, for a total of 2.58B and 2.26B updates for IPv4 and IPv6.

5.7.4 Results

We compute all the missing AS hops in the traceroutes of our dataset using the methodology described in Sec. 5.7.2, where we compare each pair of consecutive traceroutes for each pair of source and destination.

We find 1,773,680 and 3,245,881 traceroutes with at least a potential missing AS hop (without condition (4) on BGP updates), and we only keep 1,613,904 and 2,719,671 for which there exists no BGP update and where we have a BGP collector in the AS path before the star. These numbers

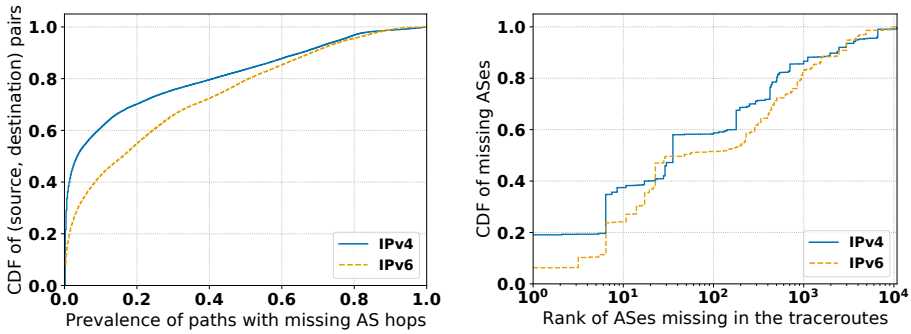


Figure 5.2: Paths with a missing AS hop are more prevalent in IPv6 (left). All types of ASes are missing in the traceroutes, from Tier 1s to small providers (right).

represent a small percentage of the total number of traceroutes (0.5% for IPv4 and 1.1% for IPv6). However 53,334 (9.7%) and 68,720 (17.5%) pairs of source and destination contain at least one traceroute with one missing AS hop for IPv4 and IPv6. It is higher for IPv6, and we believe this is because the rate limiting is a mandatory feature in ICMPv6 [42, 6], whereas it is optional in IPv4. We also remind that our methodology to infer that a traceroute contains a missing AS hop is conservative. We also find that 99% of the traceroutes with a missing AS hop have exactly one missing AS hop, so multiple missing AS hops are very rare.

How prevalent are missing AS hops in the Internet?

The number of (source, destination) pairs with at least one traceroute with a missing AS hop is both non-negligible for IPv4 (9.7%) and IPv6 (17.5%). The next question that arises next is: are these paths prevalent? In other terms, if we were to run a traceroute between a source and destination that can be affected by a missing AS hop, how likely is it that it would actually contain a missing AS hop? To answer this question, for each pair of source and destination, we compute the prevalence of each path as originally defined by Paxson [134], i.e., the frequency of this path during the five-day period. We are interested in knowing what is the probability that if we run a traceroute, it would contain a missing AS hop. The left plot of Fig. 5.2 shows the CDF of prevalence of the set of paths with a missing AS hop for each pair of source and destination. The prevalence of the set of paths with a missing AS hop is the sum of the prevalence of the paths with a missing AS hop for a given

(source, destination) pair. There are 15% and 20% of the IPv4 and IPv6 pairs of source and destination with the prevalence of paths with a missing AS hop higher than 0.5, meaning that if we run a traceroute between these (source, destination) pairs, we would get an incomplete AS path half of the time. This represents 1.5% and 3.5% of the total of (source, destination) pairs for IPv4 and IPv6, including the pairs without paths with missing AS hops, so it is non-negligible, in particular in IPv6.

Which ASes might we miss because of stars?

The right plot of Fig. 5.2 shows the CDF of the rank of the missing ASes in the traceroutes with a missing AS hop. The rank is obtained via the CAIDA AS rank dataset [106], which rank the ASes based on the reversed size of their customer cone, i.e., the number of ASes they can reach with only provider-to-customer and peer-to-peer links. The bigger the rank, the smaller the size of the customer cone. Both for IPv4 and IPv6, we observe that the missing ASes can be either Tier 1 or small providers as, for instance, in IPv4, 20% of the missing ASes correspond to AS 3356 (Level3) ranked first, which has a customer cone of 48,782, and 17% to ASes with a rank higher than 1000, corresponding to a customer cone smaller than 32.

5.7.5 Saving debugging time with MetaTrace

MetaTrace played an important role in data preprocessing, which is exactly the purpose that we want it to serve: save the time of researchers from performing tedious tasks. During the debugging of the code to build this study, we identified a bug in RIPE Atlas anchors meshed measurement. RIPE Atlas meshed measurements are supposed to happen every 15 minutes. However, we found that some (source, destination) pairs had consecutive traceroutes with very close timestamp ($<$ 5 seconds). We remove these 3,476 pairs for IPv4 and 1,702 pairs for IPv6 from our dataset, corresponding to 3.1M and 1.4M traceroutes. After having reported this problem to RIPE Atlas, they told us it was due to a bug with two instances of the program launching the meshed traceroutes, which was fixed since. We had to run a lot of predicate queries to find and confirm this bug in the data, once we had found the suspicious pairs, so MetaTrace helped us to be fast in investigating and characterizing this problem.

Second, to find the right formulation for our methodology, we had to manually look at dozens of traceroute paths with potential missing AS hops, to check at the end that the cases identified by the methodology were actually

missing AS hops. To do so, we ran dozens of predicate queries, saving us precious time compared to the time it would have taken with some manual scripts (Sec. 5.6).

5.8 Can we see Internet flattening from Ark?

Earlier work defines Internet flattening as a "reduction in the number of intermediary organizations on Internet paths" [33, 63]. However, more recent work has shown that there was no clear reduction of the average AS path length between 2007 and 2016, showing the emergence of new organizations rather than the reduction of intermediary ones [33].

We reappraise these results with a more rigorous methodology: (1) we do not choose arbitrary snapshots of traceroutes (Sec. 5.8.1); and (2) we do not aggregate traceroutes per period of time, which suffers from missing data due to the source and destination not being constant over time, but rather identify trends per (source, destination) pairs (Secs. 5.8.1 and 5.8.2). Surprisingly, we find that from 2016 to 2021, there is neither a clear trend in AS path length evolution, nor in the evolution of the hierarchical structure of the Internet paths in the Ark publicly available dataset.

5.8.1 Dataset

In this study, we consider the Ark Prefix-Probing dataset [27]. This dataset contains daily[2] traceroutes to every announced BGP prefix from a subset of Ark monitors, starting from January 2016. To account for nodes being added over time and destinations changing due to BGP dynamics, we conservatively keep the (source, destination) pairs for which we have had at least 15 traceroutes reaching the destination, per month and since 2016. We map the IP addresses to their AS using BDRPMAPIT [117]. We use windows of 15 days as was used in the BDRMAPIT paper to map the IP addresses to their AS.

We obtain 9.7M pairs, where the destinations are in 39,477 ASes representing 83% of the Internet population, according to the APNIC dataset [10], ensuring a large-scale coverage of the study.

5.8.2 Aggregate statistics and results

We make use of MetaTrace to compute two time series for each (source, destination) pair: the AS path length and the number of tier 1 in the AS

[2] In practice, we find that the time interval between two traceroutes towards the same prefix oscillates between one and two days.

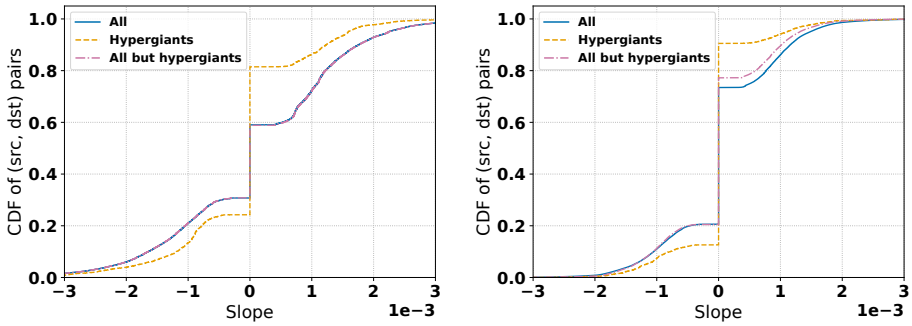


Figure 5.3: CDFs of the slope of the linear regression performed on time series of AS path length (left) and number of Tier 1 in the AS path (right) for All ASes, hypergiants, and all ASes but hypergiants.

path. Rather than aggregating these metrics per unit of time [24], which can lead to missing AS hops in the AS paths (Sec. 5.7), we seek to capture the trend on these time series. We compute a linear regression on the time series and extract the slope and the correlation coefficient between the predicted line and the actual time series. After 18 hours for inserting the 6B traceroutes in MetaTrace, it takes 15 minutes to compute the two metrics on the whole dataset.

Fig. 5.3 shows the CDF of the slope on the 847K (AS path length) and 820K (Tier 1 in the AS path) (source, destination) pairs for which the correlation coefficient is above 0.4, a threshold where the predicted line and the actual time series are moderately correlated [153]. A negative slope indicates that the metric is decreasing, while a positive slope indicates that the metric is increasing. We show the results for all paths, paths going to hypergiants [25] and paths to all ASes but hypergiants. There are two takeaways. First, there is no clear trend showing that paths are shortening, or turning into a less hierarchical structure, as there are for each metric at least the same number of paths having a slope > 0 and a slope < 0. Second, we observe that the metrics for paths to hypergiants are more stable than paths to other ASes. For instance, 59% of the paths to hypergiants have a slope of 0 for AS path length, against 27% for the other paths. This is surprising, as we would expect hypergiants to be faster than other ASes to make new peerings and reduce their Tier 1 dependency.

Limitations To be clear, these results are limited to Ark's monitors, and we are not denying here the existence of Internet flattening. Indeed, prior

work has shown that Google could reach 76% of the ASes of the Internet in one hop [12], and that they are continually expanding and limited the Tier 1's influence [12, 62]. This study only sheds light on what we can find on AS path length and Tier 1 influence with a more rigorous statistical method than prior work on the Ark dataset over 5 years, i.e., that there is no evidence of a decrease of the AS path length or of the influence of Tier 1s.

5.9 Conclusion

We presented MetaTrace, a framework for fast traceroute processing useful for both operators and researchers. We showed that MetaTrace can serve predicate queries more than 500 times faster than a multiprocessed Rust implementation and more than 3 times faster than a raw database. With MetaTrace, we show that a non-negligible number of source and destination pairs, especially in IPv6, can experience stars that can hide entire ASes from the AS path. Finally, MetaTrace allowed us to perform a study on Internet flattening over 5 years on 6 billion traceroutes, and showed the lack of statistical evidence of a trend where Tier 1 influence or AS path length decreases.

Chapter 6

Conclusion

This chapter concludes the thesis by giving a summary of the contributions and presenting the perspectives of this work.

6.1 Summary of contributions

This thesis presented contributions that scale the capture and the analysis of route tracing data.

We first designed and developed Zeph, an algorithm based on a reinforcement learning approach to make the most of multiple vantage points and maximize the discovery of IP-level topology. We also developed Iris, a production-ready measurement platform based on open-source components and code, which allows for the use of high-speed probing of single-path and multi-path IP routes at Internet-scale. With the combination of Zeph and Iris, we were able to capture 10 times more links than the previous state of the art and now provide regular IP-level topology datasets to the Internet measurement community.

The analysis of route tracing data requires to use additional metadata, such as IP geolocation or IP-to-AS resolution, to conduct meaningful studies. We have developed metrics and tools to study the data from a well-known commercial IP geolocation database and have found that geolocation data can be highly dynamic over time. We have shared these insights with the community to help them use this valuable data without introducing bias into their own studies.

Collecting a massive amount of traceroute data requires rethinking the way we analyze it at scale. We designed and developed a new MetaTrace, a large-scale traceroute processing framework, that handles the storage and

querying of route tracing data along with additional metadata in an efficient way. This enables us to fully benefit from data series such as Iris and Zeph, as well as data from other platforms like Ark [26] and RIPE Atlas [148], and metadata like IP geolocation and IP-to-AS resolution from various sources. With MetaTrace, we analyzed incomplete AS paths due to stars in traceroutes and examined Internet flattening over 5 years of route tracing data.

All of the presented tools are available to the community as free, open-source and liberally licensed software [159].

6.2 Perspectives

The contributions presented in this thesis allow scaling the capture and the analysis of the Internet topology. It leverages high-speed probing tools [21, 168, 79] that present both an opportunity and a danger. Probing faster enable more complete maps or tracking changes at a smaller temporal granularity. But these measurements can be considered harmful. Probed repeatedly, routers may have to allocate a significant amount of resources to handle probes and respond to it, potentially disrupting the traffic of other networks' users. The measurements could also trigger Intrusion Detection System (IDS) alarms that could disrupt the monitoring of network components of the Internet. Network operators may use a rate limiting configuration on router interfaces, which limit the number of packets per second crossing this interface for a particular category of traffic. It usually has a default value that can be tuned by network operators. When a probe is dropped by a router interface due to rate limiting, no response is generated, so a traceroute will display a star. This can lead to bias in the results of the measurements themselves.

The Menlo Report [92] introduced the principle of Beneficence applied to information and communication technology research. They described it as "the concept of appropriately balancing probable harm and likelihood of enhanced welfare resulting from the research". In particular, they define beneficence as "the maximization of benefits and minimization of harms". Partridge et al. [132] precised this concept to Internet measurement: "we offer that a single ICMP echo request to an IP address constitutes at best slight harm. Meanwhile, a persistent high-rate series of probes to a given IP address may well be viewed as both an attack and create serious harm (e.g., by clogging a link precisely when it is needed for an emergency)."

In this perspective paragraph, we discuss on the ethics of Internet measurements by: (1) proposing a definition of beneficence for ping and traceroute

measurements based on rate limiting; and (2) stating the challenges for making measurements that meet our definition.

6.2.1 Use of rate limiting as a beneficient proxy

Rate limiting is a way for operators to specify that traffic is abnormal. Previous studies have shown that measuring below the router's rate limit moderate the use of resources and do not affect the results of external measurements [167]. In this context, we propose to define that a ping or traceroute measurement is beneficient if it minimizes the rate limit violations of a device.

This definition is subject to debate, but has the quality of defining a measurable value, chosen by each operator, as what it considers to be abnormal traffic for its network. However, previous studies have shown that a significant portion of operators uses a default rate limit value, which does not reflect a desire to limit ping and traceroute traffic. To assist operators in choosing this setting, previous studies have shown the impact of different probing rates on the resources consumed by routers [167].

In favor of our definition, we mention that rate limiting is mandatory in IPv6 [50], and thus the rate limit will be more and more a factor to be taken into account by the different actors who make measurements in the Internet.

6.2.2 Challenges in building a beneficient system

The definition we propose for the particular case of ping and traceroute measurements is more concrete than those proposed in previous works, but it brings additional challenges. (1) Unless the operator discloses the rate limit configured on its interfaces, whoever is measuring does not know the rate limit a priori. The first challenge is therefore to measure the rate limit of the interfaces. (2) If we know the rate limit of the interfaces, we still have to design measurement algorithms that respect this rate. The second challenge is to build a system which, with the knowledge of the limit rates, will carry out its measurements to try to never exceed the limit rate.

We do not yet have definitive and evaluated solutions to meet these challenges, but we do describe paths to building solutions and the problems they will face.

How to measure the rate limit of an interface?

The rate limit can vary greatly from one interface to another, from a few packets per second to tens of thousands [167].

Since the signal for exceeding the rate limit is a loss (non-response to a packet), solutions that seem interesting are adaptive algorithms such as AIMD (additional increase, multiplicative decrease) of the TCP protocol.

In the case of ping type measurements, we can directly apply these algorithms by probing the destinations, as it is the destinations themselves that respond to the probed packets. In the case of traceroute type measurements, the problem is different, because the measurements are indirect: the destination address of the probe packet is not the address that responds, and without prior knowledge, we do not know which address will respond. A possible solution is therefore to perform initial measurements with a conservative rate (e.g., 100 pps) to find out which interfaces correspond to which correspond to traceroute probes. We can then apply our TCP-inspired algorithms to measure the rate limit. Note that although these algorithms work for any rate limit, it seems obvious that setting a maximum rate is necessary. There is no need to send tens of thousands of packets per second to an interface.

These techniques seem feasible, but raise several questions.

(1) can we scale up these techniques? From 6 measurement points, we find in Chap. 3 that the Internet topology is composed of at least 3.3 million interfaces. We propose the following rough estimate that corresponds to a realistic infrastructure: we have 6 measurement points that can probe at 100,000 pps. We take an initial rate of 1 pps, a maximum rate of 128 pps and a multiplication factor of 2. Assuming that no interface has a limiting rate below 128 pps, and assuming that we can distribute the number of interfaces between our 6 nodes, the number of probes to send per interface is therefore $\sum_{k=1}^{7} 2^k = 254$, and so the total number is 838,200,000, or 1.397 seconds, which is about 23 minutes, a time that seems reasonable.

(2) How can we be confident that a loss is due to self-induced rate limiting? It is possible that a loss is due to other factors, such as congestion, or a path change to a router that does not respond to probe packets. We can probably use signals corresponding to the two phenomena mentioned above, such as the increase in the variance of RTT for congestion, and BGP update messages collected by public collectors, BGP communities, or even path changes in public traceroutes [64].

(3) What if the initial measurements become invalid? Even if the Internet paths are globally stable, it is possible that our initial measurements are no longer valid. Indeed, with 10M prefixes /24 routed, one traceroute per prefix, and on average one traceroute path with 15 interfaces, it takes about 17 days to complete these initial measurements. Recent work showed that 16% of the paths were invalid after 17 days [64]. In our case, we can just check before launching the rate limit measurements that there is at least one

traceroute probe per interface that is still valid to measure the rate limit of this interface.

(4) What to do when the rate limit is implemented by routers in terms of bandwidth used by ping and traceroute traffic? In this case, all ping or traceroute traffic, past a certain limit will be lost on the path beyond the router that applies to this rule. Preliminary studies show that this is not a common phenomenon [167], but it can add time to the rate limit measurements of interfaces on the paths after these routers because these routers represent bottlenecks, and it may even make our rate limit measurements impossible if the interfaces beyond these routers have a higher rate limit than these routers.

How not to exceed the rate limit during a measurement?

Once the interface rate limit has been determined, the next step is to adapt the measurement so that it does not trigger this rate limit. To do this, the probes must be scheduled so that the probing rate of an interface does not exceed the threshold rate, or even a fraction of the threshold rate to be even more conservative.

As before, this is fairly straightforward to implement for ping probes, but it is more complicated for traceroutes, because we don't know in advance which interface will respond to a traceroute probe. However, we can use the results of the initial measurements to obtain the correspondence between traceroute probes and respond interfaces.

Here, however, we run into several problems.

(1) As seen previously, the Internet paths are dynamic (although globally stable), which can disturb our algorithm. Indeed, for example, if a path changes and the traceroute probe now passes through an interface that we previously probed that was close to its rate limit, this new probe can trigger the rate limit. An imprecise but simple approach is to predict that the interface that will respond is the one that responded previously. Given the stability of the paths of the Internet, if we can get regular information about traceroutes from the same source to the same destination (on a scale of a few minutes), there is little risk of error. A more accurate approach, but one that requires more thought, is to try to predict which interface will respond to a traceroute. This is the most exploratory part of this future work, but perhaps approaches based on learning about path changes is a way forward. Indeed, we can restrict the set of possible paths to which a path can change from the existing topology, assuming that one has a fairly complete view of the paths from a source, a reasonable assumption given the stability of the

paths from the Internet [64].

(2) The second problem is the same as for the measurement of interface rate limits: How can one be confident that a loss corresponds to a rate limit violation? The solution may lie in a statistical model based on previous measurements to these interfaces, which could tell whether a certain loss rate is likely or not.

(3) Finally, what to do when the rate limit is triggered despite our efforts? In this case, we have to decrease the polling on some interfaces, which will either lengthen the measurement time, or, if we decide not to send all the probes, reduce the completeness of the data. It will then be necessary to choose which probes we decide not to send. In this case, it will probably be possible to give several use cases and choose the probes not to be sent according to these cases (e.g., maximizing discovery in certain parts of the network, or maximizing the number of complete paths).

Bibliography

[1] Fantail project, 2023.

[2] RIPE Atlas presentation on Hadoop and BigQuery, 2023.

[3] D. J. Abadi, S. R. Madden, and N. Hachem. Column-stores vs. row-stores: how different are they really? In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 967–980, 2008.

[4] B. Ager, N. Chatzis, A. Feldmann, N. Sarrar, S. Uhlig, and W. Willinger. Anatomy of a large European IXP. In *Proc. ACM SIGCOMM*. ACM, 2012.

[5] R. L. C. Almeida, I. Cunha, R. Teixeira, D. Veitch, and C. Diot. Classification of Load Balancing in the Internet. In *Proc. IEEE INFOCOM*, 2020.

[6] P. Alvarez, F. Oprea, and J. Rula. Rate-limiting of ipv6 traceroutes is widespread: measurements and mitigations. *Proceedings of IETF*, 99, 2017.

[7] Amazon. AWS S3. https://aws.amazon.com/fr/s3/.

[8] D. G. Andersen, N. Feamster, S. Bauer, and H. Balakrishnan. Topology inference from bgp routing dynamics. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pages 243–248, 2002.

[9] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, et al. Understanding the mirai botnet. In *USENIX Security*, pages 1093–1110, 2017.

[10] APNIC. Visible ASNs: Customer Populations Estimations. https://stats.labs.apnic.net/aspop/.

[11] APNIC. APNIC BGP prefixes. https://blog.apnic.net/2022/01/06/bgp-in-2021-the-bgp-table/, 2022.

[12] T. Arnold, J. He, W. Jiang, M. Calder, I. Cunha, V. Giotsas, and E. Katz-Bassett. Cloud provider connectivity in the flat internet. In *Proc. ACM IMC*, pages 230–246, 2020.

[13] Asghari, Hadi. PyASN. https://github.com/hadiasghari/pyasn.

[14] B. Augustin, X. Cuvellier, B. Orgogozo, F. Viger, T. Friedman, M. Latapy, C. Magnien, and R. Teixeira. Avoiding traceroute anomalies with Paris traceroute. In *Proc. ACM IMC*, 2006.

[15] B. Augustin, T. Friedman, and R. Teixeira. Exhaustive path tracing with Paris traceroute. In *Proc. ACM CoNEXT '07*, 2006.

[16] B. Augustin, T. Friedman, and R. Teixeira. Measuring load-balanced paths in the internet. In *Proc. ACM IMC*, page 149–160, New York, NY, USA, 2007.

[17] B. Augustin, T. Friedman, and R. Teixeira. Multipath tracing with Paris traceroute. In *Proc. E2EMON '07*, 2007.

[18] B. Augustin, T. Friedman, and R. Teixeira. Measuring multipath routing in the Internet. *IEEE/ACM Transactions on Networking*, 19(3):830–840, June 2011.

[19] P. Barford, A. Bestavros, J. Byers, and M. Crovella. On the marginal utility of network topology measurements. In *Proc. ACM SIGCOMM Internet Measurement Workshop*, IMW '01, 2001.

[20] A. Bender, R. Sherwood, and N. Spring. Fixing ally's growing pains with velocity modeling. In *Proc. ACM IMC*, pages 337–342, 2008.

[21] R. Beverly. Yarrp'ing the Internet: Randomized high-speed active topology discovery. In *Proc. ACM IMC*, 2016.

[22] R. Beverly, A. Berger, and G. G. Xie. Primitives for active internet topology mapping: Toward high-frequency characterization. In *Proc. ACM IMC*, pages 165–171, 2010.

[23] R. Beverly, R. Durairajan, D. Plonka, and J. P. Rohrer. In the IP of the beholder: Strategies for active IPv6 topology discovery. In *Proc. ACM IMC*, 2018.

[24] T. Böttger, G. Antichi, E. L. Fernandes, R. di Lallo, M. Bruyere, S. Uhlig, and I. Castro. The elusive internet flattening: 10 years of ixp growth. *CoRR*, 2018.

[25] T. Böttger, F. Cuadrado, and S. Uhlig. Looking for hypergiants in peeringdb. *Proc. ACM SIGCOMM Computer Communication Review*, 48(3):13–19, 2018.

[26] CAIDA. Archipelago. https://www.caida.org/projects/ark.

[27] CAIDA. Ark prefix probing dataset. https://www.caida.org/catalog/datasets/ipv4_prefix_probing_dataset/.

[28] CAIDA. ITDK. https://www.caida.org/catalog/datasets/Internet-topology-data-kit/.

[29] CAIDA. Vela. https://www.caida.org/projects/ark/vela/.

[30] CAIDA. IPv4 routed /24 topology dataset. https://www.caida.org/catalog/datasets/ipv4_routed_24_topology_dataset/, 2008. February 1, 2008; version of July 8, 2020.

[31] CAIDA. The impact of the archipelago measurement platform. https://www.caida.org/projects/ark/impact/, 2014. July 3, 2014; version of November 15, 2019.

[32] F. Cajori. *A history of mathematical notations*, volume 1. Courier Corporation, 1993.

[33] I. Castro, J. C. Cardona, S. Gorinsky, and P. Francois. Remote peering: More peering without internet flattening. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, pages 185–198, 2014.

[34] Y.-C. Chiu, B. Schlinker, A. B. Radhakrishnan, E. Katz-Bassett, and R. Govindan. Are We One Hop Away from a Better Internet? In *Proc. ACM IMC*, 2015.

[35] k. claffy, Y. Hyun, K. Keys, M. Fomenkov, and D. Krioukov. Internet mapping: From art to science. In *IEEE DHS Cybersecurity Applications and Technologies Conference for Homeland Security (CATCH)*, 2009.

[36] ClickHouse. ClickHouse Array Functions. https://clickhouse.com/docs/en/sql-reference/functions/array-functions.

[37] ClickHouse. ClickHouse database. https://clickhouse.yandex.

[38] ClickHouse. ClickHouse dictionnaries. https://clickhouse.com/docs/en/operations/system-tables/dictionaries/.

[39] ClickHouse. ClickHouse materialized columns. https://clickhouse.com/docs/en/sql-reference/statements/alter/column/#materialize-column.

[40] ClickHouse. ClickHouse materialized views. https://clickhouse.com/docs/en/sql-reference/statements/create/view.

[41] ClickHouse. Understanding ClickHouse Data Skipping Indexes. https://clickhouse.com/docs/en/guides/improving-query-performance/skipping-indexes/.

[42] A. Conta and S. Deering. Rfc1885: Internet control message protocol (icmpv6) for the internet protocol version 6 (ipv6). https://tools.ietf.org/html/rfc1885, 1995.

[43] G. Cormode, H. Karloff, and A. Wirth. Set cover algorithms for very large datasets. In *Proc. ACM CIKM '10*, pages 479–488. ACM Press, 2010.

[44] B. C. Şenel, M. Mouchet, J. Cappos, O. Fourmaux, T. Friedman, and R. McGeer. Edgenet: A multi-tenant and multi-provider edge cloud. In *In Proc. ACM Intl. Workshop on Edge Systems, Analytics and Networking*, EdgeSys '21, 2021.

[45] J. A. Culbert. Toward Understanding the Longitudinal Stability of an IP Geolocation Database. Master's thesis, Naval Postgraduate School, Mar. 2020. http://hdl.handle.net/10945/64848.

[46] I. Cunha, P. Marchetta, M. Calder, Y.-C. Chiu, B. Schlinker, B. V. A. Machado, A. Pescapé, V. Giotsas, H. V. Madhyastha, and E. Katz-Bassett. Sibyl: A Practical Internet Route Oracle. In *Proc. USENIX NSDI*, 2016.

[47] Í. Cunha, R. Teixeira, D. Veitch, and C. Diot. Dtrack: a system to predict and track internet path changes. *IEEE/ACM Transactions on Networking*, 22(4):1025–1038, 2013.

[48] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A decentralized network coordinate system. *Proc. ACM SIGCOMM Computer Communication Review*, 34(4):15–26, 2004.

[49] DARPA Internet Program. RFC 791: Internet Protocol. https://tools.ietf.org/html/rfc791, 1981.

[50] S. Deering and R. Hinden. Rfc 8200: Internet protocol, version 6 (ipv6) specification. https://tools.ietf.org/html/rfc8200, 2017.

[51] J. Deng, G. Tyson, F. Cuadrado, and S. Uhlig. Internet scale user-generated live video streaming: The twitch case. In *International Conference on Passive and Active Network Measurement*, pages 60–71. Springer, 2017.

[52] A. Dhamdhere, D. D. Clark, A. Gamero-Garrido, M. Luckie, R. K. Mok, G. Akiwate, K. Gogia, V. Bajpai, A. C. Snoeren, and K. Claffy. Inferring persistent interdomain congestion. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 1–15, 2018.

[53] A. Dhamdhere and C. Dovrolis. The internet is flat: modeling the transition from a transit hierarchy to a peering mesh. In *Proceedings of the 6th International COnference*, pages 1–12, 2010.

[54] Docker. Docker. https://www.docker.com/.

[55] B. Donnet and T. Friedman. Internet topology discovery: a survey. *IEEE Communications Surveys & Tutorials*, 9(4):56–69.

[56] B. Donnet, M. Luckie, P. Mérindol, and J.-J. Pansiot. Revealing mpls tunnels obscured from traceroute. *Proc. ACM SIGCOMM Computer Communication Review*, 42(2):87–93, Mar. 2012.

[57] B. Donnet, P. Raoult, T. Friedman, and M. Crovella. Efficient algorithms for large-scale topology discovery. In *Proc. ACM SIGMETRICS Conf.*, SIGMETRICS '05, 2005.

[58] B. Du, M. Candela, B. Huffaker, A. C. Snoeren, and k. claffy. Ripe ipmap active geolocation: Mechanism and performance evaluation. *Proc. ACM SIGCOMM Computer Communication Review*, 50(2):3–10, may 2020.

[59] B. Eriksson, P. Barford, J. Sommers, and R. Nowak. A learning-based approach for IP geolocation. In *International Conference on Passive and Active Network Measurement*, pages 171–180. Springer, 2010.

[60] A. García-Martínez and M. Bagnulo. Measuring bgp route propagation times. *IEEE Communications Letters*, 23(12):2432–2436, 2019.

[61] M. Gharaibeh, A. Shah, B. Huffaker, H. Zhang, R. Ensafi, and C. Papadopoulos. A look at router geolocation in public and commercial databases. In *Proc. ACM IMC*, pages 463–469, 2017.

[62] P. Gigis, M. Calder, L. Manassakis, G. Nomikos, V. Kotronis, X. Dimitropoulos, E. Katz-Bassett, and G. Smaragdakis. Seven years in the life of hypergiants' off-nets. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pages 516–533, 2021.

[63] P. Gill, M. Arlitt, Z. Li, and A. Mahanti. The flattening internet topology: Natural evolution, unsightly barnacles or contrived collapse? In *International Conference on Passive and Active Network Measurement*, pages 1–10. Springer, 2008.

[64] V. Giotsas, T. Koch, E. Fazzion, Í. Cunha, M. Calder, H. V. Madhyastha, and E. Katz-Bassett. Reduce, reuse, recycle: Repurposing existing measurements to identify stale traceroutes. In *Proc. ACM IMC*, pages 247–265, 2020.

[65] V. Giotsas, M. Luckie, B. Huffaker, and k. claffy. Inferring complex as relationships. In *Proc. ACM IMC*, IMC '14, 2014.

[66] Google. BigQuery. https://cloud.google.com/bigquery.

[67] Google. Google Compute Engine (GCE). https://cloud.google.com/.

[68] M. Gouel, K. Vermeulen, O. Fourmaux, T. Friedman, and R. Beverly. Ip geolocation database stability and implications for network research. In *Network Traffic Measurement and Analysis Conference*, 2021.

[69] M. Gouel, K. Vermeulen, O. Fourmaux, T. Friedman, and R. Beverly. Longitudinal Study of an IP Geolocation Database. Technical report, 2021. https://arxiv.org/abs/2107.03988.

[70] M. Gouel, K. Vermeulen, M. Mouchet, J. Rohrer, O. Fourmaux, and T. Friedman. Zeph & iris cartographient l'internet. In *CORES 2022–7ème Rencontres Francophones sur la Conception de Protocoles, l'Évaluation de Performance et l'Expérimentation des Réseaux de Communication*, 2022.

[71] M. Gouel, K. Vermeulen, M. Mouchet, J. P. Rohrer, O. Fourmaux, and T. Friedman. Zeph & iris map the internet: A resilient reinforcement learning approach to distributed ip route tracing. *Proc. ACM SIGCOMM Computer Communication Review*, 52(1):2–9, 2022.

[72] R. Govindan and H. Tangmunarunkit. Heuristics for Internet map discovery. In *Proc. IEEE INFOCOM*, 2000.

[73] Grafana. Grafana. https://grafana.com/.

[74] Grafana. Loki. https://grafana.com/oss/loki/.

[75] B. Gueye, A. Ziviani, M. Crovella, and S. Fdida. Constraint-based geolocation of internet hosts. *IEEE/ACM Transactions On Networking*, 14(6):1219–1232, 2006.

[76] M. H. Gunes and K. Sarac. Resolving ip aliases in building traceroute-based internet maps. *IEEE/ACM Transactions on Networking*, 17(6):1738–1751, 2009.

[77] D. Hauweele, B. Quoitin, C. Pelsser, and R. Bush. What do parrots and bgp routers have in common? *Proc. ACM SIGCOMM Computer Communication Review*, 46(3):1–6, 2018.

[78] Z. Hu, J. Heidemann, and Y. Pradkin. Towards geolocation of millions of IP addresses. In *Proc. ACM IMC*, pages 123–130, 2012.

[79] Y. Huang, M. Rabinovich, and R. Al-Dalky. Flashroute: Efficient traceroute on a massive scale. In *Proc. ACM IMC*, pages 443–455, 2020.

[80] B. Huffaker, M. Fomenkov, and K. Claffy. Geocompare: A Comparison of Public and Commercial Geolocation Databases. pages 1–12, 2011.

[81] B. Huffaker, M. Fomenkov, and k. claffy. DRoP: DNS-based router positioning. *Proc. ACM SIGCOMM Computer Communication Review*, 44(3):5–13, July 2014.

[82] IP2Location. IP2Location database. https://www.ip2location.com.

[83] IPinfo. IPinfo database. https://ipinfo.io/.

[84] ITU. Measuring digital development: Facts and figures 2022. https://www.itu.int/hub/publication/d-ind-ict_mdd-2022/.

[85] V. Jacobson. 4BSD routing diagnostic tool available for ftp. Email 8812201313.AA03127@helios.ee.lbl.gov to the IETF and end2end-interest e-mail lists, 1988.

[86] P. Jacquet, M. L. Lamali, and F. Mathieu. Collecter un nombre inconnu de coupons. In *CORES 2018-Rencontres Francophones sur la Conception de Protocoles, l'Évaluation de Performance et l'Expérimentation des Réseaux de Communication*, pages 1–4, 2018.

[87] Y. Jin, C. Scott, A. Dhamdhere, V. Giotsas, A. Krishnamurthy, and S. Shenker. Stable and Practical AS Relationship Inference with ProbLink. In *Proc. USENIX NSDI*, 2019.

[88] JSON. JSON format. https://www.json.org/json-en.html.

[89] E. Katz-Bassett, J. P. John, A. Krishnamurthy, D. Wetherall, T. Anderson, and Y. Chawathe. Towards ip geolocation using delay and topology measurements. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 71–84, 2006.

[90] E. Katz-Bassett, J. P. John, A. Krishnamurthy, D. Wetherall, T. Anderson, and Y. Chawathe. Towards IP geolocation using delay and topology measurements. In *Proc. ACM IMC*, 2006.

[91] E. Katz-Bassett, H. V. Madhyastha, V. K. Adhikari, C. Scott, J. Sherry, P. Van Wesep, T. E. Anderson, and A. Krishnamurthy. Reverse traceroute. In *Proc. USENIX NSDI*, 2010.

[92] E. Kenneally and D. Dittrich. The menlo report: Ethical principles guiding information and communication technology research. *Available at SSRN 2445102*, 2012.

[93] K. Keys. Internet-scale ip alias resolution techniques. *Proc. ACM SIGCOMM Computer Communication Review*, 40(1):50–55, 2010.

[94] K. Keys, Y. Hyun, M. Luckie, and K. Claffy. Internet-scale ipv4 alias resolution with midar. *IEEE/ACM Transactions on Networking*, 21(2):383–399, 2012.

[95] S. Kim and K. Harfoush. Efficient estimation of more detailed internet ip maps. In *2007 IEEE International Conference on Communications*, pages 377–384. IEEE, 2007.

[96] Kline, E. and Duleba, K. and Szamonek, Z. and Moser, S. and Kumari, W. RFC 8805: A Format for Self-Published IP Geolocation Feeds. https://tools.ietf.org/html/rfc8805, 1981.

[97] D. Komosnỳ, M. Vozňák, and S. U. Rehman. Location accuracy of commercial ip address geolocation databases. 2017.

[98] V. Kotronis, G. Nomikos, L. Manassakis, D. Mavrommatis, and X. Dimitropoulos. Shortcuts through colocation facilities. In *Proc. ACM IMC*, pages 470–476, 2017.

[99] V. Kuleshov and D. Precup. Algorithms for multi-armed bandit problems, 2014.

[100] Y. Lee and N. Spring. Identifying and aggregating homogeneous IPv4 /24 blocks with hobbit. In *Proc. ACM IMC*, pages 151–165, 2016.

[101] H. W. Lilliefors. On the Kolmogorov-Smirnov test for normality with mean and variance unknown. *Journal of the American statistical Association*, 62(318):399–402, 1967.

[102] M. Luckie. Scamper: A scalable and extensible packet prober for active measurement of the Internet. In *Proc. ACM IMC*, 2010.

[103] M. Luckie and R. Beverly. The impact of router outages on the as-level internet. In *Proc. ACM SIGCOMM Conf.*, SIGCOMM '17, 2017.

[104] M. Luckie, R. Beverly, W. Brinkmeyer, and k. claffy. Speedtrap: Internet-scale ipv6 alias resolution. In *Proc. ACM IMC*, IMC '13, 2013.

[105] M. Luckie, A. Dhamdhere, B. Huffaker, D. Clark, and k. claffy. Bdrmap: Inference of borders between IP networks. In *Proc. ACM IMC*, 2016.

[106] M. Luckie, B. Huffaker, k. claffy, A. Dhamdhere, and V. Giotsas. AS relationships, customer cones, and validation. In *Proc. ACM IMC*, 2013.

[107] M. Luckie, B. Huffaker, A. Marder, Z. Bischof, M. Fletcher, and K. Claffy. Learning to extract geographic information from internet router hostnames. In *Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies*, pages 440–453, 2021.

[108] M. Luckie, Y. Hyun, and B. Huffaker. Traceroute probe method and forward IP path inference. In *Proc. ACM IMC*, 2008.

[109] J.-R. Luttringer, Y. Vanaubel, P. Mérindol, J.-J. Pansiot, and B. Donnet. Let there be light: Revealing hidden mpls tunnels with tnt. *IEEE Transactions on Network and Service Management*, 17(2):1239–1253, 2019.

[110] M-Lab. M-Lab. https://www.measurementlab.net.

[111] M-Lab. Network diagnostic tool (ndt). https://www.measurementlab.net/tests/ndt/index.html.

[112] H. Madhyastha, E. Katz-Bassett, T. Anderson, A. Krishnamurthy, and A. Venkataramani. iPlane Nano: Path Prediction for Peer-to-peer Applications. In *Proc. USENIX NSDI*, 2009.

[113] H. V. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani. iplane: An information plane for distributed services. In *Proc. USENIX OSDI*, pages 367–380, 2006.

[114] Z. M. Mao, R. Bush, T. G. Griffin, and M. Roughan. Bgp beacons. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, pages 1–14, 2003.

[115] Z. M. Mao, J. Rexford, J. Wang, and R. H. Katz. Towards an accurate as-level traceroute tool. In *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '03, page 365–378, New York, NY, USA, 2003. Association for Computing Machinery.

[116] P. Marchetta, V. Persico, and A. Pescapé. Pythia: yet another active probing technique for alias resolution. In *Proceedings of the Ninth ACM conference on Emerging networking experiments and technologies*, pages 229–234, 2013.

[117] A. Marder, M. Luckie, A. Dhamdhere, B. Huffaker, J. M. Smith, et al. Pushing the boundaries with bdrmapIT: Mapping router ownership at Internet scale. In *Proc. ACM IMC*, 2018.

[118] MaxMind. MaxMind database. https://www.maxmind.com/en/home.

[119] MaxMind. Significant changes to accessing and using geolite2 databases. https://blog.maxmind.com/2019/12/18/significant-changes-to-accessing-and-using-geolite2-databases/.

[120] MinIO. MinIO. https://min.io.

[121] MySQL. Comparison of B-Tree and Hash Indexes. https://dev.mysql.com/doc/refman/8.0/en/index-btree-hash.html.

[122] MySQL. MySQL. https://www.mysql.com.

[123] P. Mérindol, B. Donnet, J.-J. Pansiot, M. Luckie, and Y. Hyun. Merlin: Measure the router level of the internet. In *Proc. Conference on Next Generation Internet Networks (EURO-NGI '11)*, 2011.

[124] Neo4j. Neo4j Cypher. https://neo4j.com/developer/cypher/.

[125] Neo4j. Neo4j database. https://neo4j.com/fr.

[126] Neo4j. Neo4j logging capabilities. https://neo4j.com/docs/operations-manual/current/monitoring/logging/.

[127] NetAcuity. NetAcuity database. https://www.digitalelement.com/solutions/.

[128] R. Padmanabhan, A. Schulman, D. Levin, and N. Spring. Residential links under the weather. In *Proc. ACM SIGCOMM*, pages 145–158, 2019.

[129] V. N. Padmanabhan and L. Subramanian. An Investigation of Geographic Mapping Techniques for Internet Hosts. In *Proc. ACM SIGCOMM*, page 173–185, 2001.

[130] J.-J. Pansiot and D. Grad. On routes and multicast trees in the Internet. *Proc. ACM SIGCOMM Computer Communication Review*, 28(1):41–50, Jan. 1998.

[131] P. Papadopoulos, N. Kourtellis, P. R. Rodriguez, and N. Laoutaris. If you are not paying for it, you are the product: How much do advertisers pay to reach you? In *Proc. ACM IMC*, pages 142–156, 2017.

[132] C. Partridge and M. Allman. Ethical considerations in network measurement papers. *Commun. ACM*, 59(10):58–64, 2016.

[133] V. Paxson. End-to-end routing behavior in the internet. *IEEE/ACM transactions on Networking*, 5(5):601–615, 1997.

[134] V. Paxson. End-to-end Internet packet dynamics. *IEEE/ACM Transactions on Networking*, 7(3):277–292, 1999.

[135] V. Paxson. Strategies for Sound Internet Measurement. In *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*, page 263–271, 2004.

[136] P. Pearce, B. Jones, F. Li, R. Ensafi, N. Feamster, N. Weaver, and V. Paxson. Global Measurement of DNS Manipulation. In *USENIX Security)*, pages 307–323, 2017.

[137] PeeringDB. PeeringDB. https://www.peeringdb.com/.

[138] PlanetLab. PlanetLab Europe. https://www.planet-lab.eu.

[139] I. Poese, S. Uhlig, M. A. Kaafar, B. Donnet, and B. Gueye. IP Geolocation Databases: Unreliable? *Proc. ACM SIGCOMM Computer Communication Review*, 41(2):53–56, 2011.

[140] PostgreSQL. PostgreSQL database. https://www.postgresql.org.

[141] PostgreSQL. PostgreSQL documentation: Chapter 11. Indexes. https://www.postgresql.org/docs/current/indexes.html.

[142] Prometheus. Prometheus. https://prometheus.io/.

[143] S. Qian, Y. Wang, and K. Xu. Utilizing destination options header to resolve ipv6 alias resolution. In *Proc. IEEE GLOBECOM*, pages 1–6. IEEE, 2010.

[144] S. Qian, M. Xu, Z. Qiao, and K. Xu. Route positional method for ipv6 alias resolution. In *2010 Proceedings of 19th International Conference on Computer Communications and Networks*, pages 1–6. IEEE, 2010.

[145] Redis. Redis. https://redis.io/.

[146] Redis. Redis Persistence feature. https://redis.io/topics/persistence.

[147] Y. Rekhter, T. Li, and S. Hares. RFC 4271: A border gateway protocol 4 (BGP-4). Technical report, 2006.

[148] RIPE NCC. Atlas. https://atlas.ripe.net/.

[149] RIPE NCC. Atlas daily dumps. https://data-store.ripe.net/datasets/atlas-daily-dumps/.

[150] RIPE NCC. RIS. https://www.ripe.net/analyse/Internet-measurements/routing-information-service-ris.

[151] Q. Scheitle, O. Gasser, P. Sattler, and G. Carle. Hloc: Hints-based geolocation leveraging multiple measurement frameworks. In *Proc. IEEE TMA*, pages 1–9. IEEE, 2017.

[152] Q. Scheitle, O. Hohlfeld, J. Gamba, J. Jelten, T. Zimmermann, S. D. Strowes, and N. Vallina-Rodriguez. A Long Way to the Top: Significance, Structure, and Stability of Internet Top Lists. In *Proc. ACM IMC*, page 478–493, 2018.

[153] P. Schober, C. Boer, and L. A. Schwarte. Correlation coefficients: appropriate use and interpretation. *Anesthesia & Analgesia*, 126(5):1763–1768, 2018.

[154] Y. Shavitt and E. Shir. Dimes: Let the internet measure itself. *Proc. ACM SIGCOMM Computer Communication Review*, 35(5):71–74, Oct. 2005.

[155] Y. Shavitt and N. Zilberman. A geolocation databases study. *IEEE Journal on Selected Areas in Communications*, 29(10):2044–2056, 2011.

[156] J. Sherry, E. Katz-Bassett, M. Pimenova, H. V. Madhyastha, T. Anderson, and A. Krishnamurthy. Resolving IP aliases with prespecified timestamps. In *Proc. ACM IMC*, 2010.

[157] R. Sherwood, A. Bender, and N. Spring. Discarte: A disjunctive Internet cartographer. *Proc. ACM SIGCOMM Computer Communication Review*, 38(4):303–314, Aug. 2008.

[158] K. Sklower. A tree-based packet routing table for berkeley unix. In *USENIX Winter*, volume 1991, pages 93–99, 1991.

[159] Sorbonne Université. GitHub organization. https://github.com/dioptra-io.

[160] Sorbonne Université. Iris website. https://iris.dioptra.io/.

[161] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP topologies with rocketfuel. *Proc. ACM SIGCOMM Computer Communication Review*, 32(4):133–145, Aug. 2002.

[162] R. N. Staff. RIPE Atlas: A global internet measurement network. *Internet Protocol Journal*, 18(3):2–26, 2015.

[163] J. P. Sterbenz, E. K. Çetinkaya, M. A. Hameed, A. Jabbar, Q. Shi, and J. P. Rohrer. Evaluation of network resilience, survivability, and disruption tolerance: Analysis, topology generation, simulation, and experimentation. *Springer Telecommunication Systems*, 52(2):705–736, February 2011.

[164] University of Oregon. Route Views. http://www.routeviews.org/.

[165] D. Veitch, B. Augustin, R. Teixeira, and T. Friedman. Failure control in multipath route tracing. In *Proc. IEEE INFOCOM*, 2009.

[166] K. Vermeulen, E. Gurmericliler, Í. Cunha, D. Choffnes, and E. Katz-Bassett. Internet scale reverse traceroute. In *Proc. ACM IMC*, 2022.

[167] K. Vermeulen, B. Ljuma, V. Addanki, M. Gouel, O. Fourmaux, T. Friedman, and R. Rejaie. Alias resolution based on icmp rate limiting. In *International Conference on Passive and Active Network Measurement*, pages 231–248. Springer, 2020.

[168] K. Vermeulen, J. P. Rohrer, R. Beverly, O. Fourmaux, and T. Friedman. Diamond-miner: Comprehensive discovery of the internet's topology diamonds. In *17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20)*, pages 479–493, 2020.

[169] K. Vermeulen, S. D. Strowes, O. Fourmaux, and T. Friedman. Multilevel MDA-Lite Paris traceroute. In *Proc. ACM IMC*. ACM, 2018.

[170] F. Viger, B. Augustin, X. Cuvellier, C. Magnien, M. Latapy, T. Friedman, and R. Teixeira. Detection, understanding, and prevention of traceroute measurement artifacts. *Computer Networks*, 52(5):998–1018, Apr. 2008.

[171] Y. Wang, D. Burgener, M. Flores, A. Kuzmanovic, and C. Huang. Towards street-level client-independent ip geolocation. In *Proc. USENIX NSDI*, volume 11, page 27, 2011.

[172] Y. Wang, D. Burgener, M. Flores, A. Kuzmanovic, and C. Huang. Towards Street-Level Client-Independent IP Geolocation. In *Proc. USENIX NSDI*, volume 11, pages 27–27, 2011.

[173] Y. Wang and K. Zhang. Quantifying the flattening of internet topology. In *Proceedings of the 11th International Conference on Future Internet Technologies*, pages 113–117, 2016.

[174] Z. Weinberg, S. Cho, N. Christin, V. Sekar, and P. Gill. How to catch when proxies lie: Verifying the physical locations of network proxies with active geolocation. In *Proc. ACM IMC*, pages 203–217, 2018.

[175] W. Willinger, D. Alderson, and J. C. Doyle. Mathematics and the Internet: A source of enormous confusion and great potential. *Notices of the American Mathematical Society*, 56(5):586–599, 2009.

[176] P. Winter, R. Padmanabhan, A. King, and A. Dainotti. Geo-locating BGP prefixes. In *Proc. IEEE TMA*, pages 9–16, 2019.

[177] B. Wong, I. Stoyanov, and E. G. Sirer. Octant: A comprehensive framework for the geolocalization of internet hosts. In *Proc. USENIX NSDI*, volume 7, pages 23–23, 2007.

[178] K. Yoshida, Y. Kikuchi, M. Yamamoto, Y. Fujii, K. Nagami, I. Nakagawa, and H. Esaki. Inferring pop-level isp topology through end-to-end delay measurement. In *International Conference on Passive and Active Network Measurement*, volume 9, pages 35–44. Springer, 2009.

[179] X. Zhang, T. Sen, Z. Zhang, T. April, B. Chandrasekaran, D. Choffnes, B. M. Maggs, H. Shen, R. K. Sitaraman, and X. Yang. Anyopt: predicting and optimizing ip anycast performance. In *Proc. ACM SIGCOMM*, pages 447–462, 2021.

[180] Z. Zhang, O. Mara, and K. Argyraki. Network neutrality inference. *Proc. ACM SIGCOMM Computer Communication Review*, 44(4):63–74, 2014.

[181] Z. Zhang, Y. Zhang, Y. C. Hu, Z. M. Mao, and R. Bush. iSPY: Detecting IP prefix hijacking on my own. In *Proc. ACM SIGCOMM*, SIGCOMM '08, pages 327–338, New York, NY, USA, 2008. ACM.

[182] J. Zhu, K. Vermeulen, I. Cunha, E. Katz-Bassett, and M. Calder. The best of both worlds: high availability cdn routing without compromising control. In *Proc. ACM IMC*, pages 655–663, 2022.

[183] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.

[184] ZMap. The ZMap Project. https://zmap.io/.