Systems for analyzing routing policies and localizing faults in the Internet

Yuchen Jin

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2021

Reading Committee:

Arvind Krishnamurthy, Chair

Ratul Mahajan

Xi Wang

Program Authorized to Offer Degree:
Computer Science and Engineering

University of Washington

Abstract

Systems for analyzing routing policies and localizing faults in the Internet

Yuchen Jin

Chair of the Supervisory Committee:
Professor Arvind Krishnamurthy
Computer Science and Engineering

Our reliance on the Internet continues to grow; however, Internet communication has seen little progress over the years because it typically spans multiple Autonomous Systems (ASes) that are operated by individual Internet Service Providers (ISPs) and organizations. This inherent autonomy of the Internet limits the visibility into other networks and the velocity of change. As a result, public Internet communication has become the weak link for Internet-based services.

In this thesis, I design, build, and evaluate practical algorithms and systems that ISPs and cloud providers can use to analyze Internet routing policies and localize faults in the Internet. Knowledge of the business relationships between ASes is essential to understanding the behavior of the Internet routing system. I develop `ProbLink`, a probabilistic algorithm to infer business relationships between ASes in the Internet. By integrating noisy but useful features, it overcomes the challenges in inferring hard links such as routing violating the valley-free assumption, limited visibility, and non-conventional peering practices. I build three real-world applications on top of `ProbLink` and show that `ProbLink` has a significant impact when applied to practical applications compared to the state-of-the-art inference algorithms based on empirical rules. For Internet-based services such as video calls and online games, providing low latency is important. I design and build a system, `BlameIt`, that automatically localizes the faulty AS when there is latency

degradation between clients and clouds. `BlameIt` employs a hybrid two-phased blame assignment, combining the best parts of passive analysis (low measurement overhead) and active probing (fine-grained fault localization). `BlameIt` has been in production deployment for 3 years at Microsoft Azure and produces results with high accuracy at low overheads.

TABLE OF CONTENTS

Page

List of Figures iii

List of Tables v

Chapter 1: Introduction 1
1.1 Thesis Organization 5

Chapter 2: Background and Related Work 6
2.1 Internet Topology and Routing 6
2.2 Cloud Network 11
2.3 Measurement infrastructure and tools 13
2.4 Related Work 14

Chapter 3: Stable and Practical AS Relationship Inference with ProbLink 18
3.1 Input Datasets 20
3.2 Establishing a Benchmark for Hard Links 23
3.3 Challenges With AS Relationship Inference 30
3.4 Probabilistic AS Relationship Inference 34
3.5 Evaluation 44
3.6 Summary 46

Chapter 4: Applying ProbLink to Real-world Applications 49
4.1 Route Leak Detection 49
4.2 Inference of Complex Relationships 51
4.3 Predicting the Impact of Selective Advertisements 52

Chapter 5: Zooming in on Wide-area Latencies to a Global Cloud Provider 56
5.1 Characterizing Worldwide Latency 60

5.2 Overview of BlameIt 66
5.3 Fault Localization with Passive Measurements 68
5.4 Fine-grained Localization with Active Probes 75
5.5 Evaluation 80
5.6 Summary 89

Chapter 6: Conclusions and future work 91
6.1 Conclusions 91
6.2 Future Work 92

Bibliography 96

LIST OF FIGURES

Figure Number Page

2.1 The Internet consists of a set of Autonomous Systems (ASes). Inside each AS, end-hosts and routers are interconnected by links. 7

2.2 Hierarchical Internet topology with various tiers of ASes. 9

2.3 Example of a cloud provider's network and how it connects with other ISPs and clients. 12

3.1 Feature importance scores provided by gradient boosting tree. 28

3.2 Analysis of transit degree difference, valley-free violations, and error rates of `AS-Rank`. 30

3.3 CDF of `AS-Rank`'s error rates on paths seen from 200 different half-VP sets. . 33

3.4 Conditional probability distribution for the triplet feature describing $P(previous, next \mid middle)$. Probability values in the ranges of > 0.1, $[0.01, 0.1]$, and < 0.01, are categorized as high, medium, and low respectively in the figure. 37

3.5 The visibility of each link type derived from `CoreToLeaf` inference results on 04/01/2017 BGP paths. 39

3.6 CDF of error rates of `ProbLink` and `AS-Rank` on the snapshots of BGP paths in the past 6 years. 45

3.7 CDF of error rates of `ProbLink` and `AS-Rank` on 30 consecutive 1-day snapshots from April 1, 2016 to April 10, 2016. 46

3.8 CDFs of error rates of `ProbLink` vs. `CoreToLeaf` on *hard* links in a period of 30 days in April 2016. 47

4.1 Evaluation of route leak detection across 10 days. 50

4.2 Evaluation of complex relationships inference and the prediction of path changes due to selective prefix advertisements. 51

5.1 Map of Azure locations worldwide. 57

5.2 Fraction (%) of quartets whose average RTT was deemed to be bad. Badness thresholds are set based on Azure's region-specific RTT targets. 62

5.3 Bad quartets (%) by the hour for 1 week in USA (top) and for two ISPs (bottom). Night hours are marked. Weekend is between the 48th and 96th hours. 63
5.4 (a) Persistence of bad RTT incidents in a day (in consecutive 5-min buckets). (b) CDF of problem impact when ⟨Cloud location, BGP path⟩ are ranked by two orders. 64
5.5 Illustrative example of how ⟨Cloud location, BGP path⟩ tuples can be ranked in two different orderings. 65
5.6 CDF of the number of IP /24's sharing the same "middle segment" (different definitions) within 5 minutes. 73
5.7 `BlameIt`'s key components in production. 81
5.8 Blame fractions in a one-month period. 82
5.9 Blame fractions for one day in six cloud locations. (In each location, the blame fractions sum to 100%.) 83
5.10 Duration of cloud, middle, client segment issues in the units of consecutive 5-min time buckets. 84
5.11 Large scale traceroute based validation. Corroboration ratios of `BlameIt` to clients in 1,000 BGP paths. 87
5.12 CDF of client-time product of middle segment issues ranked by the Oracle. 88
5.13 Accuracy of active `BlameIt` under different probing frequencies with/without probes triggered by BGP churn. 89

LIST OF TABLES

Table Number Page

3.1 Size of the validation dataset vs. all links observed from all VPs. 22
3.2 Precision and recall of `CoreToLeaf` and `AS-Rank`. 25
3.3 Features fed into GBDT 27
3.4 Error rates of `CoreToLeaf` and `AS-Rank` on *hard* links on 04/01/2016. The fraction of each category of *hard* links that is in overall links vs. in the validation dataset shows that *hard* links are underrepresented in the validation dataset. 29
3.5 Average error rates on *hard* links. `ProbLink` achieves 5.9×, 2.6×, 6.1×, and 1.8× better error rate for the links observed by between 50 and 100 VPs, non-VP & non-Tier1 links, unlabeled stub-clique, and conflict than `AS-Rank` respectively. 47
3.6 Error rates of `ProbLink` with all features turning on and without each feature in turn against 04/01/2017 BGP paths. 48
5.1 Comparison with prior network diagnosis solutions on the desired properties for scalable fault localization. 58
5.2 Details of the dataset analyzed (one month in 2018). 61

ACKNOWLEDGMENTS

The past five and a half years at the beautiful University of Washington will surely be an unforgettable experience in my life. I met amazing advisors, collaborators, and friends during this journey, so I want to thank them here.

My deepest thanks go to my advisor Arvind Krishnamurthy. After graduating from my undergraduate university and entering UW, I had no idea how to do research. Arvind took me step-by-step through the process of generating research ideas and executing ideas. Arvind had endless enthusiasm and always inspired me, and he was always available when I needed help from him. He also gave me a lot of freedom to do what I was interested in. Even when my interest shifted from computer networks to neural networks, he fully supported me. One of Arvind's magic is that he can always help me find great collaborators, which made this thesis possible. Arvind is an optimist. His big smile always made me forget the worries of slow project progress and paper rejections. I believe I will miss his laughter a lot after graduation. I would like to thank other committee members of my thesis, Ratul Mahajan and Xi Wang, for their invaluable feedback, advice, and support. I am also grateful to have Tom Anderson, Shyam Gollakota, and Dan Ports advise me in the early stage of my Ph.D.

I had three wonderful internship experiences and I am grateful for all of my mentors at Microsoft Research and ByteDance: Ganesh Ananthanarayanan, Venkat Padmanabhan, Yibo Zhu, Chang Lan, Chuanxiong Guo, and Sharad Agarwal. I still often think of Ganesh taking me to the first floor of MSR to buy coffee and discuss projects, career plans, and personal life with me along the way. I learned how to work with product teams from Venkat, and he is such a humorous and kind person. Yibo is smart and can always come

up with a lot of wonderful ideas. Chang gave me a good guide, which made me more interested in ML systems. Chuanxiong has a strong vision and a principled approach to research. Sharad taught me how to do system design and how to present my work. These people were my mentors, but they are more like my friends. They cared about me and brought a lot of fun to my life.

I would also like to thank all of my collaborators because I was so honored to work with them. I want to thank Colin Scott, Amogh Dhamdhere, Vasileios Giotsas, and Scott Shenker for their help on ProbLink; Sundararajan Renganathan, Junchen Jiang, Manuel Schroder, and Matt Calder for their help on BlameIt; Tianyi Zhou, Liangyu Zhao, and Marco Canini for their help on AutoLRS. And I was fortunate to work with Haichen Shen, Lequn Chen, Bingyu Kong, Matthai Philipose, Ravi Sundaram on Nexus led by Haichen. These collaborators not only helped me in the research projects, but also motivated me and helped me build my confidence. I still remember Colin explained "Imposter syndrome" to me when I told him that I felt that I was not as suitable for doing research as my peers. When I wrote my first paper, Colin gave me a detailed description of the steps he took to write a paper, and recommended books and blogs to me.

I wish I could list all the good memories with every friend and labmate during the years at UW, but this acknowledgments will be extremely long. Lequn Chen, Tianqi Chen, Raymond Cheng, Shumo Chu, Tapan Chugh, Tianyi Cui, Pedro Fonseca, Seungyeop Han, Antoine Kaufmann, Nei Lebeck, Dianqi Li, Jialin Li (elder), Jialin Li (younger), Shu Liang, Ming Liu, Vincent Liu, Ashlie Martinez, Ellis Michael, Samantha Miller, Luke Nelson, Pratyush Patel, Simon Peter, Henry Schuh, Will Scott, Naveen Sharma, Haichen Shen, Helgi Sigurbjarnarson, Adriana Szekeres, Chenglong Wang, Xiao Sophia Wang, Xin Yang, Xieyang Xu, Irene Zhang, Kaiyuan Zhang, Qiao Zhang, Xiaoyi Zhang, Chenxingyu Zhao, Kevin Zhao, and Danyang Zhuo, they all made my Ph.D. life complete. Special thanks go to Shumo Chu, who came to the airport to pick me up when I landed in Seattle to start

my Ph.D. and later became my good friend and gym buddy. Danyang always shared his insights on research and interesting life stories with me. Qiao and Haichen gave me very useful feedback on my research, and helped me with my job search. I hope to go hiking and go to the water park again with Jialin (elder), Naveen, and Adriana.

Finally, I want to say thanks to my family. I rarely thank my parents verbally but I am always grateful in my heart. My girlfriend has unconditional trust and love for me. Thank you for everything.

DEDICATION

to my parents Yang and Qun, and my girlfriend Linh

Chapter 1

INTRODUCTION

Our lives are increasingly dependent on the Internet. As of January 2021, there were 4.66 billion active internet users worldwide, which took 53 percent of the global population [109]. The COVID-19 pandemic drastically changed the way people work and live — billions of people had to depend on the Internet for daily work, education, and social activities. The Internet traffic increased about 20% after the worldwide lockdowns started within only one week [36].

Even though the demand for highly reliable and performant Internet continues to grow, public Internet communication has seen little progress over the years because it typically spans multiple administrative domains called Autonomous Systems (ASes), and each domain is operated by an individual ISP or an organization. The network operators in an AS have little visibility into and even less control over the routing policies and decisions of the other networks in the Internet.

The limited visibility of the routing policies causes the Internet vulnerable to several security issues such as Border Gateway Protocol (BGP) prefix hijacks and BGP route leaks. Let us take route leaks as an example. Route leaks constitute a type of prevalent routing incidents that can cause significant disruptions to Internet routing. The Internet Engineering Task Force (IETF) defined a BGP route leak as "the propagation of routing announcement(s) beyond their intended scope. That is, an announcement from an Autonomous System (AS) of a learned BGP route to another AS is in violation of the intended policies of the receiver, the sender, and/or one of the ASes along the preceding AS path" [101]. On November 5, 2012, a Google peer Moratel ($AS23947$) improperly advertised Google routes to its provider, causing Moratel's providers to select the leaked routes as the pre-

ferred ones destined to Google. As Moratel could not handle such large traffic volumes, Google's services went offline in parts of Asia for half an hour [49]. Recently, attackers exploited route leaks to steal cryptocurrencies [23].

One way to combat route leaks is to build a route leak detection system by checking the "valley-free" routing property, i.e., a customer AS does not transit traffic from its provider to another, and peers do not transit traffic from one peer to another. BGP selects route based on policy, and routing policies are usually determined by the business relationships between ASes. However, business relationships are often kept confidential, which makes it hard to detect and prevent route leaks and other Internet routing issues.

In addition to the demand for Internet security, there is also a demand for high-performance Internet. To deliver high-performance Internet-based services, cloud providers such as Microsoft, Google, and Amazon constructed their own global backbone networks. These largest providers' backbone networks consist of dozens of edge sites and tens of cloud locations which are interconnected via a private Wide Area Network (WAN). The cloud locations host many interactive (latency-sensitive) services that cater to consumer and enterprise clients covering a broad set of products around productivity, search, communications, and storage. The edge locations are the first stop in the cloud network that customers hit, and with edge locations spread out worldwide, customers all over can reach the cloud resources with low latency. Moreover, to shorten the hops to the users in terms of the AS-level path, the cloud providers extensively peer with lower-tier ISPs, bringing their private networks closer to users. For example, the Microsoft Azure network has 61 cloud locations and over 160 edge locations, and peers with more than 4,000 ASes.

Users either reach the cloud providers via direct peering or via multiple ASes. While the spread of cloud locations as well as edge locations means that there are fewer ASes between the client AS and the cloud, when the latency experienced by clients increases and breaches the round-trip time (RTT) target, cloud providers want to know which AS in the path is causing the latency degradation.

Localizing the faulty AS is crucial. There could be many reasons for RTT degradation: overloaded cloud servers to congested cloud networks, maintenance issues in the client's ISP, or path updates inside a transit AS. A system to localize the faulty AS, as quickly as possible, will help cloud provider's operators trigger remedial actions such as switching egress peers, thereby minimizing the duration of user impact. But the key is the accurate and timely localization of the faulty AS in the client-cloud path.

In this thesis, I will show that **it is possible to build practical systems to help ISPs and cloud providers better analyze and understand Internet routing policies and localize the faults in the Internet to improve its performance**. This thesis makes three contributions by addressing the following problems:

How to do accurate and stable AS relationship inference? The business relationships between Autonomous Systems determines Internet routing policies. Knowledge of the AS relationships is essential to understanding the behavior of the Internet routing system. Despite significant progress in the development of relationship inference algorithms, the resulting inferences are impractical for many critical real-world applications, cannot offer adequate predictability in the configuration of routing policies, and suffer from inference oscillations. To achieve more practical and stable relationship inference, we first illuminate the root causes of the contradiction between these shortcomings and the near-perfect validation results for `AS-Rank`, the state-of-the-art relationship inference algorithm. Using a "naive" inference approach as a benchmark, we find that available validation datasets over-represent AS links with easier inference requirements. We identify which types of links are harder to infer and develop appropriate validation subsets to enable more representative evaluation. We develop a probabilistic algorithm, `ProbLink` [60], to overcome the challenges in inferring hard links, such as non-valley-free routing, limited visibility, and non-conventional peering practices. `ProbLink` reveals key AS-interconnection features derived from stochastically informative signals. Compared to `AS-Rank`, our approach reduces the error rate for all links by 1.6× and, importantly, by

up to 6.1× for various types of *hard* links.

How to build practical systems on top of `ProbLink` to improve Internet performance? Despite significant progress in the development of AS relationship inference algorithms over the past two decades, the resulting inferences perform poorly in many critical real-world applications [81, 8, 84, 79, 106]. To demonstrate the practical significance of `ProbLink`, we developed three real-world applications on top of it. We developed a route leak detection system to troubleshoot the Internet safety issues, a selective advertisement prediction system to help operators predict the impact of traffic engineering policies on the active BGP paths, and a complex relationship inference system to understand the hybrid relationships in the Internet. Compared to the current state-of-the-art AS relationship inference algorithm, `ProbLink` increases the precision and recall of route leak detection by 4.1× and 3.4× respectively, reveals 27% more complex relationships, and increases the precision of predicting the impact of selective advertisements by 34%.

How to automatically localize the faulty AS(es) in the Internet when there is performance degradation between clients and the cloud? The network communications between the cloud and the client have become the weak link for global cloud services that aim to provide low latency services to their clients. We first characterize WAN latency from the viewpoint of a large cloud provider Azure, whose network edges serve hundreds of billions of TCP connections a day across hundreds of locations worldwide. In particular, we focus on instances of latency degradation and design a tool, `BlameIt` [59], that enables cloud operators to localize the cause (i.e., faulty AS) of such degradation. `BlameIt` uses passive diagnosis, using measurements of existing connections between clients and the cloud locations, to localize the cause to one of cloud, middle, or client segments. Then it invokes selective active probing (within a probing budget) to localize the cause more precisely. We validate `BlameIt` by comparing its automatic fault localization results with that arrived at by network engineers manually, and observe that `BlameIt` correctly localized the problem in all the 88 incidents. Further, `BlameIt` issues 72× fewer

active probes than a solution relying on active probing alone, and is deployed in production at Azure.

1.1 Thesis Organization

The rest of this thesis is organized as follows. In Chapter 2, I provide background on Internet routing and introduce related work and terms I will use in the rest of the thesis. In Chapter 3, I present a probabilistic algorithm, `ProbLink`, to do accurate AS relationship inference. In Chapter 4, I build practical systems on top of `ProbLink`, and demonstrates its practical improvements over existing algorithms. In Chapter 5, I describe a system called `BlameIt` that automatically localizes the faulty AS(es) in the Internet when there is performance degradation between clients and the cloud. Finally, I conclude my thesis and discuss future work in Chapter 6.

Chapter 2

BACKGROUND AND RELATED WORK

This thesis focuses on developing tools to help network operators better analyze the Internet routing policies and localize the cause of Internet performance degradation. In order to understand the problems and my solutions, it is necessary to understand the basics of Internet routing and existing tools to understand and improve it. Therefore, before presenting details on my solutions, I provide an overview of the Internet topology and its routing protocol — BGP (§2.1.2). The routing policies are configured by the network operators to control which routes are selected and which routes are propagated to the neighbor ASes. The routing policies are largely determined by the business relationships between ASes. I introduce the AS structure and common AS relationships in §2.1.1. As cloud providers' networks become more essential to the Internet's functionality, it is important to understand their network infrastructures and how they flatten the Internet (§2.2). I describe several measurement tools that network operators often use to understand and troubleshoot the Internet (§2.3). Finally, I discuss the prior work that is related to the problems that this thesis tackles (§2.4).

2.1 Internet Topology and Routing

The Internet is a network of networks. These networks are called Autonomous Systems (ASes), each representing an administrative domain. An AS is a collection of Internet Protocol (IP) routing prefixes (e.g., 12.0.0.0/8) under the control of a single administrative entity that presents a common routing policy to the other ASes in the Internet. Typical ASes are Internet service providers (ISPs) such as Comcast and China Telecom, and organizations like University of Washington. Until 2007, each AS was assigned a globally

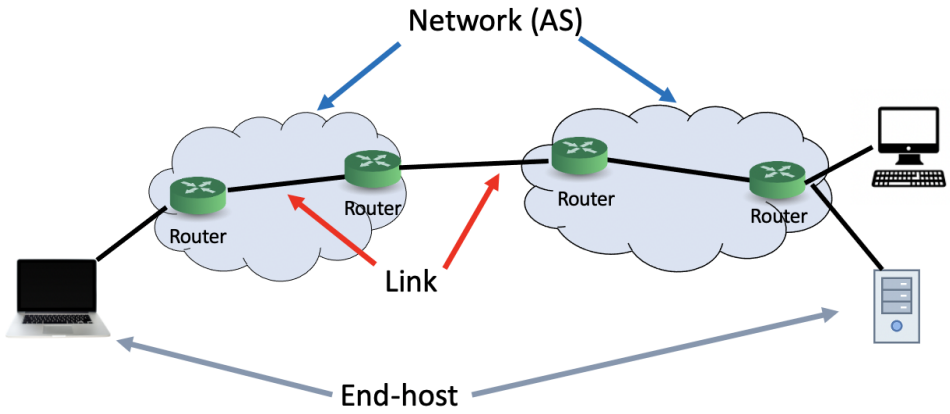


Figure 2.1: The Internet consists of a set of Autonomous Systems (ASes). Inside each AS, end-hosts and routers are interconnected by links.

unique 16-bit identification integer called *Autonomous System Number (ASN)* by the Internet Assigned Numbers Authority (IANA). Due to exhaustion, RFC 4893 expands the ASN to 32 bits [93], and this allows for over 4 billion unique ASNs. The number of ASes has exceeded 100,000 as of March 2021 [108]. Each AS contains three components (see Figure 2.1): end-hosts such as laptops, desktops, servers, and mobile devices, networking devices such as routers and switches which forward traffic between networks and end-hosts, and links that interconnect routers, end-hosts, and networks.

Routing is the process of determining which path to direct traffic from a source host to a destination. The routing protocols running inside the same AS are called *Interior Gateway Protocols (IGPs)*, and include protocols such as *Open Shortest Paths First (OSPF)* [91], *Routing Information Protocol (RIP)* [89], and *Intermediate System-Intermediate System (IS-IS)* [90]. On the contrary, *Exterior Gateway Protocols (EGP)* are used for inter-domain routing, i.e., routing between ASes. The current EGP used in the Internet is BGP. Unlike the IGPs whose goal typically is optimizing a path metric, BGP has different design goals (2.1.2). Before we dive into BGP, let us first look at inter-AS business relationships and the

Internet AS structure.

2.1.1 AS relationships and structure

AS relationships fall into two broad categories: *customer-provider* (c2p) and settlement-free *peering* (p2p). In a c2p relationship, the customer AS pays the provider AS for reachability to/from the rest of the Internet. In a p2p relationship, two networks agree to exchange traffic destined to prefixes they or their customers own without an associated fee. Peering can take two forms. Private peering is when ASes agree to exchange their traffic via a private facility. Public peering happens when ASes establish p2p relationships over shared switching fabric provided by *Internet Exchange Points (IXPs)* (See Figure 2.2). To facilitate dense peering connectivity, IXPs provide BGP route servers over which ASes establish many-to-many (multilateral) interconnections.

In practice, AS relationships can span a spectrum of types between c2p and p2p. These *hybrid* or *complex relationships* can occur when two ASes have multiple contractual agreements, one for each geographical region where an interconnection exists [48]. *Sibling* relationships exist between distinct ASes that are owned by the same organization and can exchange traffic without any cost or routing restrictions.

Based on the economic relationships of ASes, the Internet has a hierarchical structure (see Figure 2.2). A handful of ASes called Tier-1 ASes sit at the top of the Internet AS topology. These Tier-1 ASes are "transit-free", meaning that they have specific routes to all reachable BGP prefixes. Tier-1 ASes peer with all the other Tier-1 ASes, and do not have transit providers. Tier-1 ASes can reach all destination prefixes using either customer or peering links. Example Tier-1 ASes are AT&T, Verizon, and Sprint. Next down the hierarchy are the Tier-2 ASes, which engage in the practice of peering with other networks, but also purchases transit from Tier-1 ASes. Typical Tier-2 ASes are regional or country-level networks. Tier-3 ASes sit at the bottom of the AS hierarchy. These ASes are the ones that are closest to the end-hosts and connect them to the Internet by charging them money.

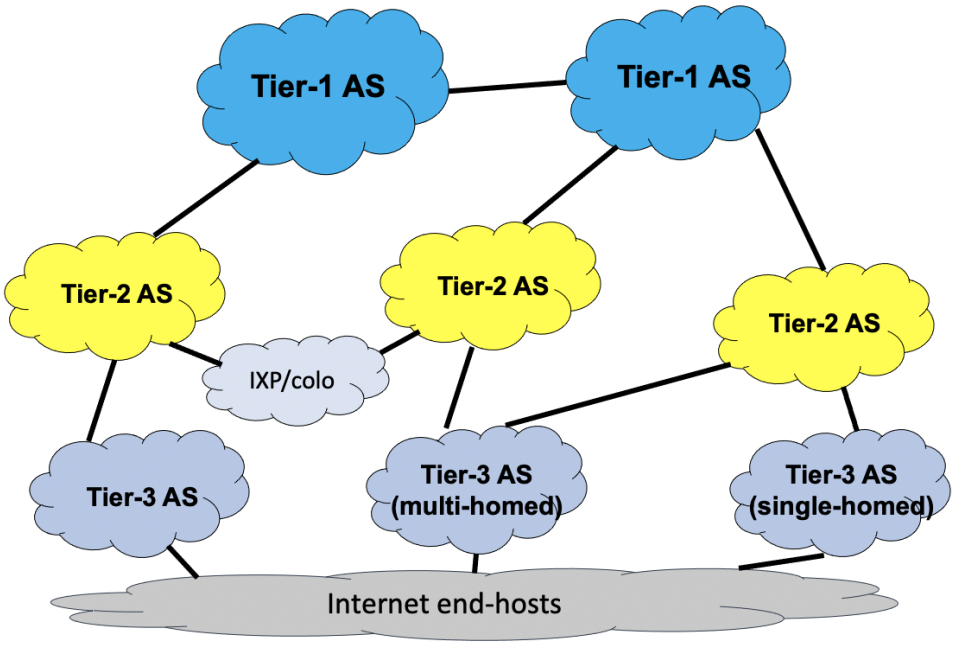


Figure 2.2: Hierarchical Internet topology with various tiers of ASes.

These ASes have to pay transit fee to Tier-2 ASes to participate in the Internet.

A *stub* AS does not carry traffic on behalf of the other ASes, i.e., it can only be a source or a destination for traffic. A stub AS is always the last AS hop in any AS path in which it appears. A *single-homed* AS only connects to one provider, while a *multi-homed* AS accepts traffic from more than one upstream provider. Being multi-homed provides extra reliability, which helps an AS survive a single provider failure, so most ASes in the Internet are multi-homed [17].

2.1.2 BGP

The Border Gateway Protocol (BGP) is the Internet's inter-domain routing protocol used by ASes to exchange *reachability information* [92].

The design of BGP was motivated by three important requirements:

1. **Scalability:** the Internet routing infrastructure should remain scalable as the num-

ber of ASes and IP prefixes increases.

2. **Privacy:** ASes want to conceal their internal network topologies and their business relationships with neighbors.
3. **Policy:** the protocol should allow each AS to implement and enforce various forms of routing policies to control where to send and receive traffic.

To fulfill these requirements, BGP is designed as a *path-vector routing protocol*. The key idea of path-vector routing is that a route annoucement contains the entire list of ASes that the route annoucement has traversed through. When crossing an AS boundary, the border router prepends its own ASN and propagates the announcement. A BGP *AS path* is a sequence of ASes denoting the routing path that the first AS in the path prefers to reach a destination IP prefix. The last AS in the path is referred to as the "origin AS" of the prefix. Being a path-vector routing protocol enables faster loop detection. The router in an AS can easily detect a loop by checking if its own ASN is in the AS path. If it is, it can simply discard the advertisement with loops.

BGP is a *policy-based protocol* which allows ASes to enforce various routing policies to control where to send and receive traffic. BGP allows policy expression by three key steps: import filtering, path selection, and export filtering.

Import filtering: When a border router hears advertisements with multiple routes to a destination prefix from its neighbor ASes, it needs to decide which route to install in its forwarding table. Typically, routes received from customers are more preferable to routes received from peers, and routes received from peers are more preferable to routes received from providers. This kind of import rule can be implemented in BGP using a local attribute called *LOCAL_PREF*.

Export filtering: Similar to the import filtering, each AS needs to make decisions on which routes to announce to its neighboring ASes. ASes may want to discard some route announcements, for example, typically an AS does not want to announce routes from one

peer to another peer because it cannot generate revenue by doing so. In BGP, an AS can modify some attributes of the route to influence how the other ASes behave. For example, an AS can prepend its ASN multiple times to the AS path attribute to artificially inflate AS path length seen by others. However, the other ASes might trim the prepended ASNs and ignore the prepending effect.

Path selection: Each AS uses a complex decision process to select the most preferred path toward each destination prefix, and install the path into its routers' routing table. Here is a partial list of the path selection criteria in the order in which a Cisco BGP router uses to select the best routes towards a destination.

1. *Highest Weight:* Weight is the first attribute used by the router and it is set locally on the router. BGP always selects the path with the highest weight.
2. *Highest LOCAL_PREF:* LOCAL_PREF is a value to express the local AS's preference for a route. A higher value means the route is more preferable. It is exchanged between internal BGP (iBGP) routers.
3. *Shortest AS path length:* An AS path is a sequence of ASNs that the route advertisement has traversed. BGP selects a route with shorter AS path length.
4. *Lowest origin type:* BGP sets higher preference to Exterior Gateway Protocol (EGP) than Interior Gateway Protocol (IGP).
5. *Lowest Multi-exit Discriminator (MED):* MED is a BGP attribute that is exchanged between ASes. It can be used to convey to a neighbor AS a preferred entry point into the local AS. MED is propagated to all routers within the neighbor AS but not passed along any other ASes. BGP selects a route with the lowest MED value.

2.2 Cloud Network

Cloud networks usually have several *points of presence (POPs)* or *edge locations* to place routers and servers. The connections between edges form the backbone or core of the

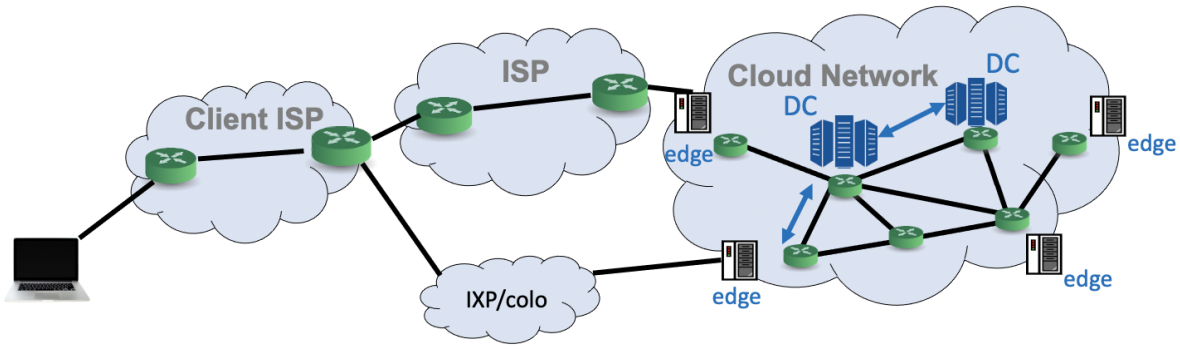


Figure 2.3: Example of a cloud provider's network and how it connects with other ISPs and clients.

cloud network (see Figure 2.3). Distributing edges globally helps reduce application latencies in two ways. Firstly, edges cache content like *content delivery network (CDN)* and server users closeby. Secondly, a TCP connection that has a large RTT to the data centers gets split into two separate connections: the short connection between the user and the edge, and the pre-established long connections with the data centers. When a user needs to communicate with a data center, the user connects to the edge over short roundtrips instead of long roundtrips. This technique is called *split TCP* [37].

The cloud providers connect their edge locations and datacenters via private WANs [53, 10, 9]. Building their own private WANs helps them bypass the public Internet for much of the traffic between their data centers and end-users. To shorten the hops to the users in terms of the AS-level path, the cloud providers extensively peer with lower-tier ISPs, bringing their private networks closer to users. For example, the Microsoft Azure network has 61 cloud locations and over 160 edge sites, owns 130,000 miles of fiber and subsea cables, and peers with more than 4,000 ASes.

Figure 2.3 shows how an end-user connects to the cloud network to access cloud services. In recent years, large content providers such as Google, Akamai, and Microsoft are more willing to directly peer with large numbers of lower-tier ASes to get free and more

efficient traffic exchange and improve user experience [104, 55]. With the rise of IXPs, these cloud providers also publicly peer with small ISPs via multilateral interconnections in IXPs. This trend is referred to as the "flattening" of the Internet [43, 10].

2.3 Measurement infrastructure and tools

BGP route collection infrastructure is operated by Routeviews [5] and RIPE NCC [4]. The infrastructure consists of routers that peer with ASes that volunteer to provide their BGP routing advertisement feeds for research or operational reasons. A route collector is configured to obtain the *best paths* from ASes it peers with, the most preferred available path towards each destination starting from that AS. Route collectors typically peer with several ASes, and thus obtain multiple best paths to each destination prefix. In March 2021, Routeviews operated 29 collectors, and RIPE NCC operated 25 collectors, which in total connect with more than 1,000 vantage points (VP) worldwide. Each RouteViews and RIPE RIS collector dumps a snapshot of their Adj-RIB-out tables every 2 hours and every 8 hours respectively. As defined in RFC 4271 [92], "the Adj-RIBs-Out contains the routes for advertisement to specific peers by means of the local speaker's UPDATE messages".

Traceroute is a commonly used tool when diagnosing network problems. It can track which routers have passed from the source host to the target host, and the time it takes to reach each router between the source and the destination.

Traceroute uses the *time-to-live (TTL)* value in the IP packet header to achieve its functions. When a host sends an IP packet, it will set a TTL value for the packet. Whenever a data packet passes through a router, its TTL value is reduced by one. When the TTL reaches 0, the router will no longer forward the packet, but an ICMP error message "TTL expired in transit" will be reported to the source IP address.

Traceroute first sends a packet with a TTL of 1 to the target host. The first router in the path between the computer sending the packet and the target host reduces the TTL of the packet by 1 when forwarding the packet. It finds that the TTL reaches 0, so an ICMP

error message is sent to the host that originally sent the packet, and thus the Traceroute program obtains the IP address of the first router on the path between the source and the target. The Traceroute program later sends packets with TTL of 2, 3, 4... to the target host one by one, and detects the IP address of each router on the path between the source host and the target host.

In this thesis, we use Traceroute to actively probe client locations from cloud locations for fine-grained fault localization.

2.4 Related Work

2.4.1 AS Relationship Inference Techniques

Beginning with the seminal work by Gao [40] in 2001, most AS-relationship inference algorithms are based on the assumption that valid BGP paths are *valley-free*, i.e., a path consists of zero or more *customer-provider (c2p)* links, followed by zero or one peering link, followed by zero or more *provider-customer (p2c)* links. This assumption captures the economic incentives that (at least partially) determine traffic exchange between ASes: an AS should not intentionally advertise routes learned from a peer or provider to another peer or provider, since this "free transit" increases infrastructure costs but provides no remuneration.

Another observation made by Gao and others [103, 29, 30, 112] is that providers usually have a higher *node degree* (i.e., the number of ASes to which an AS node directly connects to) than customers, while peers usually have similar degrees. *Node degree* is the number of neighbors an AS directly connects to, irrespective of whether the neighbors are providers, peers, or customers Willinger et al. have shown that node degree is significantly biased by the fact that the available topological data reveals only a subset of the complete Internet topology, due to limited placement of vantage points adjacent to peer-to-peer AS links [111].

The state-of-the-art AS relationship inference technique, called the "`AS-Rank`" algo-

rithm [73], makes three generally accepted assumptions: 1) there is a clique of large transit providers at the top of the hierarchy, 2) most customers purchase transit in order to be globally reachable, and 3) there are no cycles of p2c links. The `AS-Rank` algorithm takes 11 intricate steps to label each link as customer-provider (abbreviated as c2p or p2c depending on the directionality of the relationship) or peer-to-peer (p2p) as following:

1. Discard or sanitize paths with artifacts.
2. Sort ASes in decreasing order of computed transit degree, then node degree.
3. Infer a transit-free clique (i.e., Tier-1) ASes at top of AS hierarchy and label the links between every pair of ASes in the clique as p2p links.
4. Discard poisoned paths.
5. Visit ASes in order of the ranking in (2), and label a link as c2p if its previous link in a BGP path is composed of two clique members, or if its previous link in a BGP path is already labeled as c2p.
6. Infer c2p relationships from VPs inferred to be announcing no provider routes.
7. Infer c2p relationships for ASes where the customer has a larger transit degree.
8. Infer customers for ASes with no providers.
9. Infer c2p relationships between stub ASes and clique ASes.
10. Infer c2p relationships where adjacent links have no relationship inferred.
11. Infer all other links left as p2p.

It is worth noting a few properties of the `AS-Rank` algorithm. First, `AS-Rank` uses the *transit degree* attribute as one of the main sources of information in determining relationship labels. *Transit degree* is the number of ASes that appear on either side of an AS in adjacent links of BGP paths, but it does not count neighbors for which the given AS does not transit traffic. Transit connectivity is easily observable by Route Collectors (except for backup or partial-transit links [52]), therefore it provides a more robust metric to describe an AS's prominence than node degree. Second, `AS-Rank` considers ASes and links

in a specific order, using the *transit degree* information in certain cases (step 5) and not in others (step 7).

2.4.2 *Measurement of WAN latency*

WAN latencies have been studied from the viewpoint of the cloud networks. The most relevant studies are those on network path inflation (e.g., [100, 99]), alternate paths (e.g., [66]), deployment of CDN servers (e.g., [19]), IXP performance (e.g., [6]), application-level overlay paths (e.g., [50]). While these studies provide valuable insights on WAN performance in the wild, such offline analysis is not an integrated part in the online operations for fault diagnosis. Partly inspired by the prior measurement studies, in this thesis, we will seek to understand the typical latency degradations from the viewpoint of Azure, and more importantly, it provides a measurement-based solution to automate the process of root-causing the degradations.

Others seek to obtain a comprehensive view of the network conditions by redirecting existing traffic flows (e.g., Edge Fabric [98], Espresso [113]) or by performing active measurements (e.g., Entact [117]). Although network performance is explicitly measured in these schemes, such measurements are purely end-to-end.

2.4.3 *CDN Traffic Engineering*

There has been substantial work on server selection in content delivery networks (CDNs), with a range of approaches, including IP anycast [20], DNS-based redirection (including EDNS extensions to help better identify the client [22]), or anycast-based DNS server selection with co-located proxies [38]. Anycast-based selection, in particular, only has a loose connection to network performance since it depends on BGP routing to direct the client to a server. CDN performance can be significantly improved by choosing the network path and ingress point a client should be directed to as a function of path performance (e.g., [105, 98, 113, 20, 117]).

2.4.4 *Network Anomaly Diagnosis*

Diagnosing anomalies in network performance has been studied extensively (e.g., [62, 68, 75, 20]); see Table 5.1. Network tomography based solutions [21, 58, 11] passively deconstruct end-to-end performance, but run into intractable formulations at scale. In this thesis, we seek to build a diagnosis system that can be operated by cloud providers at a much larger scale than these solutions.

Other works [66, 28, 46] combine passive measurements and active measurements to identify problems, while Odin [20] probes randomly selected clients to maintain visibility to alternate network paths. But these works do not use passive data to minimize the active probes, do not trigger probes during a latency degradation (are essentially "offline"), and waste many probes due to lack of impact prioritization.

Many works make optimal use of a measurement budget for probing, e.g., Trinocular [85], Sibyl [26], iPlane [74]. However, the fault localization system we built in this thesis is distinguished from this prior work in its approach to predict the *impact* of latency degradation (e.g., how many clients would likely be affected) in deciding which problems to investigate. Finally, solutions that rely on rich information (e.g., TCP logs) collected at end-hosts for diagnosis [97, 12] are feasible in datacenters but harder to deploy in the inter-domain wide-area network.

Chapter 3

STABLE AND PRACTICAL AS RELATIONSHIP INFERENCE WITH PROBLINK

For two decades, researchers have studied the problem of inferring the different types of relationships between ASes from publicly available BGP routing data. Relationship inferences are used for a wide range of applications and areas of research, such as detecting network congestion [27], identifying malicious ASes [64, 25], deploying incentive-compatible BGP security mechanisms [44, 24], protecting the integrity of anonymization [84, 61], optimizing video streaming [67, 34, 54], and understanding Internet governance and the ramifications of public policy proposals [69, 72, 54].

In this chapter, we revisit the AS relationship inference problem. We find, as others have, that available relationship inference algorithms perform poorly in many critical applications [81, 8, 84, 79, 106]. We seek to understand why state-of-the-art algorithms are insufficient, despite extensive validation that indicates an error rate as low as 1%. In particular, we consider the sophisticated `AS-Rank` technique [73] which is carefully crafted using eleven deterministic heuristics. As a first step in assessing the performance of `AS-Rank`, we create a baseline benchmark algorithm, `CoreToLeaf`, that consists of three simple steps and only assumes valley-free paths through a core set of transit-free ASes. In spite of its simplicity, `CoreToLeaf` achieves accuracy that is almost as high as that of `AS-Rank`. At the same time, we evaluate `CoreToLeaf` and `AS-Rank` in practical applications—detection of route leaks and the analysis of selective advertisements—which reveals that the performance of both algorithms falls short of the needs of those applications.

`CoreToLeaf`'s high accuracy against the validation datasets implies that the majority

of AS-links in the validation datasets are relatively easy to infer. Yet the sub-optimal performance of both algorithms in practical applications indicates that the small minority of AS-links that are difficult to infer are crucial for those applications. We select subsets of the validation dataset that contain AS links that we consider *hard*, and find that both the `CoreToLeaf` and `AS-Rank` techniques have substantially lower accuracy on these validation subsets (confirming where these current algorithms fall short).

We next examine the challenges in developing a more accurate AS relationship inference algorithm. We observe first that the attributes of a link (and those of the paths that traverse the link) that might be used by an AS-relationship inference algorithm are noisy and often have only a weak correlation with the link's relationship type. Second, many links appear in paths that likely violate the valley-free assumption made by existing algorithms. Third, existing algorithms are sensitive to the locations of the vantage points and the order in which the link relationships are inferred. An AS relationship inference technique must address the above challenges if it is to achieve higher accuracy for hard links.

We develop a *probabilistic AS relationship inference* algorithm, `ProbLink`, to address the above issues. `ProbLink` provides a framework that allows for easy integration of many noisy but useful attributes into the relationship inference algorithm. `ProbLink` enables us to identify a set of link attributes that take into account not only observed paths but also information gleaned from the fact that certain paths are *not* observed. `ProbLink` allows for links to appear in paths that violate the valley-free property but attributes a lower probability to such occurrences. `ProbLink` uses an iterative algorithm that repeatedly infers link types based on statistical distributions of link attributes until the inferences reach a fixed point.

Our evaluation of `ProbLink` show that it achieves an error rate that is better than that of `AS-Rank` overall by 1.7×, and achieves 1.8-6.1× better error rate for various categories of *hard* links.

3.1 Input Datasets

3.1.1 BGP Paths

We collect BGP paths towards IPv4 prefixes from RouteViews [5] and RIPE RIS [4]. In September 2018, both projects operated 22 collectors, which in total connect with more than 1,000 vantage points (VP) worldwide. Each RouteViews and RIPE RIS collector dumps a snapshot of their Adj-RIB-out tables every 2 hours and every 8 hours respectively. For the purpose of evaluating the various algorithms over longitudinal data (as discussed in §3.2 and §3.5), we consider snapshots of BGP paths on the first day of April, August, and December (i.e., every four months) since 2006.

After collecting BGP paths, we parse them to remove duplicated ASes that result from BGP path *prepending*. We also filter out paths with AS loops, i.e., when an ASN appears more than once and is separated by at least one other ASN. We also sanitize the BGP paths by removing paths containing reserved ASes [94]. Loops and reserved ASes showing up in a path are artifacts of route poisoning [16, 63].

3.1.2 Sibling Relationships

We use CAIDA's AS-to-organization mapping dataset [18], which is derived from WHOIS data, to identify sibling links. This dataset provides quarterly information starting from 2009. We infer links between ASes that are operated by the same organization as sibling relationships.

3.1.3 IXP List

ASes often establish p2p relationships over shared switching fabric provided by IXPs. To facilitate dense peering connectivity, IXPs provide BGP Route Servers over which ASes establish many-to-many (multilateral) interconnections. To enable layer-3 connectivity Route Servers typically have their own ASN, but according to best practices it should be

filtered-out from the AS path since the Route Server does not participate in the routing decision process [57]. However, for debugging reasons, some IXP members append the Route Server ASN in the BGP path. We sanitize BGP paths to remove Route Server ASNs since essentially the peering links are between the IXP members, and not between the IXP and ASes. To collect a list of AS Numbers (ASNs) used by IXP Route Servers, we query PeeringDB [3] for networks of type "Route Server" and extract the ASN. We augment this list by consulting the Euro-IX IXP Service Matrix [2] and extracting the Peering LAN ASN and Route Server ASN for each IXP. There were 172 IXP ASes in this list on 12/01/2017.

3.1.4 Validation Dataset

AS operators frequently encode the relationship type with their neighbors directly in their prefix advertisements using BGP Communities, an optional transitive BGP attribute used to attach metadata on BGP paths. While the use of communities attribute is not standardized, many ASes publicly document the meaning of their BGP communities on websites and in IRR databases, enabling us to assemble a dictionary of BGP communities that denote relationship type. We used a dictionary of 1286 community values from 224 different ASes to construct a set of relationships from BGP data starting quarterly from April 2006 to April 2017. Similar to prior work [73], we treat this dataset as "best-effort" validation to evaluate existing inference techniques and our proposed approaches.

Table 3.1 shows the size of this validation dataset over the past 6 years. The coverage of our validation dataset increased from 6.6% in 2012 to about 26% of the observed links in recent years, due to the increasing popularity of BGP communities and the deployment of additional VPs that allow more communities to propagate to BGP collectors. As prior work has pointed out, links involving *Tier-1* ASes and VP ASes are over-represented, because public data on BGP communities mostly comes from large ASes [73], while communities from non-VP ASes may be stripped out during the propagation of BGP routes. However, unlike prior work, we take these biases into consideration during our evalua-

Date (MM/DD/YYYY)	# links in validation set	# links in total	Percentage
04/01/2012	7,833	117,872	6.6%
04/01/2013	11,644	133,459	8.7%
04/01/2014	44,875	159,678	28.1%
04/01/2015	47,036	176,791	26.6%
04/01/2016	52,931	204,309	25.9%
04/01/2017	56,326	213,441	26.4%

Table 3.1: Size of the validation dataset vs. all links observed from all VPs.

tion.

As noted above, the use of BGP communities has become increasingly popular [46], raising the question of whether we can eventually exclusively rely on communities to extract relationships without the need for an inference algorithm. Even with prevalent use of BGP communities however, we would face two important limitations. While communities are by default a transitive attribute, in practice operators often strip out community tags before propagating advertisements to neighbors. Indeed, if all the communities in our dictionary were transitively propagated to our BGP collectors, our validation dataset should have over 58% coverage of the visible AS links. Instead, as shown in Table 3.1 our coverage is less than 30% for the past 6 years. A second limitation with communities is partial availability of publicly available documentation of those attributes. Despite our best efforts to maximize the number of interpretable community values via automated web scraping and text processing tools, we are only able to find authoritative documentation on the meaning of 35% of visible community values.

3.2 Establishing a Benchmark for Hard Links

As explained in the previous section, the evaluation dataset is extensive but biased toward specific types of links. It is important to understand if the links over-represented in the validation dataset are easier to infer correctly, compared to the under-represented links, which may skew the overall evaluation results. To this end we develop `CoreToLeaf`, a very simple algorithm that allows us to understand which links are easy to infer. `CoreToLeaf` uses only the *valley-free* assumption and the list of Tier-1 ASes to infer relationships. We show that the inference accuracy of this algorithm is almost as high as that of the more sophisticated `AS-Rank` algorithm elaborated in §2.4.1, while the accuracy of both algorithms suffer for certain categories of links. Our findings reveal that indeed certain types of under-represented links in the evaluation dataset are harder to infer, possibly inflating the overall accuracy of past work. We address this issue by constructing distinct validation sub-datasets as benchmarks for hard links.

3.2.1 The `CoreToLeaf` *Algorithm*

`CoreToLeaf` starts by inferring a clique of Tier-1 ASes using the same inference method as `AS-Rank`. The clique inference algorithm in `AS-Rank` works as the following:

1. Find the top 10 ASes by transit degree.
2. If there are three consecutive members (X-Y-Z) in the top 10 ASes showing up in paths, and there are more than 5 ASes downstream from "X Y Z" (to make sure that the paths containing three consecutive members are not poisoned), disconnect the edge between X and Z even though X and Z are connected in some paths.
3. Find the largest clique in terms of transit degree sum among the top 10 ASes, denoted as C.
4. Visit the rest ASes top to down by transit degree, add an AS Z to C if Z has links with all members in C.

5. Similar to Step 2: If there are three consecutive members (X-Y-Z) in C showing up in paths, and there are more than 5 ASes downstream from "X Y Z", disconnect the edge between X and Z.
6. Find the largest clique in C in terms of transit degree sum as the final inferred clique.

For each path that traverses a Tier-1, we skip the first link after the Tier-1 and label all succeeding links as p2c. For example, if AS_2 is a clique member in a BGP path "AS_1, AS_2, AS_3, AS_4, AS_5, AS_6", we infer links $<AS_4$ - $AS_5>$ and $<AS_5$ - $AS_6>$ as p2c. We skip inferring the relationship for $<AS_2$ - $AS_3>$ because it could either be a p2c or a p2p, but all subsequent links need to be p2c assuming that the path is *valley-free*. (Note that if AS_1 is a clique member, we would have labeled $<AS_2$ - $AS_3>$ also as a p2c link.) Finally, we label all remaining unclassified links as p2p.

In the step of labeling p2c links, a link could be labeled more than once if it shows up in multiple paths. In some cases, a link could be labeled as a p2c in some path and as a c2p when traversing a different path. We label this link as a "conflict" link when we encounter such an inconsistency.

Note that `CoreToLeaf` does not take into account degree or transit degree information, nor does it use paths that do not go through Tier-1s. This is in contrast to other traditional algorithms; for example, Gao's algorithm [40] considers all paths, identifies the AS with the highest node degree in each AS path and treats it as the top provider, and then labels AS pairs before it as c2p or sibling and AS pairs behind it as p2c or sibling. The rationale behind `CoreToLeaf` is simply that there is greater certainty that it is customer routes that are being transitively exposed to Tier-1s and that there is less likelihood of paths being exported to Tier-1s due to complex peering mechanisms.

3.2.2 Evaluation

We evaluate this extremely simple algorithm against our validation dataset on 04/01/2017, which contains 23,528 p2p links and 32,798 p2c links (corresponding to 26.4% of the vis-

Algorithm	p2c		p2p		Conflict
	Precision (%)	Recall (%)	Precision (%)	Recall (%)	(%)
`CoreToLeaf`	98.9	95.8	95.0	98.8	0.12
`AS-Rank`	97.8	97.5	98.8	98.9	0

Table 3.2: Precision and recall of `CoreToLeaf` and `AS-Rank`.

ible topology). Table 3.2 compares the *precision* (true positives / (true positives + false positives)) and *recall* (true positives / (true positives + false negatives)) of `CoreToLeaf` and `AS-Rank`. Surprisingly, `CoreToLeaf` achieves high precision and recall for both p2c and p2p links (comparable to `AS-Rank`), with higher *precision* on p2c relationships (98.9% compared to 97.8%), and a small fraction of links labeled as 'conflict'.

The 1.1% mistakenly inferred p2c links and the links which `CoreToLeaf` labels as 'conflict' are due to *valley-free* violation, which we quantify later in §3.3.2. Since the step of labeling p2c links uses just paths through Tier-1s, it fails to capture 4.2% (95.8% *recall*) of the actual p2c links. Consequently, these links are inferred as p2p in the third step and results in a 5.0% error rate for links labeled as p2p.

The accuracy of `CoreToLeaf` and `AS-Rank` seem quite high, but they perform sub-optimally when applied to real-world applications. Route leaks constitute a type of prevalent routing incident that can cause significant disruptions to Internet routing [101, 49]. In Chapter 4, we describe how we can use inferred relationships to detect route leaks and evaluate the effectiveness of `AS-Rank` inferences. Only 19% of the route leaks detected using `AS-Rank` were real route leaks, and almost 80% of the real leaks were missed. We observe relatively poor performance for two more applications we tested, which we will discuss in detail in Chapter 4. The high application-level error rates illustrate that a better AS relationship algorithm is needed for real-world applications.

3.2.3 *Identifying Hard Links*

The surprisingly high accuracy obtained by `CoreToLeaf` has many implications. First, it indicates that simple techniques might suffice for inferring the types of many of the links in the validation dataset. Second, it underscores the need for more comprehensive validation datasets that would be more representative of AS links beyond those associated with Tier-1 and VP ASes. Third, in the absence of more comprehensive validation datasets, one way to make progress on improving and evaluating AS relationship inference algorithms is to identify specific types of links for which the current algorithms do not work well. We therefore now attempt to extract collections of *hard* links from the overall validation dataset based on the inference performance of `CoreToLeaf` and `AS-Rank`.

We feed a large set of features of every link in the validation dataset along with information on whether a link was labeled as "inferred correctly" and "inferred incorrectly" by `CoreToLeaf` and `AS-Rank` into a *gradient boosted decision tree* [39], and calculate the feature importance for accurate predictions for the two algorithms.

Gradient boosting [39] is a widely used machine learning technique for solving classification problems. In gradient boosting, *GBDT* (Gradient boosting decision trees) produces a prediction model in the form of an ensemble of multiple decision trees. It is straightforward to retrieve importance scores for features when constructing *GBDT*. An importance score (F score) describes the number of times a feature is used to split the data across all trees. The more a feature is used to make key decisions with decision trees, the higher its importance score.

To decide what features can distinguish *hard* links from *easy* links in the Internet, we first split the validation dataset into two halves. The set of links which `CoreToLeaf` or `AS-Rank` infers incorrectly are labeled as "hard", while those which are inferred correctly are labeled with "easy". Then, we feed the features listed in Table 3.3 of links along with their labels into the *GBDT* and calculate the importance score corresponding to each feature.

Feature label	Meaning
f1	Number of VPs which observe a link
f2	Max distance to Tier-1
f3	Min distance to Tier-1
f4	Max node degree
f5	Min node degree
f6	Node degree difference
f7	Max transit degree
f8	Min transit degree
f9	Transit degree difference

Table 3.3: Features fed into GBDT

Figure 3.1 plots the importance score of each feature divided by the sum of all features' scores. We can tell the features f1, f4, f7, f8, f9 are the most important ones, so we translate them into the various categories of features to characterize "hard" links in §3.2.3.

We extract the following five categories of "hard" links suggested by the feature importance analysis and the `CoreToLeaf` algorithm.

1) Links with max node degrees smaller than 100. The feature importance analysis shows that `CoreToLeaf` and `AS-Rank` do not have high accuracy for links whose endpoint ASes both have small node degrees.

2) Links observed by more than 50 but less than 100 VPs. The feature importance analysis also reveals that links observed by at least 50 VPs but not more than 100 VPs are hard to infer correctly. The reason is that p2p links are often observed by few VPs and transit links are often observed by many VPs, so it is hard to distinguish the link types for the range in the middle.

3) Non-VP and non-Tier1 links. In general, a link that is directly connected to a VP or a Tier-1 is likely to appear in many BGP paths, and the AS inference algorithm is likely

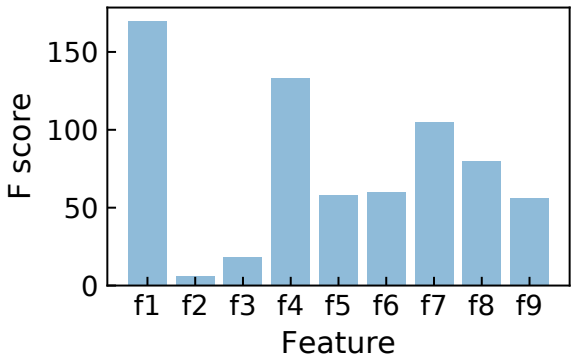


Figure 3.1: Feature importance scores provided by gradient boosting tree.

to have access to more information regarding the link. Moreover, most of our validation dataset are links that are connected to a VP or Tier-1 AS, so we want to specifically analyze the performance of inference algorithms on the "under-represented" links in our validation dataset.

4) Unlabeled stub-clique links in `CoreToLeaf`. A stub AS is one which does not carry traffic on behalf of the other ASes. A stub-clique link is a link whose one endpoint is a stub AS and the other endpoint is in the Tier-1 clique. In other words, the clique member is the only AS to which the stub AS connects. These links typically have very high transit degree difference, which is an important feature as shown by the feature importance analysis.

In `CoreToLeaf`, a stub-clique link <X, Y> (where X is a stub AS and Y is a clique AS) is inferred as a c2p *iff* there is a path containing an AS triplet "Z, Y, X" where Z is also a clique AS. We call the set of stub-clique links that are not inferred as c2p in the second step of `CoreToLeaf` (i.e., they are inferred as p2p in the later step) as "unlabeled stub-clique links".

In step 9 of `AS-Rank`, stub-clique links are classified as c2p by default based on the assumption that stub networks are extremely unlikely to meet the peering requirements of clique members. We believe this assumption should be revisited with the trend of "Internet flattening", as peering relationships between high-tier ASes and low-tier ASes

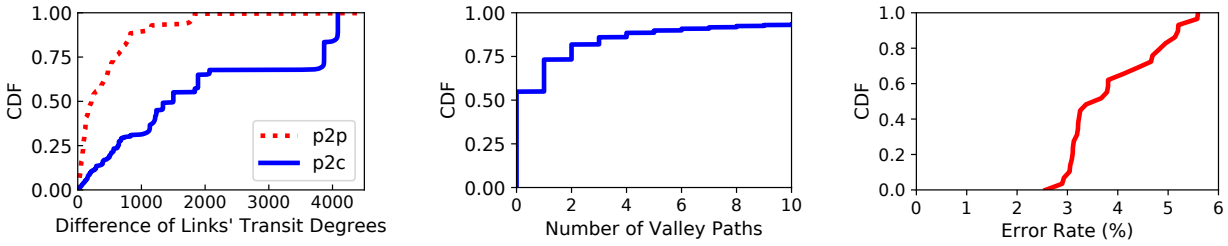
Category	CoreToLeaf **(%)**	AS-Rank **(%)**	Fraction all links	Fraction validation
Max node degree <100	13.7	8.6	16.1%	1.7%
Observed by 50-100 VPs	4.7	9.3	9.9%	8.1%
Non-VP & Non-Tier1	5.3	9.0	24.2%	11.6%
Unlabeled Stub-clique	95.5	33.4	0.3%	0.1%
Conflict	100.0	8.1	0.24%	0.16%

Table 3.4: Error rates of CoreToLeaf and AS-Rank on *hard* links on 04/01/2016. The fraction of each category of *hard* links that is in overall links vs. in the validation dataset shows that *hard* links are underrepresented in the validation dataset.

are becoming more prevalent [43].

5) Conflicts in CoreToLeaf. Recall that CoreToLeaf labels some links as "conflicts". These links appear to behave as p2c on some paths and c2p on others, and the main reason for this is violations of *valley-free* routing. We believe that this set of links is difficult to analyze because the two endpoints are likely to have unconventional routing policies.

Table 3.4 shows the error rates of inferences made by CoreToLeaf and AS-Rank on each category of hard links on 04/01/2016. We observe both algorithms yield more errors than their inferences on normal links, especially on unlabeled stub-clique links. Furthermore, the fraction of every category of hard links in the validation dataset is less than that in the overall links, especially for the "Max node degree < 100" category. This indicates that the validation dataset is skewed to *easy* links. In addition to the entire validation dataset, we will use these more specific datasets for evaluating the AS inference algorithms in the subsequent sections.



(a) CDF of absolute transit degree difference.

(b) CDF of the number of paths that violate the valley-free property traversing each link.

(c) CDF of `AS-Rank`'s error rates on 30 consecutive 1-day BGP snapshots from April 1, 2016 to April 30, 2016.

Figure 3.2: Analysis of transit degree difference, valley-free violations, and error rates of `AS-Rank`.

3.3 Challenges With AS Relationship Inference

In this section, we identify three main challenges with AS relationship inference, and describe how they hamper existing inference techniques. This analysis helps inform the design of a probabilistic algorithm for AS relationship inference.

3.3.1 Degree Inversion

An AS inference algorithm can use any observed attribute associated with a link, its two endpoint ASes, AS links, and end-to-end paths that traverse the link in order to determine the link type. However, most attributes have stochastic information value, as we will illustrate below for AS degree.

Many existing techniques for inferring AS relationships make three assumptions: highest-degree ASes sit at the top of the routing hierarchy; peering ASes have similar degrees; and providers have larger degree than customers [40, 73, 31].

Over the past four years, the top two nodes with the largest transit and node degrees

(as observed through BGP feeds from available VPs) have consistently been *AS*6939 (Hurricane Electric) and *AS*174 (Cogent Communications). However, both of these ASes are not Tier-1 ASes [110], so the assumption that the ASes with the highest degrees sit on top of the routing hierarchy is not universally valid. This fact influences the accuracy of some inference approaches since a key step in these approaches is identifying a clique of Tier-1 ASes at the top of the hierarchy [40, 73].

Figure 3.2a plots a CDF of the absolute transit degree differences of different link types. Transit degrees of ASes are computed from BGP paths observed on 04/01/2017, and link types are derived from the validation dataset on 04/01/2017 as described in §3.1.4. The validation dataset includes 55,016 links, 30,859 of which are p2c links and 24,157 are p2p links. We see that even though p2c links usually have larger degree differences than p2p links, over 14% of the p2p links have absolute transit degree differences larger than 1000, making many p2p links indistinguishable from p2c links in terms of transit/node degree difference.

According to this observation, the existence of substantial differences in node/transit degrees between peering ASes is common. This phenomenon is explained in part by the fact that, during recent years, large content providers such as Google, Akamai, and Microsoft, which usually have high degrees, are more willing to peer with large numbers of lower tier ASes to get free and more efficient traffic exchange [104, 55]. This trend is referred to as the "flattening" of the Internet [43], and it significantly influences the AS relationship inference techniques that differentiate peers from providers or customers based on transit/node degree differences, or rank ASes in decreasing order by degrees and label links based on the order in which ASes are considered (as is the case with `AS-Rank`).

3.3.2 Violation of Valley-Free Property

Next, we study the prevalence of *valley-free* violations, which is the culprit behind mistakenly inferred p2c links and 'conflict' links in `CoreToLeaf`.

3% of the BGP paths violate *valley-freeness* in the `AS-Rank` inference on 04/01/2012. We find this level of *valley-free* violations is persistent over the various snapshots in our study. Figure 3.2b shows a CDF of the number of paths that violate the valley-free property for the links in BGP paths on 04/01/2012. 47% of the links in the AS topology are traversed by paths that violate the valley-free property. This statistic is consistent with prior work that analyzes the prevalence of valley-free violations, and it is a result of the deliberate BGP policies of ASes that use unconventional economic models [47].

The existence of these violations has certain implications for AS relationship inference. First, a robust inference algorithm has to take into account the structure of all paths traversing a given link. Second, it might have to revisit and update the inference made for a given link after inferring the types of neighboring links.

3.3.3 Current Techniques are Sensitive to VP and Snapshot Selection

We observe high variation in accuracy when applying the `AS-Rank` algorithm to consecutive snapshots of BGP paths. Figure 3.2c plots a CDF of `AS-Rank`'s error rates ($1 - accuracy$) on 30 consecutive 1-day BGP snapshots in April, 2016.

Each vantage point provides its own view of the Internet AS-level topology and the flow of traffic from the VP to rest of the Internet. VPs are located in different places, belong to different tiers, and they themselves have different import and export policies.

Even though the number of VPs have been growing over time, VPs are free to join or leave the set of public collectors, so the selection of VPs we have access to is arbitrary, biased, and under flux. A good AS relationship inference algorithm should not be sensitive to the selection of these VPs.

We run the `AS-Rank` algorithm repeatedly, 200 times, against the 04/01/2017 BGP snapshot. For each of these 200 executions, we choose a random VP subset consisting of half of all available VPs (which we denote as V below) and give as input to the `AS-Rank` algorithm only the BGP paths visible to the VPs in that subset. Figure 3.3 plots CDFs of

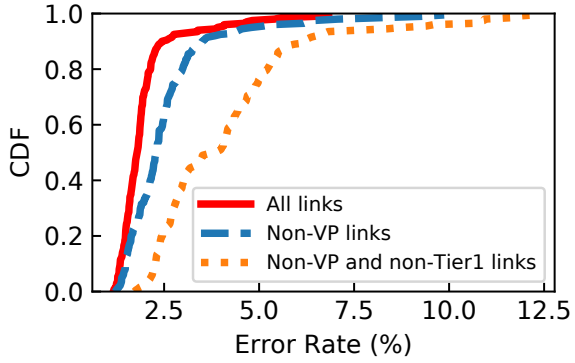


Figure 3.3: CDF of AS-Rank's error rates on paths seen from 200 different half-VP sets.

AS-Rank's error rates using paths from these VP subsets. In the plot, we examine all links in the ground-truth dataset, links in the ground-truth dataset except links that directly connect with V (i.e., non-VP links), and the links in the ground-truth dataset except V's links as well as Tier-1 links (i.e., non-VP and non-Tier1 links). The inference error rates on overall links range from 1.2% to 6.9%, and the error rates on *non-VP and non-Tier1* links range from 1.8% to 12.3%. AS-Rank's accuracy is thus quite sensitive to the VP selections, especially for links which are relatively difficult to infer (i.e., not VP or Tier1 links).

The reason for the AS-Rank algorithm's sensitivity to snapshot and VP selections lies in the first step of its inference algorithm that identifies the Tier-1 clique and the subsequent steps that labels links in a particular order starting with the Tier-1 ASes. AS-Rank first finds the biggest clique from the AS-links involving the largest ten ASes by transit degree, then visits the rest of the ASes top-to-bottom, and adds an AS to the clique if it connects with all the members in the current clique. It then labels p2c links using path segments that radiate from the Tier-1 clique. Errors that creep into the clique determination step have a significant impact on the order in which AS links are analyzed and labeled.

Let's consider two VP sets, V1 and V2, drawn from our 200 executions. AS-Rank's inference accuracy from V1 is low, while its inference accuracy from V2 is high. The largest ten ASes differ for different sets of VPs because the transit degrees of ASes are

determined by paths observed by the VPs. For example, the 9th largest AS (AS2914) observed by V2 is the 12th largest AS observed by V1, so AS2914, which is a real Tier-1 AS, shows up in the clique chosen from the top 10 ASes using V2's paths, but it does not show up in the clique chosen using V1's paths.

For V1, the AS-Rank algorithm determines the maximum clique with the largest transit degree (from the top 10 ASes) to be "AS3356, AS6939, AS8220, AS9002, AS43531". AS43531 is not considered for V2 due to a relatively lower measurement of its transit degree, and it is not a high-tier AS in reality. This affects the subsequent expansion of the clique, wherein ASes are considered in order by degree and added to the clique if they have connections to all members of the clique. So, in V1's execution, all added members are required to have a direct link with AS43531, and AS1764, AS8767, AS12389, AS12552, AS20485, AS25091, AS33891, AS43531, AS57724 are therefore all added to the clique, even though they are all low-tier ASes.

In a nutshell, the clique inference of AS-Rank algorithm is sensitive to the top 10 largest ASes ranked by transit degrees, which are determined by the selection of VPs and the selection of snapshots. Further, the clique membership determines the order in which links are analyzed by AS-Rank, impacts the computation of *customer cones* for each clique member (i.e., the set of ASes that a clique AS can reach using p2c links), and impacts the overall accuracy of the algorithm.

3.4 Probabilistic AS Relationship Inference

In this section, we present a new AS relationship inference algorithm, ProbLink, that is designed to address the challenges discussed above. First, ProbLink is a probabilistic algorithm that enables the use of link attributes with stochastic information value. Second, in determining a link's type, ProbLink simultaneously takes into account all information regarding the links and the paths that traverse it, and provides a framework for integrating conflicting information (e.g., paths that violate the valley-free property). Third, ProbLink does not prescribe a specific order in which ASes and links are considered, but

rather continually updates the link type inferences and iterates till it reaches a fixed point in terms of the underlying stochastic distributions.

Crucially, our algorithm provides a framework for integrating various link attributes that might help infer a link's type. We therefore first design a set of link features or attributes that provide noisy but still informative signals regarding the AS relationships. In particular, we design features that capture routing behavior in terms of both observed and unobserved routes as well as integrate information regarding a link's endpoints. We note that many of the features used in our algorithm are distinct from that of prior techniques, which mostly use only "valley-freeness" or "node/transit degree" features. We then describe how we use these features to build a probabilistic inference model.

3.4.1 Overview

Our algorithm starts with an initial classification of links based on the inference result of `CoreToLeaf`, so each link has deterministic relationship probabilities at the beginning. More concretely, if `CoreToLeaf` labels L as a p2p link, we will convert it to $P(\mathsf{L} = p2p) = 1.0$, $P(\mathsf{L} = p2c) = 0.0$, $P(\mathsf{L} = c2p) = 0.0$ and provide that as the input to our algorithm. Note that `ProbLink` is essentially a meta-inference algorithm that can be bootstrapped by outcomes of any algorithm. Its performance is independent on the bootstrapping algorithm we choose, which we evaluate later in §3.5.1.

For each feature, `ProbLink` computes the conditional probability distribution based on observed data and the initial set of relationship types attributed to links. In each iteration, we update the probabilities of each link's types ($P(\mathsf{L} = p2p)$, $P(\mathsf{L} = p2c)$, $P(\mathsf{L} = c2p)$) by running our probabilistic algorithm described in §3.4.4, and recompute the distributions of features using the updated probability values of each link. We repeat this process until convergence, i.e., the percentage of links that change labels between each iteration drops below a small threshold.

3.4.2 *Clique Inference*

We first attempt to infer the ASes that are at the top of the hierarchy, namely Tier-1 ASes, because it is used to derive features employed by `ProbLink`. Tier-1 ASes should have the largest *customer cones* [73], so estimating the customer cones is the core of doing clique detection.

First, we find top N ASes in terms of transit degree, denoted as D.[1] These ASes are either Tier-1 or Tier-2 ASes because of the large number of neighbours to which they provide transit. Then, we estimate the customer cone size of each AS in the graph by determining the average number of destination ASes (last hops) for which an element of D uses this AS as part of a route. This is an effective way of estimating the customer cone size because, irrespective of whether a node $d \in D$ is a Tier-1 or a Tier-2 AS, if it reaches a destination t through an AS $x \in D$, then t is likely to be in x's customer cone.

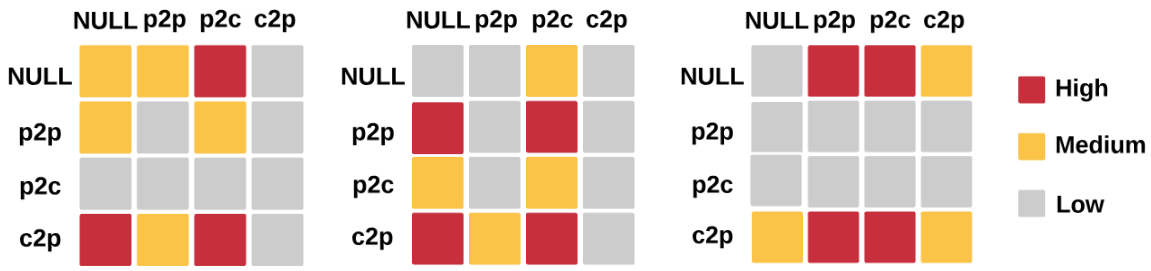
Second, we find the maximal clique C with largest estimated customer cone size sum in D. Then, we test every other AS in order by estimated customer cone size to add members to C. An AS is added to C if it has links with every other AS in C. If there are three consecutive members (X-Y-Z) in C showing up in paths, disconnect the edge between X and Z even though X and Z are connected in some paths, because no AS path should have three consecutive clique ASes. Finally, we find the maximal clique in C as the inferred clique.

3.4.3 *Feature Design*

An AS link can be characterized by the following three attributes: (A) The structure of paths that use the link; (B) The structure of paths that *do not* use the link; (C) Properties of the ASes on each side of the link. We carefully design six features that correspond to these three types of attributes.

Triplet feature (Type A). The triplet feature considers link triplets that appear in paths

[1] Any value of N between 10–40 does not affect the final result.



(a) p2p as middle link **(b)** p2c as middle link **(c)** c2p as middle link

Figure 3.4: Conditional probability distribution for the triplet feature describing $P(previous, next \mid middle)$. Probability values in the ranges of > 0.1, $[0.01, 0.1]$, and < 0.01, are categorized as high, medium, and low respectively in the figure.

and attributes probabilistic values for the relationships of the first and the last links given the relationship of the middle link. Suppose three consecutive links "L1 - L - L2" show up in a BGP path, where L1, L, L2 are three links (AS pairs). "L1 - L - L2" is called a link triplet. We break down each BGP path in a snapshot into link triplets, and, for the first and the last links in each path, we insert a " NULL" link in front of and behind it. For example: a BGP path "8793 6939 1103 198499" is decomposed into 3 link triplets: "NULL - <8793, 6939> - <6939, 1103>", "<8793, 6939> - <6939, 1103> - <1103, 198499>", and "<6939, 1103> - <1103, 198499> - NULL". As a consequence, each link in the BGP paths appears as a middle link in at least one link triplet. We take into account sibling relationships (described in §3.1.2) by skipping sibling links, i.e. treating the two ASes connected by a sibling link as a single AS, when constructing triplets.

The goal of the triplet feature is to model *valley-freeness* in a probabilistic way. For each middle link type, we compute the probability of the link type of its adjacent previous link and next link. If we put the link type of the previous link along the y-axis and the link type of the next link along the x-axis, we get a matrix view as shown in Figure 3.4, which is computed from `CoreToLeaf` initial labels. Each cube in the matrix represents a

probability that is categorized as high, medium, and low depending on it being in the range of > 0.1, $[0.01, 0.1]$, or < 0.01. For example, we can see from Figure 3.4a that when the middle link is of type p2p, the previous and next links are most likely to be <NULL, p2c>, <c2p, NULL> and <c2p, p2c>, but its previous link is unlikely to be p2c no matter what its next link's type is.

Non-path feature (Type B). In addition to observed routes, unobserved routes also provide some information regarding AS relationships. The *non-path feature* describes the probability of how many adjacent p2p or p2c links a link has, but none of them appear before this link on any of the paths. This feature is designed to capture the property that a link is unlikely to be a p2c link if it has many adjacent p2p/p2c links and none of them appear as a previous link on any of the paths containing the link.

Similar to the triplet feature, the non-path feature also models *valley-freeness* in a probabilistic way. The non-path feature is not necessarily applicable to all links. When a link does follow a p2p or p2c link or if a link does not have any adjacent p2p or p2c links, the non-path feature does not play a role in inferring the link's type.

Distance to clique feature (Type C). The *distance to clique* feature can be used to capture the observations that high-tier ASes are closer in distance (AS hops) to clique ASes than low-tier ASes, and that ASes in the same tier are likely to be peers, while high-tier ASes tend to be providers of low-tier ASes.

We first create an undirected graph by adding AS links as edges, and then compute the shortest path from each AS towards each clique member using Dijkstra's algorithm. For each AS, we compute its average distance to each member in the clique set, round it to a multiple of 0.1, and denote this value as dist(AS). We represent each link $<AS_1, AS_2>$ in the graph by a distance to clique tuple "dist(AS1), dist(AS2)".

Vantage point feature (Type C). The number of VPs observing a link also suggests the link type. The vantage point feature captures the likelihood of a certain number of

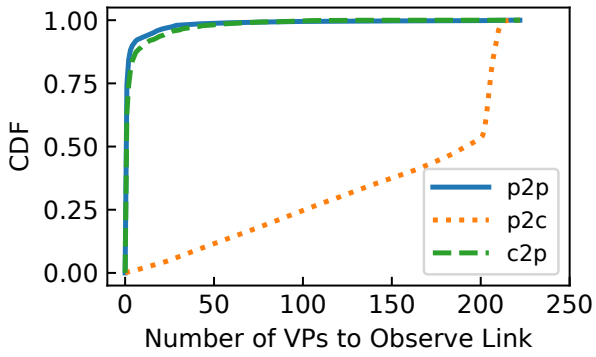


Figure 3.5: The visibility of each link type derived from `CoreToLeaf` inference results on 04/01/2017 BGP paths.

VPs with at least one path traversing a particular link given its link type. This feature naturally folds in the following intuition: p2c links are more likely to be seen by more VPs compared to p2p and c2p links. This feature considers path directions. For example, let's consider a link L (<AS1, AS2>) where AS1 is the provider of AS2. `ProbLink` computes probabilities separately for both directions by counting how many paths traverse L in the direction of <AS1, AS2>, and how many paths traverse L in the direction of <AS2, AS1>

To evaluate the informational value of this feature, we analyze the number of VPs that observe a given link and correlate that with the link's type computed by `CoreToLeaf`. Figure 3.5 shows the CDF of the number of VPs that observe a link for each link type. We observe that 93% of p2p links and 90% of c2p links are observed by ≤ 10 VPs, while 98% of p2c links are seen by more than 10 VPs.

Co-located IXP and co-located private peering facility feature (Type C). The *co-located IXP* and *co-located peering facility facility* information is extracted from PeeringDB [3]. These features are based on the intuition that the more IXPs or facilities two ASes are co-located in, the more likely they are peering with each other. Based on the validation data, 90% of transit links do not have any co-located IXPs or facilities, while more than 70% p2p links

have at least one co-located IXP or facility.

3.4.4 *Inference Algorithm*

We begin by reviewing the Naïve Bayes classifier. Given a link type variable C (which can be p2p, p2c, c2p) and a feature vector f_1 through f_n, *Bayes' theorem* states the following relationship:

$$P(C \mid f_1, ..., f_n) = \frac{P(C, f_1, ..., f_n)}{P(f_1, ..., f_n)} \tag{3.1}$$

By assuming that each feature f_i is conditionally independent of every other feature:

$$P(f_i \mid C, f_1, ..., f_{i-1}, f_{i+1}, ..., f_n) = P(f_i \mid C) \tag{3.2}$$

Using the *chain rule* to rewrite the numerator of Eq. 3.1:

$$P(C, f_1, ..., f_n) = P(C) \prod_{i=1}^{n} P(f_i \mid C) \tag{3.3}$$

So,

$$P(C \mid f_1, ..., f_n) = \frac{P(C) \prod_{i=1}^{n} P(f_i \mid C)}{P(f_1, ..., f_n)} \tag{3.4}$$

Since the denominator $P(f_1, ..., f_n)$ does not depend on the class C, the Naïve Bayes classifier assigns a link being a type $\hat{C}$ by the following function:

$$\hat{C} = \arg\max_{C} P(C) \prod_{i=1}^{n} P(f_i \mid C) \tag{3.5}$$

The inputs to `ProbLink` are BGP paths, link triplets extracted from these BGP paths, and initial relationship labels for each link as inferred by a bootstrapping algorithm. Algorithm 1 shows the pseudocode of `ProbLink`.

First, the algorithm calculates probabilities for each feature, conditional on the link type C (C in {p2p, p2c, c2p}) by accumulating probability values (line 2-3 in Algorithm 1). The parameter α is a smoothing parameter, which prevents a feature with examples

Algorithm 1: `ProbLink`: probabilistic AS relationship inference algorithm based on Naïve Bayes

Input : 1) BGP paths → link triplets

2) Initial AS relationships R

3) Feature vector f = [Triplet, Non-path, Distance to clique, VP, Co-located IXP, Co-located facility]

Output: Inferred probabilities of each link being p2p, p2c, c2p

`/* Loop until convergence */`

while $\mathsf{R} - last(\mathsf{R}) > \varepsilon$ **do**

  `/* Compute conditional distribution of each feature */`

  foreach *feature* f_i *in feature vector* f **do**

    $P(f_i \mid C) = \frac{P(f_i,C)}{P(C)} = \frac{N(f_i,C)+\alpha}{N(C)+\alpha d}$

  foreach *link* L **do**

    $all \leftarrow N(p2p)+N(p2c)+N(c2p)$

    $P(\mathsf{L}=p2p) \leftarrow P(p2p) = \frac{N(p2p)}{N(all)}$

    $P(\mathsf{L}=p2c) \leftarrow P(p2c) = \frac{N(p2c)}{N(all)}$

    $P(\mathsf{L}=c2p) \leftarrow P(c2p) = \frac{N(c2p)}{N(all)}$

    foreach *feature* f_i *in feature vector* f **do**

      $P(\mathsf{L}=p2p) *= P(f_i \mid p2p)$

      $P(\mathsf{L}=p2c) *= P(f_i \mid p2c)$

      $P(\mathsf{L}=c2p) *= P(f_i \mid c2p)$

    sum = $P(\mathsf{L}=p2p)+P(\mathsf{L}=p2c)+P(\mathsf{L}=c2p)$

    $P(\mathsf{L}=p2p) \leftarrow P(\mathsf{L}=p2p)/sum$

    $P(\mathsf{L}=p2c) \leftarrow P(\mathsf{L}=p2c)/sum$

    $P(\mathsf{L}=c2p) \leftarrow P(\mathsf{L}=c2p)/sum$

    `/* Update link's type */`

    $\mathsf{R} = \arg\max_C P(\mathsf{L}=C)$

in only one class from forcing the probability estimate to be 0 or 1. In our implementation, we use *Laplace (Add-1) Smoothing* [76], which sets the smoothing parameter to 1. The algorithm then assigns probability that link L is of each type by the prior probability distribution $P(C)$, which is the proportion of each link type in the data (line 5-8 in Algorithm 1). Then, it goes through each feature and multiplies the probability that link L is of each type by the conditional probability of the feature given each link type (line 9-12). In the end, the final probability of the link L of being each type is calculated by the fraction of each type's probability over the sum of probabilities of all possible link types (lines 14-16 in Algorithm 1). We then update L's type by picking the link type with the largest probability (line 17). We repeat this process of link type inference and updating probability distributions of features until convergence, i.e., the percentage of links that change labels between each iteration drops below a small threshold. The algorithm usually converges within four iterations.

Algorithm Design Choice: We considered alternative approaches for prediction, such as supervised learning based on a training set of labeled link types from community attributes (say using boosted trees). However, since the size of the ground truth from community attributes has not increased in recent years and since our analysis in §3.2.3 shows that the validation dataset is skewed and is only partially representative of the overall Internet, a supervised learning approach would be affected by the biases in the training set.

We instead adopted an unsupervised approach that uses EM. In particular, `ProbLink` falls in the category of techniques that use the *expectation maximization algorithm for parameter estimation in Naïve Bayes classifers* [83]. In particular, Expectation Maximization (EM) is the iterative technique used to separate out classes from a mixture, and Naïve Bayes is the classification technique used in each iteration. EM is suitable when there are hidden classes, and the observed feature distributions are a mixture of the feature distributions of different classes. In each iteration of EM, the algorithm groups together elements that

are classified together and derives the feature parameters of each class. Recall that in our setting, the feature parameters are estimates of the probability of different types of links given a particular topological feature, namely, the probability of the middle link type given previous/next link type, the probability of a certain number of VPs observing a link given each link type, and so on.

Our approach and the underlying techniques have the following implications. First, there is no ground truth in any stage of the algorithm. Second, EM does not work when the classification technique is a black-box or a non-parametric technique (such as a neural network or a boosted decision tree) since it is hard for EM to converge to a stable set of black-box parameters. In fact, in the case of decision trees, convergence would mean that not only the values in the tree nodes do not change, but also the structure of the trees remains stable across iterations. This convergence requirement is hard to satisfy, and hence non-parametric techniques such as decision trees are ill-suited for EM in spite of their high classification accuracy. Naïve Bayes, on the other hand, is a parametric technique that has been shown to work well even when there is correlation between the features used for prediction. Crucially, when Naïve Bayes is used as the classification technique, EM can converge and attribute a stable set of parameters to be used by Naïve Bayes.

It is worth noting that Naïve Bayes makes the assumption that all features are conditionally independent. This independence assumption rarely holds in practical situations, including our own. Nevertheless, despite violating the independence assumption, the classification decisions made by Naïve Bayes are often of high quality [114, 96, 115, 107]. Moreover, in our context, what is needed is a parametric classification technique as opposed to a prediction technique that attributes precise probabilities for the different classes. The Naïve Bayes classifier is appropriate for such settings, but the lack of conditional independence does have the downside that we cannot use the derived probability values as a confidence measure [14].

3.5 Evaluation

We now evaluate our probabilistic inference algorithm, ProbLink, from three aspects and show that:

- ProbLink consistently achieves low error rates across many years, reducing the average error rate of AS-Rank for all links by 1.7×, and attaining 1.8-6.1× better error rates for the different categories of *hard* links.
- ProbLink is not dependent on its bootstrapping algorithm, and it is stable with respect to snapshot and VP selection.
- Each feature used in ProbLink is meaningful and eliminating any of them harms the overall inference accuracy.

3.5.1 Accuracy

To evaluate the accuracy of ProbLink, we assemble daily snapshots of BGP paths on the first five days of April, August, and December (i.e., every four months) over the past 6 years. We apply our algorithm against these snapshots and compare it with the AS-Rank algorithm over this time period.

Figure 3.6 compares the error rates of inferences made by ProbLink and those made by AS-Rank. Our probabilistic inference algorithm consistently yields a low error rate smaller than 2%, reducing the average error rate of AS-Rank for all links from 2.1% to 1.2%.

Figure 3.7 shows a comparison between error rates of ProbLink and AS-Rank on 30 consecutive snapshots of BGP paths during April 2016. The max and average error rates across these days for ProbLink are 1.4% and 1.2%, while the error rate of AS-Rank ranges from 2.6% to 5.6%, with an average error rate of 3.9%. ProbLink is not sensitive to the specific set of paths used in a snapshot, and it achieves uniformly low error rates in spite of clique inference inaccuracies that adversely impact AS-Rank.

Figure 3.8 plots the CDFs of error rates of inferences made by ProbLink and AS-Rank

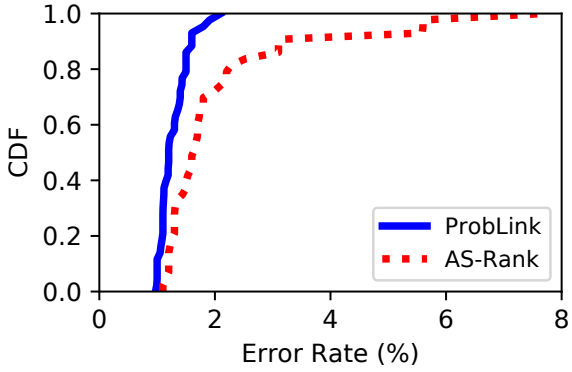


Figure 3.6: CDF of error rates of `ProbLink` and `AS-Rank` on the snapshots of BGP paths in the past 6 years.

on the four categories of *hard* links identified in §3.2.3. Not only does our algorithm yield much smaller error rates, but it also has less variation than `AS-Rank`. Table 3.5 lists the average error rate of our algorithm and `AS-Rank` for links *observed by 50 to 100 VPs*, *non-VP and non-Tier1* links, *stub-clique* links, and *conflict* links. Our probabilistic algorithm reduces the error rate on the four categories by a factor of 5.9, 2.6, 6.1, and 1.8 respectively compared to `AS-Rank`.

`ProbLink` is not dependent on the initial labels provided by the bootstrapping algorithm. Bootstrapping with `CoreToLeaf` and `AS-Rank` only results in 0.15% overall accuracy difference on average. Completely random initial assignment would not work well because our algorithm attempts to separate mixture distributions with some underlying properties. Our algorithm should, however, be robust to various types of initial label assignments as long as there is some weak correlation between the initial labeling and the actual assignments.

3.5.2 Feature Importance Analysis

Eliminating any of the features used by `ProbLink` results in lower accuracy. We next run `ProbLink` with each feature excluded in order to show that each feature adds value. Table

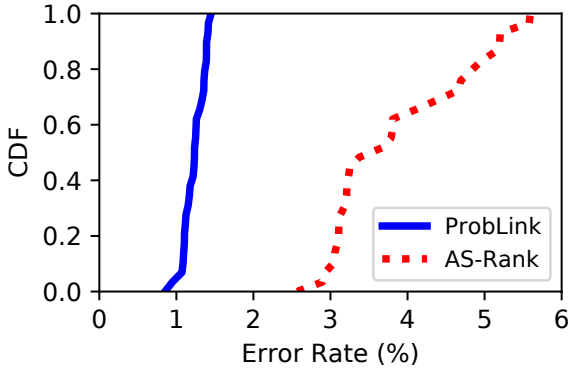


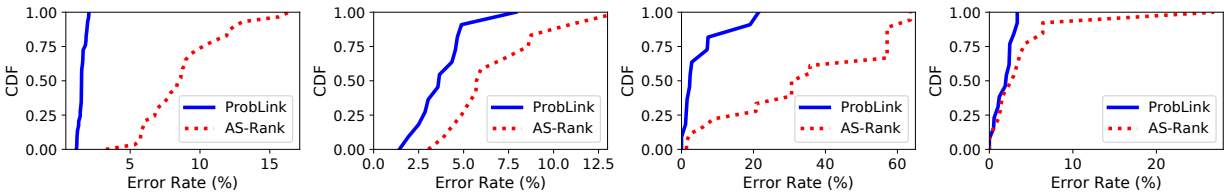
Figure 3.7: CDF of error rates of `ProbLink` and `AS-Rank` on 30 consecutive 1-day snapshots from April 1, 2016 to April 10, 2016.

3.6 lists the error rates of `ProbLink` after turning off each feature one-by-one against the 04/01/2017 snapshot of BGP paths.

Excluding any feature in `ProbLink` results in a higher error rate, which suggests that each feature adds some value. Among all the features, excluding the VP feature harms the overall accuracy the most, indicating that the visibility of BGP paths from many vantage points is a crucial attribute for inferring link types. We believe that integrating more features can further improve `ProbLink`'s accuracy.

3.6 Summary

We revisit the AS relationship problem and inference techniques. We first develop a simple inference algorithm that achieves accuracy comparable to that of the state-of-the-art inference technique, `AS-Rank`, indicating that the types of most links in validation datasets are relatively easy to infer. We then construct different subsets of the validation dataset that might be considered *hard* and use these as benchmarks for evaluating improvements in AS relationship inference. Further, we observe that many of the features that can be used by inference techniques are of a stochastic nature, so we present a probabilistic AS relationship inference algorithm that provides a framework for easy integration of many



(a) *Observed by between 50 and 100 VPs* links. **(b)** *Conflict* links. **(c)** *Unlabeled stub-clique* links. **(d)** *Non-VP and Non-Tier1* links.

Figure 3.8: CDFs of error rates of `ProbLink` vs. `CoreToLeaf` on *hard* links in a period of 30 days in April 2016.

Category	`AS-Rank`**(%)**	`ProbLink`**(%)**
Observed by 50-100 VPs	8.8	1.5
Non-VP & non-Tier1	4.4	1.7
Unlabeled Stub-clique	33.6	5.5
Conflict	6.8	3.8

Table 3.5: Average error rates on *hard* links. `ProbLink` achieves 5.9×, 2.6×, 6.1×, and 1.8× better error rate for the links observed by between 50 and 100 VPs, non-VP & non-Tier1 links, unlabeled stub-clique, and conflict than `AS-Rank` respectively.

noisy but useful attributes into the relationship inference algorithm. We show that this probabilistic algorithm is more accurate and less sensitive to the locations of vantage points and BGP paths compared to the state-of-the-art algorithms.

Feature excluded	Error rate
None	1.5%
Triplet	2.4%
Non-path	1.7%
Distance to Clique	1.7%
VP	4.3%
Co-located IXP and peering facility	1.8%

Table 3.6: Error rates of `ProbLink` with all features turning on and without each feature in turn against 04/01/2017 BGP paths.

Chapter 4

APPLYING PROBLINK TO REAL-WORLD APPLICATIONS

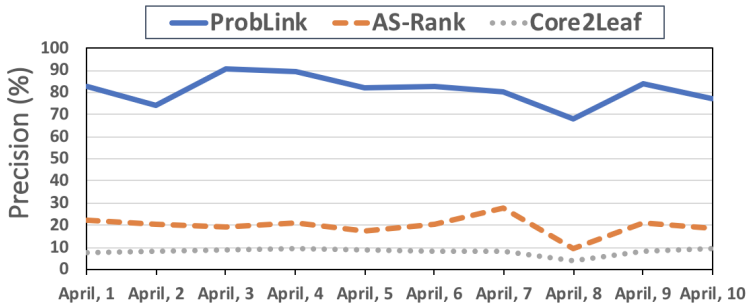
4.1 Route Leak Detection

Route leaks are a class of common routing incidents that can cause large Internet service disruptions [101]. They are caused by violations of the policies among the ASes involved. For instance, on November 5, 2012, a Google peer Moratel (*AS*23947) improperly advertised Google routes to its provider, causing Moratel's providers to select the leaked routes as the preferred ones destined to Google. As Moratel could not handle such large traffic volumes, Google's services went offline in parts of Asia for half an hour [49].

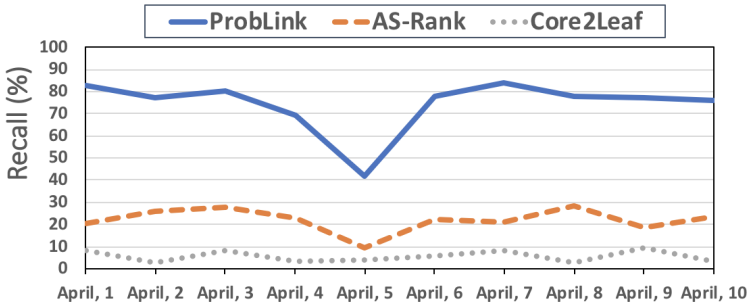
A conventional method for detecting route leaks is through checking valley-free violations in BGP paths. Mauch built a routing leak detection system based on this intuition by searching for valley paths containing three or more major networks with known relationships [56].

In the same spirit, we build a route leak detection system by detecting valley-free violations in paths based on the link relationship inference results of `ProbLink`. It is worth noting that a large fraction (more than 50%) of the valley-free violations do not result from route leaks but intended policies from ASes that are research/educational or IXPs [47]. Such ASes often establish a special type of AS relationship called indirect peering, where an AS functions as an intermediate link between two other ASes who wish to peer but go through intermediate ASes. Therefore, we ignore a path if it contains research/educational or IXP ASes when detecting route leaks.

To evaluate the performance of `ProbLink` and other AS relationship inference techniques on route leak detection, we use only those links for which we have validation data in BGP paths. For example, suppose a path has link relationships "* * p2p * c2p", where



(a) Precision

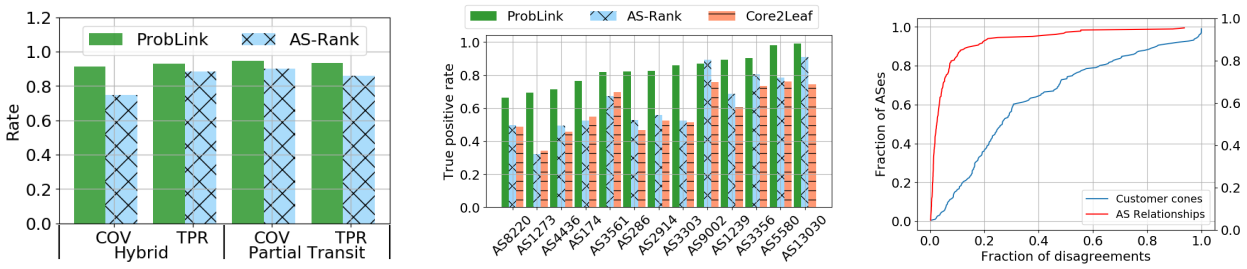


(b) Recall

Figure 4.1: Evaluation of route leak detection across 10 days.

* is unknown in the validation dataset. Even though a relationship inference algorithm should have predictions on the unknown links, we just detect route leaks by its predictions on the two known links in order to compare against the validation dataset. Figure 4.1 compares the precision and recall of `ProbLink`, `AS-Rank`, and `CoreToLeaf`, against the real route leaks implied by the validation dataset across 10 days in April 2016. The average precision for `ProbLink`, `AS-Rank`, and `CoreToLeaf` is 81.1%, 19.8%, and 8.1%, respectively; and the average recall for `ProbLink`, `AS-Rank`, and `CoreToLeaf` is 76.2%, 22.1%, and 5.6%, respectively.

While `ProbLink` significantly improves the state-of-the-art in route leak identification, the number of false-positive inferences is still relatively high (almost 20%) when used as a



(a) Comparison of coverage (COV) and True Positive Rate (TPR) between `ProbLink` and `AS-Rank` in complex relationship inference.

(b) True positive rate per relationship inference algorithm for the prediction of the impact of selective advertisement policies.

(c) Disagreements in AS relationships and ASes in the customer cone of the 200 ASes with the largest customer cones.

Figure 4.2: Evaluation of complex relationships inference and the prediction of path changes due to selective prefix advertisements.

stand-alone inference heuristic. That said, `ProbLink` shows that valley-free violations are a strong signal for the occurrence of route leaks. This indicator facilitates post processing and can be used as part of a composite detection mechanism that combines multiple sources of information, such as the detection of abrupt changes in per-prefix traffic levels [7].

4.2 Inference of Complex Relationships

AS relationships may be more complex than the traditional p2c/p2p model. Such complex agreements may take the form of a *hybrid* relationship with different relationship type for different Points-of-Presence (PoPs) or a *partial-transit* relationship, in which a provider offers transit only toward its peers and customers, but not its providers, or restricts transit to a specific geographic region [30, 35]. The state-of-the-art algorithm for inferring complex relationships (`CR` algorithm) takes as input a set of conventional relationships and iteratively refines them by combining active traceroute measurements with geolocation

data to discover the PoP-level propagation patterns of inter-domain paths [45]. Due to the high measurement cost in terms of traceroute queries required to infer complex relationships, CR utilizes customer cones to optimize the allocation of queries to traceroute probes and maximize the discovery of hybrid relationships within the limited querying budgets used by platforms such as RIPE Atlas [95]. Therefore, the quality of the p2p/p2c relationships can affect the precision, accuracy, and coverage of complex relationship inference.

To test the performance of ProbLink for complex relationship inference, we implemented the CR algorithm and executed two 2-day measurement campaigns over the RIPE Atlas platform, on 2018/09/06 and 2018/09/08 using ProbLink and AS-Rank respectively. For each measurement round, we allocated the maximum permissible number of measurement credits, which resulted in 125,529 traceroute queries from 7,870 Atlas probes. CR+ProbLink inferred 1,308 hybrid relationships and 3,163 partial transit links, while CR+AS-Rank inferred 1,029 hybrid relationships and 3,009 partial transit links. We evaluated these inferences against our validation dataset, which includes 346 hybrid links and 402 partial-transit links. As shown in Figure 4.2a, combining CR with ProbLink not only improves the True Positive Rate (TPR) of the algorithm both for hybrid and partial transit relationships, but, importantly, we significantly expand its coverage (COV) by capturing 91% of the hybrid and 95% of the partial transit relationships, compared to 76% and 90%, respectively, for CR+AS-Rank. Overall, CR+ProbLink discovers 27% more hybrid relationships than CR+AS-Rank.

4.3 Predicting the Impact of Selective Advertisements

The ability to predict the impact of traffic engineering policies on the active BGP paths can be valuable to network operators, as it would limit the need for trial-and-error experimentation, allow the configuration of more predictable and stable routing policies, and minimize the risk of propagating unintended routes [82]. However, past works have shown that the existing AS relationship datasets have poor predictive capabilities, making

them impractical for such purposes. In this section, we evaluate the impact of ProbLink's improved relationship inferences in predicting the outcome of a selective advertisement. Selective advertisement is a popular traffic engineering technique used by AS operators to achieve traffic load balancing, by advertising certain routes only to a subset of their inter-domain neighbors [86].

To predict the impact of selective advertisement "in the wild", we first need to explicitly capture the activation and the scope of such policies. We detect selectively advertised prefixes by utilizing route redistribution BGP Communities, which are increasingly utilized to implement selective prefix advertisement [87, 32]. In particular, many providers define an array of Community values that can be set by their customers, to allow them to control whether the provider should propagate or not a route to a specific peer or group of peers. For instance, if *AS*9002 (RETN) receives a prefix advertisement from a customer annotated with the BGP Community `9002:65535`, then RETN will propagate this route only to its customers, but not its peers or providers [88]. Redistribution Communities can further limit the scope of the prefix advertisement by determining a location for which the redistribution policy will be applied. For instance, when the Community `286:49` is applied on a prefix, *AS*286 (KPN) will not advertise this prefix to its US peers [65]. By parsing WHOIS records and NOC websites, we compile a dictionary of Community values that define one of the following types of selective route redistribution:

- Do not announce route to neighbors of type R.
- Do not announce route to neighbors of type R at L.
- Announce route only to neighbors of type R.
- Announce route only to neighbors of type R at L.

R indicates relationship type (customer, provider, peer) and L indicates a city-level or country-level location identifier. In total, we extracted 644 Community values from 152 ASes.

After compiling our Communities dictionary, we monitor the BGP messages of the

corresponding ASes to capture BGP Updates annotated with one of the redistribution Communities. Let us assume we observe a BGP Update for a destination prefix d annotated with a BGP Community C, which instructs AS_C to propagate p *only* to its neighbors of type R. We calculate which ASes will have to change their paths as follows: We first parse the BGP paths right before C was applied on the prefix d, and we collect all the paths P_{ALL} to d that traversed AS_C. Then, based on the inferred relationships, we find the paths $P_{R'} \subseteq P_{ALL}$ in which AS_C advertises the route toward d to a neighbor with relationship type $R' \neq R$. Since the Community C allows the prefix announcement only to neighbors with relationship type R, we infer that the paths $P_{R'}$ will be withdrawn, and the corresponding ASes in these paths will choose a different path. When C also defines a geographic scope for the prefix advertisement in addition to the relationship type, we use the techniques described in [45] to map the city-level location of AS interconnections, and calculate the affected paths in a similar manner. We validate our inferences by observing the withdrawn paths after C was applied on the path. We consider as false positive any AS $a \in P_{R'}$ that did not withdraw its path 15 minutes after we observed the BGP Update with C. We do not consider false negatives, as an AS may change its path to d for different reasons, and this change may simply coincide with the application of the Community C on the same prefix.

Figure 4.2b shows our validation results after executing the above experiment for the first week of April 2016. During that period, we found 480 prefixes tagged with redistribution communities defined by 13 ASes. Overall, 83% of `ProbLink`'s predictions were correct, compared to 62% for `AS-Rank` and 59% for `CoreToLeaf`. `ProbLink` outperformed `AS-Rank` for every AS except $AS9002$, and in some cases (e.g. $AS1273$), the true positive rate was 2x higher compared to the other algorithms. These results are surprising given that less than 4% of the relationship inferences differ between `ProbLink` and `AS-Rank`. To understand the significant improvement achieved by `ProbLink`, we investigate the impact of the relationship disagreements between the two algorithms on the customer cones obtained using the *Provider/Peer Observed* methodology proposed in [73]. We focus on

the ASes with at least 100 ASes in their downstream path. For each of these ASes we calculate the fraction of their relationships and the fraction of their customer cones that disagree between `ProbLink` and `AS-Rank`. As shown in Figure 4.2c, while less than 10% of the ASes had more than 20% relationship mismatches, over 60% of the ASes had at least 20% difference in their customer cones. This finding highlights the fact that even a few incorrect relationship inferences can lead to significant differences in properties of the resulting downstream paths and substantial deviations in the predictive capabilities of `ProbLink` and `AS-Rank`.

Chapter 5

ZOOMING IN ON WIDE-AREA LATENCIES TO A GLOBAL CLOUD PROVIDER

Global cloud services rely on three types of communication: intra-data center, inter-data center, and the public Internet. Intra-DC and inter-DC communications, which are under a single administrative domain, have seen rapid improvement in their performance and reliability over the years. On the other hand, public Internet communication, which connects clients to the cloud locations, has seen less progress because it typically spans multiple administrative domains, which limits visibility into the network and the velocity of change. As a result, the public Internet communication become the weak link for cloud services. This *cloud to client communication* is the focus of this chapter.

In this chapter, we focus on a global-scale cloud service, Azure, that hosts a range of interactive services. Azure's clients access their content, over TCP, from the hundreds of its network edge locations worldwide (see Figure 5.1). The hundreds of billions of TCP connections from the clients per day provides a measure of latency — the handshake RTT from TCP connection setup.

Measurement Characterization: Using a few trillion RTT measurements from a month, we first characterize poor (or "bad") latency for the cloud-client connections; we define badness based on Azure's region-specific RTT targets. Instances of bad latency are not concentrated in a few locations but rather widely prevalent. The duration of badness (persistence) has a long-tailed distribution: most instances of badness are fleeting (≤ 5 minutes) while a small number of them are long lasting. Finally, despite the widespread prevalence of badness among many IP-/24's, the *affected clients* themselves are concentrated only in a small number of IP prefixes.

Figure 5.1: Map of Azure locations worldwide.

Localization of faulty AS: The administrators of Azure have a strong need for a tool for *fault localization* when there is RTT degradation (i.e., the RTT breaches the target), specifically, a tool that ascribes blame to the AS(es) that caused the spike in RTT. The faulty AS could be Azure's cloud network itself or one or more of the AS'es in the cloud-client path. In our study of actual incidents (§5.5.3), we see various reasons for RTT degradation, including overloaded cloud servers to congested cloud networks, maintenance issues in the client's ISP, and path updates inside a transit AS. A tool to localize the faulty AS, as quickly as possible, will help Azure's operators trigger remedial actions such as switching egress peers, or investigating server-side issues, thereby minimizing the duration of user impact. In the absence of such a tool, the current practice is to investigate a handful of incidents from the previous day, often chosen in an untargeted manner, which could lead to wasted effort on issues that are not fixable by the cloud operators (e.g., client ISP fault) as well as the ignoring of severe issues (e.g., cloud's fault with many affected clients). Hence, the need for an accurate tool to localize the faulty AS(es). [1]

Fault localization in the network is a long-standing problem. Table 5.1 summarizes

[1]We use the terms "RTT degradation", "fault", "issue", "incident" etc. interchangeably.

Desired Property	BlameIt	Tomography [21]	EdgeFabric [98]	PlanetSeer [116]	iPlane [74]	Trinocular [85]	Odin [20]	WhyHigh [66]
Latency degradation	✓	✓	✓	✗	✓	✗	✓	✓
Internet scale	✓	✗	✓	✗	✗	✓	✓	✓
Work with insufficient coverage	✓	✗	✓	✓	✗	✓	✓	✓
Automated root-cause diagnosis	✓	✓	✗	✓	✓	✓	✓	✗
Diagnosis with low latency	✓	✗	✓	✗	✗	✓	✓	✗
Triggered timely probes	✓	✗	✗	✓	✗	✗	✗	✗
Impact-prioritized probes	✓	✗	✗	✗	✗	✗	✗	✗

Table 5.1: Comparison with prior network diagnosis solutions on the desired properties for scalable fault localization.

the main prior solutions, along with the desired properties of a fault localization solution. Our work borrows some elements from prior solutions but distills an overall approach that localizes high latency faults *while* maintaining a frugal budget for measurement probes and prioritizing the high-impact issues. In particular, prior solutions can be bucketed into "passive" techniques that analyze end-to-end RTT data alone [21, 116, 98] or "active" techniques that rely on issuing probes [74, 85]. The former is often intractable because of insufficient coverage of paths in the measurements, while the overhead of probing with the latter is often prohibitive. Even when techniques combine passive analysis with active probing [20, 66], they do not opportunistically use the passive data for localization (whenever possible), they do not trigger probes in a timely fashion (during the issue), nor do they prioritize the probes (for high impact issues). This leads to many unnecessary probes as well as probing after the incidents. Our work addresses the above issues and intelligently mixes passive analysis with prioritized active probing to achieve accurate fault localization at only a fraction of their probing costs.

Two-phased Design: Our solution `BlameIt` proceeds in two phases. The first is a *passive* phase, where `BlameIt` uses the RTT data from the existing TCP connections between clients and the cloud to *coarsely* assign blames to the "client", "cloud", or "middle" segments. We define the middle segment as the "BGP path" (set of middle AS'es between

the client and cloud). We avoid the intractability of classical tomography solutions by leveraging two strong empirical traits: (1) typically, only one of the AS'es in a client-cloud path is at fault, and (2) a smaller "failure" set is more likely than a larger set, i.e., increases in RTT for *all* the clients connecting to a cloud location is likely due to a fault at the cloud end and not due to faults at each of the client AS'es. These traits allow for a hierarchical elimination approach, starting with the cloud as the potential culprit and going down to the middle and client segments.

To resolve the location of the "middle" segment problems to the granularity of an AS (as desired by Azure's administrators), `BlameIt` resorts to an *active* phase. `BlameIt` issues traceroutes from relevant Azure locations to a targeted set of client IPs while the latency issue is ongoing. The results of these traceroutes are compared with a baseline obtained from "background" traceroutes, to localize the specific AS(es) responsible for the increase in latency. `BlameIt` is judicious in its use of traceroutes by *prioritizing* their use on those issues that are expected to last for longer and impact a larger number of clients (both, predicted from historical data). It also heavily optimizes the overheads of background probes by trading it for a small drop in accuracy. Overall, we adopt a systematic measurement-based approach to designing the various aspects in `BlameIt`'s two-phased fault localization.

Production Deployment: `BlameIt` is in production deployment, its passive component since Nov 2017 and its active component is beginning to get widely rolled out. Its outputs are used by network operators for their investigations on a daily basis. We compare the accuracy of `BlameIt`'s result (i.e., the blamed AS) to 88 incidents that had manual reports and find that it correctly localizes the fault in *all* the incidents; §5.5.3 covers a subset of the incidents. In addition, `BlameIt`'s selective active probing results in 72× fewer traceroutes than a solution that relies on active probing alone, and 20× fewer traceroutes than Trinocular that optimizes probing for WAN unreachability [85] .

Contributions: `BlameIt` makes the following contributions.

1. We present a large-scale study of cloud-client latencies using trillions of RTTs from a global cloud provider; §5.1.
2. We design a practical solution using only passive measurements for coarse-grained fault localization; §5.3.
3. We judiciously issue active probes to localize middle segment issues, while prioritizing the high-impact issues; §5.4.

5.1 Characterizing Worldwide Latency

We present a panoramic view of Azure's client latency. We first characterize the general patterns of high latency – prevalence, persistence and long-tailed duration of issues – (§5.1.2 and §5.1.3), and then identify spatial aggregates that account for much of the latency issues (§5.1.4).

5.1.1 Dataset and methodology

Azure's infrastructure consists of hundreds of network edge locations, with tens of services hosted at each location. The services are interactive (latency-sensitive) and cater to consumer and enterprise clients covering a broad set of products around productivity, search, communications and storage. Hundreds of millions of clients use the services on Azure every day.

Each client connection is over TCP to one of the nearest cloud locations. [2] For each connection, Azure records the TCP handshake RTT at the cloud server. Across all the cloud locations, Azure records hundreds of billions of RTTs each day. Table 5.2 lists the details of our dataset. We use Azure's targets as the latency "badness" thresholds and it varies according to the region and the device connectivity type. [3] The targets are in

[2]The cloud servers are organized into multiple anycast rings and clients connect to the ring that corresponds to their location and the service accessed, leaving it to BGP to direct them to the "nearest" server.

[3]Typically, mobile clients use cellular connectivity while non-mobile clients use a broadband network. Going forward, we plan to distinguish Wi-Fi connections as well.

# RTT measurements	Many trillions
# client IP	$\mathcal{O}$(100 million)
# client IP /24's	Many millions
# BGP prefixes	$\mathcal{O}$(100,000)
# client AS'es	$\mathcal{O}$(10,000)
# client metros	$\mathcal{O}$(100)

Table 5.2: Details of the dataset analyzed (one month in 2018).

keeping with the variation in RTTs across regions and are set such that no client prefix's RTT is consistently above the threshold.

Good/Bad Quartet: To understand the structural patterns of high latencies, we analyze our global-scale dataset through the lens of *quartets*. Each quartet is a 4-tuple consisting of ⟨client IP /24 prefix, cloud location, mobile (or) non-mobile device, 5 minute time bucket⟩. This definition helps us bundle together the measurements from clients that are geographically nearby using similar AS-level network paths [71], and connecting to the same cloud location at similar times. Despite the fine granularity of slicing, we typically still have many tens of RTT samples in each quartet.

We classify each quartet as "good" or "bad" depending on whether the average RTT in that quartet is below or above the corresponding badness thresholds. Note that we require each quartet to have at least 10 RTT samples for confidence in our estimates. We also verified that when the RTT samples in a quartet were randomly divided in to two sets, the Kolmogorov-Smirnov test showed that the samples in both these sets were from the same distribution.

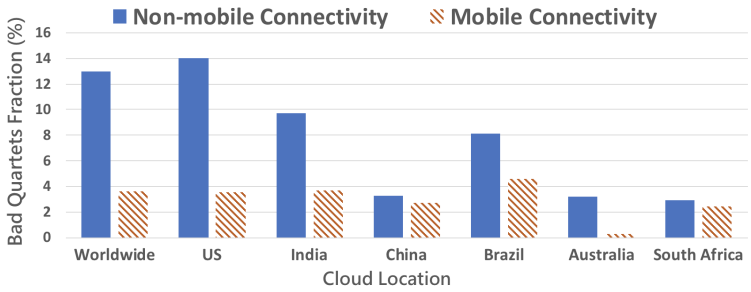


Figure 5.2: Fraction (%) of quartets whose average RTT was deemed to be bad. Badness thresholds are set based on Azure's region-specific RTT targets.

5.1.2 Spatial & temporal distribution of latencies

Our first set of findings show that the the occurrence of high (or "bad") WAN latency is widely spread, both in time and in space.

How prevalent is bad connectivity? Figure 5.2 plots the fraction of quartets whose RTT was bad, split by region. We see that high latency issues are widely distributed, with a substantial fraction of bad quartets for both mobile and non-mobile connections in all regions. While the trend on bad quartets generally decreases with the advancement in network infrastructure of the region (e.g., China and Australia), we see that surprisingly the USA has a higher fraction of bad quartets, most likely due to aggressive thresholds for the USA.

We next analyze the prevalence of latency issues from the perspective of cloud locations. We see that *one-third of the cloud locations have at least 13% bad quartets*.

How does badness vary over time? We also study the effect of time-of-day in the incidents of bad RTTs. Figure 5.3 (top) plots the fraction of bad quartets, bucketed by hours, over the course of a week. While there is an unsurprising diurnal pattern to badness, an interesting aspect is that the *fraction* of bad quartets is consistently *higher* in the nights than during work hours. We speculate that this is because the connections during off-

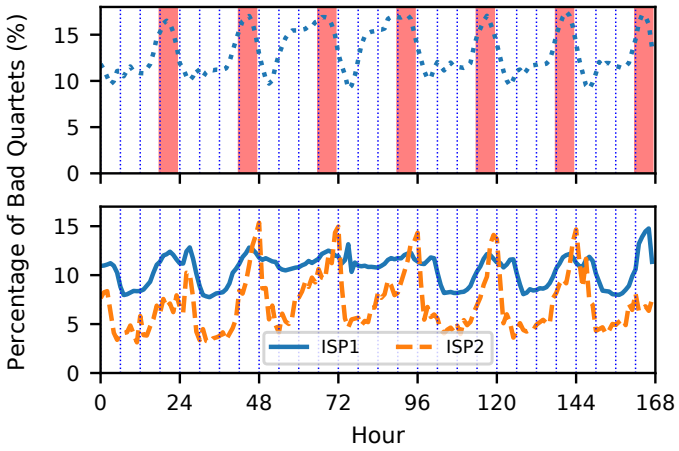


Figure 5.3: Bad quartets (%) by the hour for 1 week in USA (top) and for two ISPs (bottom). Night hours are marked. Weekend is between the 48th and 96th hours.

work hours would tend to be from home ISPs compared to the well-provisioned enterprise networks during the day. Indeed, we see that `BlameIt`'s fault localization often lays the blame for the instances of bad RTTs at night on the client ISP.

Figure 5.3 (bottom) zooms in to show that the magnitude of temporal variance can be different; the fraction of bad quartets in ISP1 varies within a range of 5%, while that of ISP2 is over 10%. Moreover, ISP1 shows a different pattern during the weekend (between the 48th and the 96th hour), in which the diurnal pattern becomes less obvious than the workdays. All of this suggests that the design of `BlameIt` should not assume general temporal predictability to badness.

5.1.3 Long-tailed distribution of badness durations

Incidents of bad latency last for different durations and we analyze the distribution of their persistence. We count the *consecutive* 5-minute time windows for which a ⟨IP /24, cloud location, device-type⟩ tuple suffers poor latency (§5.1.1). As shown in Figure 5.4a, over 60% of the issues last for $\leq$ 5 minutes while only 8% of the issues last for over 2

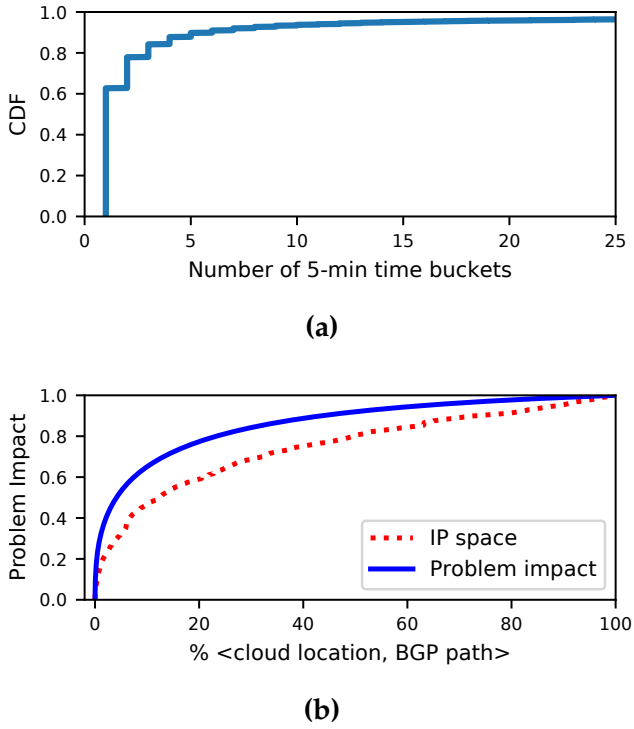
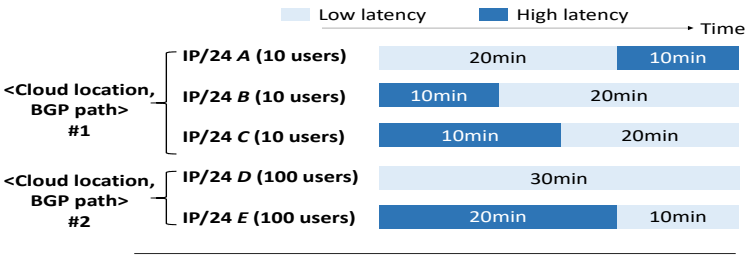


Figure 5.4: (a) Persistence of bad RTT incidents in a day (in consecutive 5-min buckets). (b) CDF of problem impact when ⟨Cloud location, BGP path⟩ are ranked by two orders.

hours (with the distribution being long-tailed). In other words, most of the issues are fleeting rather than long-lived. Our solution aims to identify the serious long-lived issues and alert `BlameIt`'s cloud operators. We also prioritize and issue traceroute probes *during these incidents* that help with their investigations.

5.1.4 Distribution of performance impact

Of relevance to performance diagnosis solutions at global scale (like `BlameIt`) is the spatial concentration of issues. For each ⟨cloud location, BGP path⟩ tuple, we calculate its "impact": defined as the number of affected users (distinct IP addresses) multiplied by the duration of the RTT degradation. BGP path is the set of middle AS'es between the



	Weighted by # of affected prefixes	Weighted by actual problem impact ∑(duration) X (# of clients)
#1	3	350
#2	1	2000

Figure 5.5: Illustrative example of how ⟨Cloud location, BGP path⟩ tuples can be ranked in two different orderings.

client and cloud. Measuring impact at the granularity of ⟨cloud location, BGP path⟩ naturally aligns with our coarse-grained segments (covered in §5.2).

We sort the ⟨cloud location, BGP path⟩ tuples by two metrics (Figure 5.5). The simplest is by the number of problematic IP-/24s contained in the ⟨cloud location, BGP path⟩ tuple, which is similar to how prior works rank the importance of a spatial aggregate [66]. We could also rank the tuples by their actual impact, as we defined above. Figure 5.4b plots the CDF of impact of the tuples when sorted in these two orderings.

When ranked by the IP-/24s, the top 60% of the ⟨cloud location, BGP path⟩ tuples cover nearly 80% of the cumulative problem impact (red line). However, to achieve the same coverage in problem impact, we only need 20% of the ⟨cloud location, BGP path⟩ tuples, if we rank them by their problem impact (blue line), or a 3× lower value.

Thus, our analysis shows that although there is a skewed distribution of the impact of latency problems in the IP space (as also seen by prior works [66]), there is a *much higher skew in the problem's impact when viewed jointly in space and time*, *i.e.*, the number of affected users and the problem's duration. It can be inefficient to simply use the number of IP prefixes to measure the impact of RTT degradations. This observation will be central to `BlameIt`'s budgeted active probing in §5.4.

5.1.5 *Summary of observations*

The takeaways from our measurement study are as follows:

1. *Spatial and temporal spread of high latencies.* Many cloud locations and client-side prefixes have experienced latency regression for a non-negligible amount of time.
2. *Long-tailed distribution of badness durations.* Most incidents of bad latencies are fleeting with only a small number of them being long-lasting and of interest for investigations.
3. *Uneven distribution of the impact of high latencies.* Despite many IP prefixes having high latencies, a substantial fraction of the affected clients are in a small number of IP prefixes.

5.2 *Overview of* `BlameIt`

The primary purpose of `BlameIt` is to help network administrators investigate reports of poor network performance, i.e., RTTs between the clients and the cloud locations being above the badness threshold. `BlameIt` attributes such *path-level* latency degradations to localize the faults at the *AS-granularity*. In this section, we explain the high-level intuitions behind our solution before elaborating on the details in §5.3 and §5.4.

5.2.1 *Two-level blame assignment*

Modeling the paths between the cloud locations and client at the AS-granularity, as per traditional approaches [21, 116], is problematic. First, the coverage of the measurements is skewed and therefore there are often not sufficient measurements to identify the contributions by each AS in the path. Second, modeling the graph at the AS granularity introduces ambiguities, e.g., a large AS like Comcast might have a problem along certain paths but not all. The impact of both these problems is compounded when paths in the Internet change.

Instead of modeling a path as a concatenation of AS'es, BlameIt views each path in two granularities—a *coarse-grained* segmentation of the path into three segments of "client" (the client AS), "cloud" (the cloud AS), and "middle" (all AS'es between the cloud and the client), and then a *fine-grained* AS-level view of only the "middle" segment. Correspondingly, BlameIt localizes the fault in two steps.

1. Attribute the blame for latency degradation on one of the three coarse segments, using *only* "passive" data.
2. If a fault is attributed to a middle segment, BlameIt (optionally) moves to the "active" phase to trigger probes for AS-level fault localization.

Note that modeling *each* path into three segments (albeit at a coarse granularity) allows BlameIt to identify localized issues in an AS that occur only in that path (but not elsewhere in that AS, thus avoiding the ambiguities explained above).

Coarse Segmentation: The three-way segmentation of each path into client, middle, and cloud segments is borne out of Azure's operational requirements. Localizing the fault to the cloud or the client AS is already actionable. A *cloud-induced* latency degradation would directly lead to personnel being deployed for further investigation. When the problem lies with the *client* network (e.g., the client's ISP), the cloud operator typically cannot fix the problem other than informing the client about their ISP choices (e.g., [1]). Thus, our segmentation proved to be pragmatically beneficial and allowed for a phased deployment with our coarse-grained fault localization solution (based on passive measurements) being deployed in production much earlier.

5.2.2 Impact-proportional budgeted probing

BlameIt uses active probes (traceroutes) only for fine-grained localization of "middle" segment blames, as these segments may contain multiple AS'es. Since probing all the paths for full coverage is prohibitive given the sheer number of paths (nearly 200 million per day; §5.5.5), BlameIt sets a *budget* on the active probes and allocates the budget to

probing bad middle segments based on their expected *impact* on clients (§5.1.4), i.e., by estimating *how many clients will be affected* and *for how long*. `BlameIt`'s budgeted probing is unique in two aspects compared to prior work (e.g., [116, 66, 74]).

Spatially, `BlameIt` defines the importance of a segment in proportion to its impact on actual *number of clients*, rather than just the addresses in the IP block (BGP-announced prefix) [66]. In Azure's operations, large IP address blocks often have fewer active clients than smaller IP blocks, thus making it desirable to predict and prioritize issues affecting the most active clients.

Temporally, by *estimating* the timespan of each latency degradation, `BlameIt` can focus on long-lived problems rather than fleeting problems that may end shortly after we probe it, thus wasting our limited budget for probing. Note that we do not need very precise estimate on the timespan of a problem because of the long tail distribution of problem durations (§5.1.3). It would suffice if we only separate the few long-lived problems from the many short-lived problems.

5.2.3 End-to-end workflow

`BlameIt`'s workflow is as follows. RTT data is passively collected at the different cloud locations and sent to a central data analytics cluster. A data analytics job periodically makes coarse-grained blame assignments to all bad RTT instances; §5.3. These blame assignments trigger two sets of actions. The middle segment issues are ordered based on their impact and active probes are issued from the appropriate cloud servers for finer localization; §5.4. All the latency inflations are prioritized based on their impact, and the top few are sent to network administrators for investigation (details in §5.5.1).

5.3 Fault Localization with Passive Measurements

In this section, we describe `BlameIt`'s fault localization method using passive RTT measurements only. We begin by laying down a set of empirical insights (§5.3.1) that leads to

the practical fault localization method (§5.3.2 – §5.3.3).

5.3.1 *Empirical insights*

At first glance, it is tempting to localize faults using standard network tomography techniques, especially given the coarse three-way segmentation (§5.2.1) of the network topology. Unfortunately, the client-middle-cloud segmentation still lacks the topological properties needed for a standard network tomography formulation. To see a concrete example, let us consider two cloud locations c_1 and c_2 serving k client prefixes $p_1 \ldots p_k$ in two distinct geographical regions using two middle segments, m_1 and m_2. Using a drastically simplified setting where there is no noise in the measurements (i.e., the latency values are not obtained from a distribution but are fixed unknown quantities), we can express the delay measurements (d_{ij}) as linear constraints of the following form: $l_{c_i} + l_{m_i} + l_{p_j} = d_{ij}$. However, even though there are $2k$ equations on $k+4$ variables, we cannot infer the individual latency values (e.g., l_{c_i}, l_{m_i} or l_{p_j}). Instead, it is easy to show that we can only solve for the following composite expressions: $l_{c_1} + l_{m_1} - l_{c_2} - l_{m_2}$ and $l_{p_s} - l_{p_t}$ (for $s,t \in 1 \ldots k$); see footnote [4]. As a consequence, we cannot simply solve for the latency parameters (even at the coarse level of cloud, middle and client segments) with the observed latency measurements. Solving linear equations is impractical, even with "boolean" tomography [41, 42, 33] (each client/middle/cloud segment is either "good" or "bad", and a path is good only if all its segments are good).

We overcome the infeasibility of the above problem formulation using two key empirical insights. We derive them from the results of manually-labeled investigations of 88 incidents over many months. In addition, we also looked at extensive traceroutes collected from 22 Azure locations every ten minutes for 14 days (described in §5.5.5) and focused on the RTTs above the badness thresholds.

[4] There are k equations of the form, $(l_{c1} + l_{m1} + l_{p_1} = d_{11}) \ldots (l_{c1} + l_{m1} + l_{p_k} = d_{1k})$ and k equations likewise for $(l_{c2} + l_{m2} + l_{p_1} = d_{21}) \ldots (l_{c2} + l_{m2} + l_{p_k} = d_{2k})$. Subtracting within the first k equations will solve for $l_{p_s} - l_{p_t}$ (for $s,t \in 1 \ldots k$). Subtracting among the first and second set of equations will solve for $l_{c_1} + l_{m_1} - l_{c_2} - l_{m_2}$.

Insight-1: Typically, *only one* of the cloud, middle, or client network segments causes the inflation. For example, inflated latency between a Azure location and the client might be due to the middle segment or the client segment but not due to moderate increases in both. All of the manual reports support this observation. In the traceroutes, 93% of the instances of RTT inflation is due to just one of the segments contributing the dominant inflation ($\geq$ 80%) in the RTT.

Insight-2: A smaller failure "set" is more likely than a larger failure set, e.g., when all the RTTs to a cloud location are bad, it is highly likely due to the cloud segment and not due to *all* the middle segments or clients experiencing an issue simultaneously. In our dataset, over 98% of incidents support this observation. Typically, each cloud location is reached via many middle segments, each of which in turn, connects clients from many client IP-/24s.

These two insights allow us to overcome the insufficiency of measurements by investigating blame assignment sequentially *starting from the cloud* segment and stopping when we have sufficient confidence to blame *one* of the segments. Opportunistically using the passive data for coarse-segmentation is a key differentiation of `BlameIt`: it allows us to overcome the intractability faced by prior passive techniques [21, 116] while also limiting the need for active probes [74, 85, 66].

5.3.2 Coarse-grained Fault Localization

Recall the definition of a "quartet" from §5.1 that aggregates RTT values from an IP-/24 block to a cloud location within a 5 minute window, with each quartet being "good" or "bad" based on the badness thresholds. Algorithm 2 shows the pseudocode to localize the cause of the fault in each bad quartet into one of three categories: the cloud, middle, client segments; it also calls out when the data is "insufficient" or "ambiguous" to classify a bad quartet.

Algorithm 2: `BlameIt` **using passive measurements.**

Input : 1) Quartets, Q:

⟨IP-/24, cloud-node, time, mobile, BGP-path, RTT⟩

2) List of cloud locations, C

3) List of BGP-Paths, B

Output: Assign Blame to each "bad" quartet

Dictionary⟨Segment, float⟩ Bad-Fraction $\rightarrow \phi$

Dictionary⟨Segment, int⟩ Num-Quartets $\rightarrow \phi$

`/* Populate dictionaries for cloud nodes */`

foreach cloud-node c ***in*** C **do**

    Num-Quartets[c] $\rightarrow$ CalcNumQuartets(c, Q)

    Bad-Fraction[c] $\rightarrow$ CalcBadFraction(c, Q, c.expected-RTT)

`/* Populate dictionaries for middle BGP-paths */`

foreach BGP-path b ***in*** B **do**

    Num-Quartets[m] $\rightarrow$ CalcNumQuartets(m, Q)

    Bad-Fraction[b] $\rightarrow$ CalcBadFraction(b, Q, b.expected-RTT)

`/* Blame assignment for each quartet */`

foreach quartet q ***in*** Q *with* q.RTT $\geq$ Threshold_RTT **do**

    `// Customized badness threshold for RTT`

    if Num-Quartets[q.cloud-node] ≤ 5 **then**

        q.Blame $\leftarrow$ "insufficient"

    else if Bad-Fraction[q.cloud-node] $\geq \tau$ **then**

        q.Blame $\leftarrow$ "cloud"

    else if Num-Quartets[q.BGP-path] ≤ 5 **then**

        q.Blame $\leftarrow$ "insufficient"

    else if Bad-Fraction[q.BGP-path] $\geq \tau$ **then**

        q.Blame $\leftarrow$ "middle"

    else if q.IP-/24 has "good" RTT to *another* cloud-node **then**

        q.Blame $\leftarrow$ "ambiguous"

    else

        q.Blame $\leftarrow$ "client"

1) Blaming the cloud: `BlameIt` starts its blame assignment sequentially beginning with the cloud's network as the potential culprit. It takes an aggregate view of the quartets that correspond to a cloud location in the same time period but spanning a wide variety of client locations. If a considerable majority ($\geq \tau$) of the IP /24's connecting to a cloud location see bad RTT values, then `BlameIt` concludes that the cloud's network is at fault (lines 5 and 12-13 in Algorithm 2). In our deployment, we set $\tau = 0.8$ and it works well in practice.

Starting the assignment of blames from the cloud segment of the network (instead of the client) gives us more diverse RTT samples. Each cloud location (in a 5 minute window) has clients connecting to it from hundreds of thousands of /24 IP blocks across thousands of client AS'es. Bad performance across this broad spectrum makes it more likely that the cloud's network is the problem than independent problems afflicting the various clients (insight-2 in §5.3.1). In contrast, the client side does not have the same richness of measurements. Hence, we start blame assignment in Algorithm 2 from the cloud segment.

A subtle aspect to note is that the CalcBadFraction() method in Algorithm 2 does *not* weight the quartets by the number of RTT samples contained in them. Even though different IP-/24s may have a varying number of connections (and hence, RTT samples) to a cloud location, weighting has the undesirable effect of a small handful of "good" IP-/24s with a large number of RTT samples masking the bad performance seen by many IP-/24s connecting to the same cloud location.

While bad quartets are identified using the RTT thresholds (§5.1), `BlameIt` also learns the typical RTT value of clients connecting to each cloud location, c.expected-RTT. It uses deviation in this expected value for blame attribution. §5.3.3 provides the justification and details of this learning.

2) Blaming the middle: If the fault does not lie with the cloud network, the next candidate is the middle section of the network. As defined earlier, this is all the AS'es *in between* the

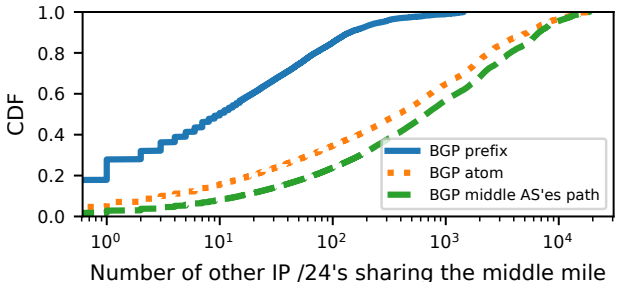


Figure 5.6: CDF of the number of IP /24's sharing the same "middle segment" (different definitions) within 5 minutes.

cloud network and the client prefix.

`BlameIt` groups all the quartets *sharing the same set of AS'es in the BGP routing path* between the client IP-/24 and the cloud location (we refer to this set of middle AS'es as "BGP path") to identify middle segment faults. If all these quartets sharing the same middle AS'es have bad RTT values, we attribute blame to the middle segment (lines 8 and 16-17 in Algorithm 2). As before, we learn the expected RTT of each middle segment (b.expected-RTT) and look for deviations.

We arrived at the decision to group clients by the BGP-path after exploring multiple options. While we considered grouping by the client AS and metro area, based on prior studies [71], we notably found on analyzing Azure's BGP tables that only 47% of clients in the same ⟨AS, Metro⟩ clients see a single consistent path from the Azure location even within a 5-minute window. Thus, we conclude that the granularity of ⟨AS, Metro⟩ is too coarse-grained.

We also considered two other subtly different options. Let the BGP tables at Azure contain two entries to client prefixes C1 and C2, (G − X_1− X_2− C1) and (G − X_1− X_2− C2); note that both C1 and C2 prefixes could be coarser than /24 addresses and in different AS'es. To analyze a bad quartet from C1, we could look at all the RTT values traversing either the path (X_1− X_2− C1) or the path (X_1− X_2− C), where C is the AS of the prefix

C1. The former (called "BGP prefix") is fine-grained while the latter (called "BGP atom" [15]) is coarser. As Figure 5.6 compares, using our solution of "BGP path" provides us with more RTT samples than both these options, and hence gives us more confidence while still being accurate. Thus, we decide to group by the BGP path, i.e., all the AS'es *in between* the cloud network and the client prefix.

3) Blaming the client: Finally, we ascribe blame to the client segment of the network path (lines 20-21). However, if the client IP-/24 has connected to *other* cloud locations in the same time period but experienced good RTT performance, we label the quartet as "ambiguous" (lines 18-19) because there is not a way to conclusively assign blame.

Finally, in any step, if the number of RTT samples is too small, `BlameIt` outputs "insufficient" (lines 10-11, 14-15).

5.3.3 Learning RTT thresholds

Central to the cloud and middle segment blame assignments in Algorithm 2 are two RTT thresholds c.expected-RTT and b.expected-RTT. These thresholds compare against the expected RTTs of clients connecting to *each of the cloud locations* and traversing *each middle segment* of the network, and are also learned separately for mobile and non-mobile clients.

Cloud Segment c.expected-RTT. Learning the RTT values separately for each cloud location helps us identify inflations in RTT at each cloud location compared to *its own* historical values. In general, c.expected-RTT is less than the (region-specific) RTT badness thresholds. We provide a simple example to show how using c.expected-RTT instead of the badness threshold in Algorithm 2 (lines 5 and 12-13) disambiguates cloud faults. Consider the case when the RTT between a client IP-/24 and cloud location X is 55ms, with the RTT threshold being 50ms. Say, this is only due to a fault inside Azure, so assuming everything else is unchanged, the cloud should be blamed.

Let us assume that the RTTs of client IP-/24s connecting to X uniformly varied between [35ms, 45ms] historically, thus c.expected-RTT is learned to be 40ms. After the fault,

due to X's increased contribution, the range changes to uniformly vary between [40ms, 70ms]. If we were to use the RTT threshold itself (of 50ms) in Algorithm 2, only $\frac{1}{3}$ of the quartets connecting to X will be bad (in between [40ms, 70ms]) and the blame will not fall on the cloud network (since $\tau = 0.8$). However, using c.expected-RTT $= 40$ms leads to the correct blame assignment as the RTTs are all above this learned value.

We use the following simple approach to learn c.expected-RTT for each cloud location. We use the median of RTT values from the last 14 days as the c.expected-RTT of connections to that location. (Using the median of historical values and $\tau = 0.8$ essentially means that we check if the distribution has shifted "right" by 30%; we picked these parameters by extensive analysis of our incidents and can vary them adaptively.) While we considered other approaches like comparing the RTT distributions, our simple approach works well in practice.

Middle Segment b.expected-RTT. We also observe considerable difference between the RTTs traversing the BGP paths ($10\times$ difference between the 10^{th} and 90^{th} percentile RTTs). As before, we set the m.expected-RTT thresholds for each middle segment (BGP path), learned as the median from RTT values of the past 14 days, and look for deviations.

5.4 Fine-grained Localization with Active Probes

We build on the techniques in §5.3 to judiciously employ *active* measurement for fine-grained fault localization of middle segment faults. We begin by explaining the rationale behind selective active probing (§5.4.1), present an overview of our approach (§5.4.2), and then explain how we optimize both our on-demand as well as background traceroutes (§5.4.3 and §5.4.4).

5.4.1 Rationale for selective active probing

Algorithm 2 blames the incidence of high RTT to one of three network segments: cloud, middle, or client. While attributing blame to the cloud or client segment automatically

identifies the specific AS responsible for the latency degradation, attributing blame to the middle segment of the network could be due to any of the AS'es in the BGP path between the cloud and client could be causing the degradation.

We considered extending the approach of "hierarchical elimination" of Algorithm 2 to the AS-level graph, with the blame being attributed to the AS common to the paths of the majority of connections that are experiencing high RTT. In practice, however, the data density is insufficient for such AS-level analysis. While we considered coarsening the definition of a quartet (e.g., larger client IP aggregates than /24's or wider timer buckets than 5 minutes), we decided against it as it would hurt the accuracy of the fault localization.

Active measurements – issuing continuous traceroutes from the different Azure locations to client locations – is a natural approach to localize faults. [74, 85, 66] However, for accurate localization of the middle AS, we need continuous traceroutes so that we have data to compare *before and after an incident*. As our calculations in §5.5.5 will show, that works out to nearly 200 million per day, an amount that is prohibitive and infeasible in Azure's production deployments. Besides the overhead, such volumes could also trigger security alarms in the various AS'es since the traceroute packets and their ICMP responses could be mistaken for probing attacks [70].

Due to routing asymmetries [51], the "forward" (cloud-to-client) and "reverse" (client-to-cloud) Internet paths can be different. Our current solution only uses traceroutes issued from the cloud locations (for ease of deployment) but we believe that reverse traceroute techniques [58] can be incorporated into `BlameIt`'s active phase. Azure already has many users with rich clients [20] that can be coordinated to issue traceroutes to measure the client-to-cloud paths.

5.4.2 Approach for fine-grained localization

Our approach for fine-grained fault localization is as follows: judiciously issue "on-demand" traceroutes based the passive analysis of RTTs (§5.3), and then compare these measure-

ments with the *baseline* established through infrequent "background" traceroutes to obtain the picture prior to the fault. We illustrate how these traceroutes are compared using an simple example.

Illustrative Example: Consider the following example, modeled on a real-world investigation in India, on how the background and on-demand traceroutes are compared. The AS-level path between cloud X and client c is X - m1 - m2 - c. The RTT observed by the background traceroutes from X to the *last* hop in each of these four AS'es was 4ms, 6ms, 8ms and 9ms, respectively. However, after the latency degradation, the corresponding values became 4ms, 60ms, 62ms and 64ms. Specifically, the individual contribution of the middle AS m1 went up from $(6-4) = 2$ms to $(60-4) = 56$ms, thus pointing to m1 as the reason behind the performance degradation. [5]

There are two key challenges to realizing the above approach at scale. 1) First, we have to be judicious about when to issue "on-demand" traceroutes since fine-grained localization might be of little interest for issues that are ephemeral and/or only affect a small number of clients. 2) Second, background measurements have to be optimized so that they are not wasteful or cause a significant increase in network traffic. We next present BlameIt's solution to both issues – prioritized on-demand measurements (§5.4.3) and optimized background measurements (§5.4.4).

5.4.3 *Prioritized On-Demand Measurements*

BlameIt prioritizes on-demand traceroutes for latency degradations that, (1) have lasted for a long duration, (2) are expected to persist for longer durations, and (3) would potentially impact many clients. Prioritization helps BlameIt focus on issues of most impact to clients while avoiding the high probing overheads of prior solutions [74, 85, 66]. As §5.1.3 and §5.1.4 showed, there is considerable variation across the middle-segment issues. They may impact just 10 clients or all the way up to 4 million client IPs. While some

[5] While the latency measurements to a later hop in traceroutes could be smaller than to an earlier hop [13, 77, 102], this issue is less prevalent when we look at latencies at the level of AS'es.

issues are ephemeral, others can last multiple hours.

Client-time Product: To prioritize the middle segment issues, we use "client-time product" as our metric. Client-time product is simply the product of (*a*) the predicted duration of bad performance of the middle segment, multiplied by, (*b*) the number of clients that are expected to access the cloud via that middle segment in that period (and hence, will be impacted). This is akin to the metric in §5.1.4 but with estimated values.

Budgeted Prioritization: `BlameIt` operates under a "budget" for traceroutes defined in the number of traceroutes permissible out of each of Azure's cloud locations per time window (say, every 5 minutes or every day). For simplicity, we avoid setting budgets per AS and instead employ a larger budget for traceroutes emanating from a cloud location.

Our prioritization takes all the quartets with middle segment issue (from Algorithm 2), and groups them by their AS-level BGP path (`m1 - m2` in §5.4.2's example). This grouped set `P` contains all the paths with at least one problematic AS. For each path `p` in `P`, we *predict* the two metrics ((a) and (b)) that are needed to calculate the client-time product. We sort paths in descending order of client-time product and issue traceroutes within the budget (one per middle segment issue). We next explain how the two two metrics are estimated.

(a) Predicting the duration of degradation. `BlameIt` estimates the probability of a problem persisting *given that* the problem has lasted thus far. Specifically, if an issue has lasted for a duration t, it estimates the probability of it lasting for an *additional* duration T, $\mathrm{P}(T|t)$. Using estimates of $\mathrm{P}(T|t)$ for different values of T, we can calculate the expected duration, $\Sigma\sum_{T=1}^{T_{\max}} P(T|t) * T$, for which the issue will persist. We increase T in increments of 5 minutes. We obtain $\mathrm{P}(T|t)$ for various T values based on historical fault durations in *each* BGP path. Given the long-tailed distribution of problem durations (§5.1.3) we only need to separate out the long-lived problems rather than precisely estimate the duration of each problem.

(b) Predicting the number of impacted clients. `BlameIt` uses historical data to also predict the number of clients that are likely to connect (and hence, be impacted) during an issue. We empirically observe that predicting the number of *active* clients in a BGP-path using the same 5-minute time window in the previous days is more accurate than using recent history (e.g., a few prior time windows in the same day). Hence, we use the average number of clients that connected via the same middle BGP-path in the same time window in the past 3 days.

The above approaches work well in practice with our prioritization of traceroutes closely matching an oracle (§5.5.5).

5.4.4 Optimized Background Measurements

Recall from the example in §5.4.2 that `BlameIt` uses traceroutes from *before* an incident of RTT degradation to localize the faulty AS. These "background" traceroutes are in addition to those issued for Algorithm 2's middle segment faults. Background traceroutes are issued *periodically* to each BGP path as well as *triggered* based on BGP path change at Azure's border routers.

To keep the overhead manageable, the periodic traceroutes are performed infrequently, two times a day to each BGP path from each Azure location. As we report in §5.5.5, the relative stability of Internet paths in normal times means that the above low frequency is a good "sweet spot" between traceroute overheads and accuracy of fault localization.

In addition, we use information from Azure's BGP listener, which connects to all of the border routers over IBGP, to determine if the AS level path to a client prefix has changed at a border router or a route has been withdrawn. In either case, we issue a traceroute to that client prefix from the cloud location which connects to the border router in question. Favorably for us, analysis of `BlameIt`'s BGP messages show that nearly two-thirds of the BGP paths at the routers do not see any churn in an entire day, thus limiting our overheads.

The combination of periodic (but infrequent) traceroutes and traceroutes triggered by BGP churn enables BlameIt to maintain an accurate picture of paths from the cloud to clients in various locations with low overhead.

5.5 Evaluation

BlameIt is in production deployment at Azure (§5.5.1 and §5.5.2); here are the highlights.

1. BlameIt's fault localization matches the results from manual investigations in 88 real-world incidents. §5.5.3
2. As further validation, we also use large-scale traceroutes to corroborate BlameIt's accuracy. §5.5.4
3. Active probing with BlameIt prioritizes the right middle-segment issues and is 72× cheaper. §5.5.5

5.5.1 Implementation and Deployment Details

BlameIt is in production deployment, its passive component since Nov 2017 and its active component beginning to get widely rolled out. Figure 5.7 shows all its relevant components. RTTs (from TCP handshakes) are continuously collected as clients connect to Azure and are aggregated at an analytics cluster where the BlameIt script runs periodically. Its outputs trigger prioritized alerts to operators, and targeted traceroutes to clients.

RTT Collector Stream: Azure's cloud locations generate two data streams, one containing the client's IP and the other the RTT (in addition to many other fields). These two streams had to be joined (via a "request id") for BlameIt's Algorithm 2 to be executed. Since these streams were large in size, their joining happened only once every day to limit the load on the data analytics cluster. As part of BlameIt's deployment, we modified the RTT stream to also include the client IP field, thus not needing to wait for the joining.

Periodic execution: BlameIt's Algorithm 2 is scheduled to execute every 15 minutes (which is sufficient given the long-tailed distribution we saw in §5.1.3). We encountered

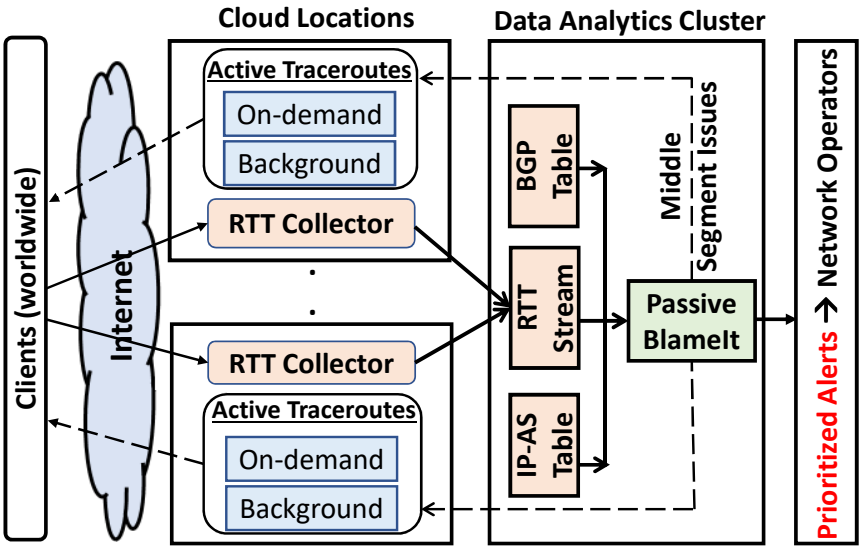


Figure 5.7: `BlameIt`'s key components in production.

one tricky issue in running `BlameIt` every 15 minutes. Every hour, a few hundred "storage buckets" are created afresh and each RTT tuple is written into a *randomly* chosen bucket. This leads to a loss of temporal ordering *within the hour*, so each run of `BlameIt`, even though it needs only the last 15 minutes of data, has to read all the buckets filled thus far in that hour (and filter out RTTs). We are currently working on creating finer buckets.

Active traceroutes: `BlameIt`'s outputs are used to prioritize middle segment issues and issue traceroutes to the clients. This module periodically fetches the destinations from the analytics cluster to issue on-demand traceroutes (§5.4.3) . Finally, it also issues background traceroutes (§5.4.4) at a regular cadence. For issuing traceroutes, we use the native Windows tracert command.

Network Operators: `BlameIt` helps prioritize the issues based on their business impact and the top few are automatically "ticketed" for investigation. The detailed outputs of `BlameIt` are auto-included in these tickets for ease of investigation. `BlameIt`'s coarse segmentation of the issues helps route the tickets to the appropriate teams to investigate

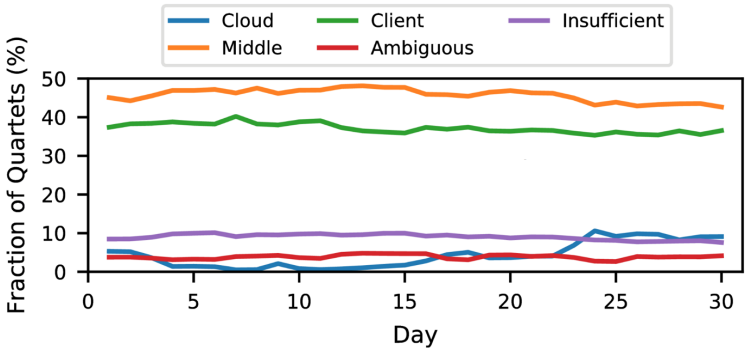


Figure 5.8: Blame fractions in a one-month period.

server issues, networking issues, peering relationships, etc. Crucially, they only investigate high-impact and relevant issues.

5.5.2 Blame Assignments in Production

Each day, `BlameIt` assigns blames for millions of "bad" quartets whose RTT are above the badness thresholds. We present a flavor of the results of `BlameIt`'s Algorithm 2 – the blame fractions and durations of badness.

Figure 5.8 plots the fraction of blame categories worldwide for a one month period. We notice a general stability of the fractions accounted by each blame category, with middle segment issues slightly higher than client issues. Even though cloud segment issues generally account for less than 4% of bad quartets, these are investigated with high priority. Their increase around day-24 in Figure 5.8 is due to a scheduled maintenance.

Figure 5.9 takes one of the days and splits the blame fractions by cloud regions. One notable aspect is that middle segment issues dominate in India, China and Brazil. This is likely due to the still-evolving transit networks in these regions, compared to relatively mature regions like the US. Another aspect we notice is that the "insufficient" and "ambiguous" categories constitute a high fraction, but this presents an interesting quandary: we can relax our minimum RTT samples and use coarser grouping for middle prefixes

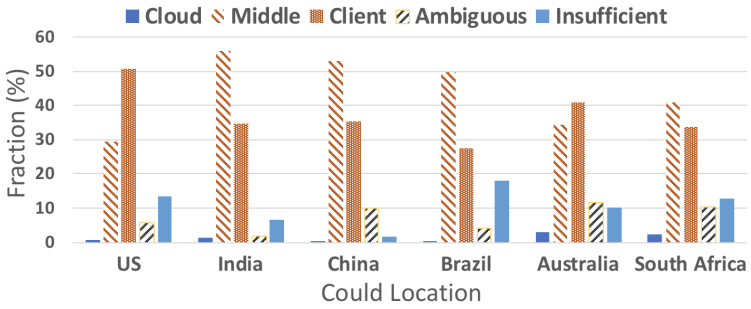


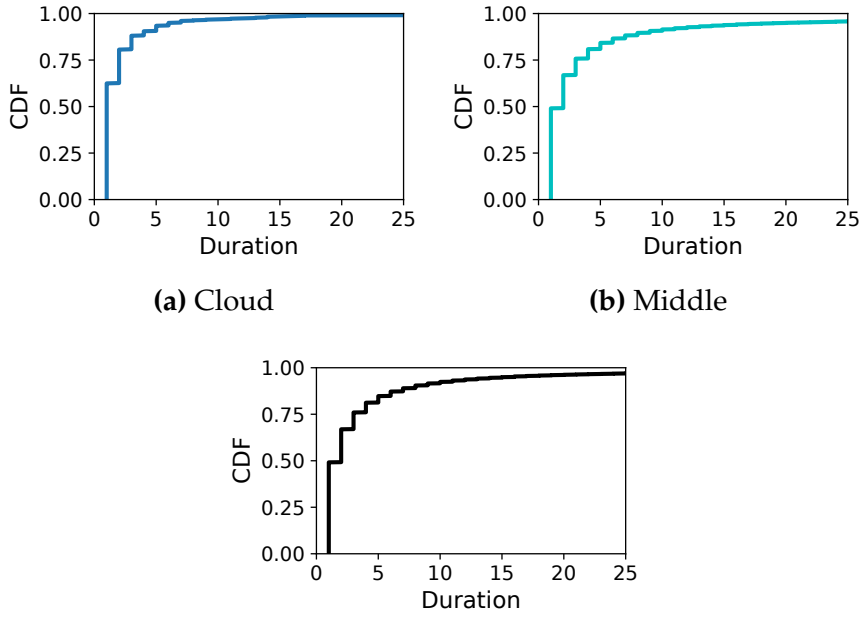
Figure 5.9: Blame fractions for one day in six cloud locations. (In each location, the blame fractions sum to 100%.)

(like AS, Metro) to reduce these categories, but it would likely come at the expense of accuracy of our fault localization. We could also issue traceroutes to the clients in these two categories, but that would increase the overheads of active probing.

Finally, how long do the different badness incidents last? Back in §5.1.3 we had observed a long-tailed distribution of all badness incidents, and Figure 5.10 breaks their durations by blame categories; similar distributions persist across the categories. Cloud issues generally last for lesser durations than middle or client segment issues, possibly explained by Azure dedicating a team to fix them at the earliest.

5.5.3 Real-world Case Studies

We analyzed investigation reports from 88 incidents of latency degradation in production that were investigated by Azure's network administrators. As part of their investigations, they look at performance logs, network captures, as well as communicate with administrators in other AS'es. Their reports document the cause behind the degradation and identify the faulty AS. An encouraging result is that `BlameIt`'s localization of the faulty AS matched the conclusions of the network administrators for *all* the incidents. We now discuss a few in detail to present a flavor of the faults, including some where `BlameIt` helped with the investigation.



(a) Cloud **(b)** Middle

(c) Client

Figure 5.10: Duration of cloud, middle, client segment issues in the units of consecutive 5-min time buckets.

1) Maintenance in Brazil: Azure's cloud network in one of the locations in Brazil had internal routing issues due to an unfinished maintenance operation that considerably increased the latency of the southern American clients that were connecting to the location. Since the incomplete maintenance was not detected, clients were not rerouted to a different location and investigations focused on other segments of the network including peering AS'es. The issue was finally fixed after a couple days. This incident was before the deployment of `BlameIt` and *post facto* analysis of the RTT logs shows the correct localization of the fault on the cloud segment.

2) Peering fault: There was a high-priority issue where many customers of Azure's services experienced high latencies. This was a widespread issue affecting many clients in the USA in the east coast, west coast, and central regions. `BlameIt` correctly identified

it to be a "middle segment" issue, thus avoiding the unnecessary involvement of Azure's internal-networking teams. Finer localization revealed that the issue was due to changes inside a peering AS, with which Azure peers at multiple locations.

This incident provided an interesting comparison with other monitoring systems. One system was based on periodic traceroutes *from a small fraction* of Azure's clients, but these clients happened to not be impacted much by this issue and hence did not detect the problem (in other words, lack of coverage of client "vantage points", a problem that `BlameIt` does not face). Another system made web users download a small web object to measure the latency, but these active measurement were conservatively deployed to limit overheads (since they were not triggered in a targeted manner, like `BlameIt` does using analysis of passive data). Finally, the system to monitor RTTs in each AS, metro also did not raise an alarm because no single AS, metro was excessively affected even though there were many affected clients countrywide (speaks to the value of `BlameIt`'s fine-grained analysis using BGP-paths and client IP-/24s). Combining large-scale aggregated measurements, we can detect and localize even slight but widespread increases in latency.

3) Cloud overload in Australia: Recently clients of Azure connecting to a cloud location in Australia experienced higher RTTs than the RTT targets. During this incident the median RTT went up from 25ms usually to 82ms. An interesting aspect in the investigation of this incident is that many clients sharing the BGP paths to reach this specific location (in Australia) saw increases in their RTTs. However, `BlameIt`'s approach of starting the blame assignment from the cloud (Algorithm 2) ensured that the blame was correctly pinpointed to the cloud segment (and not the middle BGP paths). As a validation, even though the same BGP paths were also used to connect to other nearby cloud locations in Australia, those clients did not experience bad RTTs (Insight-2 in §5.3.1). Investigations tracked the issue to an increase in CPU usage of the servers (overload) that was leading to the spike in RTTs.

4) Traffic shift from East Asia to US West coast: Due to some unforeseen side-effects

of changes in BGP announcements, Azure clients in east Asia were starting to get routed to Azure's locations in the US west coast instead of the locations in east Asia. This substantially increased their latencies and `BlameIt` blamed the middle segment. The ISPs of east Asian clients did not have good peers and connections to route them to the US west coast, since traffic rarely gets routed that direction, and thus, the middle segments contributed to the substantial increase. The issue was fixed with the east Asian clients being redirected back

5) Client ISP issues in Italy: The median RTT of Azure users in a major city in Italy increased from the usual value of 9ms to 161ms. Given the substantial increase in RTT, the persistence of the issue, and the high number of users that were affected, an investigation was launched that concluded that the increase was due to a maintenance inside the client ISP, for which no advance notice was provided.

This incident was prior to `BlameIt`'s deployment, but our analysis shows that `BlameIt` would have blamed the client AS with a high confidence of 93% (confidence is obtained by calculating the proportion of quartets blamed in each category of Algorithm 2). It would have avoided this wasted effort as there was little that Azure could do in fixing the issue.

5.5.4 Large-scale Corroboration

The case studies above are encouraging and notes by network operators help us see the value provided by `BlameIt` in their diagnoses. However, we also corroborate the results of `BlameIt` at a larger scale by comparing its results with those from continuous traceroutes. We continuously issue traceroutes (every minute) from different Azure locations to clients in $1,000$ select BGP paths. We treat the latency contributions (of each AS) from the traceroutes as the "ground truth". When latency goes beyond the targets, we compare each AS'es "normal" contribution to the end-to-end latency with the contributions *just after* the incident. We call the AS with the most increase in its contribution as the

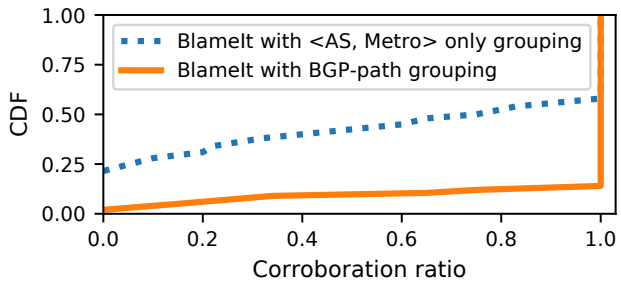


Figure 5.11: Large scale traceroute based validation. Corroboration ratios of `BlameIt` to clients in 1,000 BGP paths.

culprit AS. We compare this result to `BlameIt`'s identification of the culprit AS. Given the overhead of such large-scale traceroutes, we had to restrict this experiment to only three Azure locations in the US east coast for one day.

For each BGP path, we measure its *corroboration ratio*, defined as the fraction of latency issues where `BlameIt`'s passive diagnosis (of the culprit segment) matched the traceroutes. Note that this can include any of the AS'es in the middle or the client AS or Azure; recall from §5.3 that the blame is ascribed to only one segment. Figure 5.11 plots the CDF of corroboration ratios of the 1,000 BGP paths. We observe near-perfect corroboration (ratio of 1.0) for nearly 88% of the paths (orange line), thus showing that despite the two-level approach used by `BlameIt`, it does not lead to loss in accuracy. The figure also vindicates our decision to use BGP paths to group clients in the same middle segment as opposed to the traditional practice of grouping them by ⟨AS, Metro⟩ [71], which significantly lowers the corroboration ratio (blue line).

To reiterate, `BlameIt`'s high accuracy is without the high overhead of continuous traceroutes (which we deployed only for obtaining the ground truth for large-scale corroboration). We next analyze the aspects of our selective probing from §5.4.

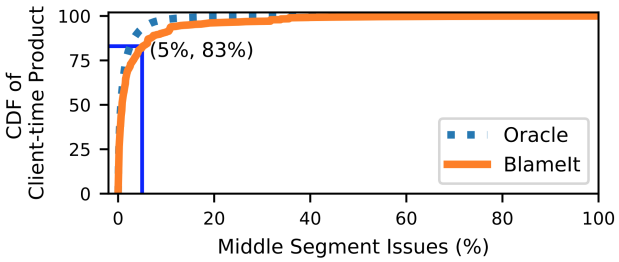


Figure 5.12: CDF of client-time product of middle segment issues ranked by the Oracle.

5.5.5 Active Probes for Middle Segment Issues

We dig deeper into `BlameIt`'s active traceroutes for fine-grained localization of middle segment issues.

For extensive data-driven evaluations, we issued traceroutes from 22 Azure cloud locations to 85,100 client IP /24's in 23,000 BGP paths, once every 10 minutes for a period of 14 days. As in §5.5.4, we use the traceroutes as ground truth for contributions by each AS. We identify middle segment issues, and identify the faulty AS in the middle by comparing against its historical contributions to the end-to-end latency.

Client-time product: Recall from §5.4.3 that `BlameIt` prioritizes on-demand traceroutes for middle segment issues based on an estimate of their client-time product. We had settled on simple approaches for our estimations and Figure 5.12 illustrates that as a result of our accurate estimates, we are able to prioritize the traceroutes as good as an oracle; Figure 5.12 sorts the middle segment issues (on the x-axis) by their client-time product as calculated by the oracle. Note that 5% of the middle segment issues cover over 83% of the cumulative client-time product impact. Thus, a 5% budget would suffice for diagnosing the high-impact middle segment issues.

Background probing frequency vs. Accuracy: There is an intrinsic trade-off between the frequency of background traceroutes (§5.4.4) and the accuracy of the fault localization. Too fine a frequency increases the accuracy but at the expense of overheads. Sending a

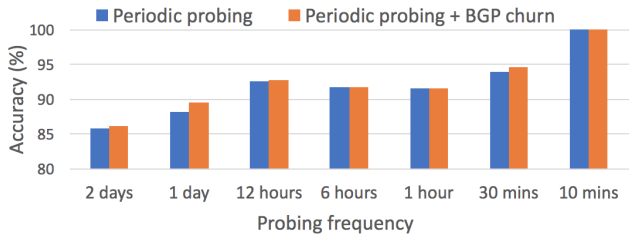


Figure 5.13: Accuracy of active `BlameIt` under different probing frequencies with/without probes triggered by BGP churn.

probe every ten minutes to cover all the BGP paths achieves high accuracy, but to cover all the BGP paths seen across all the Azure locations, this amounts to nearly 200 million traceroutes out of Azure each day, an unacceptably high overhead. Figure 5.13 plots how the accuracy drops at lower probing frequencies. We notice a "sweet spot": with a frequency of once per 12 hours (along with BGP churn triggered probes), we still obtain an accuracy of 93%. This frequency represents a traceroute overhead that is 72× lower, and this is feasible for Azure.

We also compare `BlameIt` to Trinocular [85] that diagnoses WAN unreachability (not latency inflation) using an optimized network model to trigger active probes. Compared to Trinocular, `BlameIt` issues 20× fewer active probes.

5.6 Summary

We presented `BlameIt`, a tool that automatically localizes the faulty AS when there is latency degradation between clients and cloud locations, using a combination of analysis of passive measurements (TCP handshake RTTs) and selective active measurements (traceroutes). Such a tool is highly valuable for global cloud providers like Azure. `BlameIt` smartly leverages the passively collected measurements and a small amount of active probes for its fault localization. In doing so, `BlameIt` avoids the problems of intractability that stifled prior tomography based solutions and prohibitively high overhead that

plagued probing solutions based on global vantage-points. `BlameIt` is in production deployment at Azure and produces results with high accuracy at low overheads.

Chapter 6
CONCLUSIONS AND FUTURE WORK

In this chapter, I summarize the work presented in this thesis, and outline some lines of future work that could improve the systems I built and the Internet performance in general.

6.1 Conclusions

This thesis supports the following thesis statement: **it is possible to build practical systems to help ISPs and cloud providers better analyze and understand Internet routing policies and localize the faults in the Internet to improve its performance**. The work presented in this thesis made the following contributions:

We revisit the AS relationship problem and inference techniques. We first develop a simple inference algorithm that achieves accuracy comparable to that of the state-of-the-art inference technique, `AS-Rank`, indicating that the types of most links in validation datasets are relatively easy to infer. We then construct different subsets of the validation dataset that might be considered *hard* and use these as benchmarks for evaluating improvements in AS relationship inference. Further, we observe that many of the features that can be used by inference techniques are of a stochastic nature, so we present a probabilistic AS relationship inference algorithm, `ProbLink`. It provides a framework for easy integration of many noisy but useful attributes into the relationship inference algorithm. We show that this probabilistic algorithm is more accurate and less sensitive to the locations of vantage points and BGP paths compared to the state-of-the-art algorithms. Our evaluation of `ProbLink` shows that it achieves an error rate that is better than that of `AS-Rank` overall by 1.7×, and achieves 1.8-6.1× better error rate for various categories of

hard links.

We developed three real-world practical applications on top of `ProbLink`. We developed a route leak detection system to troubleshoot the Internet safety issues, a selective advertisement prediction system to help operators predict the impact of traffic engineering policies on the active BGP paths, and a complex relationship inference system to understand the hybrid relationships in the Internet. Compared to the current state-of-the-art AS relationship inference algorithm, `ProbLink` increases the precision and recall of route leak detection by 4.1× and 3.4× respectively, reveals 27% more complex relationships, and increases the precision of predicting the impact of selective advertisements by 34%.

We developed a system, `BlameIt`, that automatically localizes the faulty AS when there is latency degradation between clients and cloud locations, using a combination of analysis of passive measurements (TCP handshake RTTs) and selective active measurements (traceroutes). Such a system is highly valuable for global cloud providers such as Microsoft, Google, and Amazon. `BlameIt` smartly leverages the passively collected measurements and a small amount of active probes for its fault localization. In doing so, `BlameIt` avoids the problems of intractability that stifled prior tomography based solutions and prohibitively high overhead that plagued probing solutions based on global vantage-points. `BlameIt` is in production deployment at Azure and produces results with high accuracy at low overheads.

6.2 Future Work

6.2.1 Performance-based Routing Optimization With SD-WAN

Over the course of doing the research work in my thesis, I realized that BGP has a few limitations. BGP was designed for exchanging routing reachability information, but it lacks a method to do performance-based route selection. For example, when there is performance degradation between clients and the cloud, the BGP path will not automatically

switch to a more performant path. Moreover, BGP provides ASes with little influence over the other ASes' routing decisions. For example, if the passive phase of `BlameIt` identifies an issue in the middle ASes between the clients and the cloud, the cloud provider cannot influence the BGP path from the clients to the cloud. The only way is to contact the network administrators of other ASes and expect them to solve the problem as soon as possible, which may take days or even weeks.

Unfortunately, inter-domain traffic engineering remains a black art where each AS adapts the BGP policies on its routers based on its local view. Even though systems like Google Espresso [113] and Facebook EdgeFabric [98] measure the application end-to-end performance in real-time and manipulates its outbound traffic, it is hard to be optimal. Inbound traffic engineering is even harder to achieve because of limited control over which paths will be selected by the other ASes. Some BGP techniques like AS path prepending, changing the MED attribute value, using the BGP community attribute value to hint the other ASes, and BGP prefix de-aggregation all have their limitations. And these BGP techniques are oftentimes ignored by the other ASes because most ASes make routing decisions based on the peering policy.

Although doing performance-based route selection is difficult, new trends and technologies bring opportunities to cloud providers. An increasing number of enterprises choose to connect their branches to cloud providers' networks to improve their branch-to-branch routing and the performance of accessing their cloud applications and services. One of these types of cloud services is Azure Virtual WAN (vWAN) [80]. Optimizing routing performance for cloud enterprise customers is hard for two reasons. First, ISPs in-between customers and cloud do not always get customers to the best edge locations to enter the cloud network backbone. Second, enterprise WAN often has very complex topologies, including branch offices widely spread across the world, virtual networks (VNets) deployed in the cloud which form their own private networks, and virtual WAN

hubs[1] placed in different regions.

To optimize the routing performance for such cloud enterprise customers, a cloud provider needs to solve the following three problems. (1) Routing problem: how to route a customer to a particular ISP to a particular edge location, then from the edge location to a particular vWAN hub and a particular VNet. (2) Measurement problem: how to do measurements to compare the performance of different routes. (3) Optimization problem: how to select the best ISPs, edges, Azure regions to provide the optimal performance to the customer and save their cost. The real challenge of solving these three problems is scale, because the number of possible routes from the customer to Azure is very large.

For the routing problem, a cloud provider has four knobs to steer vWAN customer traffic: 1) which cloud regions to host vWAN hubs, 2) which edge site(s) to ingress/egress customer traffic, 3) which peering connection(s) to ingress/egress customer traffic, 4) The SD-WAN controller at the customer side dynamically controls the route through which ISP to enter the cloud network. The *Software-defined Wide Area Network (SD-WAN)* controller can be informed by the cloud provider about which egress ISP they should select.

The traditional function of enterprise WAN was to connect users at the branch office to applications hosted on servers in the enterprise's own data center. Typically, dedicated MPLS circuits were used for secure and reliable connectivity. It is now a cloud-centric world: enterprises use a lot of cloud applications. The traditional WAN is no longer suitable mainly because backhauling all traffic to the enterprise's data center where firewalls are deployed introduces latency and hurts application performance. SD-WAN is a software-defined approach to managing the enterprise WAN, and it is a popular technology that many enterprises have adopted.

SD-WAN contains two components, SD-WAN appliance and SD-WAN controller. SD-WAN appliance, or customer premises equipment (CPE), is a specialized router deployed at the branch site, which only does packet forwarding. The CPE devices are all connected

[1]Virtual WAN hub is a regional core of the virtual WAN which contains a hub gateway that serves as a connection point for branches and virtual networks.

to a centralized SD-WAN controller. The SD-WAN controller is the brain, and decides which packets should go which route. The SD-WAN controller is often located in the SD-WAN vendor's cloud.

By leveraging the SD-WAN devices deployed at the enterprise customers' side, a cloud provider has the opportunity to do performance-based ingress traffic engineering by redirecting user traffic to carefully advertised unicast addresses.

6.2.2 Internet Measurement Tools

The future research and development of measurement tools will help accelerate the improvement of Internet performance. `BlameIt` relies on Traceroute to do fine-grained fault localization. However, Traceroute has a few shortcomings.

Not all of the devices along the path respond to Traceroute, which makes the router IP address and RTT information in-between the source and destination incomplete, and makes it difficult for `BlameIt` to locate the fault. Some network administrators block ICMP traffic, while some firewalls block UDP traffic while allowing ICMP to get through. A technique that can improve the chance of successful Traceroute probing will help `BlameIt`. Moreover, the ICMP latency might be a wrong indicator of the latency of real production traffic [78].

We do not have a bidirectional Traceroute technique so far. The Internet is bidirectional and the Internet paths can be asymmetric. Due to routing asymmetries, the "forward" (cloud-to-client) and "reverse" (client-to-cloud) Internet paths can be different. `BlameIt` only uses traceroutes issued from the cloud locations, but we believe that bidirectional Traceroute techniques can be incorporated into `BlameIt` 's active phase to pinpoint the faulty AS more accurately.

BIBLIOGRAPHY

[1] Google Video Quality Report. https://support.google.com/youtube/answer/6013340?hl=en.

[2] IXP Service Matrix. https://www.euro-ix.net/en/tools/ixp-service-matrix/.

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

[4] RIPE (RIS). http://www.ripe.net/ris/.

[5] U. Oregon Route Views Project. http://www.routeviews.org/.

[6] Bernhard Ager, Nikolaos Chatzis, Anja Feldmann, Nadi Sarrar, Steve Uhlig, and Walter Willinger. Anatomy of a large european ixp. *ACM SIGCOMM Computer Communication Review*, 42(4):163–174, 2012.

[7] B. Al-Musawi, P. Branch, and G. Armitage. BGP anomaly detection techniques: A survey. *IEEE Communications Surveys Tutorials*, 19(1):377–396, Firstquarter 2017.

[8] Ruwaifa Anwar, Haseeb Niaz, David Choffnes, Ítalo Cunha, Phillipa Gill, and Ethan Katz-Bassett. Investigating interdomain routing policies in the wild. In *Proceedings of the 2015 Internet Measurement Conference*, pages 71–77. ACM, 2015.

[9] Todd Arnold, Ege Gürmeriçliler, Georgia Essig, Arpit Gupta, Matt Calder, Vasileios Giotsas, and Ethan Katz-Bassett. (how much) does a private wan improve cloud performance? In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pages 79–88. IEEE, 2020.

[10] Todd Arnold, Jia He, Weifan Jiang, Matt Calder, Italo Cunha, Vasileios Giotsas, and Ethan Katz-Bassett. Cloud provider connectivity in the flat internet. In *Proceedings of the ACM Internet Measurement Conference*, pages 230–246, 2020.

[11] Behnaz Arzani, Selim Ciraci, Luiz Chamon, Yibo Zhu, Hongqiang (Harry) Liu, Jitu Padhye, Boon Thau Loo, and Geoff Outhred. 007: Democratically finding the cause of packet drops. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 419–435, Renton, WA, 2018. USENIX Association.

[12] Behnaz Arzani, Selim Ciraci, Boon Thau Loo, Assaf Schuster, and Geoff Outhred. Taking the blame game out of data centers operations with netpoirot. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 440–453. ACM, 2016.

[13] Brice Augustin, Xavier Cuvellier, Benjamin Orgogozo, Fabien Viger, Timur Friedman, Matthieu Latapy, Clémence Magnien, and Renata Teixeira. Avoiding traceroute anomalies with paris traceroute. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 153–158. ACM, 2006.

[14] Paul Bennett. Assessing the calibration of Naive Bayes' posterior estimates. Technical report, September 2000. Computer Science Department, School of Computer Science, Carnegie Mellon University.

[15] A. Broido and k. claffy. Analysis of RouteViews BGP data: policy atoms. In *Network Resource Data Management Workshop*, Santa Barbara, CA, May 2001.

[16] Randy Bush, Olaf Maennel, Matthew Roughan, and Steve Uhlig. Internet optometry: assessing the broken glasses in internet reachability. IMC '09.

[17] CAIDA. AS types. https://www.caida.org/research/routing/astypes.

[18] CAIDA. Inferred AS to organization mapping dataset. http://www.caida.org/data/as-organizations/.

[19] Matt Calder, Xun Fan, Zi Hu, Ethan Katz-Bassett, John Heidemann, and Ramesh Govindan. Mapping the expansion of google's serving infrastructure. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 313–326. ACM, 2013.

[20] Matt Calder, Ryan Gao, Manuel Schröder, Ryan Stewart, Jitendra Padhye, Ratul Mahajan, Ganesh Ananthanarayanan, and Ethan Katz-Bassett. Odin: Microsoft's scalable fault-tolerant CDN measurement system. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, Renton, WA, 2018. USENIX Association.

[21] Rui Castro, Mark Coates, Gang Liang, Robert Nowak, and Bin Yu. Network tomography: Recent developments. *Statistical science*, pages 499–517, 2004.

[22] Fangfei Chen, Ramesh K Sitaraman, and Marcelo Torres. End-user mapping: Next generation request routing for content delivery. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 167–181. ACM, 2015.

[23] Cloudflare. Bgp leaks and cryptocurrencies. https://blog.cloudflare.com/bgp-leaks-and-crypto-currencies/.

[24] Avichai Cohen, Yossi Gilad, Amir Herzberg, and Michael Schapira. Jumpstarting BGP security with path-end validation. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 342–355. ACM, 2016.

[25] Giovanni Comarela, Evimaria Terzi, and Mark Crovella. Detecting unusually-routed ASes: methods and applications. In *Proceedings of the 2016 Internet Measurement Conference*, IMC '16, pages 445–459, New York, NY, USA, 2016. ACM.

[26] Ítalo Cunha, Pietro Marchetta, Matt Calder, Yi-Ching Chiu, Brandon Schlinker, Bruno VA Machado, Antonio Pescapè, Vasileios Giotsas, Harsha V Madhyastha, and Ethan Katz-Bassett. Sibyl: A practical internet route oracle. In *NSDI*, pages 325–344, 2016.

[27] Amogh Dhamdhere, David D. Clark, Alexander Gamero-Garrido, Matthew Luckie, Ricky K. P. Mok, Gautam Akiwate, Kabir Gogia, Vaibhav Bajpai, Alex C. Snoeren, and Kc Claffy. Inferring persistent interdomain congestion. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, pages 1–15, New York, NY, USA, 2018. ACM.

[28] Amogh Dhamdhere, David D Clark, Alexander Gamero-Garrido, Matthew Luckie, Ricky KP Mok, Gautam Akiwate, Kabir Gogia, Vaibhav Bajpai, Alex C Snoeren, and Kc Claffy. Inferring persistent interdomain congestion. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 1–15. ACM, 2018.

[29] Giuseppe Di Battista, Maurizio Patrignani, and Maurizio Pizzonia. Computing the types of the relationships between autonomous systems. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, volume 1, pages 156–165. IEEE, 2003.

[30] Xenofontas Dimitropoulos, Dmitri Krioukov, Marina Fomenkov, Bradley Huffaker, Young Hyun, George Riley, et al. AS relationships: Inference and validation. SIGCOMM '07, 2007.

[31] Xenofontas Dimitropoulos, Dmitri Krioukov, Bradley Huffaker, George Riley, et al. Inferring AS relationships: Dead end or lively beginning? In *International Workshop on Experimental and Efficient Algorithms*, pages 113–125. Springer, 2005.

[32] Benoit Donnet and Olivier Bonaventure. On BGP communities. *ACM SIGCOMM Computer Communication Review*, 38(2):55–59, 2008.

[33] Nick Duffield. Network tomography of binary network performance characteristics. *IEEE Transactions on Information Theory*, 52(12):5373–5388, 2006.

[34] Qilin Fan, Hao Yin, Geyong Min, Po Yang, Yan Luo, Yongqiang Lyu, Haojun Huang, and Libo Jiao. Video delivery networks: Challenges, solutions and future directions. *Computers & Electrical Engineering*, 66:332–341, 2018.

[35] Peyman Faratin, David D Clark, Steven Bauer, William Lehr, Patrick W Gilmore, and Arthur Berger. The growing complexity of internet interconnection. *Communications & Strategies*, (72):51, 2008.

[36] Anja Feldmann, Oliver Gasser, Franziska Lichtblau, Enric Pujol, Ingmar Poese, Christoph Dietzel, Daniel Wagner, Matthias Wichtlhuber, Juan Tapiador, Narseo Vallina-Rodriguez, et al. The lockdown effect: Implications of the covid-19 pandemic on internet traffic. In *Proceedings of the ACM Internet Measurement Conference*, pages 1–18, 2020.

[37] Tobias Flach, Nandita Dukkipati, Andreas Terzis, Barath Raghavan, Neal Cardwell, Yuchung Cheng, Ankur Jain, Shuai Hao, Ethan Katz-Bassett, and Ramesh Govindan. Reducing web latency: the virtue of gentle aggression. In *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, pages 159–170, 2013.

[38] Ashley Flavel, Pradeepkumar Mani, David A Maltz, Nick Holt, Jie Liu, Yingying Chen, and Oleg Surmachev. Fastroute: A scalable load-aware anycast routing architecture for modern cdns. *connections*, 27:19, 2015.

[39] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[40] Lixin Gao. On inferring autonomous system relationships in the internet. *IEEE/ACM Transactions on Networking*, 9(6):733–745, Dec 2001.

[41] Denisa Ghita, Katerina Argyraki, and Patrick Thiran. Network tomography on correlated links. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 225–238. ACM, 2010.

[42] Denisa Ghita, Can Karakus, Katerina Argyraki, and Patrick Thiran. Shifting network tomography toward a practical goal. In *Proceedings of the Seventh COnference on Emerging Networking EXperiments and Technologies*, CoNEXT '11, pages 24:1–24:12, New York, NY, USA, 2011. ACM.

[43] Phillipa Gill, Martin Arlitt, Zongpeng Li, and Anirban 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.

[44] Phillipa Gill, Michael Schapira, and Sharon Goldberg. Let the market drive deployment: A strategy for transitioning to BGP security. In *SIGCOMM '11*, volume 41, pages 14–25. ACM, 2011.

[45] V. Giotsas, M. Luckie, B. Huffaker, and k. claffy. Inferring complex AS relationships. In *Internet Measurement Conference (IMC)*, pages 23–30, Nov 2014.

[46] Vasileios Giotsas, Christoph Dietzel, Georgios Smaragdakis, Anja Feldmann, Arthur Berger, and Emile Aben. Detecting peering infrastructure outages in the wild. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 446–459. ACM, 2017.

[47] Vasileios Giotsas and Shi Zhou. Valley-free violation in internet routing—analysis based on BGP community data. In *Communications (ICC), 2012 IEEE International Conference on*, pages 1193–1197. IEEE, 2012.

[48] Vasileios Giotsas, Shi Zhou, Matthew Luckie, and kc claffy. Inferring multilateral peering. In *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '13, pages 247–258, New York, NY, USA, 2013. ACM.

[49] Sharon Goldberg. Why is it taking so long to secure Internet routing? *Communications of the ACM*, 57(10):56–63, 2014.

[50] Osama Haq, Mamoon Raja, and Fahad R Dogar. Measuring and improving the reliability of wide-area cloud paths. In *Proceedings of the 26th International Conference on World Wide Web*, pages 253–262. International World Wide Web Conferences Steering Committee, 2017.

[51] Yihua He, Michalis Faloutsos, Srikanth Krishnamurthy, and Bradley Huffaker. On routing asymmetry in the internet. In *GLOBECOM'05. IEEE Global Telecommunications Conference, 2005.*, volume 2, pages 6–pp. IEEE, 2005.

[52] Yihua He, Georgos Siganos, Michalis Faloutsos, and Srikanth Krishnamurthy. Lord of the links: a framework for discovering missing links in the Internet topology. *IEEE/ACM Transactions on Networking (ToN)*, 17(2):391–404, 2009.

[53] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. Achieving high utilization with software-driven wan. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, pages 15–26, 2013.

[54] Geoff Huston. The death of transit and the future internet. Second ITU Workshop on Network 2030, December 2018.

[55] Google Inc. Google peering policy. https://peering.google.com.

[56] Mauch Jared. BGP routing leak detection system. https://puck.nether.net/bgp/leakinfo.cgi.

[57] E. Jasinska, N. Hilliard, R. Raszuk, and Bakker N. RFC 7947: Internet exchange BGP route server. https://tools.ietf.org/html/rfc7947.

[58] Junchen Jiang, Rajdeep Das, Ganesh Ananthanarayanan, Philip A Chou, Venkata Padmanabhan, Vyas Sekar, Esbjorn Dominique, Marcin Goliszewski, Dalibor Kukoleca, Renat Vafin, et al. Via: Improving internet telephony call quality using predictive relay selection. In *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*, pages 286–299. ACM, 2016.

[59] Yuchen Jin, Sundararajan Renganathan, Ganesh Ananthanarayanan, Junchen Jiang, Venkata N Padmanabhan, Manuel Schroder, Matt Calder, and Arvind Krishnamurthy. Zooming in on wide-area latencies to a global cloud provider. In *Proceedings of the ACM Special Interest Group on Data Communication*, pages 104–116. 2019.

[60] Yuchen Jin, Colin Scott, Amogh Dhamdhere, Vasileios Giotsas, Arvind Krishnamurthy, and Scott Shenker. Stable and practical AS relationship inference with problink. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 581–598, Boston, MA, February 2019. USENIX Association.

[61] Joshua Juen, Aaron Johnson, Anupam Das, Nikita Borisov, and Matthew Caesar. Defending tor from network adversaries: A case study of network path prediction. *Proceedings on Privacy Enhancing Technologies*, 2015(2):171–187, 2015.

[62] Partha Kanuparthy and Constantine Dovrolis. Pythia: Diagnosing performance problems in wide area providers. In *USENIX Annual Technical Conference*, pages 371–382, 2014.

[63] Ethan Katz-Bassett, David R Choffnes, Ítalo Cunha, Colin Scott, Thomas Anderson, and Arvind Krishnamurthy. Machiavellian routing: improving Internet availability with BGP poisoning. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, page 11. ACM, 2011.

[64] Maria Konte, Roberto Perdisci, and Nick Feamster. Aswatch: An AS reputation system to expose bulletproof hosting ases. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 625–638. ACM, 2015.

[65] KPN NOC. BGP Communities For AS286. https://as286.net/AS286-communities.html, September 2018.

[66] Rupa Krishnan, Harsha V. Madhyastha, Sushant Jain, Sridhar Srinivasan, Arvind Krishnamurthy, Thomas Anderson, and Jie Gao. Moving beyond end-to-end path information to optimize cdn performance. In *Internet Measurement Conference (IMC)*, pages 190–201, Chicago, IL, 2009.

[67] Simon Kuenzer, Anton Ivanov, Filipe Manco, Jose Mendes, Yuri Volchkov, Florian Schmidt, Kenichi Yasukata, Michio Honda, and Felipe Huici. Unikernels everywhere: The case for elastic CDNs. *SIGPLAN Not.*, 52(7):15–29, April 2017.

[68] Anukool Lakhina, Mark Crovella, and Christophe Diot. Diagnosing network-wide traffic anomalies. In *ACM SIGCOMM Computer Communication Review*, volume 34, pages 219–230. ACM, 2004.

[69] Paul Laskowski and John Chuang. Network monitors and contracting systems: Competition and innovation. SIGCOMM '06, pages 183–194, 2006.

[70] Felix Lau, Stuart H Rubin, Michael H Smith, and Ljiljana Trajkovic. Distributed denial of service attacks. In *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, volume 3, pages 2275–2280. IEEE, 2000.

[71] Youndo Lee and Neil Spring. Identifying and aggregating homogeneous ipv4 /24 blocks with hobbit. In *Internet Measurement Conference (IMC)*, Santa Monica, CA, 2016.

[72] Aemen Hassaan Lodhi. *The economics of Internet peering interconnections*. PhD thesis, Georgia Institute of Technology, 2014.

[73] Matthew Luckie, Bradley Huffaker, Amogh Dhamdhere, Vasileios Giotsas, et al. AS relationships, customer cones, and validation. IMC '13.

[74] Harsha V Madhyastha, Tomas Isdal, Michael Piatek, Colin Dixon, Thomas Anderson, Arvind Krishnamurthy, and Arun Venkataramani. iplane: An information plane for distributed services. In *Proceedings of the 7th symposium on Operating systems design and implementation*, pages 367–380. USENIX Association, 2006.

[75] Ajay Anil Mahimkar, Zihui Ge, Aman Shaikh, Jia Wang, Jennifer Yates, Yin Zhang, and Qi Zhao. Towards automated performance diagnosis in a large iptv network. In *ACM SIGCOMM Computer Communication Review*, volume 39, pages 231–242. ACM, 2009.

[76] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[77] Morley Mao, Jennier Rexford, Jia Wang, and Randy Katz. Towards an accurate as-level traceroute tool. In *ACM SIGCOMM*, 2003.

[78] Pietro Marchetta, Valerio Persico, Antonio Pescapé, and Ethan Katz-Bassett. Don't trust traceroute (completely). In *Proceedings of the 2013 workshop on Student workhop*, pages 5–8, 2013.

[79] Riad Mazloum, Marc-Olivier Buob, Jordan Auge, Bruno Baynat, Dario Rossi, and Timur Friedman. Violation of interdomain routing assumptions. In *International Conference on Passive and Active Network Measurement*, pages 173–182. Springer, 2014.

[80] Microsoft. Azure Virtual WAN Overview. https://docs.microsoft.com/en-us/azure/virtual-wan/virtual-wan-about.

[81] Asya Mitseva, Andriy Panchenko, and Thomas Engel. The state of affairs in BGP security: A survey of attacks and defenses. *Computer Communications*, 124:45 – 60, 2018.

[82] Wolfgang Mühlbauer, Anja Feldmann, Olaf Maennel, Matthew Roughan, and Steve Uhlig. Building an AS-topology model that captures route diversity. SIGCOMM '06, pages 195–206, 2006.

[83] Kamal Nigam, Andrew Kachites McCallum, Sebastian Thrun, and Tom Mitchell. Text classification from labeled and unlabeled documents using EM. *Machine learning*, 39(2-3):103–134, 2000.

[84] Rishab Nithyanand, Oleksii Starov, Phillipa Gill, Adva Zair, and Michael Schapira. Measuring and mitigating AS-level adversaries against Tor. In *23rd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016*, 2016.

[85] Lin Quan, John Heidemann, and Yuri Pradkin. Trinocular: Understanding internet reliability through adaptive probing. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 255–266. ACM, 2013.

[86] B. Quoitin, C. Pelsser, L. Swinnen, O. Bonaventure, and S. Uhlig. Interdomain traffic engineering with BGP. *IEEE Communications Magazine*, 41(5):122–128, May 2003.

[87] Bruno Quoitin, Sébastien Tandel, Steve Uhlig, and Olivier Bonaventure. Interdomain traffic engineering with redistribution communities. *Computer Communications*, 27(4):355–363, 2004.

[88] RETN NOC. BGP communities For AS9002. http://retn.net/support/bgp-communities/, September 2018.

[89] RFC. RFC 1058: Routing Information Protocol. https://tools.ietf.org/html/rfc1058.

[90] RFC. RFC 1142: OSI IS-IS Intra-domain Routing Protocol. https://tools.ietf.org/html/rfc1142.

[91] RFC. RFC 1583: OSPF Version 2. https://tools.ietf.org/html/rfc1583.

[92] RFC. RFC 4271: A Border Gateway Protocol 4 (BGP-4). https://tools.ietf.org/html/rfc4271.

[93] RFC. RFC 4893: BGP Support for Four-octet AS Number Space. https://tools.ietf.org/html/rfc4893.

[94] RFC. RFC 6996: Autonomous System (AS) reservation for private use. https://tools.ietf.org/html/rfc6996.

[95] RIPE Atlas. User-defined measurements - rate limits. https://atlas.ripe.net/docs/udm/#rate-limits.

[96] Irina Rish et al. An empirical study of the Naive Bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, pages 41–46. IBM New York, 2001.

[97] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, and Alex C Snoeren. Passive realtime datacenter fault detection and localization. In *NSDI*, pages 595–612, 2017.

[98] Brandon Schlinker, Hyojeong Kim, Timothy Cui, Ethan Katz-Bassett, Harsha V Madhyastha, Italo Cunha, James Quinn, Saif Hasan, Petr Lapukhov, and Hongyi Zeng. Engineering egress with edge fabric: Steering oceans of content to the world. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 418–431. ACM, 2017.

[99] Ankit Singla, Balakrishnan Chandrasekaran, P Godfrey, and Bruce Maggs. The internet at the speed of light. In *Proceedings of the 13th ACM Workshop on Hot Topics in Networks*, page 1. ACM, 2014.

[100] Neil Spring, Ratul Mahajan, and Thomas Anderson. The causes of path inflation. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 113–124. ACM, 2003.

[101] Kotikalapudi Sriram, Doug Montgomery, D McPherson, Eric Osterweil, and Brian Dickson. Problem definition and classification of BGP route leaks. https://tools.ietf.org/html/rfc7908.

[102] Richard Steenbergen. A practical guide to (correctly) a practical guide to (correctly) troubleshooting with traceroute. In *NANOG*, 2017.

[103] Lakshminarayanan Subramanian, Sharad Agarwal, Jennifer Rexford, and Randy H Katz. Characterizing the Internet hierarchy from multiple vantage points. IEEE, 2002.

[104] Akamai Technologies. Akamai Network partnerships. https://www.akamai.com/us/en/products/network-operator/akamai-network-partnerships.jsp.

[105] Vytautas Valancius, Bharath Ravi, Nick Feamster, and Alex C Snoeren. Quantifying the benefits of joint content and network routing. In *ACM SIGMETRICS Performance Evaluation Review*, volume 41, pages 243–254. ACM, 2013.

[106] Jinu Susan Varghese and Lu Ruan. Computing customer cones of peering networks. In *Proceedings of the 2016 Applied Networking Research Workshop*, pages 35–37. ACM, 2016.

[107] Qiong Wang, George M Garrity, James M Tiedje, and James R Cole. Naive Bayesian classifier for rapid assignment of rRNA sequences into the new bacterial taxonomy. *Applied and environmental microbiology*, 73(16):5261–5267, 2007.

[108] Wikipedia. Autonomous system (internet). https://en.wikipedia.org/wiki/Autonomous_system_(Internet).

[109] Wikipedia. Global internet usage. https://en.wikipedia.org/wiki/Global_Internet_usage.

[110] Wikipedia. List of Tier-2 ASes. https://en.wikipedia.org/wiki/Tier_2_network.

[111] Walter Willinger and Matthew Roughan. Internet topology research redux. *ACM SIGCOMM eBook: Recent Advances in Networking*, 2013.

[112] Jianhong Xia and Lixin Gao. On the evaluation of AS relationship inferences. In *Global Telecommunications Conference, 2004. GLOBECOM'04. IEEE*, volume 3, pages 1373–1377. IEEE, 2004.

[113] Kok-Kiong Yap, Murtaza Motiwala, Jeremy Rahe, Steve Padgett, Matthew Holliman, Gary Baldus, Marcus Hines, Taeeun Kim, Ashok Narayanan, Ankur Jain, et al. Taking the edge off with espresso: Scale, reliability and programmability for global internet peering. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 432–445. ACM, 2017.

[114] Harry Zhang. The optimality of Naive Bayes. In Valerie Barr and Zdravko Markov, editors, *Proceedings of the Seventeenth International Florida Artificial Intelligence Research Society Conference (FLAIRS 2004)*. AAAI Press, 2004.

[115] Harry Zhang and Jiang Su. Naive Bayesian classifiers for ranking. In *European conference on machine learning*, pages 501–512. Springer, 2004.

[116] Ming Zhang, Chi Zhang, Vivek S Pai, Larry L Peterson, and Randolph Y Wang. Planetseer: Internet path failure monitoring and characterization in wide-area services. In *OSDI*, volume 4, pages 12–12, 2004.

[117] Zheng Zhang, Ming Zhang, Albert G Greenberg, Y Charlie Hu, Ratul Mahajan, and Blaine Christian. Optimizing cost and performance in online service provider networks. In *NSDI*, pages 33–48, 2010.