Evaluating and Improving Internet Load Balancing with Large-Scale Latency Measurements

by

Yibo Pi

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in the University of Michigan
2021

Doctoral Committee:

Associate Professor Sugih Jamin, Chair
Associate Professor Harsha Madhyastha
Associate Professor Chinedum Okwudire
Professor Atul Prakash

Yibo Pi

yibo@umich.edu

ORCID iD: 0000-0003-1287-3311

To my family and friends.

ACKNOWLEDGEMENTS

I am very grateful for the opportunity to study here at the University of Michigan. I would first like to thank my advisor, Professor Sugih Jamin, for his support and guidance through this journey. Sugih is always very patient with me and allows me to grow at my own pace. He treats me more like a friend than a student. He respects my ideas, decisions and feelings, and always spares no effort to help me, especially during difficult times. He teaches me how to become an independent researcher and, more importantly, how to work with others and treat everyone with kindness, respect and compassion. I truly appreciate the experience of working with him in the past six years.

I am also thankful for all my collaborators and sponsors for their generous support. I thank Peter Danzig for devoting his personal time working with me for four years. Peter always guided me to think deeper on the practical impact of my work and generously introduced my work to his industry colleagues. I thank Professor Feng Qian for his valuable comments on the paper, which changed my career path. I greatly appreciate Professor Atul Prakash, Harsha Madhyastha, and Chinedum Okwudire serving as my committee members.

I am fortunate to meet many wonderful friends at Ann Arbor and would like to thank you all for your company through this journey. I thank labmates, Yike Liu, Haojun Ma, Yulong Cao, Qingzhao Zhang, for countless time of studying together and my roommate, Shichang Xu, for being such a supportive and considerate friend. I would like to specially thank Winnie Song for accompanying me through the most difficult times. At last, I would like to sincerely thank my parents and girlfriend, Yujing Zhong, for their unconditional love and support. They are always by my side through ups and downs.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

Table

ABSTRACT

Load balancing is used in the Internet to distribute load across resources at different levels, from global load balancing that distributes client requests across servers at the Internet level to path-level load balancing that balances traffic across load-balanced paths. These load balancing algorithms generally work under certain assumptions on performance similarity. Specifically, global load balancing divides the Internet address space into client aggregations and assumes that clients in the same aggregation have similar performance to the same server; load-balanced paths are generally selected for load balancing as if they have similar performance. However, as performance similarity is typically achieved with similarity in path properties, e.g., topology and hop count, which do not necessarily lead to similar performance, performance between clients in the same aggregation and between load-balanced paths could differ significantly.

This dissertation evaluates and improves global and path-level load balancing in terms of performance similarity. We achieve this with large-scale latency measurements, which not only allow us to systematically identify and evaluate the performance issues of Internet load balancing at scale, but also enable us to develop data-driven approaches to improve the performance. Specifically, this dissertation consists of three parts. First, we study the issues of existing client aggregations for global load balancing and then design AP-atoms, a data-driven client aggregation learned from passive large-scale latency measurements. Second, we show that the latency imbalance between load-balanced paths, previously deemed insignificant, is now both significant and prevalent. We present Flipr, a network prober that actively collects large-scale latency measurements to characterize the latency imbalance

issue. Lastly, we design another network prober, Congi, that can detect congestion at scale and use Congi to study the congestion imbalance problem at scale. For both latency and congestion imbalance, we demonstrate that they could greatly affect the performance of various applications.

CHAPTER I

Introduction

Internet load balancing is used to distribute network traffic load across multiple resources for better utilization and performance. At the Internet level, for scalable management, global load balancing divides Internet address space into aggregation units of proper sizes for the accurate assignment of clients to their nearby servers. Akamai's global load balancing system routes trillions of clients' requests to servers per day in its content delivery network (CDN) [1]. At the path level, load balancers forward packets across multiple paths to balance traffic. The paths between about 90% of source-destination pairs are reported to traverse a load balancer [2]. Global load balancing assumes that clients in the same aggregation have similar performance to the same server such that they can be managed together and share the same server redirection decisions. Path-level load balancing assumes that load-balanced (LB) paths have similar performance such that load balancers can choose among LB paths with simple hashing algorithms to distribute traffic as if it makes no difference using one path or another [3].

1.1 Why Existing Methods Fail to Achieve Performance Similarity?

Existing methods take a simple way to achieve performance similarity by assuming that similar properties lead to similar performance. This assumption makes the problem of achieving performance similarity much easier, because properties of an object, e.g.,

topology of a path and geographic locations of clients, are typically easy to obtain and stable over time. However, this assumption does not always hold. When it fails, objects could differ significantly in performance. This dissertation focuses on the performance dissimilarity issue in DNS-based global load balancing and between load-balanced paths.

DNS-based Global Load Balancing As there are billions of clients in the Internet, these clients are generally aggregated into aggregation units for scalable management, where clients in the same aggregation should have similar performance to the same server. Choosing the proper size of client aggregations is a complex problem, which determines the accuracy of server assignment and the number of aggregation units that a mapping system has to maintain. DNS-based client aggregation has been adopted by many CDN provides like Google [4] and Akamai [5]. Since the local DNS nameserver (LDNS) of a client is typically co-located with the client, it is common to aggregate clients by their LDNSs. After that, to make redirection decisions for billions of clients, a mapping system only needs to measure the performance from servers to hundreds of thousands of LDNSs. However, a recent study on Akamai's CDN found that aggregation by LDNS results in poor client redirection for nearly 20% of client demand, where clients use either public resolvers or LDNSs remote to the clients [1]. In contrast, we can use /24 IP blocks of fine granularity as client aggregations, which may be accurate in redirection but not scalable as the mapping system has to maintain performance from servers to millions of /24s. As a good tradeoff between scalability and accuracy, /20 IP blocks have been proposed in [1]. There are also other aggregation methods aggregating clients by their geographical locations [6] and BGP prefixes [7]. However, all existing aggregations rely on the property similarity between clients, which does not equate similar performance.

Path-Level Load Balancing Load balancers commonly use equal-cost multi-path routing (ECMP) to find LB paths of equal cost (e.g., hop count) such that LB paths have similar performance [3]. However, as the cost of a path is not universally defined and equal cost

does not necessarily equate similar performance, LB paths could differ significantly in performance. Further, large flows could cause one path to be congested while the other paths are uncongested. Extensive efforts have been made to measure the topology of LB paths [2, 8, 9], but the performance difference (or *performance imbalance*) between LB paths is still under-explored. To better understand this issue, this dissertation studies latency and congestion imbalance between load-balanced paths. Latency imbalance is the latency difference between LB paths, where the latency is the minimum path latency without inflation, while congestion imbalance occurs when LB paths experience difference levels of congestion. Latency and congestion imbalance affects application performance in different ways. Congestion imbalance in general has more significant impacts, because it affects all aspects of path performance, including latency inflation, packet loss and throughput drop.

1.2 Why is the Performance Dissimilarity Issue Important?

The performance dissimilarity issue mentioned above affects application performance at different levels and in different ways. In global load balancing, when clients in the same aggregation have different performance to a server, a redirection decision to the server will not work well for some clients. For these clients, another server may provide better service. It is similar that in path-level load balancing, latency and congestion imbalance indicates that one path has better performance than another.

Latency-Sensitive Applications Using a path or a server that leads to lower latency would improve the quality of experience (QoE) for applications sensitive to latency. In the era of 5G, the latency in cellular networks can be as low as 1ms and the latency bottleneck will be shifted to the Internet [10]. The problem of latency dissimilarity in the Internet load balancing will become more prominent for the end-to-end latency. Moreover, latency imbalance makes the existing methods insufficient for measuring the minimum path latency, because they only consider the latency inflation in single paths [6], not the difference

between paths. This affects geolocation applications or network coordinated systems relying on the accurate minimum path latency [7,8]. Also, latency imbalance, if measured separately on the forward or return path, is an indicator to path asymmetry, affecting applications relying on symmetric paths to accurately estimate one-way delay, e.g., clock synchronization via networks [9, 10].

Throughput-Sensitive Applications Throughput-sensitive applications, e.g., large file transfer and video streaming, could benefit similarly as the latency-sensitive applications do. When congestion imbalance occurs, the congested path could experience severe throughput drop. Choosing the uncongested path would greatly improve application performance. This also applies to global load balancing, where one server could provide higher throughput than the others.

Other Impacts Besides latency and throughput, applications sensitive to packet loss, e.g., Voice over IP (VoIP), can benefit similarly from considering performance dissimilarity in Internet load balancing. Moreover, latency imbalance has an impact on all applications relying on measuring latency to estimate distance, e.g., network coordinate systems used in peer-to-peer systems [11, 12]. Congestion imbalance indicates that multi-path transport protocol could be more useful in the Internet. Both latency and congestion imbalance would serve as an important metric for the measurement studies examining latency characteristics.

1.3 Challenges in Evaluating Performance Dissimilarity at Scale

Considering the scale of global and path-level load balancing, we want to evaluate the performance dissimilarity issue at the Internet scale. This requires a dataset describing the performance between clients and servers for global load balancing and a dataset describing the performance between LB paths for path-level load balancing. Both datasets will be of very large scale to cover Internet-scale clients and paths. These datasets can be collected ei-

ther passively without incurring measurement overheads or actively with network probers. We will use passive measurements for global load balancing and active measurements for path-level load balancing.

Passive Measurements for Global Load Balancing Considering that the daily traffic between clients and servers is enormous, we can passively collect performance data at the server side without incurring extra measurement overheads. Although the total volume of passive measurements is large, the average amount of samples for individual addresses is sparse and sporadic. The major challenge is how to increase data density by proper data aggregation and extract useful information from the aggregated data. Large-scale passive measurements from the server's perspective are readily available to content providers, but are mostly proprietary and not available to the public, making it hardly a common way of obtaining large-scale performance data. To tackle the data sparsity issue, our approach increases data density by clustering samples of similar addresses and obtains path performance with pattern recognition techniques.

Active Measurements for Path-Level Load Balancing Compared with passive measurements, we can collect active measurements at scale with network probing. Specifically, we send probes to random Internet addresses and use their responses to infer path performance. An experiment that scans the Internet-wide addresses can be done at a single vantage point, which makes network probing a popular way of data collection in network measurements. However, due to ICMP rate limiting, probes need to be sent at a low rate. Moreover, routers generally de-prioritize ICMP responses and generate them in the slow paths, which could cause the inferred path performance to be inaccurate. It is well-known that network probing can accurately measure path latency, but can only perform coarse estimation on throughput [13, 14]. Throughput is commonly measured from vantage points to instrumented servers in public measurement infrastructures, e.g., M-Lab. The major challenge in collecting active measurements is how to measure each target with as few samples

Problem scope	Project name
The sizing of client aggregations for global load balancing	AP-atoms: A High-Accuracy Data-Driven Client Aggregation for Global Load Balancing
Latency imbalance between LB paths	Latency Imbalance Among Internet Load-Balanced Paths: A Cloud-Centric View
Congestion imbalance between LB paths	Congi: Measuring Congestion Imbalance Among InternetLoad-Balanced Paths at Scale

Table 1.1: Summary of dissertation work

as possible such that it can be easily scaled to measuring Internet-wide addresses. In this dissertation, we will correlate latency inflation with throughput drop and packet loss, and use sampling to efficiently infer latency inflation.

1.4 My Thesis and Contributions

By studying the performance dissimilarity issue in global and path-level load balancing, this dissertation supports the following thesis: *large-scale latency measurements allow us not only to systematically identify and evaluate the performance issues of Internet load balancing at scale, resulting in a comprehensive understanding of its impacts on applications, but also to develop data-driven approaches to improve its performance.*

As summarized in Table 1.1, this dissertation includes three projects, where the first project uses large-scale latency measurements to study the sizing of client aggregations for global load balancing and the second and third projects focus on evaluating the latency and congestion imbalance among LB paths at scale with large-scale latency measurements respectively. This dissertation work makes the following contributions.

A Data-driven Client Aggregation for Global Load Balancing To the best of our knowledge, we are the first to conduct a comparative study of the performance of existing client aggregation methods. We find that even for the best existing aggregation, almost 17% of clients have latency 50 ms apart from other clients. By studying the causes for widely

dispersed clients, we find that the wide dispersal of client latencies in existing aggregations are caused by aggregating clients based on attributes other than path performance. To address this issue, we propose a data-driven aggregation, AP-atoms, that can flexibly trade off scalablity for accuracy. Our method relies on the passive measurements of existing traffic between service providers and clients and thus incurs no extra measurement overheads. The data-driven property enables our mechanism to dynamically adapt to changing network conditions.

A Cloud-Centric View of Latency Imbalance Among LB Paths We develop a methodology of efficiently measuring latency imbalance at the Internet scale. We carefully discuss the tradeoff between the accuracy and efficiency of our method and verify the accuracy of measured imbalance. We present a global view of latency imbalance from a cloud-centric view, where latency imbalance is found to be both significant and prevalent. We use path analysis to explain the difference between latency imbalance seen from different cloud providers and show that latency imbalance is stable over time. We evaluate the impact of latency imbalance on three applications and propose potential solutions to improve their performance. We make our tool and data publicly available at [15].

Measuring Congestion Imbalance Among LB Paths at Scale We present Congi, a network prober that uses SVM classifiers to detect congestion imbalance at scale. Congi is developed in a systematic way and is verified to be capable of detecting throughput, latency and packet loss imbalance between LB paths. We use Congi to detect congestion imbalance from our VPs to Internet-wide addresses and find that congestion imbalance is prevalent in the Internet. We use web page load as an example and further evaluate how congestion imbalance affects application performance.

1.5 Thesis Organization

The dissertation is organized as follows. Chapter II presents the data-driven client aggregation, AP-atoms, for global load balancing. Chapter III characterizes congestion imbalance among LB paths and its impacts on applications. Chapter IV presents Congi, our network prober to measure congestion imbalance at scale. Chapter V summarizes the related work. We conclude the dissertation in Chapter VI.

CHAPTER II

AP-Atoms: A Data-Driven Client Aggregation for Global Load Balancing

In this chapter, we focus on evaluating and improving the existing client aggregations for global load balancing with large-scale latency measurements. We first conduct a comparative study of the existing client aggregations and find out the causes for clients in the same aggregation to have different performance to the same server. Inspired by the insights from the cause analysis, we propose a data-driven client aggregation, called *AP-atoms*. AP-atoms are learned from the latency data passively collected at the server side and thus incur no extra measurement overheads. As AP-atoms are generated from data continuously collected over time, it adapts to changing network conditions. Further, AP-atoms can flexibly trade off between accuracy and scalability based on the needs.

2.1 Introduction

In Internet mapping, IP address space is divided into a set of client aggregation units, which are the finest-grained units for global load balancing. Choosing the proper level of aggregation is to find a good tradeoff between scalability and accuracy. Using /24 IP blocks as client aggregations may be accurate in redirection, but not scalable in terms of performance estimation. In contrast, aggregating clients by their LDNSs is scalable, but not

accurate for remote clients. IP blocks with /20 network prefix have been proposed to be a good tradeoff between scalability and accuracy [5]. Clients can also be aggregated by their geographic locations [16] and BGP routing paths [17]. We use Internet-wide measurements provided by a commercial global load balancing service provider to study the performance of existing client aggregations.

We find that even for the best existing aggregation, almost 17% of clients have latency 50 ms apart from other clients. By studying the causes for widely dispersed clients, we find that the wide dispersal of client latencies in existing aggregations are caused by aggregating clients based on attributes rather than path performance. To address this, we propose AP-atoms that are data-driven and group clients based directly on their path performance (e.g., latency) to servers.[1] Since the path performance between clients and servers can be passively measured from user traffic available to service providers [19], AP-atoms can be obtained without incurring extra measurement overheads. To obtain AP-atoms, we use machine learning algorithms to identify distinct latency patterns and cluster clients based on these patterns. Besides the next-generation mapping system, AP-atoms also have potential usages in other emerging applications, e.g., consumer cloud storage [20], that could benefit from the scalable and accurate estimation of path performance between clients and servers.

2.2 Evaluating Existing Client Aggregations

To motivate this study, we first compare the performance of existing client aggregations. To our best knowledge, these existing aggregation methods have been studied separately [16, 5, 7, 21], but there has been no comparative study of the four methods. We use Internet-wide measurements from a commercial global load balancing service provider for the comparative study.

[1] In the literature, performance measurements are commonly used to estimate performance between servers and pre-determined client aggregations for better load balancing [18, 4], but not used to obtain client aggregations.

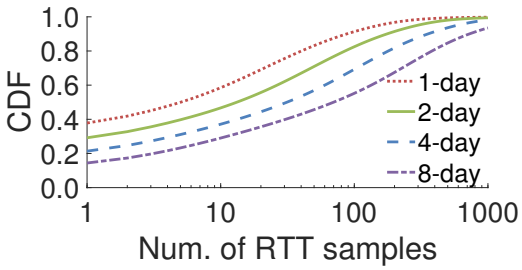


Figure 2.1: Data density

2.2.1 Data Description and Methodology

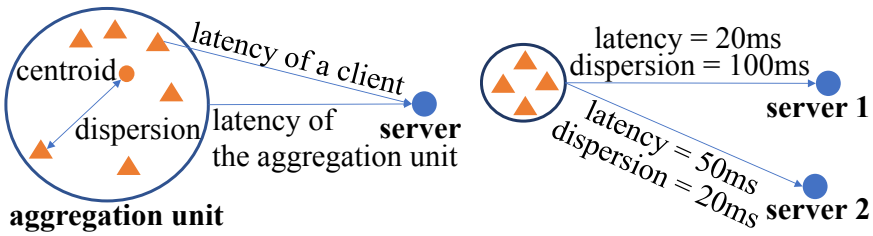
Our dataset is provided by a commercial global load balancing service provider, collecting over 1 billion measurements per day as part of their normal operations. Each time when a client browses an instrumented web hosted by a participating provider, a Javascript script is downloaded and executed to download test objects. The test objects are small enough to fit into a single TCP packet. The time to first byte is recorded as the latency to the participating provider. This project is called Radar and is described in detail in [22]. Measurements are conducted when the browser is idle such that user experience is not affected. Since each visit to the web pages will trigger only a few measurements, the measurements from single clients are sparse over time. Our dataset includes measurements from both wireless (Wi-Fi only) and wired networks, where the percentage of measurements from wireless networks is 12.8%. Since measurements in our dataset are triggered by user activities, our dataset resembles passive measurements that global load balancers would obtain from user traffic. The anonymization of addresses in our dataset does not limit the use of our dataset as explained in §2.3.1.1.

Our dataset includes 30 days of measurements. Each day's data contains about 1.5 billion measurement records, occupying approximately 550 GB. Clients in our dataset cover 86% of countries in the world and are from 4.2 million /24s. Path latencies are measured between these clients and 160 sites of CDN and cloud service providers, including Akamai, Amazon, Google, Level3 and Microsoft Azure, and others. Our dataset only includes measurements on network performance (e.g., RTT and throughput) and does not contain

any sensitive information about clients (e.g., traffic content and passwords). Further, individual addresses (32-bit addresses) are anonymized with the last byte of addresses masked for privacy. In other words, addresses in our dataset are represented as /24 IP blocks. However, most providers do not have enough traffic with clients for analysis purposes. We only use the top 10 servers having the most traffic with clients in our experiments. Figure 2.1 shows the distribution of the number of RTT samples between clients and servers in period lengths of 1 to 8 days. The measurements of individual addresses in the same /24 block are aggregated and are considered from the same single client. In 1-day periods, 38% of clients only have one sample. As the period length increases, the number of RTT samples from clients increases. Due to the *sparsity* of our dataset, we use 2-day periods for latency pattern identification in our experiments. We also include only clients with at least 5 measurements in a 2-day period.

The general principle in client aggregation is to pool together clients that are similar. Existing aggregations define client similarity mainly based on one of four attributes: (1) geographic locations, (2) fix-sized prefixes, (3) BGP routing paths, or (4) LDNSs. Using the first attribute, clients within a given geographic radius are gathered together into *geo-blocks* [16]. Aggregation by the second attribute simply groups clients by the first k bits of their IP addresses. Researchers studying Akamai's end-user mapping suggested */20 prefixes* as a good tradeoff between scalability and accuracy [5]. Aggregation by the third attribute exploits shared BGP routing paths amongst clients. Since clients within the same routable prefix share a portion of their BGP paths, it is reasonable to aggregate clients into routable *BGP prefixes*. Aggregation by the fourth attribute is commonly used by CDNs, where clients using the same LDNS are aggregated into the same aggregation unit [5, 4].

Using our dataset, we simulate the use of aggregation methods above in real systems. In Internet routing, routers choose next hops using the longest prefix match. Packets sent from a server to a client are routed to the prefix that shares the longest common bits with the client's IP address. The latency along the routes is the latency between the server and



(a) Latency dispersion of clients (b) Impact of dispersion

Figure 2.2: Illustration of latency dispersion

the client. We thus use the latency measurements between clients and servers in our dataset to simulate the latency along the routes in Internet routing. To compare the performance of the four client aggregation methods, we present a metric referred to as *latency dispersion*, which measures the differences in latency between clients in the same aggregation unit. We study the performance of existing client aggregations in terms of latency. The same concept of dispersion applies to other performane metrics such as bandwidth and packet loss.

2.2.2 Latency Dispersion

As shown in Figure 2.2(a), an aggregation unit is an aggregation of clients. Given a server, we can have two types of latencies: 1) the latency from *each client* in the aggregation unit to the server and 2) the latency from the aggregation unit *as a whole* to the server[2]. For scalable management of the Internet, client redirections are determined by the latencies of aggregation units to servers [4, 5]. To understand the effects of using latency of aggregation units to perform redirection for all clients inside the unit, we define two metrics for both clients and aggregation units as follows.

The *dispersion of a client to a server* is the difference between the latency of the client to the server and the centroid of the aggregation unit to the server, where the centroid is the average latency of clients in the aggregation unit. The dispersion of clients tells us

[2]In [4], one client in the aggregation unit is selected randomly and its latency is considered as representative of the unit.

how many clients are far away from other clients in the same aggregation unit in terms of latency. The *dispersion of an aggregation unit to a server* is the largest dispersion among all clients in the aggregation unit, which tells us the worst performance of existing aggregation units. We use the largest dispersion instead of a percentile-based one because the sizes of aggregation units range from a few to thousands of clients and the percentile-based dispersion is biased against small ones, where each client takes a significant portion of its aggregation unit.[3] More importantly, using largest dispersion gives us a worst-case performance for aggregation units. We use largest dispersion only in comparing the performance of aggregation methods, not part of aggregation methodology.

Figure 2.2(b) shows an example on the impact of dispersion on server selection. Based on latencies, server 1 is a better choice than server 2 for the aggregation unit. However, since the dispersion of the aggregation unit to server 1 is 100 ms, directing all clients in the unit to server 1 will cause some clients to experience 50 ms larger latency than if they had been redirected to server 2. We want to use latency dispersion as a metric to indicate the quality of aggregation units. Small latency dispersion enables an accurate estimation of latency to aggregation units and thus an accurate server redirection. Since clients in an aggregation unit could be redirected to mutliple servers, we prefer that the dispersion of the aggregation unit to all possible servers be small. Although latency dispersion may give us false positive results[4] in some cases, these cases cover at most 0.8% of clients for all aggregation methods. In the following, we first show the latency dispersion of existing client aggregations to a single server and the maximum latency dispersion to all servers.

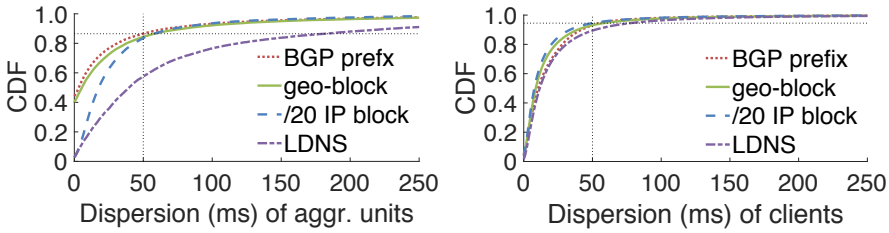


Figure 2.3: Latency dispersion to a single server

2.2.2.1 Latency Dispersion to a Single Server

We use the database of routable BGP prefixes from CAIDA [23] for aggregation by BGP routing paths and the IP-to-location database from IP2Location [24] for aggregation by geographic locations. We use $/20$ IP blocks for fix-sized prefixes and aggregate clients by their LDNSs. Addresses in the IP-to-location database are represented as IP blocks and each IP block has an estimated location. For comparison with other aggregations, we aggregate IP blocks into geo-blocks that have a distribution of prefix size similar to BGP prefixes, where IP blocks in the same geo-block have locations in a circle with a radius less than 200 miles. We compare the latency dispersion of the four aggregation methods above on two granularities: aggregation units and clients. Since the last byte of addresses in our dataset is anonymized for privacy reasons, *each client in our dataset is a /24 IP block*, which is actually an aggregation of individual addresses (32-bit).

To calculate latency dispersion, we use the latency measurements from all clients to one server.[5] Among all clients, we have obtained 1.7 million clients that have stable latencies (i.e., the single-mode latency in Figure 2.6(a)) to the server. For comparison among aggre-

[3]For large aggregation units, we could have a small portion of clients far away from others, but we do not consider them as outliers, in contrast, representatives of a small group of clients that are aggregated improperly. The reasons for this are that 1) the clients we can use to calculate dispersion for large aggregation units only takes a small portion of all clients in the units, which implies that the actual dispersion could be larger, and that 2) the latencies of clients are accurately identified as shown in Section 2.4.3.

[4]False positive results occur when clients in an aggregation unit use the same gateway (not middleboxes, see Section 2.5.0.1) to the rest of the Internet and have different path performance. Such clients would have large dispersion (larger than 50 ms) to all servers.

[5]Using latency measurements to other servers gives us similar results, not presented here. We do not average latency dispersion of clients over servers because each client has communications with a different set of servers.

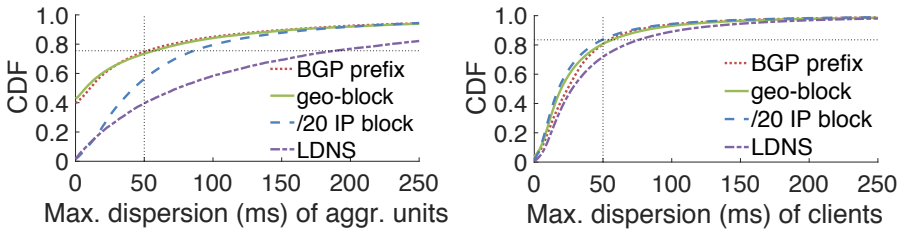


Figure 2.4: Maximum latency dispersion to multiple servers

gation methods, we group these clients to aggregation units for each aggregation method respectively. For LDNS, clients using the same LDNS are in the same aggregation unit. For BGP prefixes, /20 IP blocks and geo-blocks that are represented as prefixes, we use the longest prefix match to group clients. Each client is mapped to the aggregation unit having the longest common prefix with the client's address. Then, we calculate the dispersion of clients and aggregation units. Figure 2.3(a) shows the cumulative distribution function (CDF) of latency dispersion of *aggregation units* to the server. BGP prefixes in general have smaller dispersion than other aggregations, but still have a significant percentage (about 14%) of aggregation units with dispersion larger than 50 ms. Following the study of Google's CDN [4], we use 50 ms as a threshold to indicate a significant difference in latency. Looking at the distribution of latency dispersion of *clients* in Figure 2.3(b), /20 IP blocks have smaller dispersion than other aggregations. About 6% of clients in /20 IP blocks have dispersion larger than 50 ms. Moreover, the distribution of dispersion of clients has a long tail, where the 99-th percentile dispersion is about 150 ms for all aggregations. In Figure 2.3(a), BGP prefixes and geo-blocks include a portion of aggregation units with zero dispersion. This is due to half of aggregation units from BGP prefixes and geo-blocks being of size /24, comprising of only one client each.

2.2.2.2 Latency Dispersion to Multiple Servers

In global load balancing, since clients of an aggregation unit could be directed to one of a set of candidate servers [25], the worst-case performance of clients is determined by

the server to which the aggregation unit has the maximum dispersion among all candidate servers. Thus, we want to know the maximum dispersion of aggregation units and of clients to multiple servers. To calculate the maximum dispersion, we use latencies from clients to all 10 servers. Among all clients, we have obtained 2.3 million clients having stable latencies to at least one server. The dispersion of clients to each server is first calculated. If a client has no measured latency to a server, the dispersion of the client to the server is considered undetermined. Each client then will have dispersions calculated for up to 10 servers. Among these dispersions, the maximum one is used as the maximum dispersion of clients. Recall how we obtained the dispersion of aggregation units to each server from the dispersion of clients. Similarly, we can obtain the maximum dispersion of aggregation units. Figure 2.4(a) shows the distribution of the maximum dispersion of aggregation units to the 10 servers. BGP prefixes and geo-blocks have smaller dispersion than /20 IP blocks and LDNS[6], but almost 26% of BGP prefixes and geo-blocks have dispersion larger than 50 ms to at least one server. Figure 2.4(b) shows that about 17% of clients in /20 IP blocks and geo-blocks have dispersion larger than 50 ms to at least one server.

Comparing Figures 2.4(a) and 2.4(b), we can see that even though BGP prefixes and /20 IP blocks have similar latency dispersion of clients, BGP prefixes have 20% more aggregation units with dispersion less than 50 ms than /20 IP blocks do. This is because BGP prefixes and geo-blocks have 40% of aggregation units with zero dispersion, where 98% of these aggregation units are /24s. However, compared to /20 IP blocks, BGP prefixes and geo-blocks both have very large aggregation units, which could easily have large dispersion if clients included are not similar. Indeed, among the aggregation units with dispersion larger than 250 ms in BGP prefixes, 73% are of size at least $/19$ while only 7% are of size $/22$ or smaller. Similarly, among aggregation units with dispersion larger than 250 ms in geo-blocks, 80% are of size at least $/19$ and only 4% are of size $/22$ or smaller. As the numbers show, aggregating clients by attributes could result in widely dispersed clients,

[6]Since geo-blocks and BGP prefixes are similar in the distribution of prefix sizes, they are close in dispersion in Figure 2.3 and 2.4.

regardless of aggregation unit sizes.

2.2.3 Causes for Large Dispersion

An aggregation unit has large dispersion when a small portion of clients (*minorities*) have latencies significantly different from other clients in the same aggregation unit,[7] or when the aggregation unit is overlarge and includes clients that should be divided into smaller aggregation units (*over-aggregation*).

2.2.3.1 Identifying Minorities and Over-aggregation

To study the two causes, we look at the aggregation units with dispersion larger than 50 ms and determine the minimum set of clients that must be pruned to obtain dispersion less than 50 ms. We use a greedy algorithm to minimize the number of clients in the pruned set. The greedy algorithm starts from all the clients in the aggregation unit and first prunes the client that has latency furthest from the average latency of clients. Then, the process is repeated with the rest of the clients until the dispersion of the aggregation unit is less than 50 ms. Then, we calculate the ratio of the number of clients in the pruned set to the total number of clients (*pruned ratio*).

Figure 2.5 shows the distribution of pruned ratios for the four aggregation methods. We consider an aggregation unit as containing *minorities* if they have a pruned ratio 0.1 or less. Since a /20 IP block includes at most 16 /24 clients, almost no /20 IP blocks have a pruned ratio less than 0.1. Of the other aggregation methods, only 25% to 31% of their aggregation units have a pruned ratio less than 0.1. Since small aggregation units include a fewer number of clients, they are less likely to have a small pruned ratio. Among aggregation units with a pruned ratio less than 0.1, over 84% of them are of size at least /18 for BGP prefixes and geo-blocks. If we now consider aggregation units with a pruned ratio larger

[7] We have verified that atypical latencies are not the reason to large dispersion. Among aggregation units with large dispersion, less than 2% of them include minorities that could experience *atypical* latencies (20 ms larger than latencies in neighboring periods) such as caused by network congestion.

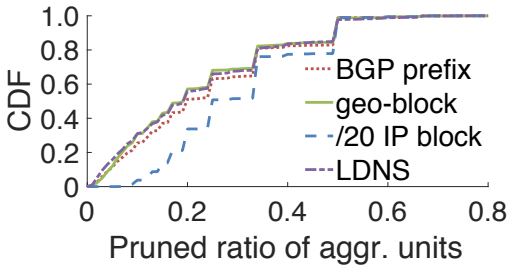


Figure 2.5: Distribution of the pruned ratio

than 0.2 to be due to *over-aggregation*, over 40% of aggregation units have large dispersion due to over-aggregation for all aggregation methods. Since each client takes a significant portion in small aggregation units, small aggregation units with large dispersion are more likely due to over-aggregation. Among aggregation units of size /18 or larger, 10% of geo-blocks, 9% of BGP prefixes and 4% of aggregations by LDNSs with large dispersion are due to over-aggregation.

2.2.3.2 Inflexibility of Existing Aggregations

Existing aggregation methods partition Internet address space by clustering clients similar in certain attributes. The attributes do not necessarily reflect path performance of clients. Moreover, given an attribute by which clients are aggregated, partitioning of the address space is fixed, not adaptive to changing network conditions. This inflexibility of partitioning causes the minorities and over-aggregation problems. In this section, we analyze the relationship between inflexible address space partitioning due to the use of arbitrary, non-performance related attributes and the presence of minorities and over-aggregation.

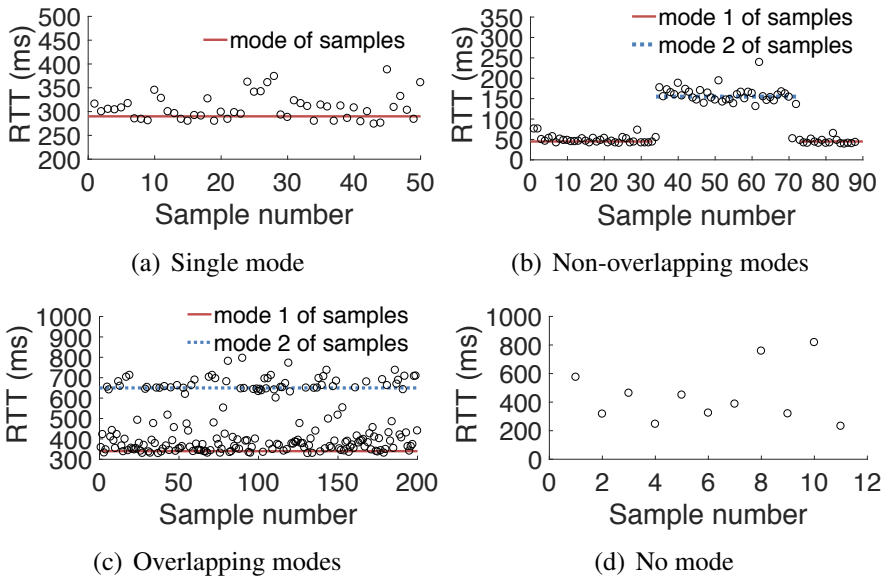
Minorities and over-aggregation are due to clients' having large distances in their latencies (*latency distance*). We now study how latency distance relates to the distance in IP addresses (*address distance*), i.e., how clients far away in latency can be separated by splitting address space. We use the same definition of address distance used by Lee and Spring [26]. For each client in the pruned set, we choose the top 10% furthest clients in latency and refer to them as *distant clients*. Then, we calculate two address distances: 1)

from the client in the pruned set to the distant clients and 2) between distant clients. For the first measure, we calculate the address distance between the client and each of the distant clients, and use the average as the address distance. For the second measure, we calculate the address distance between each pair of the distant clients and use the average as the address distance.

For existing aggregation methods, we find that about 37% to 47% of minorities are closer to the distant clients in address space than the distant clients are among themselves. Thus to separate them from the distant clients would require segregating them into their own, small aggregation units, e.g., /23s. For the remaining 53% to 63% of minorities, since they are further apart from the distant clients than the distant clients are apart amongst themselves, we could further divide the address space. In the case of over-aggregation, only 7% of clients are closer to the distant clients than the distant clients are apart amongst themselves, in address space. This means that for most clients in the case of over-aggregation, being far apart in latency to distant clients implies being far apart in address space. Thus, these clients should be easily separated from the distant clients by further dividing the address space. In the following, we introduce AP-atoms which takes advantage of the above observations and allow for flexible address space boundaries that adapt dynamically to network condition.

2.3 AP-Atoms

In contrast to existing aggregation methods, AP-atoms are data-driven aggregation. It aggregates clients based on their latencies. Each AP-atom includes clients with similar latencies. The similarity in latencies, which determines the dispersion of clients, can be controlled depending on the requirements on the accuracy of client redirection. Since AP-atoms aggregate clients based on their latencies and the latencies of clients change with *network events*, e.g., route changes, AP-atoms dynamically adapt to changing network conditions, resulting in high-accuracy aggregation. Further, AP-atoms are *server-independent*:



(a) Single mode (b) Non-overlapping modes

(c) Overlapping modes (d) No mode

Figure 2.6: Four patterns of latencies

the dispersion of aggregation units and of clients is small whichever server they are redirected to. To achieve server-independence, we find a set of aggregation units that have small dispersion from the view of each single server, referred to as *AP-atom candidates*. Then, we merge the view of each server (i.e., AP-atom candidates) to obtain AP-atoms.

2.3.1 AP-Atom Candidates: Single-Server View

AP-atom candidates are generated from clusters of clients with similar patterns in their latency measurements. Before discussing how to cluster clients based on their latency patterns, we first present latency patterns and the identification of latency patterns. We start with the latencies between a server and its clients, which can be obtained, for example, from the TCP round-trip time estimates of the clients' connections at the server. Client latencies, observed over time, change with network events and thus have different patterns.

2.3.1.1 Latency Patterns

Before introducing patterns in latency measurements, we first discuss how to determine latency. When an individual address (a 32-bit IP address) has a large number of RTTs

available [4, 11], the minimum or median RTT is generally used as latency. However, when this method is used for passive measurements, it faces two problems: 1) a significant portion of individual addresses could have an insufficient number of measurements in passive measurements,[8] and 2) the measurements of individual addresses could be inflated due to network congestions, not representative for typical latencies. To solve these problems, we aggregate RTTs and use the collective behaviors in the aggregated RTTs to determine latency. As the finest address prefix granularity in our dataset is /24 for privacy purposes, we aggregate individual addresses in the same /24 IP block. As recent work has shown, it is possible for /24 IP blocks to include individual addresses that are dissimilar in latencies [27]. We use the modes of RTTs as latency, where the modes of RTTs are the prevalent RTTs (see Section 2.3.1.2 for more details). Since RTTs can have multiple modes, using modes instead of the minimum or median latency helps us distinguish dissimilar addresses within the same /24 IP block.

Figure 2.6 shows four patterns of RTTs taken from a client (i.e., a /24 IP block). When individual addresses in the same /24 block are similar in latencies, their combined RTTs either have a single mode as in Figure 2.6(a) or multiple modes that do not overlap in time as in Figure 2.6(b). A *single mode* occurs when RTTs are measured in a stable period of the network, where the network could be either uncongested or persistently congested. *Multiple non-overlapping modes* occur when RTTs are measured in a period including network condition changes, e.g., congestion or route changes. As individual addresses are similar in latency, their RTTs change with the changing network conditions, resulting in non-overlapping multiple modes. Figure 2.6(c) shows a pattern including multiple modes that overlap in time, which we refer to as *overlapping modes*. Due to insufficient or noisy measurements, the modes of RTTs could be unidentifiable, referred to as *unidentifiable modes*. Figure 2.6(d) shows an example of noisy RTTs, where RTTs take a large range of

[8] An individual address could easily have sparse data in passive measurements due to 1) the individual address may have infrequent or no communications with the servers of interest and 2) latency can be measured passively only at limited stages of communications, e.g., during TCP's three-way handshake [19].

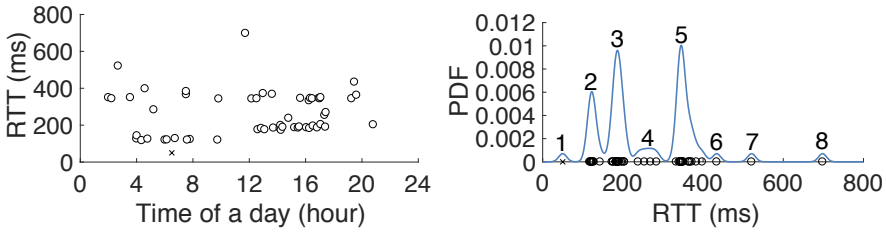
values and thus no mode can be identified.

2.3.1.2 Identifying Latency Patterns

We identify latency patterns using two machine learning algorithms, i.e., mean shift clustering (MSC) [28] and total variation denoising (TVD) [29]. MSC is used to cluster RTT samples, where samples in the same cluster (called *MSC cluster*) are close in values. TVD is used to determine the relations between MSC clusters. In Figure 2.7(a), we use an example to show how to obtain MSC clusters and the relations between MSC clusters. The example shows one-day RTT trace from a /24 block to a given server. At time 6, an erroneous sample, much less than other samples, is marked as a cross. Before and after time 12, there is a change of the minimum RTTs, which could be due to a network event. After applying MSC on the RTT samples, we obtain the MSC clusters in Figure 2.7(b). The erroneous sample is isolated in MSC cluster 1. Each MSC cluster has a peak and the RTT corresponding to the peak is the *mode* of the cluster. MSC clusters 2, 3 and 5 have a much higher peak than other clusters, indicating that they include a much larger number of samples.

To distinguish between MSC clusters (denoted as x and y), we identify three relations: 1) x is *noise* to y, i.e., x has a much less number of RTTs than y, and RTTs in x and y are interleaved in time, e.g., MSC clusters 1 and 5; 2) x and y are *non-overlapping*, i.e., x and y are not noise to any other clusters and their RTTs almost do not overlap in time, e.g., MSC clusters 2 and 3; 3) x and y are *overlapping*, i.e., x and y are not noise to any other clusters and their RTTs are interleaved in time, e.g., MSC clusters 2 and 5. Only the modes of MSC clusters that are not noise to any other clusters are considered stable and used to calculate latency dispersion in experiments. In the following, we use TVD to quantitatively determine the relations between MSC clusters.

The most well-known application of TVD is noise removal. Given a noisy signal as input, TVD recovers the original signal by smoothing out the noise. If the noisy signal is



(a) RTT samples (b) MSC clusters

Figure 2.7: An example of applying MSC to obtain MSC clusters

denoted as $\boldsymbol{r} = (r_1, \ldots, r_m)$, where r_i is the i-th element, the objective of TVD is to find a sequence $\boldsymbol{z} = (z_1, \ldots, z_m)$, an approximation to the original signal by minimizing the following cost function,

$$\arg\min_{\boldsymbol{z}} \sum_{i=1}^{m} |r_i - z_i|^2 + \lambda \sum_{i=2}^{m} |z_i - z_{i-1}| ,$$

where λ is a tuning parameter penalizing the change to z_i. Given two MSC clusters x and y, we first set all RTTs in each MSC cluster to be the mode of that cluster and then merge the RTTs in both clusters in the order of their measured time. Using the merged RTTs as input $\boldsymbol{r}$ to TVD, we obtain the output $\boldsymbol{z}$ as previously described. Setting all RTTs in each cluster to the same value before applying TVD emphasizes the impact of one cluster on the other. If r_i is in x, $|z_i - r_i|$ is then the impact of RTTs in y on r_i after TVD. Let us denote the change of r_i, i.e., $|z_i - r_i|$, as Δ_i and use a threshold Δ_{th} to determine if the change is significant.

After TVD, we calculate the percentages of RTTs having significant changes in x and y, denoted as P_x and P_y respectively. Let h be a threshold to determine if most RTTs have significant changes. The relations between x and y are summarized as follows: 1) If $P_x \leq 1 - h$ and $P_y \geq h$, most RTTs in y have significant changes, while most RTTs in x do not have significant changes, i.e., y is noise to x. Similarly, if $P_y \leq 1 - h$ and $P_x \geq h$, x is noise to y; 2) If $P_x \leq 1 - h$ and $P_y \leq 1 - h$, RTTs in x and y have small impact on each other, i.e., x and y are non-overlapping; 3) Otherwise, overlapping. It should be noted that x and y in the last two relations cannot be noise to any other clusters. Using the

pairwise relations between MSC clusters, we can identify latency patterns as follows. If two MSC clusters are overlapping, the latency pattern has overlapping modes. If two MSC clusters are non-overlapping and other clusters are noise to the non-overlapping clusters, the latency pattern has non-overlapping modes. If all other clusters are noise to one cluster, the latency pattern has a single mode.

In the algorithms above, we have three parameters λ, Δ_{th} and h. To determine λ and Δ_{th}, we establish a relation between them. Since RTTs in x and y are set to the mode of each cluster respectively before merging, the merged RTTs are comprised of alternating segments, where each segment includes RTTs of the same value and consecutive segments include RTTs of different values. We represent the merged RTTs as $\boldsymbol{r} = (\boldsymbol{s}_1, \ldots, \boldsymbol{s}_k)$, where $\boldsymbol{s}_i$ is the i-th segment, and the number of RTTs in $\boldsymbol{s_i}$ as $N(\boldsymbol{s_i})$. The relation between λ and $N(\boldsymbol{s_i})$ is as follows.

Theorem 1. *Let d_x and d_y be the modes of x and y, and the merged RTTs have k segments. For $\lambda \in (0, |d_x - d_y|/2)$, we have that $\Delta_i = \frac{\lambda}{2N(\boldsymbol{s}_j)}$ if r_i is in $\boldsymbol{s}_j$ for $j \in \{1, k\}$ and that $\Delta_i = \frac{\lambda}{N(\boldsymbol{s}_j)}$ if r_i is in $\boldsymbol{s}_j$ for $j \in \{2, \ldots, k-1\}$.*

Proof. Without loss of generality, we assume $d_x < d_y$. When the optimal solution is reached, for any $i \in \{1, \ldots, m\}$, we must have $d_x \leq z_i \leq d_y$. For z_i's less than d_x or larger than d_y, we can always get a smaller cost by setting these z_i's to d_x or d_y whichever is closer in distance. Similarly, when the optimal solution is reached, for r_i's in the same segment, we must not have oscillating Δ_i's, e.g., $\Delta_i > \Delta_{i-1}$ and $\Delta_i > \Delta_{i+1}$. For such Δ_i's, a smaller cost can be achieved by setting $\Delta_i = (\Delta_{i-1} + \Delta_{i+1})/2$. Thus, for all r_i's in the same segment, we have a monotonic sequence of Δ_i's. Without loss of generality, assuming that the sequence of Δ_i's in the same segment is non-decreasing, we have $|z_i - z_{i-1}| = |\Delta_i - \Delta_{i-1}| = \Delta_i - \Delta_{i-1}$. By denoting $d_y - d_x$ as Δd and $z_i - z_{i-1}$ as Δz_i, we can rewrite the cost function as

$$f = \sum_{i=1}^{m} \Delta_i^2 + \lambda \left(\Delta d - \Delta_1 - \Delta_{N(\boldsymbol{s}_1)+1} + \sum_{i=N(\boldsymbol{s}_1)+2}^{m} |\Delta z_i| \right).$$

When f is minimized, we have that for each r_i where $i \in \{2, \ldots, N(s_1)\}$, $\partial f / \partial \Delta_i = 2\Delta_i = 0$. However, since the sequence of Δ_i is non-decreasing, we have that f is minimized when $\Delta_i = \Delta_1$ for $i \in \{2, \ldots, N(s_1)\}$. This implies that the cost is minimized when all r_i's in the same segment have the same change. Let Δ_{s_i} be the change of RTTs on the i-th segment. We have

$$f = \sum_{i=1}^{k} N(\boldsymbol{s}_i)\Delta_{\boldsymbol{s}_i}^2 + \lambda \left((k-1)\Delta d - \sum_{i=1}^{k} \Delta_{\boldsymbol{s}_i} - \sum_{i=2}^{k-1} \Delta_{\boldsymbol{s}_i} \right).$$

Taking the derivatives of f with respect to Δ_{s_i}, we have that the optimal solution is achieved when $\Delta_{\boldsymbol{s}_i} = \frac{\lambda}{2N(s_i)}$ for $i \in \{1, k\}$ and $\Delta_{\boldsymbol{s}_i} = \frac{\lambda}{N(\boldsymbol{s}_i)}$ for $i \in \{2, \ldots, k-1\}$. For the similar reason as Δ_i's, we must not have oscilating $\Delta_{\boldsymbol{s}_i}$'s across segements, i.e., $\frac{\lambda}{N(\boldsymbol{s}_i)} + \frac{\lambda}{N(\boldsymbol{s}_j)} \leq \Delta d$ is true for any $N(\boldsymbol{s}_j)$ and $N(\boldsymbol{s}_j)$ when $i \neq j$. We thus have $\lambda \leq \Delta d/2$. □

Theorem 1 tells us that the changes to RTTs depend on the number of RTTs in their segment. We set a threshold N_{th} to determine if the number of RTTs in a segment is significant. We refer segments with at least N_{th} RTTs as *significant segments*. Since we expect RTTs in significant segments have insignificant changes after TVD, we set the threshold Δ_{th} equal to λ / N_{th}. The relation between λ and Δ_{th} holds for λ between 0 and $|d_x - d_y|/2$. We thus can set λ to any value in the range. The parameters h and N_{th} both determine the percentages of different latency patterns. Higher h and N_{th} result in a larger percentage of latency patterns to be identified as having overlapping modes. As will be discussed in later sections, to guarantee the accuracy of AP-atoms, we prefer relatively large h and N_{th}. We use $h = 0.8$ and $N_{th} = 5$ in our later experiments, where the setting of N_{th} also considers the data density in our dataset as discussed in §2.2.1. To avoid high-variation RTTs that are greatly inflated, we require the largest MSC cluster include at least N_{th} RTTs.

2.3.1.3 Clustering Clients Based on Latency Patterns

After having the latency patterns of clients, we want to cluster clients such that clients in the same cluster have similar latency patterns. The latency patterns of clients are identified for a given time period and can change over different time periods, depending on the network events in the periods. To ensure that clients that experienced the same network event, i.e., having the same latency pattern, can be found, we cluster clients based on latency patterns identified for the same time period. In the next section, we will discuss how to consolidate latency patterns over different time periods.

Once we have identified latency patterns, we divide clients having the same latency pattern into the same group and cluster clients in each group seperately. For clients with overlapping modes, since they already include individual addresses with dissimilar modes, we consider that each of them is a cluster by itself. Due to the data granularity limitation of our dataset, even if we find that addresses in some /24 clients have a distance between latencies greater than the predefined threshold, we cannot further split these clients into smaller clusters. Nonetheless, this is a limitation of our dataset, not our method. Our method can be used to split /24 clients if the measurements of individual addresses in the /24 block are available. For clients with non-overlapping modes, we determine if two clients should be in the same cluster by merging the RTTs of the two clients and then checking the latency pattern of the merged RTTs. If the merged RTTs continue to have non-overlapping modes, the two clients shoud be in the same cluster; otherwise, they are in different clusters. Two clients with non-overlapping modes are in the same cluster only if the modes of latencies of the two clients are similar both in values and in time duration. The degree of similarity is determined by the TVD algorithm discussed above. To cluster clients with a single mode, we use the quality threshold (QT) clustering algorithm [30]. Since all clients only have a single mode, each of them is associated with the value of the mode and the QT algorithm clusters clients based on the values of their modes. The QT algorithm takes the pre-defined threshold as a parameter for clustering. After clustering,

clients in the same cluster have difference in modes less than the pre-defined threshold.

Since the algorithms to identify latency patterns and the QT algorithm are of superlinear complexity [28, 31], clustering all clients together is of high complexity. We thus divide the Internet address space into prefixes of proper size, which we refer to as the *top prefix*, and cluster clients using the QT algorithm only within each top prefix, not across them. For the sizing of top prefixes, there is a tradeoff between scalability and computational complexity. As will be discussed in Section 2.3.1.5, the number of top prefixes affects the number of AP-atoms. Considering both scalability and computional complexity, we use /16s as top prefixes and need 45K /16s to cover the entire Internet address space of the 592K routable BGP prefixes provided by CAIDA.

2.3.1.4 Evolution of Client Clusters

Given a server, we can use latency patterns of clients in a single period to obtain the clusters of clients in that period (*single-period clusters*). Due to changing network conditions and the emergence of new clients, single-period clusters can only reflect the similarity of clients in one period. For each top prefix, we want a set of client clusters that dynamically adapts to the changing network conditions and cumulatively accommodates new clients. Further, since the latency patterns of clients could be misidentified in any one time period, we want to be able to correct the misidentification over time. We refer to such client clusters as *cumulative clusters*. As cumulative clusters use latencies of clients from multiple (both past and current) periods, cumulative clusters include more clients than single-period clusters and more accurate latency patterns of clients.

Suppose we want to obtain the cumulative clusters in the $(n+1)$-st period. We first consolidate clients in the cumulative clusters in the n-th period together with clients in the single-period clusters in the $(n+1)$-st period, and then correct the latency patterns of clients that are misidentified. The cumulative clusters in the n-th period are obtained from the latency patterns of clients from the first to the n-th periods. When n is equal to

1, the cumulative clusters are the single-period clusters in the first period. Depending on whether a client has identifiable latency patterns on the n-th and $(n+1)$-st time periods, we have three cases (not including the case when the client has no identifiable latency patterns on both periods). For each case, we must treat clients differently based on their latency patterns.

In the first two cases, a client has an identifiable latency pattern in the $(n+1)$-st period. We cluster the client using the latency pattern and record the latency pattern for correction later. In the third case, a client has no identifiable latency pattern in the $(n+1)$-st period. We want to use the client's latency pattern in the n-th period to infer the one in the $(n+1)$-th period for the consolidation. More specifically, in the first two cases, if the client has overlapping and non-overlapping modes, the client is in the single-period cluster of the client in the $(n+1)$-st period, where single-period clusters are obtained by clustering clients based on their latency patterns in that period. If the client has a single mode, we calculate the moving average of the mode and use the moving average to cluster the client with other single-mode clients later. Using moving average is to smooth the mode and thus be more resilient to atypical modes.

In the third case, if the client has overlapping modes in the n-th period, it is a cluster by itself in the $(n+1)$-st period. If the client has a single mode, the moving average of its mode in the n-th period carries to the $(n+1)$-st period and the moving average is used to cluster the client with other clients. If the client has non-overlapping modes, we check if the client is in the same cumulative cluster with other clients in the n-th period, referred to as *in-cluster clients*. If there is no in-cluster client, the client is a cluster by itself in the $(n+1)$-st period. If in-cluster clients exist, we use the most frequent pattern of these clients in the $(n+1)$-st period as the latency pattern of the client. It is possible that none of the in-cluster clients have identifiable pattern in the $(n+1)$-st period or the most frequent pattern still has non-overlapping modes. In this case, the client and its in-cluster clients are still in the same cluster in the $(n+1)$-st period. If the most frequent pattern has overlapping

modes, the client is in a cluster by itself in the $(n+1)$-st period. If the most frequent pattern has a single mode, the average mode of the in-cluster clients is used as the mode of the client in the $(n+1)$-st period. We calculate the moving average using the mode for the client and use the moving average to cluster the client with the other clients later.

After the operations above, clients in the n-th and $(n+1)$-st periods are consolidated. We record the most recent latency pattern (i.e., the one in the $(n+1)$-st period) and correct the latency patterns of clients to their most frequent latency patterns in record if the most recent one differs from the most frequent. This is to avoid the possibility that the most recent latency pattern is misidentified. However, we do not correct the latency patterns of clients if their most frequent latency patterns are non-overlapping modes, as non-overlapping modes are temporary due to network events. Since clients with non-overlapping and overlapping modes are already clustered, we must cluster clients with a single mode using the QT algorithm, which operates on the moving averages of their modes. All the resulting clusters combined comprise the cumulative clusters of clients in the $(n+1)$-st period. In the following, we use cumulative clusters to get AP-atom candidates and simply refer to cumulative clusters as clusters.

2.3.1.5 AP-Atom Candidates: Optimal Prefix Splitting

After clustering, we obtain clusters including clients with similar latency patterns to a given server. However, these clusters only include clients with identifiable latency patterns in our dataset. We cannot directly use these clusters for global load balancing. Instead, we use them as a guide to generate a set of prefixes that cover the entire Internet address space.

Given a server, we can have a set of client clusters in each top prefix. Our goal is to find the minimum set of prefixes (i.e., AP-atom candidates) for each top prefix such that 1) these prefixes can cover the entire address space of the top prefix and 2) using the longest prefix match, clients in the same cluster are matched into the same prefix and clients in different clusters are matched to different prefixes. The longest prefix match algorithm [32], widely

used for Internet routing, always finds the prefix that shares the most number of common bits with the address of the client. This property guarantees that even when one prefix is included in another prefix, the client can still be matched to the correct one. In the following, we refer to this problem as *optimal prefix splitting*.

To obtain the minimum set of prefixes, we construct a binary tree from all the prefixes covered by the address space of the top prefix. Each node in the tree is a prefix, where the root of the tree is the top prefix and the leaves are /24 clients. The children of a prefix $/x$ are two $/(x+1)$ prefixes that evenly split the $/x$ prefix. Clusters of /24 clients are indexed and each /24 client, if it has an identifiable latency pattern, is associated with the number of the cluster that the client is in. Clients in the same cluster have the same cluster number and must be matched to the same prefix. Clients with no identifiable pattern has no cluster number and can be matched to any prefix depending on how the address space is split. Among all the nodes in the tree, we want to select the minimum number of nodes that can achieve the optimal prefix splitting, where the prefixes at the selected nodes comprise the minimum set of prefixes. For each node in the tree, we have two choices, either selecting the node or not.

Let us start the node selection from the top prefix. If the top prefix is not selected, the minimum set of prefixes is the union of the minimum sets of prefixes needed to achieve the optimal prefix splitting for the left and right children of the top prefix. If the top prefix is selected, there could be multiple clusters of clients in the subtree of the root node, where the *subtree of a node* consists of the node and all descendents of the node. We must determine which cluster of clients the top prefix should cover under the longest prefix match. Because clients in different clusters should be matched to different prefixes, each prefix can only cover one cluster. Suppose the top prefix is selected to cover the i-th cluster. To guarantee that clients in the i-th cluster are matched with the top prefix under the longest prefix match, for other prefixes we must select the largest ones whose subtrees do not include clients in the i-th cluster; otherwise, since these prefixes are smaller than the top prefix, clients in

the i-th cluster in the subtree of these prefixes will be matched with them rather than the top prefix under the longest prefix match. We refer to such prefixes as *complementary prefixes* to the top prefix. To generalize the definition, the complementary prefixes to a prefix are the largest prefixes that are in the subtree of the prefix and do not include any client from the cluster covered by the prefix under the longest prefix match. When the top prefix is selected, the minimum set of prefixes to achieve the optimal prefix splitting is the top prefix and the union of the minimum set of prefixes needed for each complementary prefix to the top prefix, where the set of complementary prefixes depends on which cluster the top prefix covers.

From the selection process above, we can see that the problem of optimal prefix splitting can be divided into similar subproblems. We can use dynamic programming to solve the problem as follows. Suppose we have a node u and want to know the optimal splitting of the prefix at node u. We denote the minimum number of prefixes needed as $F_{opt}(u)$. The set of prefixes that achieve the minimum number of prefixes is the minimum set of prefixes. We denote the number of prefixes needed by selecting node u as $F_{in}(u, i)$, where i is the cluster number of the clients that the prefix at node u must cover under the longest prefix match. We index clusters from 1 and use $F_{in}(u, 0)$ for the case that no cluster is included in the subtree of node u. Suppose the i-th cluster is chosen to be covered by the prefix at node u. We denote the set of complementary prefixes to the prefix at node u as $P_{u,i}$. The subtree of each complementary prefix includes a set of clusters—the set may be empty. We denote the set of clusters in the subtree of node v as S_v, where S_v is empty if no cluster is in the subtree of node v. If node u is not selected, no cluster is covered by the prefix at node u and the optimal splitting is determined by the children of node u, denoted as C_u. We denote the number of prefixes needed by not selecting node u as $F_{out}(u)$. Thus, the optimal solution is:

$$F_{opt}(u) = \min \left\{ \min_{i \in S_u} \left\{ F_{in}(u, i) \right\}, F_{out}(u) \right\},$$

where $\min_{i \in S_u} \left\{ F_{in}(u, i) \right\}$ is the minimum number of prefixes among all cases when dif-

ferent clusters in the subtree of node u are chosen to be covered. Based on the selection process, we also have that

$$F_{in}(u,i) = 1 + \sum\nolimits_{v \in P_{u,i}} F_{opt}(v)$$

and

$$F_{out}(u) = \sum\nolimits_{v \in C_u} F_{opt}(v).$$

We use dynamic programming to compute the optimal solution. When traversing the binary tree, we stop at a node either when the subtree of the node includes no cluster or a single cluster. Suppose we stop at node v. If node v includes no cluster, we set $F_{in}(v,0) = 1$, because no further search is needed. If node v includes a single cluster and the cluster number is i, we set $F_{in}(v,i) = 1$. In both cases, we set $F_{out}(v)$ to ∞ such that the prefix at node v must be included to guarantee that either the entire address space is covered or each cluster is covered by a prefix. After we obtain the minimum set of prefixes, each prefix is an AP-atom candidate.

2.3.2 AP-Atoms: Multi-Server View

We now have a set of AP-atom candidates for each server, where clients in the same AP-atom candidate have small distances between their latencies to the server. Since the sets of AP-atom candidates to different servers may be different, we next merge these sets to obtain AP-atoms. The goal of this merging operation is to achieve server-independence. Under the longest prefix match rule, the merging can be easily done by combining the set of AP-atom candidates of each server. Suppose we have a client that is in the prefix $a_1.b_1.c_1.d_1/x_1$ to server 1 and is in the prefix $a_2.b_2.c_2.d_2/x_2$ to server 2. For the two prefixes, since they cover the same client, the larger prefix must include the smaller one, i.e., the address space covered by the smaller prefix is within the address space covered by the larger one. After combining the sets of AP-atom candidates, we have both prefixes in the set of AP-atoms.

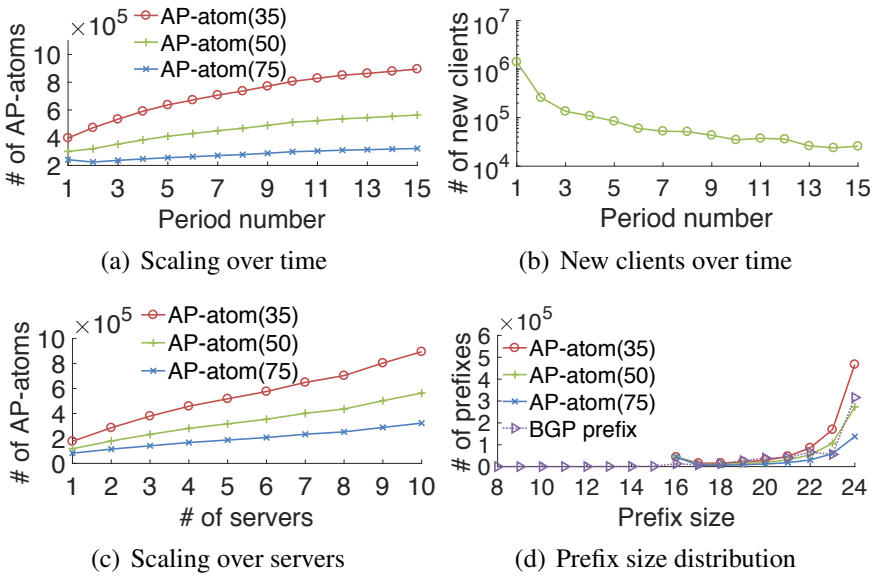
Figure 2.8: Locations of servers

Under the longest prefix match rule, the client is matched to the smaller prefix. Since the larger prefix includes the smaller one, the client is also in the larger prefix and thus has small dispersion to both servers.

Combining AP-atom candidates to each server gives us the set of AP-atoms, but this set includes many *empty prefixes*, i.e., the ones that no clients are matched to. In other words, the address space covered by an empty prefix is the combination of the address spaces covered by other small prefixes. For instance, if the set of AP-atoms includes a $/x$ prefix and two $/(x+1)$ prefixes that evenly divide the address space of the $/x$ prefix, the $/x$ prefix is an empty prefix. After pruning all empty prefixes, we obtain the finalized set of AP-atoms.

2.4 Performance Evaluation

We use 30 days of latency measurements between clients and 10 servers in September 2015, where the locations of servers are shown in Figure 2.8. Due to data sparsity, as discussed in the §2.2.1, we divide 30 days into 15 two-day periods and use measurements from two consecutive days to identify latency patterns. To avoid atypical latencies, we smooth the latencies by their moving average with the typical weights of moving average in TCP. To present the capability of AP-atoms in trading off scalability for accuracy, we



(a) Scaling over time (b) New clients over time

(c) Scaling over servers (d) Prefix size distribution

Figure 2.9: Scalability of AP-atoms

obtain three sets of AP-atoms using different pre-defined thresholds for the QT algorithm. The numbers of AP-atoms on these sets are chosen to be on scales that are larger than, equal to, and smaller than the number of aggregation units under existing aggregation methods respectively. Each set of AP-atoms is compared against existing aggregation methods in terms of scalability and server-independence.

2.4.1 Scalability of AP-Atoms

We look at the scalability of AP-atoms in three aspects: 1) the scaling of AP-atoms over time, 2) the scaling of AP-atoms over the number of servers and 3) the distribution of prefix sizes of AP-atoms. The third aspect tells us the reasons for the advantages of AP-atoms over existing aggregation methods in terms of scalability.

2.4.1.1 Scaling of AP-Atoms over Time

Since AP-atoms evolve, we want to know the scaling of AP-atoms over time and the factors that affect the number of AP-atoms. Figure 2.9(a) shows the number of AP-atoms over 15 periods, where AP-atom(k) denotes the AP-atoms obtained by setting the pre-

defined threshold equal to k ms, meaning that the radius of clusters in the QT algorithm is less than k ms. In general, as the period number increases, the total number of AP-atoms under all thresholds increases. The increment is mainly caused by the appearance of *new clients*, clients that never have an identifiable latency pattern in the previous periods. Depending on the latency patterns, new clients can cause the number of AP-atoms to increase in three ways: 1) If the new client has a single mode and the mode is far away from the modes of other clients, a separate cluster will be needed for the client. 2) If the new client has non-overlapping modes not similar to the latency patterns of other clients, the client will be in a separate cluster. 3) If the new client has overlapping modes, the new client will be in a separate cluster by itself.

Figure 2.9(b) shows the number of new clients in each period. In the first five periods, there are at least 100K new clients in each period and even until the last period, there are still about 20K new clients. When the number of new clients becomes stable at the 10-th period, the number of AP-atoms(50) and AP-atoms(75) increases slowly. In contrast, AP-atoms(35) are sensitive to new clients and increase with a rate of 15K per period after the 11-th period. Comparing AP-atoms(35) and AP-atoms(75), we see that a large threshold helps accommodate new clients without incurring a significant number of new AP-atoms. This is because most clients have a single mode, which can be easily accommodated with other clients when the threshold is large. When the last period ends, we obtain 895K AP-atoms(35), 563K AP-atoms(50) and 322K AP-atoms(75), which are $1.51\times$, $0.95\times$, and $0.54\times$ the 592K number of BGP prefixes. To cover the address space of BGP prefixes, we need 693K /20 IP blocks. Since geo-blocks generally have smaller aggregation units than BGP prefixes, we need more geo-blocks to cover the entire address space.

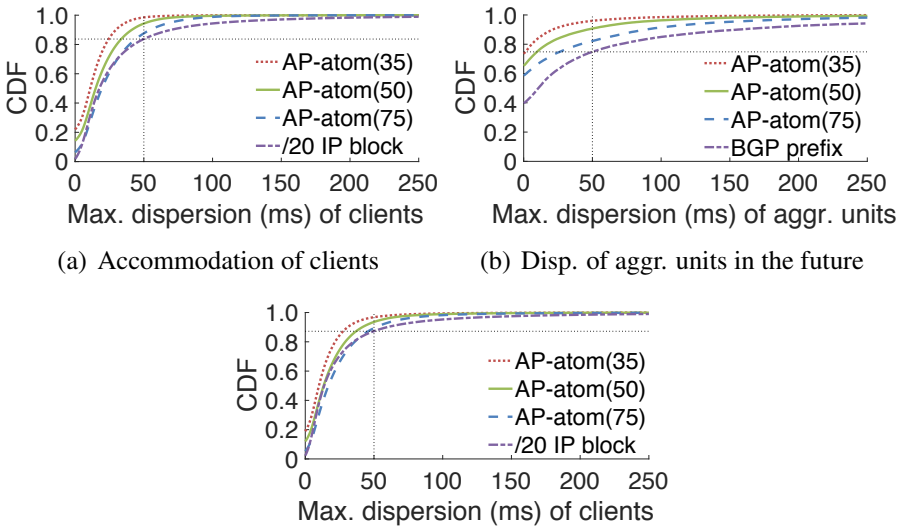
It is noticeable that even when there are more than 100K new clients appearing in the second period, the number of AP-atoms(75) in the second period is still less than that in the first one. The decrement is caused by the correction of latency patterns. From the first to the second period, clients identified to have non-overlapping and overlapping modes in the

first period are corrected to have a single mode in the second one. Under a large threshold, these clients changing to have a single mode can be clustered with other clients, causing the total number of AP-atoms(75) to decrease. The latency patterns of such clients also change when the thresholds are equal to 35 ms and 50 ms, but as AP-atoms(35) and AP-atoms(50) are more sensitive to new clients, the total still increases.

2.4.1.2 Scaling of AP-atoms over the Number of Servers

To be server-independent, AP-atoms are obtained by merging the AP-atom candidates of each server. We want to know the scaling of AP-atoms over the number of servers. Figure 2.9(c) shows the number of AP-atoms under different numbers of servers. As the number of servers increases, the number of clients accommodated into AP-atoms increases sharply (not shown here). At 8 servers, about 98% of clients have been accommodated into AP-atoms, but this does not slow down the linear increment of AP-atoms from server 9 to 10. The key reason for the increment is because AP-atom candidates of each server are obtained separately without considering others. More specifically, AP-atom candidates of a server are generated by the optimal prefix splitting based on the set of clients to the server, while servers have different sets of clients, which results in different sets of AP-atom candidates. An optimization across servers would further decrease the number of AP-atoms, which is a subject of our future research.

For small-scale CDNs with servers in 30 to 40 locations [33], the scaling of AP-atoms is not a concern. Even when the AP-atom candidates of all servers are used, based on the trend of AP-atoms(75) in Figure 2.9(c), the number of AP-atoms is approximately 470K for 40 servers, which is $0.8\times$ the number of BGP prefixes. For large-scale CDNs with more than 1K locations of servers [34, 5], we can select servers whose views of AP-atom candidates differ significantly and only merge AP-atom candidates of these servers.



(a) Accommodation of clients

(b) Disp. of aggr. units in the future

(c) Disp. of clients in the future

Figure 2.10: Server-independence

2.4.1.3 Distribution of Prefix Sizes

We have compared AP-atoms with existing aggregation methods in terms of the total number of aggregations needed to cover the Internet address space, but the total number cannot tell us the reasons for the scale of aggregation units. We look at the distribution of prefix sizes to understand the advantages of AP-atoms over existing aggregations. Figure 2.9(d) shows the distributions of prefix sizes of BGP prefixes and the three sets of AP-atoms, where the 1.6% of BGP prefixes smaller than /24 are not counted for the distribution. Since the IP2Location LITE database does not include the entire Internet address space, geo-blocks are not compared here. The largest prefix size of BGP prefixes is /8, while as AP-atoms are split from /16 top prefixes, AP-atoms have the largest prefix size equal to /16.

The three sets of AP-atoms include about 43K /16s, which are the 97% of all /16s we use to cover the entire address space. Large prefixes, due to covering large address space, are preferred to minimize the number of aggregation units. For each prefix size smaller than /16, AP-atoms(75) use a smaller number of prefixes than BGP prefixes and thus have a total number that is half the number of BGP prefixes. AP-atoms(50) have almost equal

number of prefixes as BGP prefixes for each prefix size and thus have the same scale as BGP prefixes, while AP-atoms(75) include higher numbers of /23s and /24s than BGP prefixes. From the figure, we can see that the different scales of AP-atoms are mainly due to using different numbers of /23s and /24s. In other words, top prefixes generally must be split into small prefixes under a small threshold. It is interesting to note that only 5% of AP-atoms(50)'s /24 aggregation units prefixes are also BGP /24 prefixes. The /24 aggregation units in AP-atoms(50) are constructed based on path performance whereas /24 BGP prefixes reflect commercial organizational decisions that do not necessarily reflect different network performances.

2.4.2 Server-Independence

To be server-independent, an aggregation unit must have small dispersion to all servers, i.e., the maximum dispersion of the aggregation unit to all servers must be small. We have presented the maximum dispersion of existing aggregations in Section 2.2.2 and use the best existing aggregation to compare against AP-atoms. We first check if clients in the past and current periods have been properly accommodated in AP-atoms and then look at the performance of AP-atoms to deal with clients in future periods.

2.4.2.1 Capability of Accommondating Clients

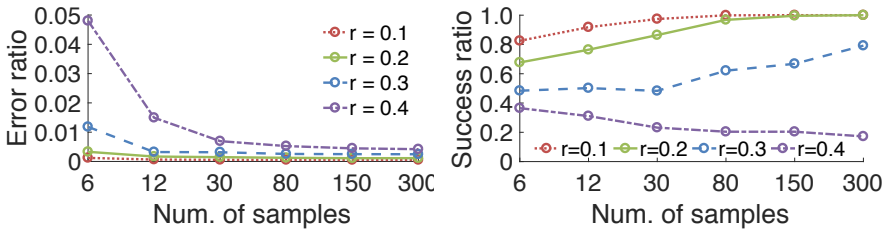
AP-atoms accommodate clients based on the latency patterns of clients in the past periods. If clients are properly accommodated in AP-atoms, we would expect the maximum dispersion of the clients calculated using the latencies in the past periods to be small. We use the AP-atoms in the last period for the experiments. Since the maximum dispersion of /20 IP blocks is the smallest among existing aggregations (Figure 2.4(b)), we compare the three sets of AP-atoms with /20 IP blocks. Figure 2.10(a) shows the maximum dispersion of clients calculated using latencies from the first to the 15-th period. Compared to /20 IP blocks that have 17% of clients with maximum dispersion larger than 50 ms, AP-atoms(35)

only have 1.5% of clients with the maximum dispersion larger than 50 ms. This implies that AP-atoms(35) properly accommodate clients in terms of latency. The main reason for AP-atoms(35) not able to control the dispersion of all clients less than 50 ms is because the threshold is 35 ms, which allows clients in the same AP-atom to have the maximum dispersion up to 70 ms. As the threshold increases to 75 ms, 12% of clients have maximum dispersion larger than 50 ms.

2.4.2.2 Dispersion and Latency Prediction

At the end of each period, we update AP-atoms and use them for the next period. AP-atoms used in the $(n+1)$-st period are AP-atoms updated when the n-th period ends. From Figure 2.9(b), we can see that the number of new clients starts to slow down from the 10-th period. We consider the 10-th period as the time that AP-atoms have accumulated sufficient clients. From the 10-th to the 15-th period, we use AP-atoms in the n-th period as the aggregation units in the $(n+1)$-st period and calculate the dispersion of clients in the $(n+1)$-st period. Given a server, the dispersion of a client to the server is the maximum dispersion of the client over all periods to the server. Using the dispersion of a client to each server, we obtain the maximum dispersion of the client to all servers.

Figure 2.10(b) shows the distribution of the maximum dispersion of aggregation units. Because the maximum dispersion of aggregation units in BGP prefixes is the smallest among existing aggregations (Figure 2.4(a)), we compare AP-atoms with BGP prefixes. While BGP prefixes have 26% of aggregation units with dispersion larger than 50 ms, AP-atoms(35) can reduce such aggregation units to 4%. The aggregation units with zero dispersion are mainly /24s. It is interesting to note that the percentage of /24s in AP-atoms(75) is 12% less than that in BGP prefixes, but in Figure 2.10(b), the percentage of aggregation units with zero dispersion in AP-atoms(75) is higher than that in BGP prefixes. This is because /24s in AP-atoms are identified by their traffic in the past and clients with past history of traffic to a service provider are generally more likely to have traffic to the service



(a) Error ratio (b) Success ratio

Figure 2.11: Tolerance to latency inflation

provider in the future. Thus, /24s in AP-atoms(75) are more likely to have traffic than the /24s in BGP prefixes.

Figure 2.10(c) shows the distribution of the maximum dispersion of clients, where /20 IP blocks are compared with AP-atoms. About 13% of clients in /20 IP blocks have dispersion larger than 50 ms. Even with a scale of aggregation units that is $0.46\times$ the number of /20 IP blocks, AP-atoms(75) can reduce this number to 10%. With a scale that is $1.3\times$ the number of /20 IP blocks, AP-atoms(35) reduces this number to 3.5%.

2.4.3 Tradeoff Between Tolerance and Responsiveness

Clients could have high-variation latencies to a server if their paths are greatly inflated due to network events, e.g., queueing and route changes. Our latency identification algorithm can tolerate a certain level of inflation and still accurately identify the underlying latency pattern. A larger set of measurements is always preferred for latency identification, but collecting measurements takes time, influencing the responsiveness of AP-atoms to network changes. We want to study the tradeoff between tolerance and responsiveness.

To test our algorithm, we create synthetic latency samples with different levels of inflations, controlled by the probability of latency samples being inflated (denoted as r). Suppose the base latency without inflation is denoted as l. In our experiments, each sample is chosen to be inflated with a probability of r. If a sample is inflated, we first determine the inflation that is uniformly distributed between 0 and l,[9] and then add the inflation to the

[9]We use a simple uniform distribution to model latency inflation. The maximum inflation is set to l,

base latency; otherwise, the sample is equal to the base latency. We continue generating new samples until the desired number of samples (denoted as n) is reached. We then apply our algorithm on these samples to identify the base latency. If a single latency is found in these samples, we consider that our algorithm succeeds in identifying the base latency and calculate the *identification error* (i.e., the difference between the identified base latency and the actual base latency).

In our experiments, we set the base latency to 100 ms such that the maximum inflation is 100 ms, large enough to simulate variable network conditions. To avoid randomness, we repeat the same experiment 1000 times and use the average results. Figure 2.11 shows the average success and error ratios under different r's and n's, where the *success ratio* is the percentage of repeated experiments in which our algorithm succeeds in identifying the base latency and the *error ratio* is the ratio of the *identification error* to the base latency. Figure 2.11(a) shows that the identified base latency is accurate with an error less than 5% for all r's, but the cost of having accurate identification is that the success ratio is relatively low when the number of samples is small. Using a larger number of samples can improve the success ratio to near 100% when $r = 0.1$ or 0.2, but this fails when r increases to 0.4. This implies that the limit to the tolerance of our algorithm to latency inflation is reached when 30% of samples are inflated 50 ms on average.

The success ratio determines the percentage of clients whose latencies can be identified. A better tolerance to inflation leads to a higher success ratio. We want a better tolerance in order to have more clients with identifiable latencies. However, collecting a larger number of measurements generally takes a longer time, resulting in slow response to network changes. Considering both tolerance and responsiveness, using 12 samples for latency identification is a reasonable choice.[10] When no base latency is identified at 12 samples, we can either consider the clients have multiple latencies and regroup them or

proportional to the base latency, as a path with larger latency is likely to traverse more links and experience larger inflation.

[10]In previous experiments, we divide time into fixed-length periods and use measurements from the same periods for comparison purpose.

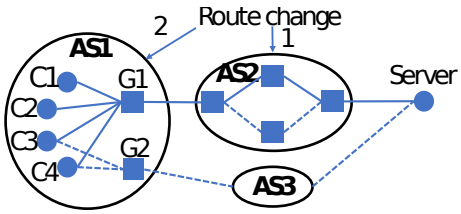


Figure 2.12: Impact of network changes on client aggregation

wait for more samples to be collected. The actual time taken to collect 12 samples depends on the traffic rates of clients. We can complement passive measurements with active probes to improve the responsiveness to network changes.

Compared to AP-atoms, other aggregation methods are not adaptive to network changes. Geolocations and LDNSs of clients are static. Although BGP prefixes may be changed, as BGP prefixes are designed for Internet routing, changing BGP prefixes would cause many BGP issues like convergence and incur communication overheads. As found in [35], except for new networks, most new BGP prefixes are due to BGP misconfigurations. Moreover, BGP prefixes are managed by tens of thousands of service providers separately. Cooperation from these service providers is needed to change BGP prefixes.

2.5 Discussion

We know how to obtain AP-atoms, but it is unclear how AP-atoms perform under specific network scenarios: middleboxes, high-variation latencies and network changes.

2.5.0.1 Aggregating Clients behind Middleboxes

Middleboxes generally hide their clients' addresses from external networks. When a server communicates with clients behind a middlebox, only the public addresses (i.e., publicly routable addresses) of the middlebox are visible to the server. From the server's perspective, all measurements are from the same address. If these measurements exhibit a single latency (e.g., clients behind a NAT have similar latencies to the server), the public

address is further aggregated with other clients based on the latency. In contrast, if these measurements exhibit more than one latency (e.g., clients behind a NAT have evenly distributed latencies from 100 to 200 ms to the server), the public address will be treated as a separate aggregation unit. In AP-atoms, middleboxes that hide the private addresses from the rest of the Internet implicitly aggregate these private addresses behind them. Nonetheless, middleboxes can be further aggregated to reduce the number of aggregation units that a mapping system has to maintain. The potential improvements of AP-atoms are discussed in Section 2.5.0.4.

2.5.0.2 Aggregating Clients with High-Variation Latencies

Clients could have various path performance depending on their network conditions. Clients in mobile networks (e.g., cellular) are prone to experiencing more variations in latency than those in wired networks. For clients with high-variation latencies, if they are behind a middlebox, they will be aggregated by the middlebox as discussed above. In cellular networks, a majority of clients (70% in [36]) are reported behind a middlebox and have private addresses. For clients with public addresses, if they have large-variation latencies, AP-atoms will group them into small aggregation units. To understand the resilience of AP-atoms to latency variations, we conduct experiments in Section 2.4.3, which shows that our algorithm is able to identify latency patterns even when 30% of samples are inflated by 50% on average. As AP-atoms aggregate clients starting from /24 blocks. The worst case is that AP-atoms treat each /24 block as an aggregation unit, which increases the total number of aggregation units. This weakness of AP-atoms can be overcome if combined with active probing as discussed in Section 2.5.0.4.

2.5.0.3 Impact of Network Changes on AP-atoms

When network conditions change, the performance of clients using the network might be affected, but no re-grouping of clients is needed if clients in the aggregation unit still

have similar path performance after the change.[11] Only when network changes cause parts of an aggregation unit to have different performance than the others, clients in that aggregation unit should be re-grouped. Since AP-atoms group clients based on measurements, the re-grouping of clients requires new measurements be collected after the change. If a network change is long-term, AP-atoms would be updated to accommodate the change after it is detected. On the other hand, if the network change is transient, the update of AP-atoms may not be responsive to the change. The time taken to collect sufficient new measurements for re-grouping clients depends both on the portion of clients affected by the change and the traffic rates of clients. An experimental study on the responsiveness of AP-atoms to network changes is provided in Section 2.4.3.

In Figure 2.12, we use route changes as an example of network changes to show how client aggregation is affected. Before the route changes, all clients in autonomous system AS1 take a path traversing gateway G1 and AS2 to the server (represented by the solid line) and have similar latencies. The first route change occurs between two routers in AS2. Since all clients still share the same path to the server, the performance of clients is changed equally and thus no client re-grouping is needed. The second route change causes clients C3 and C4 to use a new gateway G2 to the server. If the new path increases the latency to the server, clients C3 and C4 will be re-grouped into a separate aggregation unit. In contrast, if the route change does not affect the latency to the server, all clients remain in the same aggregation unit.

2.5.0.4 Weaknesses and Improvements of AP-Atoms

AP-atoms are obtained from passive measurements, which incurs no extra measurement overheads. Unfortunately, this also leads to several weaknesses of AP-atoms: 1) AP-atoms aggregate clients with high-variation latencies into small aggregation units, which impacts the scalability of AP-atoms; 2) AP-atoms cannot promptly react to abrupt network

[11] The reasons for this could be that clients take a new path with the same performance or that the performance of clients is changed by the same amount.

changes, e.g., a BGP routing change. The responsiveness of AP-atoms to network changes is determined by the traffic rates of clients; 3) since AP-atoms aggregate clients based on measurements, if the network has few measurements available, AP-atoms may not be accurate for clients in that network. These weaknesses can be overcome by using targeted active probing to complement AP-atoms [5]. Suppose we find a large IP block consisting of small aggregation units, each of which includes clients with high-variation latencies. It is possible that this large IP block includes addresses from cellular networks. We can send probes to discover if the clients in this large IP block are similar in topology (e.g., using the same gateway connecting with the Internet[12]). Similarly, we can complement passive measurements with active measurements to be more responsive in detecting network changes. The sending rate of active probes is determined by the requirements on responsiveness.

2.5.0.5 Extension of AP-Atoms to IPv6 networks

AP-atoms are specifically designed for IPv4 networks. Since IPv6 has a much larger address space that is presently more sparsely allocated and carries much less traffic than IPv4 [37], the proper use of AP-atoms for IPv6 networks remains an interesting future research topic.

2.6 Summary

In this chapter, we presented a comparative study of existing client aggregation methods and found that even for the best existing client aggregation, a significant portion (17%) of clients are far away from other clients in the same aggregation unit in terms of latency. We analyzed the causes for the widely dispersed clients and found the main cause to be that existing methods aggregate clients based on attributes other than path performance. To address this, we proposed a data-driven aggregation called AP-atoms, which group clients

[12]It has been reported that major cellular carriers in US connect their networks with the Internet through a few ingress points[26].

based on their path performance. The data-driven property enables AP-atoms to dynamically adapt to network changes. We studied the scalability of AP-atoms and compared AP-atoms against existing aggregation methods. Our results showed that AP-atoms can flexibly trade off scalability and accuracy. Using the same scale of aggregation as existing methods, AP-atoms can reduce widely dispersed clients by $2\times$ and reduce the 98-th percentile difference in clients' latencies by almost 100ms.

CHAPTER III

Latency Imbalance Among Internet Load-Balanced Paths: A Cloud-Centric View

In this chapter, we evaluate the latency imbalance among LB paths and its impacts on various applications. We show that the latency imbalance, previously deemed insignificant, is now prevalent from the perspective of the cloud and affects various latency-sensitive applications. To characterize latency imbalance, we conduct the first large-scale measurement study of latency imbalance from a cloud-centric view. Using public cloud around the globe, we measure latency imbalance both between DCs in the cloud and from the cloud to the public Internet. We further evaluate the impact of latency imbalance on three applications (i.e., NTP, delay-based geolocationand VoIP) and propose potential solutions to improve application performance.

3.1 Introduction

Latency imbalance was first studied by Augustin *et al.* in 2007 between 22K source-destination pairs, of which only 2% exhibit significant latency imbalance [8]. Low latency imbalance at that time was not enough to raise concerns. Although measurement tools, e.g., *Tokyo ping* [38], have been proposed to enforce probes to follow the same LB path to avoid instability of latency samples due to latency imbalance, existing measurement tools

and applications do not consider latency imbalance by default. For instance, `ping` and `traceroute` are commonly used to measure latency in their default modes, allowing probes sent to the same destination to take different LB paths. In this work, we conduct the first large-scale measurement study of latency imbalance from a cloud-centric view, i.e., using data centers (DCs) as vantage points, and evaluate its impact on applications. In 2019, 12 years after the first study, we observed that 15.6% of 48M DC-destination pairs had significant latency imbalance, affecting various latency-sensitive applications.

To measure latency imbalance at the Internet scale, we need an efficient way of measuring it for each source-destination pair, which could have a large number of LB paths in between [9]. The major challenges are three-fold: 1) how to efficiently ensure that all LB paths are covered to calculate latency imbalance for a given source-destination pair? 2) how to efficiently obtain stable latency under transient network events for LB paths? 3) how to dissect latency imbalance to better understand the causes for latency imbalance? To address these challenges, our methodology uses three key strategies (§3.3.3). First, instead of enumerating LB paths that is inherently heavy in measurement overhead, we sample the latency of LB paths with sufficient sample size such that almost all LB paths are covered with high probability. Second, as we want to know how LB paths differ in latency, we use the range of latencies to measure the latency difference and obtain stable latencies only for LB paths that could potentially have the lowest and highest path latencies. Third, to better understand latency imbalance, we focus on measuring one-way latency imbalance occurring only on the forward paths that can be discovered with TTL-limited probes.

Using our methodology, we collect a global view of latency imbalance from the perspective of the cloud. The global view includes two parts: latency imbalance from DCs around the globe to about 3M /24 prefixes (§3.5) and latency imbalance between DCs in the cloud (§3.6). Overall, we find that latency imbalance is both significant and prevalent between DCs in the cloud and from the cloud to the public Internet. To better interpret the significance of latency imbalance, we determine the baseline of latency imbalance for

three applications based on each application's sensitivity to latency imbalance (§3.4). We highlight our key findings below.

- Latency imbalance is prevalent from the perspective of DCs. Even the best DC has latency difference between LB paths larger than 5ms to about 20% of public IPv4 addresses and the latency difference increases to 40ms to the same amount of addresses for the worst DC (§3.5). The prevalence of latency imbalance in the 1s of milliseconds could affect applications, e.g., Internet geolocation and clock synchronization via NTP, very sensitive to latency imbalance (§3.4). High latency imbalance in the tens of milliseconds mainly occurs on long-distance LB paths and thus would affect long-distance communications, e.g., international VoIP calls (§3.7).

- Latency imbalance differs significantly between cloud service providers: Google's DCs have consistently lower latency imbalance to public addresses than Amazon's and Alibaba's. Detailed path analysis tells us that Google uses private wide area networks (WANs) to route packets to an egress point close to the destinations and that its private long-distance paths are well balanced. Other cloud service providers, on the other hand, forward packets to the public Internet closer to traffic sources, which is less balanced than the providers' own networks (§3.5.2).

- The distribution of latency imbalance seen from most of the DCs is stable over months, which implies that latency imbalance is not due to temporary behaviors of the Internet (§3.5.3).

- Significant latency imbalance is found both between DCs in the same cloud (*intra-cloud*) and between DCs across different clouds (*inter-cloud*). Google's DCs (in premium network service tiers) have lower intra- and inter-cloud latency imbalance than those of other cloud providers due to using well-balanced paths in its own private WANs (§3.6).

- We further evaluate the impact of latency imbalance on three applications affected by latency imbalance in different ways. (1) Under latency imbalance, `ping` would overestimate the minimum path latency by $>$ 5ms for 17% of addresses if measured from the cloud. Using the minimum path latency rather than the *ping-time* (the path latency measured by `ping`) could improve the accuracy of Internet geolocation by 40% for 20% of addresses for cloud-based applications, e.g., cloud data geolocation [39]. (2) Clock synchronization via NTP assumes symmetric paths to estimate one-way latency between the NTP client and server. Our experiments show that estimation error of the one-way latency can be reduced by as high as tens of milliseconds. (3) We simulate the visionary idea of using an overlay network, consisting of DCs around the globe as relays, for VoIP [40]. About 14% of long-distance VoIP calls between certain countries (e.g., US and CN) could have better call quality by using lower-latency paths.

3.2 Background

3.2.1 Types of Load Balancing

Load balancing can be performed per flow, per destination, or per packet. In per-flow load balancing, load balancers commonly hash 5 fields in the packet header to determine the next hop to forward the packet, where the 5 fields are used as *flow identifier* and described as a five-tuple: ⟨source address, source port, destination address, destination port, protocol number⟩ [2, 41]. For the same source-destination pair, we can change port numbers to discover LB paths in between [9]. Per-destination load balancing allows routers to balance traffic based on the destination address. In per-packet load balancing, packets in the same flow could be forwarded to different paths causing reordering of packets. Besides the five-tuple in TCP and UDP packets, routers also load-balance ICMP packets based on their checksum fields [2]. Both per-flow and per-packet load balancing could cause latency

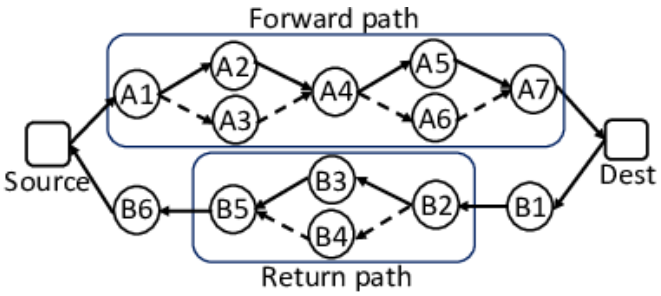


Figure 3.1: An example of LB paths

imbalance between source-destination pairs. Per-flow load balancing affects about 40% of Internet paths, while per-packet load balancing only affects 2% [8]. We focus on per-flow load balancing in this chapter, under which paths to the same destination can be controlled via port numbers.

3.2.2 Load-Balanced Segments or Diamonds

Starting from the source, paths to a given destination diverge at the first load balancer (*divergence point*) and eventually converge back (at the *convergence point*) to reach the destination. A multi-homed source could be a divergence point and similarly a multi-homed destination could be a convergence point. Following the literature, we refer to the multiple path segments between the divergence and convergence points (or *end points*) as the *load-balanced segments* or, in aggregate, as a *diamond*. Each diamond is defined by its divergence point (x_1) and convergence point (x_2) and referred to as the $[x_1, x_2]$ diamond. Since diamonds include the path segments where LB paths differ from each other, they can be used to dissect latency imbalance.

Figure 3.1 shows an example of LB paths between a source and a destination. Due to path asymmetry, the forward and return paths are different and thus include different diamonds. In Figure 3.1, we have two diamonds, $[A1, A4]$ and $[A4, A7]$, on the forward paths and $[B2, B5]$ on the return paths. Suppose that the path segments $A1$-$A2$-$A4$ and $A1$-$A3$-$A4$ in $[A1, A4]$ differ in path latency. We refer to $[A1, A4]$ as *imbalanced diamond*, which are the causes for latency imbalance.

3.3 Measurement Methodology

This section presents our focus and key challenges in measuring latency imbalance and our measurement methodology. We begin by defining latency imbalance.

Definition of Latency Imbalance When LB paths exist between a source-destination pair, these paths could have different path latencies. To characterize the significance of the path latency difference, we define *latency imbalance* as the difference between the highest and lowest latencies of LB paths. We want latency imbalance to be stable and thus consider *path latency* as the minimum time needed for packets to travel through the path, excluding transient network dynamics (e.g., congestion). Such stable path latency is widely used in applications, e.g., Internet mapping [4], geolocation [42] and network clock synchronization [43].

3.3.1 Our Focus and Key Challenges

Our goal is to characterize latency imbalance at the Internet scale. We first discuss the exact type of latency imbalance to be measured, and then identify the key challenges and describe a strawman approach to highlight the limitations of existing measurement tools.

Measuring One- or Two-Way Latency Imbalance? Between a source-destination pair, latency imbalance could occur on both forward and return paths. *One-way imbalance* is the latency imbalance occurring only on forward paths, calculated as the difference between the highest and lowest one-way delays (OWDs) of LB forward paths. *Two-way imbalance* is the combination of latency imbalance on both forward and return paths. As we have no control over destinations, we can only use TTL-limited probes to obtain full topology on forward paths but not on return paths. While *Reverse Traceroute* can discover return paths using ICMP packets [44], the return paths cannot be pinned down by flow identifiers. This means that we can understand one-way imbalance with path analysis (§3.5.4 and §3.5.2),

but cannot do the same for two-way imbalance. We thus focus on studying one-way latency imbalance in this chapter, which affects all applications mentioned above.

Key Challenges and A Strawman Approach To measure one-way imbalance at the Internet scale, we face three challenges: 1) how to efficiently discover LB paths between each source-destination pair? 2) how to efficiently measure the stable latencies of LB paths for the calculation of latency imbalance? 3) how to measure one-way imbalance? A strawman approach is to first enumerate LB forward paths using *Paris Traceroute*, the most popular tool to enumerate LB paths to a given destination [2], and then measure their stable OWDs. However, the overhead of path enumeration is inherently heavy. Even with the lightweight version, MDA-Lite, recently developed by Vermeulen *et al.* [9], it could take several minutes to enumerate paths between a single source-destination pair for complex network topologies [45]. Moreover, OWD can be accurately measured only if we have access to both ends of the path [46]. Access to a large number of hosts is needed for large-scale experiments. These issues make it difficult to scale the approach.

3.3.2 Overview of Our Methodology

As no existing measurement tool and public dataset are suitable for studying one-way imbalance, we develop our own measurement tool and use it to collect data by actively probing the Internet. Our tool is built on Yarrp [47] with functions (1000+ lines of code) added for the scheduling and processing of packets. To ensure global coverage, we send probes to IPv4 addresses from DCs around the globe. We initiate a prober process in each DC selected and the prober measures one-way imbalance to one representative address in each /24 prefix. Of 10.8M /24 prefixes probed, we were able to measure one-way imbalance to about 3M representative addresses (see Table 3.2). The prober has no access to the destination and thus measures one-way imbalance from a single end. The key ideas of achieving this include two points. First, we strategically manipulate the UDP probes

to ensure the ICMP responses always traverse the same return path, thus eliminating the impact of return path load balancing. Second, we sample LB paths to the destination by sending UDP probes with different flow identifiers, specifically port numbers. Since the path of a UDP probe is passively selected by load balancers based on its flow identifier, for sufficiently large sample size, most of the LB paths would be covered with high probability (see §3.3.3.2). We use UDP rather than TCP probes, as we have no way of controlling the return paths of TCP responses (i.e., TCP ACKs or RSTs).

3.3.3 Measuring One-Way Imbalance

We want to efficiently measure one-way imbalance from a single end. This section first presents our methodology and then evaluates its accuracy in measuring one-way imbalance.

3.3.3.1 Fixing the Return Path

Each RTT sample measured at the prober is the sum of the traversal time of a UDP probe on the forward path and that of the corresponding ICMP response on the return path. For UDP probes sent to the same destination, we want to fix the return paths of ICMP responses such that the difference between RTTs is the same as the latency difference between forward paths. Since except for per-packet load balancers, ICMP responses are load-balanced based on their checksums [2], we fix the return path by enforcing ICMP responses to use the same checksum. Per-packet load balancers on the return path could cause ICMP responses with the same checksum to take different paths, but only 2% of paths in the public Internet were found to traverse a per-packet load balancer in Augustin *et. al*'s 2007 survey [8]. In case there could be more per-packet load balancers in the public Internet nowadays or from the cloud to the public Internet, we further filter out unstable path latencies that could be affected by per-packet load balancing as in §3.3.3.4. Augustin *et al.* have proposed a general method of obtaining the desired ICMP checksum by manipulating the UDP data, which requires extra probing to the destination [8]. We simplify the method

by sending UDP probes without UDP data.

3.3.3.2 Achieving High Path Coverage with Probabilistic Guarantees

We refer to UDP samples with the same flow identifier as a *flow*. Instead of enumerating LB paths to a destination, we sample them by sending flows with different identifiers (specifically port numbers), which may take different paths to the destination. We want to determine the *sample size* (i.e., the number of flows) needed to achieve high path coverage.

Defining Path Coverage Suppose there are n LB paths between a source-destination pair with latencies ordered as $l_1 \leq l_2 \leq \cdots \leq l_n$, and that the probability of taking the i-th path is p_i. We have m flow samples with the lowest and highest latencies denoted as l_Y and l_Z, where Y and Z are the path indices. Since latency imbalance is the difference between the lowest and highest latencies, all paths with latencies inclusively between l_Y and l_Z are covered. We thus define *path coverage* as the sum of the probability of the covered paths. Let $s[i, j]$ be the probability of taking a path with latency in interval $[l_i, l_j]$, i.e., $s[i, j] = \sum_i^j p_i$. The path coverage is then equal to $s[Y, Z]$. This definition of path coverage implicitly makes us favor covering the paths of high probability over those of low probability for high path coverage. As will be discussed later, a large number of extra samples is needed to cover a path of low probability with latency much lower or higher than other paths.

Probability of Achieving High Path Coverage We want $s[Y, Z]$ to be higher than a threshold, called *required coverage*. We denote the required coverage as x and the probability of $s[Y, Z] > x$ as $p(s[Y, Z] > x)$. Let $w_{x,i}^+$ be the largest index with $s[i, w_{x,i}^+] \leq x$ and

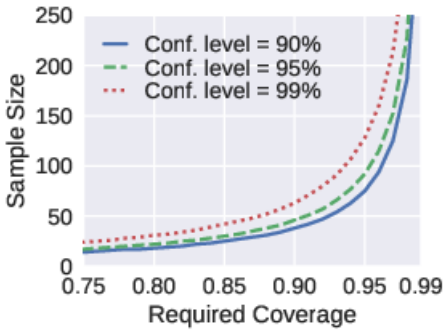


Figure 3.2: Tradeoff between accuracy and efficiency

$w^-_{x,i}$ be the lowest index with $s[w^-_{x,i}, i] \leq x$. We can calculate $p(s[Y,Z] > x)$ as

$$\begin{aligned} & p(s[Y,Z] > x) = 1 - p(s[Y,Z] \leq x) \\ & = 1 - p(s[Y,Z] \leq x | Y \geq w^-_{x,n}) - p(s[Y,Z] \leq x | Y < w^-_{x,n}) \\ & = 1 - s[w^-_{x,n}, n]^m - \sum_{i=1}^{w^-_{x,n}-1} p(Y=i) \left(\frac{s\left[i, w^+_{x,i}\right]}{s\left[i,n\right]} \right)^{m-1}, \end{aligned} \tag{3.1}$$

where $p(Y=i) = p(Y \geq i) - p(Y \geq i+1) = s\left[i,n\right]^m - s\left[i+1,n\right]^m$.

Coverage Guarantee and Tradeoff We want to achieve high coverage with probabilistic guarantees. More formally, we want the actual coverage to be no less than the required coverage (x) with probability at least $1-\delta$, i.e., $p(s[Y,Z] > x) \geq 1-\delta$. Under the same δ, we need larger sample size to achieve higher path coverage; there is a tradeoff between the *measurement accuracy* (determined by path coverage) and the *measurement efficiency* (sample size). Given the required coverage and the probability distribution of paths (p_i's) between a source-destination pair, we can use Equation (3.1) to obtain the minimum sample size needed to achieve a certain probabilistic guarantee. In practice, however, as the probability distribution of paths is not known in advance, we cannot customize the sample size for each pair. Instead, we choose a sample size that achieves the coverage guarantee for all source-destination pairs under various probability distributions of paths.

To collect a diverse range of LB paths, we measured LB paths between 232K source-destination pairs from 6 DCs belonging to 3 cloud providers to random IPv4 addresses. For

each pair, we approximate the probability distribution of paths using their frequency distribution [48]. More specifically, we measure the IP-level paths of 600 flows with different identifiers for each pair and count the frequency of each path. The normalized frequency of a path is used as its probability. Figure 3.2 shows the sample sizes needed to achieve different required coverages with confidence levels of 90%, 95% and 99% respectively. Although the sample size increases exponentially with the required coverage, the increase rate is low when the required coverage is under 0.85 and we can use 30 samples to achieve the required coverage of 85% at the 95% confidence level.

It is worth noting that cloud providers may place many load balancers close to DCs to distribute high-volume traffic and these load balancers provide a large number of unique interfaces, resulting in hundreds of IP-level LB paths between DCs and destinations. The probability distributions of LB paths from Amazon's and Alibaba's DCs to destinations are nearly uniform. Only $< 1\%$ of the DC-destination pairs fail the chi-squared test for uniformity at the 95% significance level. The large number of LB paths further necessitates the use of sampling to measure latency imbalance from a cloud-centric view.

3.3.3.3 Obtaining Stable Latencies for Latency Imbalance

We refer to flows with the highest and lowest latencies as *extreme flows*. Latency imbalance is then the latency difference between extreme flows. A simple approach to calculating latency imbalance is to first measure stable latency to each flow and then select the extreme flows. To reduce measurement overhead, we only measure flows that could potentially be the extreme flows. We first sample each flow once and select the flow with the highest RTT as the candidate for the highest-latency flow. Since the RTT of the candidate could possibly be inflated, we probe the candidate for six consecutive rounds and consider the candidate is the highest-latency flow if its RTT maintains the minimum over the six rounds. We use six probes to effectively mitigate the effects of latency inflation. If the flow's RTT is higher than that in any of the six rounds, we update the flow's RTT to the minimum RTT in the six

rounds to mitigate inflation. We then choose the next flow with the highest RTT among all flows and probe it over another six rounds until the highest-latency flow is found. Similarly, we can determine the lowest-latency flow. It is possible that the RTTs to some addresses are highly varying and the extreme flows cannot be determined within a reasonable amount of probes. We stop probing an address if no extreme flows could be determined after 90 rounds. After probing, each extreme flow will have at least 7 RTTs, which will be used to prune probing results.

3.3.3.4 Pruning Probing Results

Although the minimum RTT of a flow is conventionally used as its latency in existing methods [4], it cannot eliminate latency inflation. Moreover, it could cause flow latency to be underestimated due to measurement errors, i.e., the minimum RTT is an outlier much lower than other RTTs of the flow [4]. The inflation of the highest flow latency and the underestimation of the lowest flow latency both result in the overestimation of latency imbalance. We further prune probing results as below. Flows with inflated latencies typically have varying RTTs over time. We use the variability of RTTs to prune the highest-latency flows with unstable latencies. The variability of RTTs is measured by the median absolute difference (MAD), robust to outliers [49]. We empirically prune a highest-latency flow if it has a MAD $> 2\%$ of the flow's latency. For the lowest-latency flow, we detect measurement errors by checking if the flow's minimum RTT is much lower than other RTTs. We pruned about 5% of lowest-latency flows with the minimum RTT lower than the second minimum one for $> 5\%$ of the flow latency.

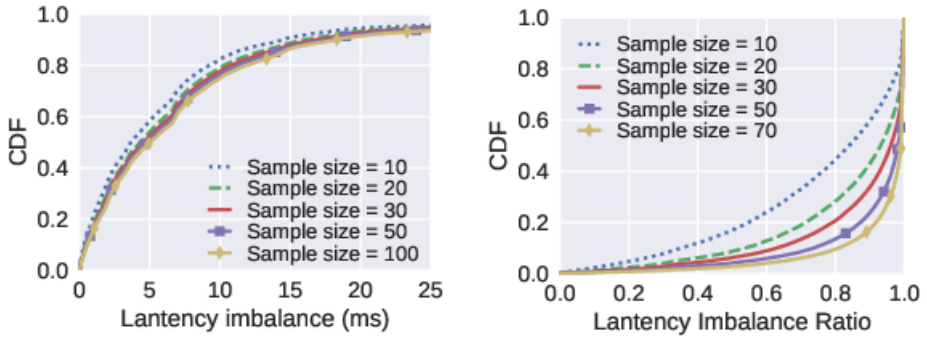
3.3.3.5 Impact of Sampling on Accuracy and Sample Size Selection

To understand how sampling affects accuracy, we want to know the distance between the maximum latency imbalance and the measured one. We begin by estimating the maximum latency imbalance.

Estimating the Maximum Latency Imbalance Obtaining the maximum latency imbalance requires enumerating all LB paths and measuring their path latencies. When 100s of LB paths are common between source-destination pairs, it is impractical to directly measure the maximum imbalance for a large number of destinations. Instead, we approximate the maximum latency imbalance using our method with a large sample size. When the sample size is 100, we can achieve path coverage greater than 95% at the 97% confidence level. In other words, the probability of the uncovered paths only sums up to $\leq 5\%$ at the 97% confidence level. We thus consider these uncovered paths to be rare. We use the latency imbalance measured with 100 flow samples to approximate the maximum latency imbalance and study the accuracy loss due to sampling.

Accuracy of Sampling and Sample Size Selection We evaluate the accuracy of sampling in measuring two types of latency imbalance: 1) *aggregate imbalance*, the collection of latency imbalance for a large number of source-destination pairs and 2) *pairwise imbalance*, latency imbalance for single source-destination pairs. In our experiments, we measure latency imbalance with different sample sizes back to back for the same source-destination pair, where a pair is discarded if path changes are detected while the path is being measured. We use CDF to characterize the distribution of latency imbalance in aggregate and measure the *difference between CDFs* as $|F_1(x) - F_2(x)|$, where $F_1(x)$ and $F_2(x)$ are two CDFs of latency imbalance.

Figure 3.3(a) shows the CDFs of latency imbalance under different sample sizes for 469K source-destination pairs from 6 DCs to random IPv4 addresses. Sampling is in general effective in measuring the aggregate imbalance. Even with 10 samples, the measured CDF only has at most 11% difference from that measured with 100 samples. The maximum difference decreases to 4% for 30 samples and further decreases to 2% for 50 samples. Moreover, the difference between CDFs reaches the maximum at low latency imbalance ($<$ 3ms) and decreases as the latency imbalance increases. Considering both accuracy and



(a) Aggregate Imbalance (b) Pairwise imbalance

Figure 3.3: Impact of sampling on accuracy

efficiency, we use 30 samples when measuring aggregate latency imbalance.

For each source-destination pair, we calculate the ratio of the latency imbalance measured with each sample size to that with 100 samples respectively. Figure 3.3(b) shows the distribution of the ratios under different sample sizes. For 10 samples, only 40% of pairs have a ratio > 0.9. The percentage increases with the sample size and reaches 80% for 70 samples. Compared to aggregate imbalance, we apparently need a much larger sample size to accurately measure pairwise imbalance. We use 100 samples to measure inter-DC latency imbalance in §3.6.

3.3.4 Source and Destination Selection

Source Selection We probe from 16 DCs in 14 cities around the globe, including 4 in Europe, 3 in North America, 6 in Asia and 1 in Australia. All but four of the DCs are located in different cities. Of the four, two are located in London and belong to different cloud providers, and the other two are located in Sydney, also belonging to different cloud providers. We intend to use these four DCs to observe the effect of cloud provider choices (§3.5.2).

Destination Selection Considering the similarity of addresses in the same /24 prefix [26], we probe only one responsive address for each /24 prefix to reduce measurement overhead.

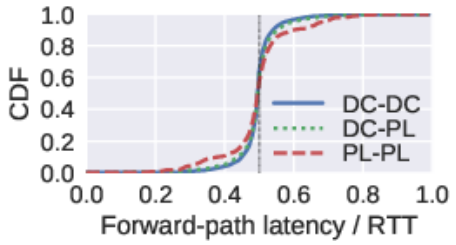


Figure 3.4: Distribution of path asymmetry

For /24 prefixes without any responding addresses, we use the responding router closest to the destination as a *proxy* for the entire /24. To ensure that probes to the proxy follow the same path as probes to the intended /24 prefix, the probes are directed to a (non-responding) destination within the /24 prefix with TTLs expiring at the proxy.

3.3.5 Metrics for One-Way Imbalance

Absolute and Relative Latency Imbalance The latency imbalance defined above is the absolute latency difference between the highest- and lowest-latency paths, called *absolute imbalance*. However, the same absolute imbalance is of higher significance for low path latency than for high path latency. To capture the relative significance of absolute imbalance to path latency, we introduce a new metric called *relative imbalance*, which is the ratio of absolute imbalance to the forward-path latency of the highest-latency path. Relative imbalance is calculated with respect to the forward-path latency because one-way imbalance only occurs on forward paths. Relative imbalance can be *interpreted* as the percentage of latency reduction on the forward paths by switching from the highest-latency flow to the lowest-latency one. As forward-path latency is OWD, which cannot be accurately measured from a single end, we thus estimate forward-path latency from its RTT.

Estimating Forward-Path Latency In 2008, Pathak *et al.* showed that estimating OWD of a path as 0.6 times its RTT would overestimate the OWD in 90% of cases, due to path asymmetry [46]. Conservatively, we want to use 0.6 times RTT of the highest-latency flow as its forward-path latency in computing relative imbalance to avoid overestimation.

Table 3.1: Applications affected by imbalance and baseline

Applications	Latency (OWD)	Accuracy	One-Way Imbalance Baseline
NTP	10s of ms	1s of ms	max(1ms, 5% of OWD)
Geolocation	10s of ms	$<$ 400km	2ms
VoIP (Intl.)	100s of ms	$\times$	20% of OWD

To verify that their results still apply in the current Internet, especially when data center networks are involved, we re-did the experiments in 2019 using public instances in 24 DCs and 31 PlanetLab (PL) nodes that are geographically distributed. An NTP daemon runs on each node to synchronize its local clock and estimates the synchronization error. A sufficient number of flows are exchanged between each pair of nodes to explore alternate paths in between. As in [46], flows with a synchronization error $>$ 3% of their RTTs are pruned. Figure 3.4 shows the distribution of path asymmetry for 35,328 flows from 552 DC-DC pairs, 85,910 flows from 1,354 DC-PL pairs and 48,747 flows from 765 PL-PL pairs. PL-PL pairs generally have the worst path asymmetry, but still have 90% of flows with the ratio of forward-path latency to its RTT $>$ 0.6. This confirms that using 0.6 times RTT of a path to conservatively estimate its OWD is still reasonable in the current Internet. We use "absolute imbalance" and "relative imbalance" to mean absolute and relative one-way imbalance by default unless stated otherwise.

3.4 Impact of One-Way Imbalance on Applications and Baseline

Before looking at the measurement results on one-way imbalance, we first want to intuitively understand its impact on applications. We take three applications (in Table 3.1) as examples and discuss how latency imbalance affects their performance. We set a *baseline* of one-way imbalance for each application to help evaluate the significance of the impact.

Clock Synchronization by NTP To synchronize time, an NTP client estimates the NTP server's clock and adjusts its local clock to the estimated server's clock. The NTP client

sends a message to the server and estimates the server's clock as the sending time of the response plus the estimated OWD from the server to the client. Since the OWD is typically estimated to be half of the RTT, a major source of error for NTP is path asymmetry [50]. We define $\overline{\Delta}_d$ as the average of one-way imbalance on the forward and return paths. Through both analysis and experiments (see §3.7.1), we show that the maximum reduction in synchronization error by considering latency imbalance is $\leq \overline{\Delta}_d$ for all source-destination pairs and $> \overline{\Delta}_d/2$ for most of the pairs.

A study on NTP servers shows that the OWDs between most NTP servers and their peers (other servers used for synchronization) are in the 10s of milliseconds and that the synchronization errors are in the 1s of milliseconds [51]. We simply use 10% of OWDs as reference for typical synchronization errors, where the error is proportional to OWD because longer distance between NTP servers and clients is likely to result in larger synchronization error. We consider *significant* error reduction to be $> 25\%$ of the synchronization error, i.e., 2.5% of OWD. Since the maximum error reduction is mostly at least half the amount of one-way imbalance, we set the baseline of one-way imbalance to the maximum of 1ms and 5% of OWD, where 1ms is used when OWDs are low.

Delay-Based Geolocation Delay-based geolocation is an important technique to build IP geolocation database used by online services to enforce access policy and to determine advertising content [52]. To geolocate an Internet host, delay-based geolocation uses a set of hosts with known locations, called *landmarks*, and measure the latency between each landmark and the host. The measured latency is then used to estimate the distance (r) to the host. From the perspective of each landmark, the host is in a circle with radius r centered at the landmark. The intersection of these circles is the estimated geolocation region of the host. We can see from this process that larger-than-actual latency causes overestimated distance, resulting in a larger estimated geolocation region. Since city-level accuracy is an important metric to the existing geolocation databases [52], we want to the overestimated distance to be at least $<$ 400km, where 400km is simply used to approximate the diameter

of the largest city in the world. We use the worst case to set the baseline such that latency imbalance higher than the baseline would be *significant* to delay-based geolocation. Using the travel speed of signals in fiber-optic cable (2/3 of the speed of light in vacuum), we obtain that it takes 2ms for signals to travel 400km. We thus set the baseline to be 2ms. Since landmarks in the same subcontinental region (e.g., country) of the target host are preferred for better accuracy, we use the average OWD in North America (in the 10s of milliseconds) as a reference for the typical latency in delay-based geolocation [53].

VoIP International VoIP calls have been rising in recent years and have taken a large portion of Internet-based calls (about 46% of Skype calls are international [40]). However, international calls are more likely to experience high RTT and have poor call quality. According to [40], about 5% of Skype calls use paths with RTT over 400ms. For these calls, the fraction of calls with poor quality (with a rating of 1 or 2) can be reduced by $>$ 10% when OWD is decreased by 20%. We set the baseline to be 20% of OWD and consider one-way imbalance larger than the baseline to be *significant* to call quality. Since call quality is more sensitive to latency reduction at high path latencies than at low ones, we would expect that latency imbalance have more significant impacts on calls experiencing high latency. We use 100s of milliseconds as the latency of interest for VoIP.

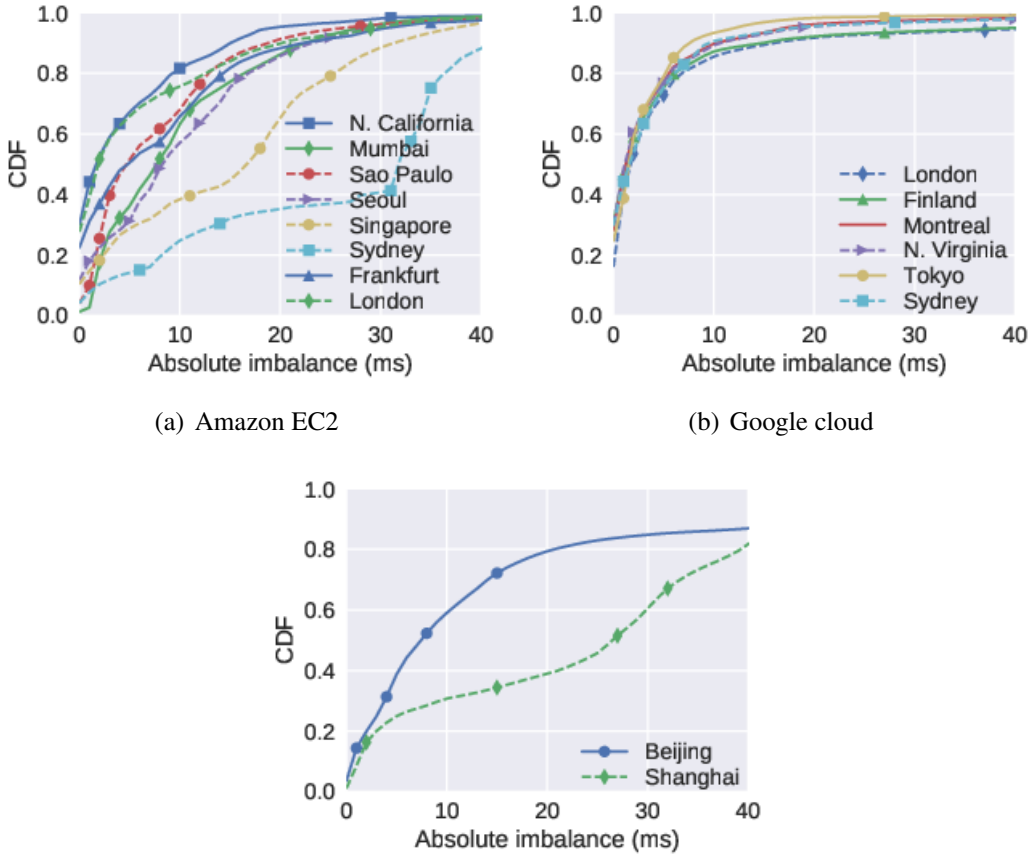
Other Applications Other applications vulnerable to high imbalance include networked AR/VR [54], low-latency gaming [55], real-time video conferencing [56], industrial IoT [10], etc.

Baseline The baseline for each application is summarized in Table 3.1. Latency imbalance affects applications with various operating latencies and applications have different degrees of sensitivity to latency imbalance.

Table 3.2: /24 address prefix reachability

Responsive /24s			Proxies			Probed /24s
Stable	Pruned	Expired*	Stable	Pruned	Expired	
2.7M	598K	736K	254K	92K	29K	10.8M

* No extreme flows can be obtained within 90 probes.



(a) Amazon EC2

(b) Google cloud

(c) Alibaba cloud

Figure 3.5: Distribution of relative imbalance from DCs to public IPv4 addresses.

3.5 Imbalance from Data centers to Public IPv4 Addresses

We probed from the 16 DCs to 10.8M /24 prefixes covered in the CAIDA's *IPv4 prefix-to-AS mapping* database [57]. The instance types we used in Amazon's, Google's and Alibaba's DCs were c5.large, n1-standard-2 and ecs.ic5.large respectively, all of which had 2 virtual CPUs and at least 2GB memory. Table 3.2 characterizes these /24 prefixes by the degree to which we were able to reach them. A /24 prefix is considered "Responsive" if

at least one address within the address space responded to our probe. We are unable to determine imbalance for 736K /24 prefixes and 29K proxies (termed "Expired") because we cannot determine the extreme flows after sending 90 probes. About 598K prefixes and 92K proxies are pruned because their extreme flows (defined in §3.3.3) have unstable latencies. In total, we have latency imbalance measured to roughly 2.7M /24 prefixes and 254K proxies. The rest 6.4M /24 prefixes are not measurable in that we were not able to reach a responsive host and we could not establish 30 flows to the last-responding router proxy.

In the following, we report only on the results of probing the 2.7M /24 prefixes and 254K proxies with stable imbalance from the 16 DCs, or about 48M source-destination pairs in total. These /24 prefixes are widely spread covering 96% of countries in the world and 68% of ASs in the CAIDA's prefix-to-AS database.

3.5.1 Distribution of One-Way Imbalance

Absolute Imbalance Figure 3.5 shows the distribution of absolute imbalance seen from the perspective of each DC. Latency imbalance is prevalent between DCs and public addresses. Even the DC with the narrowest distribution (Google Tokyo) has absolute imbalance $>$ 5ms to about 20% of probed addresses. Moreover, latency imbalance has a wide range; the distributions of absolute imbalance for most DCs are long-tailed. The narrowest distribution (Google Tokyo) has a tail covering 7% of addresses with absolute imbalance $>$ 10ms. The absolute imbalance for Amazon's Sydney and Alibaba's Shanghai DCs are more widely distributed with shorter but heavier tails and have absolute imbalance $>$ 20ms to more than 60% of addresses. Moreover, latency imbalance differs significantly between cloud providers, as will be studied further in §3.5.2.

Relative Imbalance Figure 3.6 shows the distribution of relative imbalance under different ranges of OWDs for 5 DCs. We do not compute relative imbalance for low OWDs ($<$

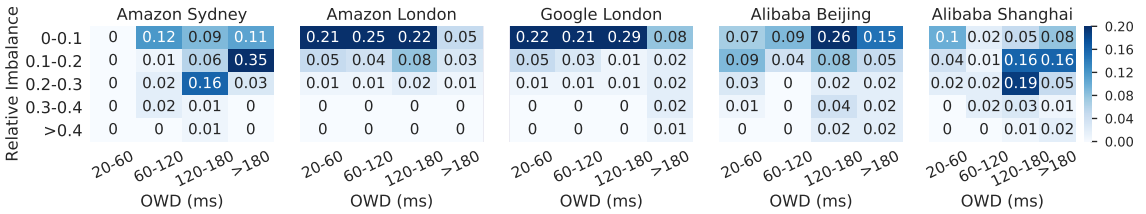


Figure 3.6: Distribution of relative imbalance under different ranges of OWDs

20ms) to mitigate the effects of measurement errors and our dataset only includes 5% of source-destination pairs with OWD $<$ 20ms. Figure 3.6 shows that Amazon's Sydney and Alibaba's Shanghai DCs, with the highest two absolute imbalance, have most of the high imbalance occurring at high path latencies (OWD $>$120ms). If we only consider addresses with OWD $>$ 120ms from DCs, Amazon's Sydney and Alibaba's Shanghai DCs have relative imbalance $>$ 0.2 to about 26% and 41% of them, and the other 3 DCs also have relative imbalance $>$ 0.2 to 7%, 13% and 21% of them respectively. This implies that traffic in the WANs between DCs and public addresses would experience significantly less latency by choosing a path of lower latency. Potential applications include using DCs as relays for international VoIP calls [40] or as globally accessible cloud VPN [58].

Besides imbalance in high path latencies, most DCs also have large relative imbalance to a significant portion of addresses with low path latencies (OWD $<$ 60ms). Except for Amazon's Sydney DC, other DCs have relative imbalance $>$ 0.1 to about 6% (Amazon London) to 13% (Alibaba Beijing) of all reported addresses. Amazon's Sydney DC has nearly zero relative imbalance in low path latencies due to a small number of responsive domestic addresses ($<$ 10K). If we only consider addresses with OWD $<$ 60ms from DCs, Amazon's and Google's London DCs have relative imbalance $>$ 0.1 to about one-fifth of them, and Alibaba's Beijing and Shanghai DCs have imbalance $>$ 0.1 to no less than one-third of them. This implies that the accuracy of delay-based geolocation and NTP could both be improved for a large portion of addresses by using lower-latency paths. Potential applications include geolocating cloud VPN servers [42] and time synchronization for

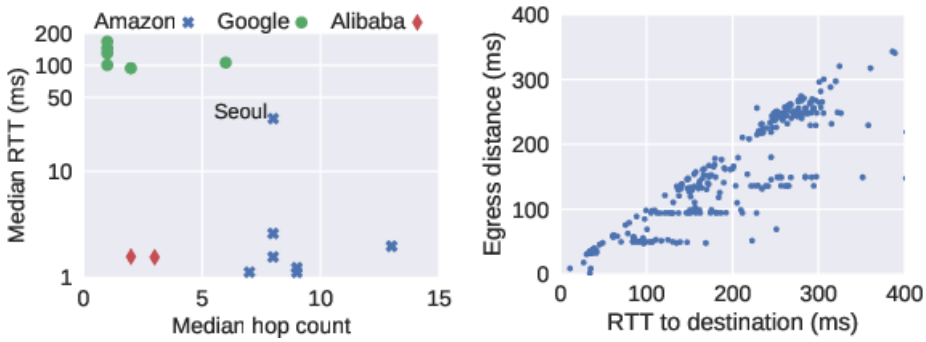
cloud-enabled IoT, e.g., real-time earthquake detection [59].

3.5.2 Why Cloud Providers Differ in Latency Imbalance?

As shown in Figure 3.5, Google and Amazon both have DCs in Sydney and London, but Google's Sydney DC has much lower imbalance than Amazon's Sydney DC. To understand the causes, we divide the distance from a DC to a destination into 1) the distance from the DC to the first router in its neighbor AS, called *egress distance*, and 2) that from the router to the destination. Since cloud providers operate DCs and routers within the egress distance in their own ASes, a longer egress distance indicates more control over the entire path to the destination. We will show that Google's DCs have much longer egress distances than other cloud providers' DCs.

Given the path from a DC to a destination, the difficulty in determining the egress distance is to locate which router along the path is the border router of the DC's neighbor AS. Inferring the border routers of ASes is a complicated problem and simply mapping a router to the origin AS of its interfaces could be incorrect for many reasons [60]. To obtain accurate router-to-AS mapping, we considered using CAIDA's ITDK datasets including Internet-scale routers' ownership, inferred from traceroute paths from vantage points to destinations. However, as traceroute discovers router interfaces facing the vantage points, only about 20% of router interfaces in our dataset are covered by the ITDK dataset [61], where our dataset includes the traceroute paths from each DC to responsive addresses. We thus use *bdrmapIT* to obtain routers' ownership using our dataset, together with other inputs including prefix-to-AS mapping supplemented with RIR delegations [62, 63, 64, 65, 66], IXP prefixes [67, 68, 69] and AS relationships [70]. Using the derived router-to-AS mapping, we first identify the interdomain link (the near side in the DC's AS and the far side in its neighbor AS) from the traceroute path for each DC-destination pair and then measure the egress distance in both hop count and RTT.

Figure 3.7(a) shows the median egress distance (measured in hop count and RTT) from



(a) Distance to neighboring ASs (b) Egress distance of Google Tokyo

Figure 3.7: Imbalance difference between cloud providers

each DC to responsive addresses. All Google DCs have the median RTT $>$ 94ms, while Amazon's and Alibaba's DCs (except for Amazon Seoul) have the median RTT $<$ 3ms. Looking into the egress distances of Google's DCs, we found that egress distance generally increases with the distance (measured in RTT) from the DC to destinations and is capped at certain values for some destinations. Figure 3.7(b) shows the relation between egress distance and distance to destinations for 1,000 destinations seen from Google Tokyo. The egress distance is capped at around 50ms, 100ms and 140ms for some destinations. This indicates that Google uses its own private WANs to route packets to an egress point close to traffic destinations. This observation is further confirmed by Google's own documentation on network service tiers [71], where traffic in premium network tiers is forwarded as close to the destination's ISP and all our VMs in Google cloud are in premium network tiers. As a result, the imbalance incurred in Google's own networks dominates the latency imbalance for the entire paths. For instance, for 77% of destinations probed from Google Tokyo, more than 60% of the entire path latency is spent on travelling from the DC to its neighbor ASes. Despite the long egress distance, the latency imbalance incurred by Google's networks is low. The worst DC has absolute imbalance $<$ 10ms before entering its neighbor ASes for at least 88% of destinations. This shows that the low imbalance of Google's DCs is due to the use of well-balanced long-distance paths in Google's own networks.

As shown in Figure 3.7(a), Amazon's and Alibaba's DCs are close to neighbor ASes

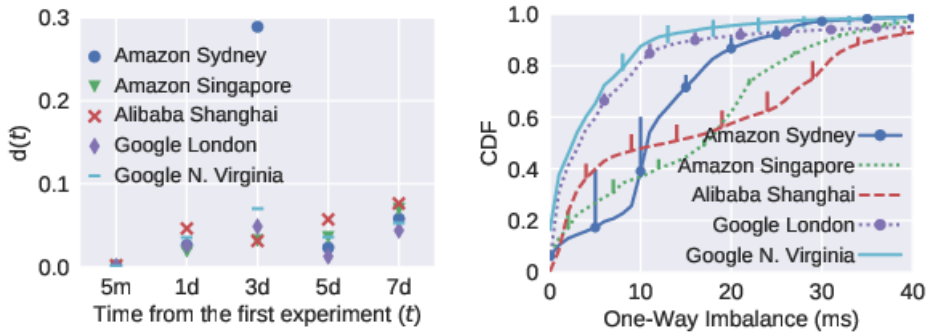
and thus should forward packets to the public Internet close to traffic sources. The low egress distance of Amazon's and Alibaba's DCs results in low latency imbalance to neighbor ASes, where all of these DCs have the median absolute imbalance < 0.8ms. Amazon's Seoul DC has the median RTT equal to 32ms, but the median absolute imbalance to neighboring ASs is only 0.2ms. Since the latency imbalance before entering the neighbor ASs is low, we can conclude that the major imbalance of Amazon's and Alibaba's DCs to destinations occurs in the public Internet.

3.5.3 Is Latency Imbalance Stable Over Time?

We consider *aggregate latency imbalance*, which is a collection of latency imbalance measured to a set of destinations in a single experiment. We want to know if aggregate latency imbalance is stable over experiments. Specifically, for each DC, we first measure latency imbalance to the same set of destinations in experiments starting at different times. We then use the *first* experiment as reference and compare the aggregate latency imbalance in subsequent experiments respectively with that in the first experiment to check if the difference increases with time or remains stable. We use a CDF to characterize the distribution of latency imbalance in aggregate and measure the *difference between aggregate latency imbalances* by the maximum difference between their CDFs. Suppose the first experiment starts at time 0 and that $F_t(x)$ denotes the percentage of destinations with latency imbalance $\leq x$ in the experiment starting at time t. The difference between CDFs of latency imbalance in the first experiment and the experiment at time t is then $d(x,t) = F_t(x) - F_0(x)$ with the maximum equal to $d_{max}(t) = \max_{x \geq 0} |d(x,t)|$ over all latency imbalance.

Figure 3.8(a) shows the $d_{max}(t)$'s for 5 representative DCs in experiments starting at different t's over a week, where latency imbalance is measured from each DC to 200K[1] destinations. Except for Amazon Sydney, all other DCs have stable distributions of latency

[1]We conduct our medium-sized experiments at the scale of 100s of thousands of destinations, a typical scale for many existing measurement campaigns [9, 72]. As a reference, 200K random destinations cover roughly about 87% of countries, 13K ASes.



(a) Latency imbalance over time (b) Distribution of latency imbalance

Figure 3.8: Stability of latency imbalance over time

imbalance across experiments with $d_{max}(t) < 0.08$ for all t's. We further examine if $d_{max}(t)$'s are reached at low or high latency imbalance. Figure 3.8(b) shows the distribution of latency imbalance in the first experiment of the week with error bars being $d_{max}(x) = \max_{t>0} |d(x,t)|$, the maximum difference between CDFs of latency imbalance over all experiments in the week at latency imbalance equal to x. We find that the differences between experiments are large (the high error bars) mostly at low latency imbalance and decrease as latency imbalance increases. From Figure 3.8(b), we thus can conclude that except for Amazon Sydney, other DCs have stable latency imbalance for at least one week. To check if these DCs have stable latency imbalance for longer periods of time, we compare the first experiment in Figure 3.8 (starting on September 1, 2019) with that in Figure 3.5, conducted in February, 2019. We find that these DCs (except for Amazon Sydney) still have small $d_{max}(t) < 0.10$ occurring at low latency imbalance. This indicates that most of the DCs have stable aggregate latency imbalance even over months.

Compared to other DCs, Amazon Sydney has more variable aggregate latency imbalance due to changing network conditions. At t = Day 1, $d_{max}(t)$ is reached at latency imbalance equal to 9ms. Comparing the destinations with latency imbalance around 9ms (specifically from 8 to 11ms) in all experiments for Amazon Sydney, we found that the experiment at Day 1 only shared $< 20\%$ of these destinations with other experiments, while other experiments shared $> 80\%$ of these destinations among themselves. Considering

all experiments probe to the same set of destinations, this indicates that the network conditions from Amazon Sydney to a large number of destinations changed temporarily at t = Day 1 and reverted back to the previous conditions at later days. The large difference between Amazon Sydney's latency imbalance in Figure 3.8 and Figure 3.5 is likely due to network condition changes (e.g., caused by traffic engineering) or the use of different source addresses in the two experiments, resulting in LB paths traversing different parts of the Internet.

3.5.4 Diving Deeper Into One-Way Imbalance

As mentioned in §3.2.2, imbalanced diamonds with unequal latencies on their path segments are the causes for one-way imbalance. In this section, we want to 1) verify that our measured one-way imbalance is indeed due to imbalanced diamonds, 2) analyze imbalanced diamonds to understand the causes, and 3) discuss the challenges of load balancing in terms of latency imbalance.

3.5.4.1 Verifying Our Measured One-Way Imbalance

We want to verify that our measured one-way imbalance reflects the imbalance experienced by LB paths rather than measurement artifacts. We first collected 10K samples of one-way imbalance from each prober to its nearby routers at different times throughout our experiments (in §3.5.1) and found that the average one-way imbalance is $<$ 0.2ms for all probers. This confirms that the probers have almost no impact on the measured imbalance. Measurement errors could also come from the destinations due to 1) slow ICMP responses, generated in the slow-path [38], and 2) irregular router behaviors, e.g., adding IP options to ICMP responses or modifying their contents [8]. It is challenging to directly estimate these errors [73] and we instead verify the accuracy of our measured imbalance to a destination using its last-hop router(s) as follows.

The key idea is that if latency imbalance between the extreme flows (i.e., the highest-

and lowest-latency flows) is caused by traversing an imbalanced diamond, the latency imbalance will be carried to hops after the diamond. In other words, if our measured imbalance is due to imbalanced diamonds, we would expect the latency differences between the extreme flows seen at a destination (denoted as I_1) and its last-hop router(s) (denoted as I_2) to be similar, unless non-negligible latency imbalance occurs in between. In our experiments, we only use destinations in the same /24 subnet as their last-hop routers, where the distance between them is likely to be close and results in negligible imbalance. We define the *error* of the measured imbalance as the absolute difference between I_1 and I_2. We found that our measured imbalance had an error $<$ 1ms for 95% of source-destination pairs of all DCs, where the worst DC (Google Tokyo) had an error $<$ 1ms for 81% of pairs and an error $<$ 2ms for 89%. Looking into the worst DC, we confirmed that accuracy of path latency was not an issue for the pairs with an error $>$ 1ms, because the back-to-back imbalance samples to the same destination had a difference $<$ 0.5ms for all destinations. We further checked that only 1% of these pairs experienced ICMP responses with different checksums. The error thus should be mainly due to the imbalance between the destinations and their last-hop routers. This implies that our measured imbalance should have higher accuracy than the reported. Since from the perspective of destinations, the imbalance possibly caused by middleboxes or firewalls is the same as that caused by unequal path lengths, we consider them as part of imbalanced diamonds.

3.5.4.2 Top Imbalanced Diamonds and Potential Causes

We first discuss how to discover imbalanced diamonds and then the potential causes for imbalanced diamonds.

Discovering Imbalanced Diamonds Figure 3.9 shows an example of discovering imbalanced diamonds. By merging the common hops of the extreme paths (solid and dashed lines) in the forward direction, we can identify diamond $[A1, A4]$ from the topology. Sup-

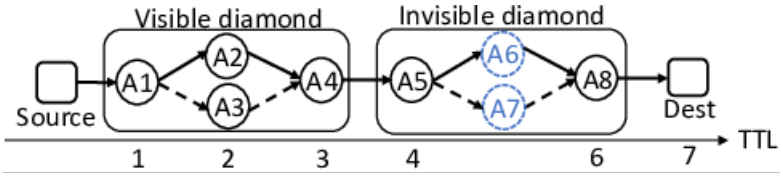


Figure 3.9: Visible and invisible diamonds

pose we want to estimate the *imbalance of a diamond*, i.e., the latency difference between the diamond's path segments. We first measure the latency difference between the extreme paths at both the convergence and divergence points of the diamond, denoted as d_1 and d_2 respectively. The difference between d_1 and d_2 is then the change in latency difference due to traversing the diamond, equal to the imbalance of the diamond. However, not all diamonds can be discovered by topology. Routers could be unresponsive or even invisible to TTL-limited probes, e.g., routers in MPLS networks [74]. We refer to diamonds discoverable by topology as *visible diamonds* and those not as *invisible diamonds*. Diamond $[A5, A8]$ in Figure 3.9 is an invisible diamond with unresponsive routers $A6$ and $A7$. We identify an invisible imbalanced diamond from its imbalance.

Imbalanced Diamonds and Potential Causes For each source-destination pair, we first discover visible diamonds and group the hops in the rest of path segments into pairs of two, e.g., $A4$-$A5$-$A8$ in Figure 3.9 is divided into $A4$-$A5$ and $A5$-$A8$. We consider each pair as a potential invisible diamond and calculate its imbalance. Considering that the error of our measured imbalance is $<$ 1ms at one hop (§3.5.4.1), we would expect an error at most 2ms for diamond imbalance. We use a relative large threshold, 5ms, to determine imbalanced diamonds and found 4.1M visible and 2.1M invisible imbalanced diamonds. Although invisible diamonds can be identified from the topology, it is error-prone to identify the actual end points of a diamond at the IP level due to IP aliasing [8]. Since we do not use alias resolution techniques, we limit our focus on invisible diamonds with their end points having consecutive TTLs such that no aliases exist in the diamonds. We further require each invisible diamond to have at least 10 samples and use the median sample as the diamond imbalance. After pruning, we obtain 65K invisible imbalanced diamonds, where 32% have

imbalance $<$ 10ms and 50% have imbalance between 10ms and 20ms. For such invisible diamonds, load balancing between their end points could occur between multiple paths in MPLS networks [75] and between links in link aggregation groups (LAGs) [38].

Using the database of router ownership in §3.5.2, we can categorize these diamonds into two types: 1) *inter-AS*, if the convergence and divergence points are in different ASes and 2) *intra-AS*, otherwise. We found that 80% of these diamonds are intra-AS and that almost half of these intra-AS diamonds are from the top 10 imbalanced ASes (containing the most number of intra-AS diamonds), including 4 Tier-1 providers: AS174 (Cogent), AS7018 (ATT-INT), AS2914 (NTT America) and AS6762 (Telecom Italia). We confirmed that 7 of the top 10 imbalanced ASes use MPLS from existing studies of MPLS networks [76, 74]. Further, invisible diamonds directly between border routers were found in all of these ASes except AS7922 (Comcast Backbone) and AS4230 (Embratel). Since MPLS tunnels are widely used inside ASes to connect border routers [75], these invisible diamonds are very likely due to invisible MPLS tunnels with imbalanced load balancing. AS12389 (Rostelecom) and AS6762 have respectively 68% and 100% of invisible diamonds between border routers, which could indicate wide deployment of invisible MPLS tunnels. Other than invisible MPLS tunnels directly between border routers, we can also see invisible MPLS tunnels with the last hop of the egress border router visible to traceroute when the Penultimate Hop Poping (PHP) is activated [74]. We found that 6% and 30% of invisible diamonds in AS4230 and AS12389 respectively used such invisible MPLS tunnels. Identifying other types of diamonds requires careful probing analysis [77] and validation requires information from the operators. Both are subject to future research. We also found 8,601 inter-AS invisible diamonds spreading across 3,340 pairs of ASes, of which 7.6% are peers. This indicates that latency imbalance also occurs between border routers of inter-domain links. Both LAG and multi-paths in invisible MPLS networks are the possible causes for the latency imbalance.

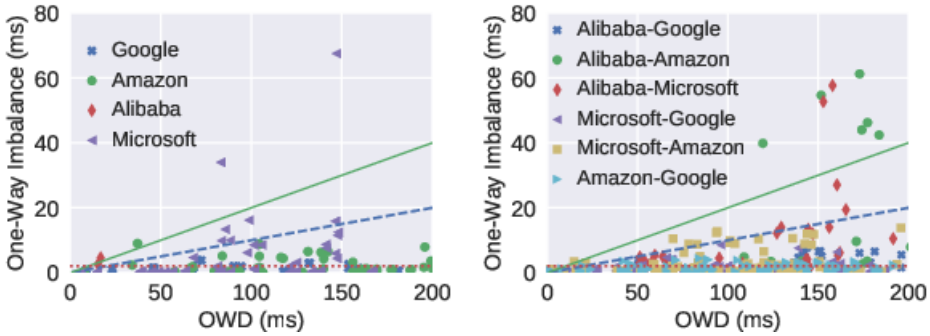
Challenges of Load Balancing The definitions of path cost are diverse and a certain definition of path cost (e.g., latency) can at best ensure the related performance metric to be equal across LB paths. However, applications have a wide range of requirements from latency- to bandwidth-sensitive. In addition to changing network conditions, it is impossible to have LB paths with equal performance for all applications. LB paths optimized for equal bandwidth may differ greatly in latency [78]. When performance imbalance is significant, undifferentiated selection of LB paths, as done by hashing-based algorithms, would degrade application QoE. Latency imbalance emphasizes the importance of differentiated services. Diffserv-aware MPLS traffic engineering (MPLS-TE) is a good option for providing differentiated services in the Internet. However, MPLS-TE generally monitors and computes label switching paths at the router level, while load balancing could occur at the link level [75]. Since multiple links could exist between routers, it is challenging to monitor link-level path performance.

3.6 Imbalance Between Data centers

Inter-DC WANs, e.g., Microsoft's SWAN [79] and Google's B4 [80], are designed to carry traffic between DCs for various interactive services highly sensitive to latency [79]. We thus want to understand the latency imbalance between DCs.

3.6.1 Data Collection

To have a more rounded view of latency imbalance between DCs, we added 8 Microsoft Azure's DCs located in New South Wales (Australia East), Tokyo, Chennai (South India), California, Virginia, Toronto, London, and Paris. We refer to two connected DCs as a *DC pair*. Due to path asymmetry, connectivity of DC pairs is *directional*. We denote the set of paths from DC a to DC b as $a \mapsto b$. Along with the 16 DCs in Figure 3.5, we have 552 DC pairs amongst the 24 DCs. We sample the latency imbalance between each pair every 20 minutes for one day using the same methodology in §3.3. We use 100 flows for each



(a) Intra-cloud (b) Inter-cloud

Figure 3.10: Intra-cloud and inter-cloud latency imbalance

source-destination pair as studied in §3.6. No latency imbalance sample can be obtained if no extreme flows are found within the first 160 probes or extreme flows have unstable latency. After the data pruning in §3.3.3, we have 5 pairs with < 10 samples left. We prune these pairs and analyze the stability of latency imbalance for the rest of pairs. To avoid outliers, we use the median latency imbalance sample as representative for each DC pair in our study.

3.6.2 Intra- and Inter-Cloud Latency Imbalance

We separate DC pairs into intra- and inter-cloud pairs for analysis. An intra-cloud pair is between DCs belonging to the same cloud provider, while an inter-cloud pair consists of two DCs from different cloud providers.

3.6.2.1 Intra-Cloud Latency Imbalance

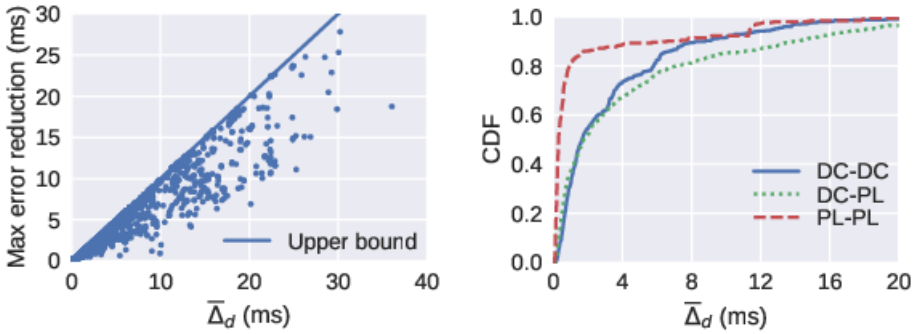
Figure 3.10(a) shows the one-way imbalance of all intra-cloud pairs with respect to their OWDs, where the three lines present one-way imbalance equal to 2ms, 10% of OWD and 20% of OWD respectively. In general, Google has the least latency imbalance with only 10% of its intra-cloud pairs having imbalance slightly $>$ 2ms, while Microsoft has 7 out of 56 pairs with imbalance $>$ 10% of their OWDs. Of the 7 pairs, 2 pairs have imbalance as high as 34ms and 67ms, much larger than 20% of their respective OWDs,

and they have very stable latency imbalance over time. The high latency imbalance would cause Microsoft's traffic between the same DC pair to experience disparate path latencies, causing inconsistent performance for clients in latency-sensitive applications. We only have two intra-cloud pairs between Beijing and Shanghai for Alibaba with OWD around 15ms and one-way imbalance $>$ 2ms.

We further analyze the paths between DCs as in §3.5.2. We found that all intra-cloud pairs have all discovered intermediate hops in the ASs of their respective cloud providers. As Google's private WAN is well-balanced, its intra-cloud pairs have low latency imbalance. Looking into Microsoft's pairs with high imbalance ($>$ 10% of OWD), we found that 1) two pairs with the highest imbalance use flows of disparate path lengths (measured in the total hop count), where the lowest-latency flow takes a path of length only about half of that of the highest-latency flow and that 2) even when paths have similar lengths, the long-distance inter-continental paths are not well-balanced and cause most of the imbalance.

3.6.2.2 Inter-Cloud Latency Imbalance

The percentage of inter-cloud pairs with high imbalance ($>$ 10% of OWD) is similar to that of intra-cloud pairs. Pairs involving Google's DCs generally have lower latency imbalance than other pairs, because Google uses its well-balanced paths to directly forward inter-cloud traffic from its own ASs to the ASs of the Amazon's and Microsoft's DCs. The traffic from Google's DCs to Alibaba's DCs first traverses the major ISPs in China before entering Alibaba's ASs, but most of the time is spent on traveling Google's own networks, resulting in low latency imbalance. Pairs involving Alibaba's DCs have higher latency imbalance than other pairs mainly because the long-distance paths between Alibaba's and other DCs are not well-balanced. Looking at the AS-level paths between Alibaba's and Amazon's DCs, we found that the high imbalance from Alibaba's to Amazon's DCs largely occurs at the inter-AS connections between AS 4837 (China Unicom) and AS2914 (NTT), while the high imbalance from Amazon's to Alibaba's DCs mainly occurs within AS 4134



(a) Max error reduction (b) Average one-way imbalance

Figure 3.11: Maximum error reduction for clock offset in NTP

(China Telecom), a major ISP in China.

3.7 Applications

As discussed in 3.5.4.2, it is inherently difficult to guarantee that LB paths have similar latency at the network layer or below. In this section, we will evaluate the impact of latency imbalance on three applications and discuss potential solutions.

3.7.1 Clock Synchronization by NTP

An NTP client synchronizes time by adjusting its local clock to the NTP server's clock. Suppose that of all LB paths, the i-th path is used for time synchronization with its forward- and return-path latency equal to d_i^F and d_i^R respectively. The error in estimating the server's clock time is equal to the estimation error of the OWD, i.e., $|d_i^F - d_i^R|/2$ [50]. When multiple LB paths exist, choosing forward and return paths that are more symmetric can reduce the estimation error of the OWD. Suppose A is the set of LB paths. The *maximum* reduction of the estimation error is $\left(\max_{i\in A}\{|d_i^F - d_i^R|\} - \min_{j\in A}\{|d_j^F - d_j^R|\}\right)/2$.

Upper Bound of Error Reduction Suppose there are n LB paths between the NTP client and the NTP server and that the maximum and minimum estimation errors occur at the i-th and j-th LB paths respectively. The maximum reduction of estimation error (denoted as Δ)

then becomes $\left(|d_i^F - d_i^R| - |d_j^F - d_j^R|\right)/2$. Using triangle inequality, we have that

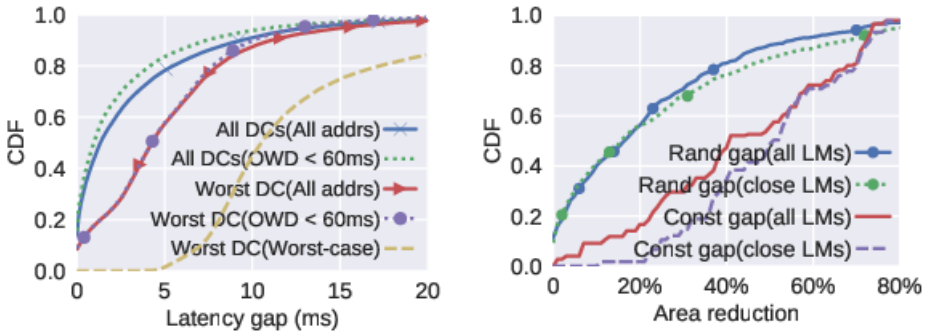
$$\begin{aligned}\Delta &\leq \left|(d_i^F - d_i^R) - (d_j^F - d_j^R)\right|/2 \leq \left(|d_i^F - d_j^F| + |d_i^R - d_j^R|\right)/2 \\ &\leq (\Delta_F + \Delta_R)/2,\end{aligned}$$

where Δ_F and Δ_R are the one-way imbalance on the forward and return paths respectively.

Relation Between One-Way Imbalance and NTP Accuracy Using the same dataset of OWDs between the 55 nodes (24 data center instances and 31 Planetlab (PL) nodes) in §3.3.5, we calculate the maximum error reduction for each pair of nodes as defined above. Figure 3.11(a) shows the relation between the maximum error reduction and $\overline{\Delta}_d$ for all pairs of nodes. The maximum error reduction can be as high as 27ms and more importantly, most of the maximum error reductions are larger than $\overline{\Delta}_d/2$. Figure 3.11(b) shows the CDF of the average latency imbalance of different types of pairs. The latency imbalance between PL-PL pairs is much lower than that between pairs involving DCs.

Practical Impact Although NTP uses DNS-based load balancing to serve clients based on locality, NTP servers could, in practice, still see a median OWD of 175-300ms to world-wide clients [81]. Since long-distance paths are more likely to have high latency imbalance, considering latency imbalance in NTP would benefit a large portion of clients. Moreover, NTP servers could also see traffic from cloud providers [82], and popular IoT clouds (e.g., AWS IoT [83]) would synchronize their servers with cloud-connected IoT devices. The communications between clouds and public Internet would experience similar latency imbalance as characterized in §3.5.1.

Potential Solution Using the same dataset, we simulate the schemes of choosing a random path and the lowest-latency path for time synchronization and repeat the experiment 1,000 times. In the experiments, only source-destination pairs with potential improvement



(a) Distribution of latency gap (b) Distribution of area reduction

Figure 3.12: Impact of latency imbalance on geolocation

($\overline{\Delta}_d \geq$ 1ms) are used and 10 random flows are used in finding the lowest-latency path. We found that using the lowest-latency path could achieve on average 64% of the maximum error reduction and reduce the standard deviation of synchronization error by 1ms compared to using a random path. However, choosing the lowest-latency path incurs extra overhead. This solution works for any client-server pair, complementary to stratum server selection.

3.7.2 Delay-Based Geolocation

Delay-based geolocation techniques commonly use `ping` to measure latency and use the measured latency to estimate the distance from landmarks to the host. However, under latency imbalance, `ping` may fail to measure the *minimum path latency* (minRTT), resulting in overestimated distance. We refer to the measured minRTT by `ping` as *ping-time* and the difference between the ping-time and the minRTT as *latency gap*. We show how the accuracy of CBG++ [42], the state-of-the-art geolocation technique, can be improved by considering latency imbalance.

We first present a global view of latency gap and then discuss how latency gap affects geolocation accuracy. The global view of latency gap is from the perspective of DCs, providing insights to the geolocation for cloud-based applications, e.g., location verification of cloud data and proxies [42, 39].

A Global View of Latency Gap We measure ping-time and the minRTT, in parallel, from DCs in 15 cities around the globe to 400K random destinations. We use Amazon's and Alibaba's DCs as vantage points to reflect more closely traffic sent from client hosts in the Internet (see § 3.5.2). For each DC-address pair, we repeat the method in §3.3.3 with UDP, TCP and ICMP probes respectively to discover as many alternate paths as possible and use the minimum RTT sample as minRTT. We use `fping`, an extension to `ping`, for fast probing destinations in parallel. In geolocation applications, *ping-time* is typically the minimum of several RTT samples to mitigate queueing delay [84, 16]. In our experiments, we use more samples (i.e., 30 samples) to further exclude queueing delay for *ping-time*. Figure 3.12 shows the distribution of latency gaps under different cases. The worst DC (with the largest median latency gap) has latency gap $>$ 5ms to 40% of addresses. As landmarks close to the host are found to be more effective in geolocation [42], we further look at DC-address pairs with OWD $<$ 60 ms and find that the distribution of latency gap is similar. This indicates that latency gap occurs at latencies of a wide range. The worst case assumes that `ping` probes sent from the worst DC always take the highest-latency path, indicating the maximum possible latency gap. The figure shows that the worst-case latency gap is much larger than those under other cases.

Impact of Latency Gap on Geolocation Accuracy We simulate delay-based geolocation using the real-world data of 55 PlanetLab nodes, where their locations can be found in [85] and the RTT samples between them are obtained from the iPlane project dataset [86]. To simulate the process of geolocating an Internet host, we randomly select one PlanetLab node as the host and use the rest as landmarks. Based on the latencies between landmarks and the host, we use CBG++ to predict a region containing the host. To study the impact of latency gap, the region is predicted under both minRTT (without latency gap) and ping-time (with latency gap). The minRTT between each landmark and the host is the minimum RTT sample between them in the iPlane project dataset and the ping-time is simulated as

the minRTT plus latency gap. We use the DC-address pairs in Figure 3.12(a) to resemble the landmark-host pairs in applications using existing measurement platforms to geolocate cloud addresses, where the host resides in a DC and landmarks are common Internet addresses outside the cloud. We consider two geolocation schemes: 1) all landmarks are used for geolocation, where the latency gap follows the distribution when all DC-address pairs are included in Figure 3.12(a) and 2) only landmarks close to the host (with OWD < 60) are used for geolocation, where the latency gap follows the distribution when only DC-address pairs with OWD < 60 are included.

The accuracy of geolocation is typically measured by the area of the predicted region. Since the predicted region under ping-time is no less than that under minRTT, we consider the percentage of *area reduction* as the accuracy improvement by considering latency imbalance. We repeat the geolocation process above for 1000 random hosts (selected with replacement from the 55 PlanetLab nodes) under each geolocation scheme. Figure 3.12(b) shows the distribution of area reduction (in percentage), where area reduction is $> 20\%$ for 40% of hosts. The area reduction would be larger if the host is in the worst DC. We further verify if latency imbalance larger than the baseline in Table 3.1 is significant to geolocation. As latency gap includes latency imbalance on both forward and return paths, we set latency gap to 4ms for each landmark-host pair, equivalent to one-way imbalance equal to 2ms (the baseline). About 85% of hosts have area reduction $> 20\%$ for both geolocation schemes.

Practical Impact The results above are obtained using our measured imbalance from the cloud to the public Internet. Applications that possibly see similar improvement include: 1) geolocating VPN services to enforce accurate geographic access control [42] and 2) geolocating data in the cloud to verify that replicas are in diverse geolocations [87] or to prevent storage providers from relocating the data [39]. As imbalanced diamonds exist in the public Internet (§3.5.4.2), other geolocation applications would also benefit.

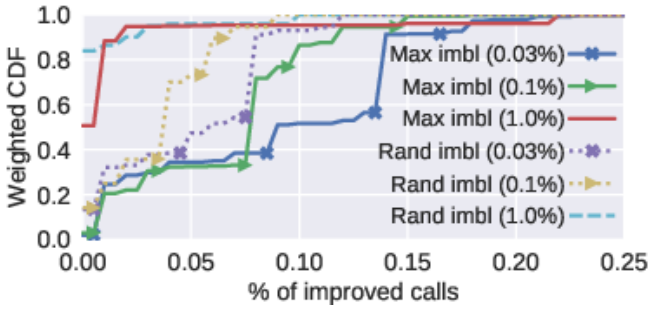


Figure 3.13: Weighted distribution of the percentage of improved calls for country pairs

Potential Solution The experiments above used our method to measure the minRTT (see Appendix 3.7.2 for details), where UDP packets with random ports are used to uncover LB paths and the minimum path latency is used as the minRTT. `Ping` uses ICMP echo packets, which have higher reachability than UDP packets [72], but UDP packets may uncover more LB paths than ICMP echo packets as more per-flow load balancers balance UDP packets than ICMP echo packets. Moreover, sending UDP packets with random ports, rather than incrementing ones as traceroute does, helps UDP packets uncover LB paths more efficiently.

3.7.3 VoIP

Long-distance Internet-based calls have been rising in recent years with 46% of Skype calls being international [40]. However, long-distance Internet paths could have high RTT, packet loss and jitter. Distributed DCs around the globe are envisioned as relays to provide high-quality paths [40]. In this context, we show how to improve call quality when latency imbalance is considered.

Experiments Our experiment simulates a relay network formed by 24 DCs around the globe, where a path between the caller and callee consists of multiple *logical links* (including a caller-relay link, a relay-callee link and relay-relay links, if used) and each link could include multiple alternate paths. Our simulated relay network is similar to the ones used

by Google+ Hangout and Skype [88], except that we use relay networks to improve performance, not just to provide connectivity between clients. The *best path* is defined to have the minimum latency between the caller and callee. Without considering latency imbalance, each link randomly uses one of the alternate paths. When latency imbalance is considered, each link always uses the lowest-latency path. We select the best paths in both cases and consider the difference between call quality as the improvement by considering latency imbalance. The call quality is estimated using an analytical model (E-Model defined by the ITU [89]) that computes the Mean Opinion Score (MOS) from network metrics, where MOS values range from 1 ("unacceptable") to 5 ("excellent"). Since our focus is on the impact of latency imbalance on call quality, we fix the jitter parameter in the E-model to the default value [90] and assume constant packet loss for all paths.

We randomly select callers and callers in different countries for each call and consider that call quality is improved if its MOS increases by one or more levels. We simulate one million calls in total with three packet loss rates (0.03%, 0.1% and 1%) and two types of latency imbalance: 1) *random imbalance*, where an alternate path is randomly selected for each link, and 2) *maximum imbalance*, where the highest-latency path is selected for each link. We group calls from the same pair of countries (*country pair* henceforth) and calculate the percentage of calls with improved quality for each country pair. Figure 3.13 shows the distribution of the percentage of improved calls weighted by the number of calls in respective country pairs. The percentage of improved calls decreases as the packet loss increases and becomes the major bottleneck. Nonetheless, calls between some countries still benefit significantly from latency reduction. When packet loss is 0.03% (packet loss between DCs [91]), the percentage of improved calls is about 8% under random imbalance and 14% under maximum imbalance for 10 country pairs comprising 40% of total calls.

Potential Solution Requests on multiple flows can be sent simultaneously to the relay or destination, and the first received request is considered taking the shortest path. Although

we only consider stable path latency, this method is also useful to explore the multi-path capability to reduce latency inflation as in [92].

3.8 Discussion

Our methodology and measurement results have several limitations.

Limited View: All of our hosts are in DCs, but ideally we want more heterogeneous ones such as residential hosts. Latency imbalance in public-only Internet may differ from that between the cloud and the public Internet.

Measurement Bias: Our method is biased towards destinations with stable flow latencies. Destinations pruned due to unstable latencies could also have high latency imbalance.

Causes Not Validated: Our tool has limited visibility in MPLS tunnels and has no alias resolution, which prevents us from fully characterizing and understanding the causes for latency imbalance. The possible causes are also not validated by network operators.

Compounding Errors: In §3.3.3, although we have carefully examined each type of error sources in our methodology, these errors could compound and greatly affect the accuracy of pairwise latency imbalance. For instance, the probabilistic guarantee of high path coverage may fail together with the stability check of path latency during temporary network congestion. If the goal is to accurately measure pairwise latency imbalance, multiple measurements for the same pair over time would help either explore the rare paths with extreme latencies or reduce the measurement errors due to temporary network events.

3.9 Summary

In this chapter, we presented a methodology of measuring latency imbalance among Internet load-balanced paths at scale and used it to collect a global view of latency imbalance from the perspective of the cloud. We found that latency imbalance is both significant and prevalent between DCs in the cloud and from the cloud to the public Internet. We analyzed

the reasons for cloud providers to have different latency imbalance and further showed that latency imbalance could affect application performance in various ways. Moreover, we verified the accuracy of our measured imbalance and discussed the potential causes for latency imbalance and the challenges of load balancing. In the era of 5G, with the last-mile ratio link having sub-millisecond delay and emerging applications being more delay-constrained, the problem of Internet latency imbalance will become more prominent. This requires us to consider latency imbalance in the performance evaluation of future applications. Our work has limitations in that we measure latency imbalance all from DCs and only focus on latency.

CHAPTER IV

Congi: Measuring Congestion Imbalance Among Internet Load-Balanced Paths at Scale

In the previous chapter, we discussed the latency imbalance among LB paths, which focuses on the minimum path latency stable over time. In this chapter, we evaluate the congestion imbalance among LB paths that occurs with network congestion. As congestion is typically short-lived, this makes the detection of congestion imbalance much more difficult than latency imbalance. Latency imbalance is extensively studied in datacenters, but is still under-explored in the Internet. We take the first step to measure congestion imbalance in the Internet. Congi is our network prober designed to measure congestion imbalance at scale, which uses SVM classifiers to detect congestion imbalance with a very small number of samples. We verify Congi in terms of its ability to detect the throughput, packet loss and latency imbalance between LB paths. We further use web page load as an example and evaluate the impact of congestion imbalance on applications.

4.1 Introduction

Congestion imbalance has been mostly studied in datacenter (DC) networks [93, 3], where paths between servers can be carefully planned—to share minimal common bottleneck links—such that not all of the paths would be congested at the same time. Given

the capability to carefully plan paths, congestion-aware load balancing can better utilize bandwidth in data center networks. In contrast, despite the prevalence of load balancers in the Internet, Internet LB paths are likely to overlap, especially in access networks at the edges [94]. We are not aware of any study reporting on the prevalence of congestion imbalance in the Internet. In this work, we take the first step to measure congestion imbalance in the Internet at scale. We want to understand the prevalence of congestion imbalance and the extent to which it causes LB paths to differ in performance. Considering that congestion affects all performance aspects (throughput, latency, and packet loss) of a path, we would expect that harnessing Internet congestion imbalance would benefit a wide range of Internet applications.

Broadly speaking, congestion imbalance exists as long as LB paths experience different levels of congestion. As a first step, we limit our focus on recognizing a congested path from an uncongested one, without differentiating the congestion levels. To measure congestion imbalance at scale, we need to address two major challenges: 1) how to detect congestion, and 2) once a congested path is detected, how to quickly search among a pool of LB paths for an uncongested path while the congestion persists? To detect congestion, we use network probing from a single vantage point (VP) directed towards public Internet addresses. Network probing infers path performance towards an address based on the destination's responses to the probes. Due to ICMP rate limiting, probes sent to the same destination must be controlled at a very low rate, which makes us unable to detect transient congestion. We focus on congestion with durations ranging from seconds to minutes, which are of interest to many applications, e.g., web page load and VoIP. Our approach can also be easily extended to detect the diurnal persistent inter-domain congestion in [95].

Our approach uses latency inflation to infer congestion similar to the Time Series Latency Probing (TSLP) technique [96]. However, to detect congestion at scale (the *first* challenge), we cannot use latency time series collected in the long term as in TSLP. Instead, we develop *Congi*, our network prober that leverages a support vector machine (SVM) clas-

sifier to detect congestion with a small number of latency samples. As Congi is designed to detect congestion imbalance, not just congestion, it also includes two SVM classifiers trained to detect uncongested paths, which focus on the *speed* and *accuracy* of detection respectively. The goal of the *speed-focused* classifier is to fast search among LB paths for potential uncongested paths, whose results are then further verified by the *accuracy-based* classifier. These two classifiers work together to address the *second* challenge. The SVM classifiers in Congi are trained on a ground-truth dataset that we build from a collection of path performance data between a large number of geographically-diverse source-destination pairs.

We verify Congi's performance with both trace-driven simulation and real-world experiments, and find that Congi excels at detecting significant congestion imbalance, where the uncongested paths are on average 3x greater in throughput and 1.8% lower in packet loss rate than the congested ones. In our large-scale experiments, we ran Congi from 12 DCs around the globe to Internet-wide addresses. We found that most DCs experience long imbalance lasting 2 minutes to about 10% of Internet addresses and short imbalance lasting 30s to at least 35% of addresses. We further evaluate the impact of congestion imbalance on web page load by downloading web pages using the congested and uncongested paths respectively. We found that half of the download times can be reduced by 50%, using the uncongested paths.

4.2 Measurement Methodology

4.2.1 Our Goal

Congestion imbalance occurs between a source and a destination when the connecting paths experience different levels of congestion. In this work, we simply categorize paths as either congested or uncongested, with no further distinction of congestion levels. Congestion imbalance between a source and destination pair occurs when an uncongested path

exists concurrently with a congested one. Our goal is to measure congestion imbalance at scale from a single vantage point (VP) with network probing, assuming no direct access to a large number of instrumented routers or end hosts. We focus on measuring short-term congestion imbalance lasting from seconds to minutes, which affects many applications like web page load and video streaming. Nonetheless, our approach can also be used to measure transient congestion imbalance within sub-seconds, if probe rate limiting is not a concern, and to measure long-term congestion imbalance lasting hours.

4.2.2 Overview of Our Methodology

We want to design *Congi* (short for *Congestion Imbalance*), a network prober that collects latency samples to detect congestion imbalance. The core of Congi is a set of SVM classifiers capable of detecting both congested and uncongested paths with a small number of latency samples. Of these classifiers, some focus on the accuracy of congestion imbalance detection, while others focus on the speed of searching for uncongested paths amongst a set of LB paths. To obtain these classifiers, we must address two key questions:

1. *Can we differentiate between congested and uncongested paths from their latency samples?*

2. *Can we accurately detect congestion imbalance with a small number of latency samples?*

A positive answer to the first question requires the existence of a *latency-based metric* capable of such differentiation. To that end, we collect real performance data (latency, throughput, and packet loss) of Internet paths, use the metric to classify these paths as either congested or uncongested based on latency, and check if the uncongested paths have lower throughput and higher packet loss rates than the uncongested ones. However, as there could be many LB paths between a source-destination pair, to collect path performance of all of them is a very resource intensive process. Instead, we collect the performance of one

path continuously over a long period of time and check if a metric can differentiate between congested and uncongested periods of the path. This not only simplifies data collection, but it also ensures that congestion would be the reason for performance degradation with all other factors being equal. We experiment on several latency-based metrics (*latency elevation*, *latency deviation*, and *latency inflation*) and found that *latency elevation* (increase in mean latency over a sustained period of time) best differentiate congested and uncongested periods of a path (see §4.2.3.4 for more precise definitions and detailed descriptions). In §4.3.2 we verify that *latency elevation* can also be used to detect congestion across multiple paths.

Latency elevation enables us to detect congestion imbalance without actually measuring throughput and packet loss rates of LB paths. Unfortunately *latency elevation* relies on long running latency time series to be effective. We want to answer the second question by finding classifiers capable of accomplishing the same task but with a very small number of latency samples. To train these classifiers, we create a large ground-truth dataset consisting of latency time series each of which has been labeled by the *latency- elevation* based metric as being of a congested or uncongested paths. These lightweight classifiers enable us to scale the detection of congestion imbalance. We combine them to build Congi, verify its ability to detect congestion imbalance, and conduct measurement campaigns from VPs around the globe to Internet-wide addresses.

4.2.3 How to Choose the Latency-Based Metric?

The intuition behind a latency-based metric is that when a path is congested, its capacity is over utilized, causing packets to be buffered, resulting in inflated measured latency. Dhamdhere *et al.* [95] have shown that inflation in latency time series can be used to detect congested periods. However, we are not aware of any study to compare different latency-based metrics and to systematically explore their use in detecting congestion periods. In this section, we will compare alternative metrics for describing latency inflation and deter-

mine thresholds that properly separate congested periods from uncongested ones. We start by the collection and processing of the *path-perf dataset* that includes the performance of a diverse range of paths.

4.2.3.1 Data Collection: The Path-Perf Dataset

This dataset is used to evaluate several latency-based metrics, which involves correlating path latency with throughput and packet loss. In this dataset, all the three path metrics are collected for each path included. As public clouds enable us to easily access computing resources in data centers (DCs) around the globe, we create virtual machines (VMs) in DCs as our vantage points (VPs). For throughput measurement, we run the Network Diagnostic Tool (NDT) test from our VPs to M-Lab NDT servers [97]. Since we use latency to infer congestion, our latency and throughput measurements do not overlap in time, to avoid self-induced latency inflation. The TCP trace made public by M-Lab only provides latency during TCP sessions and thus does not fit our needs [97].

Measurement Methodology We want to measure latency and throughput for the same path between a source-destination pair, including both the forward and return directions. As 98% of load-balanced paths include only per-flow and per-destination load balancers, traffic with the same flow identifier (source IP, destination IP, source port, destination port, protocol ID) will follow the same path when there is no path change [8]. Given a source-destination pair, we enforce all measurement traffic to use the same source and destination ports. The impact of path changes will be considered during data processing (§4.2.3.2). Since NDT tests use TCP, we measure latency to the NDT hosts by sending TCP ACK to the port numbers used in NDT tests. The measured latency of the path is the elapsed time between the sending of the TCP ACK and receiving the corresponding TCP RST. Measured packet loss rate is the percentage of TCP ACK probes for which we do not receive responses.

To understand how latency is correlated with throughput, we want latency and throughput to be measured under various network conditions, including both congested and uncongested periods. Since collecting one latency sample only takes one probe, we periodically measure path latency to all NDT servers in parallel. In contrast, each NDT test measures upload and download TCP throughput separately, each taking about 10 seconds. We only run one NDT test from each VP to a target server at the same time to avoid multiple NDT tests competing for bandwidth. Considering the overheads of throughput measurements and that congested periods are typically less common than the uncongested ones, it is impractical and unnecessary to periodically measure throughput to all NDT servers. We instead drive NDT tests by latency variation: we trigger a NDT test to measure a path when the path latency is seen varying and follow up with another NDT test when the path latency returns back to stable. In this way we distribute throughput measurements more evenly between paths of varying and stable latencies, and still covers paths within the full range of latency variations. Moreover, we want to focus on latency variation due to congestion rather than route changes, where path latency during congestion is constantly varying while a route change causes path latency to suddenly increase or decrease and remain stable afterwards. We thus define *latency variation* as the average absolute difference between neighboring latency samples, i.e., $\sum_{i=1}^{n} |r_i - r_{i-1}| / (n-1)$, where n is the number of samples and r_i is the i-th latency sample.

Experiment Setup In our experiments, we used 12 geographically-distributed VPs (2 in Europe, 5 in Asia, 2 in North America, 1 in South America, 1 in Africa, and 1 in Oceania) and measured path performance from these VPs to 471 NDT-7 (NDT version 7) servers to create this dataset. As cloud providers may differ in how they route traffic from their DCs to the public Internet [98], these 12 VPs are further distributed evenly among 3 cloud providers (Google Cloud, Microsoft Azure, and Alibaba Cloud).[1] Each VP sent TCP ACK

[1]We did not have VPs in Amazon AWS for this dataset because the download TCP throughput from the NDT-7 servers to our VPs in Amazon DCs was throttled to a low level, which decorrelated latency and

probes to all NDT-7 servers every 2 seconds to measure path latency and packet loss. Latency variation was calculated every 60 seconds with the most recent 90 latency samples, where samples below the 10-th percentile and above the 90-th percentile were not used, to mitigate the impact of outliers. For each source-destination pair, we triggered a NDT test when latency variation was greater than 2ms, a relatively small threshold to capture small latency changes. When a NDT test was done, we waited at least 15 minutes before checking if the latency variation has returned back to stable (below the threshold) and we could conduct a follow-up NDT test.

4.2.3.2 Data Processing: Time Series Segmentation

We want to prepare the dataset for correlating latency with throughput and packet loss. Given a latency time series between a pair of VP and NDT server, we want to 1) segment it into periods of varying latency (or *varying periods*) and periods of stable latency (or *stable periods*) such that latencies in the same period reflect the same state of the path, and 2) match each period with the throughput samples measured in it. Note that a varying period is not necessarily a congested one because apart from congestion, other factors, such as delayed response, could also cause varying latency [99]. A congested period is a varying period that actually causes performance degradation. We will find proper thresholds for concluding if a path is congested or uncongested in §4.2.3.4. In the following, we first discuss the current method of detecting varying periods and its drawbacks, and then introduce a more accurate method for time series segmentation.

Drawbacks of the Current Mean-Shift Detection Method Observing that latency samples in varying periods are typically inflated, have higher means, than those in stable periods, Dhamdhere *et al.* proposed to use latency mean shifts in time series to identify varying periods [95]. Their method uses a moving window to sequentially scan the latency

throughput.

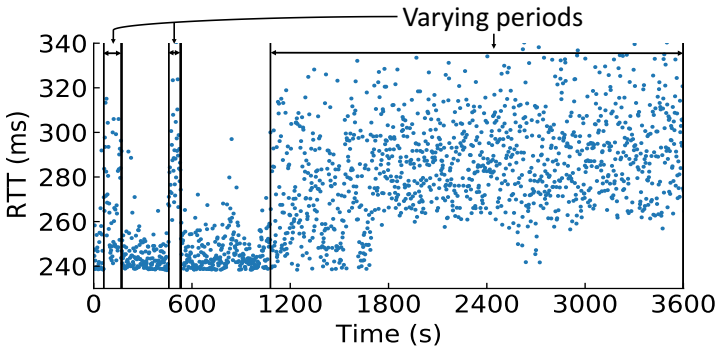


Figure 4.1: An example of time series segmentation

time series and, with a statistical test, checks if there is a mean shift in the moving window. However, this method relies on a moving window, including only a partial view of the entire time series, to make decisions and is thus prone to false positives when the entire moving window is in a varying period. Moreover, the statistical test can detect level shifts but is not capable of telling the exact boundary between varying and stable periods, which is critical for segmenting time series.

An Improved Method: Changepoint Detection To address the above issues, we introduce a new method called *changepoint detection*, which segments time series by observing the entire time series [100]. Specifically, the changepoint detection method segments the latency time series into time periods with different latency means. Let P_i be the i-th period, y_{ij} be the j-th sample in P_i, and m be the number of periods. The optimal segmentation is obtained by minimizing the loss function

$$\min_{\{\bar{y}_1,\ldots,\bar{y}_m\}} \sum_{i=1}^{m} \sum_{y_{ij}\in P_i} |y_{ij} - \bar{y}_i|^2 + \beta m,$$

where $\bar{y}_i$ is the latency mean of the i-th period and βm is a penalty term to avoid over-fitting. We use one of the common choices for the penalty term, i.e., the Bayesian Information Criterion penalty by setting β to $\sigma^2 \log(n)$, where n is the total number of samples and σ^2 is the variance of samples [100]. The `ruptures` package in Python [101] is used to

Table 4.1: The Path-Perf dataset summary

Cloud Providers	#VP-Server Pairs	#Throughput Samples	#Latency Samples
Google Cloud	342	1,903	22,239,395
Alibaba Cloud	629	2,683	30,464,281
Microsoft Azure	627	2,789	40,717,975

segment time series in our dataset. The minimum period length is set to 15 samples to obtain periods lasting for at least 30s (inter-sample time is 2s). The minimum period length is chosen to be relatively small such that short varying periods can still be detected.[2] Figure 4.1 shows an example of time series segmentation with the changepoint detection method, where we can clearly see the boundaries between neighboring periods.

Matching Segmented Periods with Throughput Samples After time series segmentation, it is straightforward to associate each segmented period with the throughput samples measured by time. We do need to handle some edge cases where throughput samples are measured close to the boundaries. To avoid associating throughput samples with the wrong periods, we only use throughput samples measured at least 10s away from both the start and end times of their associated periods.

4.2.3.3 Resulting Data Distribution

Table 4.1 summarizes the *path-perf* dataset, which only counts the VP-server pairs (pairs of VP and NDT server) that have at least one varying and one stable periods, both with throughput samples. Google Cloud has the least number of VP-server pairs with varying periods, likely because probes from VPs in Google Cloud to NDT servers spend most of their times traversing Google's well-provisioned private WANs [98, 80]. Since we run only one NDT test at a time from each VP, when the paths to multiple NDT hosts have

[2] A long minimum period forces the method to consider short varying periods as noise and may result in long stable periods with short varying periods inside them. A very short minimum period makes the method over-reactive to outliers.

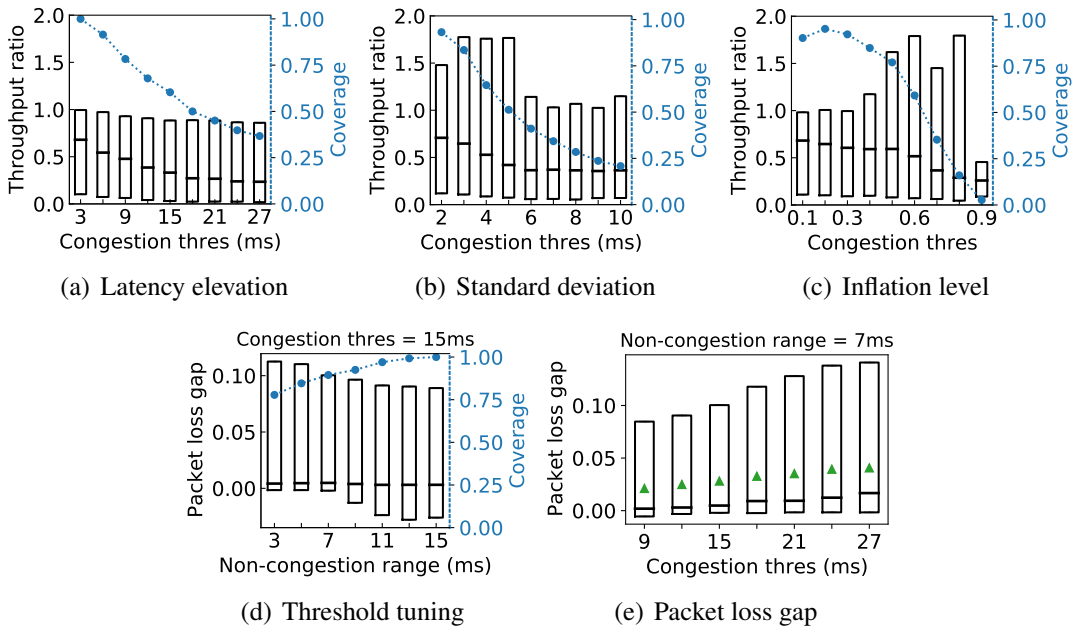
varying latencies, only one will be measured—the others would have to wait for future rounds of measurement. This case happens more often for the VPs in Alibaba Cloud and Microsoft Azure, leaving them with fewer throughput samples for each pair, on average. To compensate for this difference, we run experiments longer for VPs in Alibaba Cloud and Mircrosoft Azure. We next use this dataset to compare different latency-based metrics.

4.2.3.4 Comparative Study of Latency-based Metrics

We want to find a latency-based metric that can differentiate between congested and uncongested periods of a path by the gap in path performance during these periods. We will compare three latency-based metrics that describe latency inflation from different aspects: 1) *latency elevation*, the difference between latency mean and the minimum latency, 2) *latency deviation*, the standard deviation of latency samples, and 3) *inflation level*, the percentage of inflated latency samples.

Defining the Performance Gap We want to define the performance gap between the congested and uncongested periods for two path metrics, throughput and packet loss. For each source-destination pair, the throughput gap is defined as the ratio of the average throughput during congested periods to that during uncongested ones, also referred to as the *throughput ratio*. Similarly, the *packet loss gap* is defined as the difference between the average packet loss rate during congested periods and that during uncongested ones. We use the average path performance to alleviate the impact of measurement errors that are infrequent but difficult to exclude. For example, we could not tell if low throughput during a stable period is because the bottleneck link, though uncongested, has high utilization [94] or because the NDT server is under heavy load, which we want to exclude as it is not network related.

Ability to Maximize Throughput Gap To compare the alternative metrics, we use each metric to classify periods in our dataset as congested or uncongested, and calculate the



(a) Latency elevation (b) Standard deviation (c) Inflation level

(d) Threshold tuning (e) Packet loss gap

Figure 4.2: Comparing latency-based metrics and tuning thresholds

throughput gap between the congested and uncongested periods. The best metric is the one that achieves the maximum throughput gap. Given a metric, we consider a period *congested* if the metric calculated from the latency samples during the period is above a pre-defined *congestion threshold*; otherwise, we consider the period *uncongested*. Per the definition of throughput gap, it can only be calculated for pairs having at least a congested period and an uncongested one. This means that the number of pairs with a throughput gap is determined by the choice of the metric and the congestion threshold. In our dataset, we obtain the maximum number of pairs having a throughput gap when we use the i*latency elevation* metric, with congestion threshold of 3ms. The maximum number of pairs having a throughput gap is about 47% of the total VP-server pairs present. We consider this the total number of available pairs and calculate the *coverage* (i.e., the percentage of pairs with a throughput gap) at other congestion thresholds with respect to this number. Figure 4.2(a) shows the throughput gap/ratio and coverage under different congestion thresholds, where each box represents the distribution (10-th, 50-th and 90-th percentiles) of throughput ratios for the given threshold. We can see that as the congestion threshold increases, the throughput ra-

tio decreases, indicating a larger throughput gap. Meanwhile, the coverage also decreases, which implies that a higher throughput gap is achieved at the cost of coverage. There is thus a tradeoff between throughput gap and coverage. When the congestion threshold is 15ms, we can achieve a median throughput ratio of 0.33 and cover 60% of pairs. The average throughput ratio between the 10-th and 90-th percentile is only 0.37.

Figure 4.2(b) and 4.2(c) show the throughput ratios when latency deviation and inflation level are used as the metrics. We calculate standard deviation with samples between the 10-th and 90-th percentiles to mitigate the impact of outliers and consider a sample inflated if it is larger than the minimum latency for 5ms. Using standard deviation with a threshold of 6ms can achieve a median throughput ratio of 0.37 but covers only 45% of pairs, which is much less coverage than using latency elevation. When inflation level is 0.9, the median throughput ratio can be as low as 0.18 and even the 90-th percentile is below 0.4, but only a very small faction of pairs are covered. We thus consider latency elevation as the best latency-based metric and use 15ms as the congestion threshold to obtain a good tradeoff between throughput gap and coverage. We next discuss the ability of latency elevation to describe the packet loss gap.

One Threshold or Two? Given that the congestion threshold is 15ms, we calculate the packet loss gaps for available pairs and find that 11% of pairs have a negative packet loss gap. This implies that some pairs have their congested periods misidentified as uncongested, resulting in the packet loss rates in uncongested periods higher than those in congested ones. We revisit our way to differentiate between congested and uncongested periods, and find that the culprit is that since only one threshold is used to separate the congested periods from the uncongested ones, there is a sudden switch in categorization. We thus add a second threshold to identify uncongested periods and enforce a buffer region between the two thresholds. A period is *uncongested* if it has latency elevation below the second threshold. We refer to the range from zero to the second threshold as

the *non-congestion range*. Figure 4.2(d) shows the packet loss gaps under different non-congestion ranges, where the 10-th percentile packet loss gap approaches zero when the non-congestion range is 7ms or less. The use of a buffer region may reduce the coverage. We calculate the coverage at different non-congestion ranges with respect to that when the buffer region is zero, i.e., both the non-congestion range and the congestion threshold are 15ms. We use 7m as the non-congestion range, which reduces coverage by 15%.

Relating Latency Elevation with the Packet Loss Gap After the non-congestion range is determined, we can calculate the packet loss gaps under different congestion thresholds. As shown in Figure 4.2(e), both the median and mean packet loss gaps increase with the congestion threshold. This provides additional validation that latency elevation is an effective latency-based metric in separating congested periods from uncongested ones. Nonetheless, it is noticeable that compared with the throughput gap, the packet loss gap is in general very small, with the median being as low as 0.6% when the congestion threshold is 15ms. It is more interesting to look at the packet loss gaps in the high percentiles, where the 75-th and 90-th percentiles are about 5% and 10% respectively. We refer to this method of using latency elevation and thresholds to identify congested and uncongested periods as the *metric-based method*.

4.2.4 Building a Ground-Truth Dataset

The main drawback of the metric-based method is that it relies on long latency time series. We want lightweight classifiers that can achieve a similar accuracy in detecting congested and uncongested paths as the metric-based method but does so with much fewer samples. Specifically, we 1) want a dataset consisting of the latency time series of paths, where each time series is labeled either belonging to a congested or uncongested path, and 2) train the classifiers to determine if a path is congested or uncongested with a small portion of samples from its latency time series.

4.2.4.1 Large-Scale Ground-Truth Dataset

Instead of using the path-perf dataset, we want the ground-truth dataset to be of a larger scale and include a more diverse range of paths such that the classifiers can be accurately trained and perform well when used in real-world scenarios. To create such a large-scale dataset, we collect latency time series from VPs in DCs around the globe to a large amount of random IPv4 Internet addresses. Since we want the classifiers to achieve similar performance as the metric-based method, we apply the metric-based method on the collected data to construct the ground truth (see §4.2.4.3). We start by introducing how latency time series are collected.

4.2.4.2 Data Collection

We randomly select addresses from the IPv4 Internet address space and probe from 12 geographically-diverse VMs to the last-hop routers of these addresses to avoid affecting the end hosts. Latency is measured as the elapsed time between sending a UDP probe with the TTL expiring at the last-hop router and receiving the ICMP response from it. For each target, we force all UDP probes and ICMP responses to take the same path by sending UDP probes with the same source and destination ports [98]. The rate of UDP probes sent to each router is controlled at 0.5 packet/s, unlikely to cause packet drop due to ICMP rate limiting on routers [102]. To capture short-term congestion of various durations, we periodically measure each last-hop router for 1 hour and stop immediately once there is a route change, which is detected by path discovery to the last-hop router every 5 minutes with TTL-limited probes. After 7 days of measurements in October 2020, we collected latency and path information from our VPs to 342,139 Internet addresses, where each target has 1,800 latency samples.

4.2.4.3 Data Labeling

We first segment latency time series into periods of different latency means with the changepoint detection method. For each period, we calculate the latency elevation and label those with latency elevation greater than 15ms as "congested", less than 7ms as "uncongested", and the rest as "undetermined".

4.2.5 Training Congi's SVM Classifiers

Given a target, Congi first collects data with its probing module and then uses its SVM classifiers to detect congestion imbalance with the collected data. We will first describe the classifiers and discuss how we design a concurrent probing and training regime of the classifiers.

4.2.5.1 Accuracy- and Speed-Focused SVM Classifiers

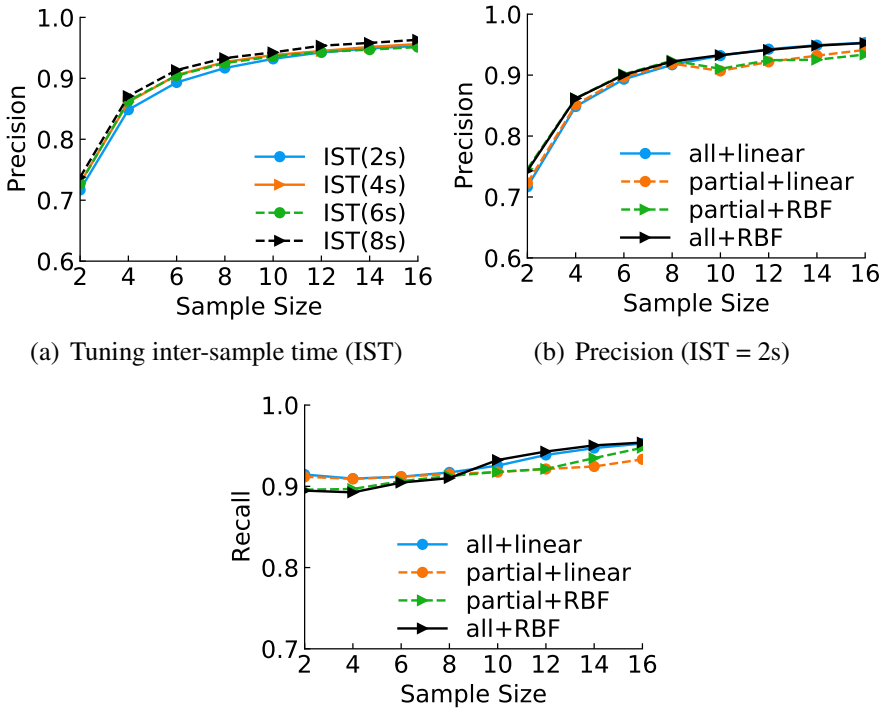
Congi includes two types of SVM classifiers: the *accuracy-focused* and the *speed-focused* classifiers. The *accuracy-focused classifiers* aim to detect both congested and uncongested paths with high accuracy, while the *speed-focused classifiers* aim to detect uncongested paths with a reasonable accuracy but with much fewer samples. For each target host, Congi starts the probing by detecting congestion with an accuracy-focused classifier, because if an uncongested path is misclassified as congested, all subsequent measurements, e.g., the search of an uncongested path, will be wasted. Congi also ends the probing by reconfirming congestion imbalance with the accuracy-focused classifiers. The speed-focused classifiers are used to search for an uncongested path once a congested path is detected. Between a source and a destination pair, there could be tens of LB paths [9]. It is crucial to check each path fast such that an uncongested path can be found while congestion persists. We choose SVM as our classifiers because 1) the separating hyperplane between classes is determined only by the data points near the hyperplane (a.k.a. support vectors), making it more robust to outliers compared to logistic regression, and 2) if needed, we can sep-

arate non-linearly separable classes by mapping them to higher dimensional spaces with non-linear kernels [103].

4.2.5.2 Creating Training and Testing Datasets from the Ground Truth

The ground-truth dataset above includes a collection of labeled latency time series. For each time series, Congi's classifiers only take a small portion of samples as input. The process of picking these samples from the time series simulates how probing works. There are two design parameters for probing, the number of samples to collect (denoted as m) and the inter-sample time (denoted as d). As different probing schemes lead to different data being collected, we will train the SVM classifiers on datasets collected under different d's and m's to choose the best probing scheme.

Consider a probing process that collects 3 samples with an inter-probe time of 4s. To simulate this process on a ground-truth time series, we first randomly select a sample in the time series as the start point and then select the 1st, 3rd, and 5th samples as input to the classifiers, where the start point is the 1st sample and the inter-sample time is 2s in the ground-truth dataset. As probing could begin from any sample within the time series, we repeat the probing process at different start points to average out randomness. Let d^* and m^* denote the upper bounds of m's and d's in the design space. To simulate a probing scheme with $m = m^*$ and $d = d^*$, we need latency time series of length at least equal to $l^* = d^* \times (m^* - 1) + 1$. We require each time series in the dataset to support simulating the probing scheme with $m = m^*$ and $d = d^*$ such that all probing schemes with d's and m's within the upper bounds can be simulated on the same dataset for fair comparison. Another issue of this dataset is that we have much more uncongested paths than congested ones. We handle this unbalanced class problem by randomly selecting the same amount of uncongested and congested paths in the dataset before simulating the probing process. For each pair of d and m, this random selection is repeated to average out randomness. We use two thirds of the data for training and the rest for testing.



(a) Tuning inter-sample time (IST)

(b) Precision (IST = 2s)

(c) Recall (IST = 2s)

Figure 4.3: Performance in detecting congested paths

4.2.5.3 Training the Classifier for Detecting Congested Paths

As mentioned in §4.2.4.3, our dataset includes three classes of paths (congested, uncongested, and undetermined), which implies that this is a multi-class classification problem. As we are not interested in detecting the undetermined paths, we take a one-versus-rest approach to detect the congested and uncongested paths respectively [104]. That is, when our goal is to detect one class, we consider it as the positive class and the rest two together as the negative class. This requires us to have separate classifiers for detecting the congested and uncongested paths. Considering the number of samples as input to the classifiers is limited, we generate a feature from each sample and form a feature vector of dimension equal to the number of samples. Specifically, given m samples, we form a m-dimensional feature vector by 1) considering the minimum sample as the minimum latency, 2) subtracting it from each sample to obtain the inflated part, and 3) sorting the inflated parts of all samples in an increasing order.

Figure 4.3(a) shows the precision in detecting congested paths under different m's and d's, where a linear kernel is used and the precision is the fraction of true positives among all claimed positives. We can see that the precision increases gradually with the sample size, but the improvement becomes very marginal when the sample size is greater than 12. As the sample size becomes larger, samples are more likely to include outliers far away from other samples, which causes more uncongested paths to be misclassified as congested and counteracts the improvement by using a higher dimension. We can also see a slight improvement in precision by using a larger inter-sample time (IST), because a larger IST makes samples more independent from each other and is thus more likely to result in larger difference between latency samples for congested paths. Considering that the improvement for using a large inter-sample time is limited and that congestion is typically short-lived, we want probing to be done fast and thus choose the inter-sample time to be 2s.

In Figure 4.3(a), all m samples are used to construct the feature vector. We wonder if excluding the extreme samples that might be outliers could improve precision and that how kernel selection affects precision. With the inter-sample time being 2s, Figure 4.3(b) and 4.3(c) show how feature and kernel selection affects the precision and recall in detecting congested paths, where "all" means that all samples are used for prediction and "partial" means that only samples between the 10-th and 90-th percentiles are used, to exclude outliers. When a percentile lies between two samples, the smaller one is used for the 10-th percentile and the larger one is used for the 90-th percentile. In Figure 4.3(b), the scheme of using partial samples begins to drop samples when sample size reaches 10, which results in a lower-dimensional feature vector and causes a sharp decrease in precision. This indicates that the precision gain by excluding outliers cannot cover the precision loss due to using a higher-dimensional feature vector. We therefore use all samples as input for prediction. Comparing the two schemes of using all samples in Figure 4.3(b) and 4.3(c), we find almost no difference in precision between using a linear and a radial basic function (RBF) kernels and a slight 1% difference in recall when sample size is 10. When sample size is 12, both

classifiers with the linear and RBF kernels can achieve a high precision of 95% and a high recall of 94%. We will elaborate how Congi uses these classifiers in §4.2.6.

4.2.5.4 Training Classifiers for Detecting Uncongested Paths

We want to train two types of classifiers for detecting uncongested paths: an accuracy-focused classifier and a speed-focused classifier. The uncongested paths constitute the positive class and the other two classes together constitute the negative class. We train these classifiers for detecting uncongested paths similarly to the way we train the classifiers for detecting congested paths described in the previous subsection. We use different sample sizes but with a fixed inter-sample time of 2s in the training regime. We find that for both classifiers with linear and RBF kernels, the precision and recall increase with the sample size and the increase rate begins to slow when the sample size reaches 8 (not shown). When the sample size is 8, using a RBF kernel results in a precision of 94%, almost the same as using a linear kernel, and a recall of 87%, about 2% higher than using a linear kernel. For the speed-focused classifier, we want a small sample size that can provide a reasonable precision and recall such that each LB path can be checked quickly. When the sample size is 4, using a RBF kernel achieves a precision of 88% and a recall of 81%, while using a linear kernel achieves a precision of 91% and a recall of 76%. Since a high recall is desired while searching for uncongested paths, we choose the RBF kernel for the speed-focused classifier.

4.2.6 Putting It All Together to Build Congi

We want to combine the classifiers above to build Congi. Figure 4.4 shows Congi's high-level workflow, which consists of three stages.

Stage I: Detecting Congestion Given a target address, Congi first detects if congestion exists in a path to the target. To avoid affecting the end hosts, we often use the last respond-

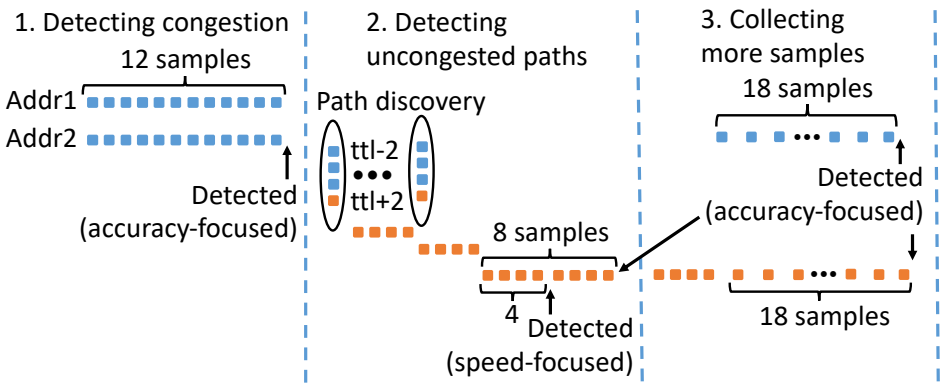


Figure 4.4: Congi's high-level workflow

ing router along the path to the target as a proxy and measure congestion imbalance from the source to the router. This means that all probes are still sent to the target with TTLs expiring at its last responding router such that congestion imbalance seen by the router will also be seen by the target. To force all probes to take the same path, Congi sends UDP probes with the same source and destination ports. To avoid triggering ICMP rate limiting, Congi sends one UDP probe to the router every 2 seconds until the 12 samples are collected. Congi then feeds the 12 samples into its accuracy-focused classifier to detect congestion. If no congestion is detected, Congi moves on to the next target; otherwise, Congi enters into stage II. It is possible that the target router has a very low budget of ICMP responses and drops most of our probes. Congi allows a packet loss rate less than 30% for each target and stops probing a target if the packet loss rate is above the threshold. As shown in Figure 4.4, targets are probed in parallel and the total sending rate depends on the number of targets being probed simultaneously.

Stage II: Searching for an Uncongested Path Once a congested path is detected, Congi begins to search for a new path to the router and check if it is uncongested. This process is repeated until an uncongested path is found or failed to be found after 15 trials. The discovery of new paths is done by sending UDP probes to random destination port numbers. Although the paths of UDP probes are passively selected by load balancers based on their

port numbers, with a sufficiently large number of trials, we can discover an uncongested path, if one exists, with a high probability. For instance, assuming that LB paths have equal probability of being taken by UDP probes with random port numbers, even when the portion of uncongested paths is less than 10% among all LB paths, Congi can still find one with a probability of $1 - 0.9^{15} = 79.4\%$ with 15 trials. This is more efficient than finding new paths by hop-by-hop comparison with the previously probed ones. Note that although UDP probes may take different forward paths due to having different port numbers, all ICMP responses take the same return path because Congi manipulates UDP probes such that ICMP responses can have the same checksum, the factor load balancers use to determine the paths ICMP packets take [98]. For each new path, Congi has to estimate the TTL from the source to the target router. Congi uses the TTL of probes in the congested path as reference and search neighboring TTLs (± 2) for the new path. Once a new path is found, Congi collects 4 samples and use the speed-focused classifier to check if this path is uncongested. If so, Congi continues to collect 4 more samples and verify this with the accuracy-focused classifier. Congi either enters into Stage III after a successful verification or returns back to finding a new path if the verification fails.

Stage III: Collecting More Samples Congi collects more samples for both the congested and uncongested paths for two reasons. First, as it takes time to find the uncongested path, during which the congestion may have ended, Congi confirms congestion imbalance with the most recent 12 samples of each path at the end of the probing. Second, we want to collect 30 samples for each path in total to estimate their mean path latencies for later study of the latency difference between LB paths under congestion imbalance (§4.4.3).

4.3 How Does Congi Actually Perform?

Before conducting large-scale measurements with Congi, we want to verify its performance in detecting congestion imbalance with both trace-driven simulation and real-

Table 4.2: Trace-driven simulation

Method	Throughput Ratio				Packet Loss Gap			
	25-th	50-th	75-th	Mean*	25-th	50-th	75-th	Mean*
Metric-based	0.12	0.32	0.65	0.37	0	0.6%	5.0%	2.0%
Congi	0.11	0.27	0.59	0.34	0	0.7%	3.4%	1.6%

* Mean of the samples between the 10-th and 90-th percentiles.

world experiments. In trace-driven simulation, we use the path-perf dataset to simulate how Congi works and expect Congi to detect similar throughput and packet loss imbalance as the latency-based metric does in §4.2.3.4. In the real-world experiments, we run Congi from VPs to NDT servers to verify its ability to detect the throughput imbalance between LB paths, and from VPs to the public Internet addresses to verify its ability to detect the packet loss imbalance.

4.3.1 Trace-driven Simulation

Recall that in the path-perf dataset, each throughput sample is associated with a segmented period of latency samples. We use Congi to determine if a period is congested or uncongested based on its latency samples and compute the throughput and packet loss gaps as in §4.2.3.4. To simulate how Congi works, for each throughput sample, we feed the 12 latency samples measured right before starting the throughput sample into Congi's accuracy-focused classifier to detect congested paths. If the result comes back positive, we then feed the 12 latency samples right after the completion of the throughput sample into the same classifier. We consider the associated period congested if both the results are positive. We check congestion both before and after the throughput sample to ensure that the throughput sample is measured while congestion persists. Similarly, we consider the period uncongested if both the results come back positive from Congi's accuracy-focused classifier for detecting uncongested paths. Table 4.2 compares the throughput ratios and packet loss gaps under Congi and the metric-based method. We can see that Congi achieves similar throughput ratios at different percentiles and lower packet loss gaps at the high percentiles compared to the metric-based method. Considering the small number of latency samples

Table 4.3: Real-world experiments

Method	Throughput Ratio				Packet Loss Gap			
	25-th	50-th	75-th	Mean*	25-th	50-th	75-th	Mean*
Congi	0.10	0.28	0.61	0.34	0	0.4%	1.5%	0.7%

* Mean of the samples between the 10-th and 90-th percentiles.

Congi uses to perform its measurements, we are pleased by how close the results are to the metric-based measurements and how well they track each other.

4.3.2 Real-World Experiments

We next verify Congi's ability to detect the throughput and packet loss imbalance between LB paths with real-world experiments.

4.3.2.1 Ability to Detect Throughput Imbalance

If Congi is capable of detecting throughput imbalance, the congested and uncongested paths detected by Congi should differ significantly in throughput. In our experiment, we ran Congi to measure congestion imbalance from VPs in 9 geographically-diverse DCs (3 DCs per cloud provider) to 471 NDT-7 servers, where NDT servers are used as destinations because we want to measure path throughput with NDT tests. Each VP periodically detects congestion imbalance to all NDT servers every 3 minutes. Once congestion imbalance is detected between a congested and an uncongested path to a NDT server, we immediately stop Congi and measure throughput for each path respectively with a NDT test. Since two back-to-back NDT tests take at least 40 seconds, it is possible that congestion imbalance ceases during the NDT tests. After the NDT tests are done, we resume Congi to collect more samples and double check if congestion imbalance still exists. This ensures that congestion imbalance persists when throughput samples are measured.

The left column in Table 4.3 shows the percentiles of throughput ratios between uncongested and congested paths in real-world experiments. Among all congestion imbalance events detected by Congi, the uncongested paths on average have throughput 3x greater

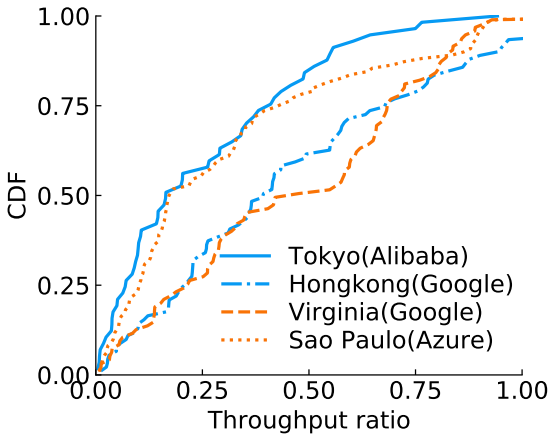


Figure 4.5: Verifying ability to measure throughput imbalance

than the congested ones. This implies that Congi is capable of differentiating between the congested and uncongested paths in real-world settings. Moreover, by comparing the throughput gaps in the real-world experiment and the trace-driven simulation, we find that Congi, though trained on the congested and uncongested periods of the same path, can achieve similar throughput gaps between different LB paths in real-world experiments. This validates our design choice of using the performance data of the same path to choose the best latency-based metric in §4.2.2 and build the ground-truth dataset in §4.2.4.

To further illustrate the utility of congestion-imbalance detection, we show in Figure 4.5 the distribution of throughput ratios for four DCs. The figure shows that Alibaba Tokyo's and Microsoft Sao Paulo's DCs have the two smallest mean throughput ratios of 0.22 and 0.26, and that Google's DCs in Hongkong and North Virginia have the two largest mean throughput ratios of 0.43 and 0.46. We can see that although throughput imbalance varies across DCs, all DCs experience significant congestion imbalance on their paths to NDT servers. The average thoughput of uncongested paths from Alibaba Tokyo's DC was almost 5x that of the congested paths. As a point of record, Congi uses the same set of classifiers for all DCs for the simplicity of design.

4.3.2.2 Ability to Detect Packet Loss Imbalance

Unlike throughput that can be measured with a single NDT test, packet loss rate is a statistic calculated from a large number of collected samples. To obtain the packet loss gap between a congested and uncongested paths, we have to probe each path for a long period of time, all the while monitoring that they remain in the same state (either congested or uncongested) the whole time. Recall that when creating the ground-truth dataset for training the SVM classifiers in §4.2.4, we segmented the latency time series into varying or stable periods. To verify Congi's ability to detect packet loss imbalance, we similarly create a dataset comprising latency time series of LB paths, segmented into varying and stable periods. To build this dataset, we first randomly select an IPv4 address. Then from each of our VPs (listed in §4.2.4.2), we alternately probe two randomly selected LB paths to the selected IPv4 address. For each VP-destination pair, we measure only two LB paths at a time to avoid triggering ICMP rate limiting. Upon completion of probing a pair of LB paths, we segment the resulting latency time series using the changepoint detection method introduced in §4.2.4 and pair the segmented periods of the two LB paths that overlap in time. We then use Congi to detect congestion imbalance for each pair of segmented periods, where Congi picks a random starting point and takes latency samples from the pair of segmented periods. The packet loss gap is calculated for pairs where each period includes at least 200 samples and congestion imbalance is detected by Congi.

The right column in Table 4.3 shows the percentiles of packet loss gaps between the congested and uncongested paths by Congi, where the mean is 0.7%. As this dataset includes latency time series of paths, we can also apply the metric-based method to detect packet loss imbalance, which achieves a mean packet loss gap of 1.2%. Congi achieves a lower packet loss gap than the metric-based method for the same reason as in the trace-driven simulation: it makes decisions based on a small number of samples, a local view of the period. Moreover, we can see that the packet loss gaps in the real-world experiments are smaller than those in the trace-driven simulation. This difference is likely because 1) dif-

ferent sets of source-destination pairs are measured and 2) the dataset in this experiment is collected with UDP probes and ICMP responses, while the dataset used in the trace-driven simulation is collected using TCP ACK/RST pairs. The use of different source-destination pairs and different types of probes make it hard to fairly compare the two. We use UDP probes in this real-world experiment to obtain a packet loss gap that reflects more closely the gap in later large-scale measurement campaigns that also use UDP probes.

4.4 Congestion Imbalance: A Cloud-Centric View

We ran Congi from 12 DCs of 4 cloud providers to 10.8M routable /24s in CAIDA's IP-to-Prefix dataset [57]. This run took about 10 days in January of 2021. The average probing rate of Congi was controlled at 500 packets/second or approximately 110 Kbps to avoid self-induced queueing delay. For each /24, we randomly select an address and probe to the first responsive router closest to the address, so as not to affect the end host.

4.4.1 Does Congi Measures Short or Long Congestion Imbalance?

Recall that Congi checks congestion imbalance twice (in Stage II and III respectively) in a complete cycle of probing a destination. We refer to the initial congestion imbalance detection as *short imbalance* and secondary detection as *long imbalance*. The initial detection differs from the secondary detection in that the initial detection uses samples collected in sequence for both the congested and uncongested paths, whereas the secondary detection uses interleaved samples collected alternately between the two paths. This results in the initial detection being prone to false positives: an uncongested path is found because congestion ceases prior to the end of measurement. We don't use the sample interleaving method for initial detection because it is inefficient when there is no congested path.

We first focus on the long imbalance measured by Congi and design separate experiments to measure short imbalance later in §4.4.4. Another detailed discussion of short imbalance can be found in §4.5, where we use short imbalance as a guide for real applica-

Table 4.4: Prevalence of long imbalance

	Google Cloud			Amazon AWS			Microsoft Azure			Alibaba Cloud		
	London	Tokyo	HK*	London	Tokyo	Cali	London	Tokyo	SP*	London	Tokyo	Shanghai
Congestion	251K	171K	172K	331K	349K	339K	283K	295K	318K	362K	392K	1,693K
Long imbl	15K	15K	16K	37K	48K	35K	36K	32K	39K	59K	43K	32K
%	6.0%	8.8%	9.3%	11.1%	13.7%	10.3%	12.7%	10.8%	12.3%	16.3%	11.0%	1.9%

* HK = Hongkong, SP = Sao Paulo

tions. In a complete cycle of probing, Congi collects 60 samples for each destination (30 samples for each path), which means that long imbalance lasts for at least 120 seconds.

4.4.2 Prevalence of long imbalance

Table 4.4 summarizes the number of congestion and long imbalances seen by each DC in our experiments. Most DCs see long imbalances to 9%-13% of probed addresses. Google Cloud observes much less congestion than other cloud providers. This is due to Google using its own, well-provisioned private WANs to route traffic to egress points closest to the destinations [98]. For Alibaba Cloud, DCs in London and Tokyo observe similar amounts of congestion as Amazon's and Microsoft's DCs. This may be because unlike Google Cloud, they rely on the public Internet to route traffic to destinations [98]. Alibaba Shanghai observes a much higher level of congestion events than other DCs, which indicates the presence of persistent under provisioning of network resources between China and the rest of the world. Despite that congestion events are frequent, Alibaba Shanghai has the lowest percentage of congestion imbalance among all DCs, because when congestion occurs, almost all LB paths are congested.

4.4.3 Latency Imbalance as a Result of Congestion Imbalance

Congestion imbalance causes LB paths to differ significantly in latency. We analyze the latency imbalance between LB paths under congestion imbalance from several aspects.

Imbalance in Latency Elevation Although we have verified in §4.3 that Congi is capable of detecting both congested and uncongested paths, we would like to understand how much

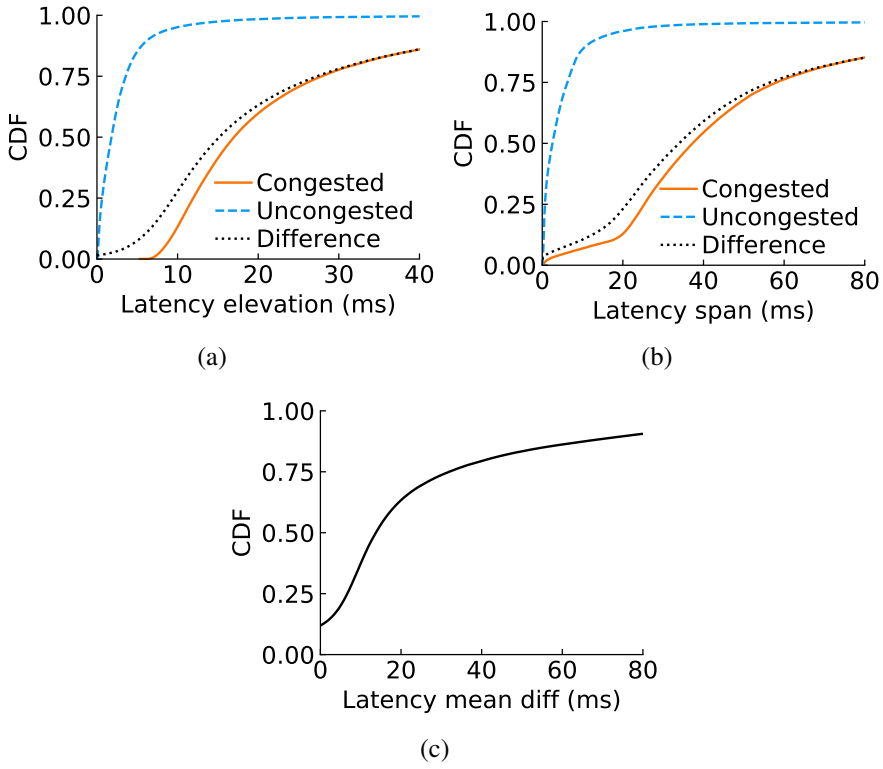


Figure 4.6: Latency imbalance as a result of congestion imbalance

latency is elevated under congestion imbalance in our large-scale experiments. One caveat is that we only have 30 samples for each path and thus cannot accurately estimate the minimum latency as in previous experiments. We simply use the minimum sample as the minimum latency to roughly estimate latency elevation, i.e., the difference between latency mean and the minimum latency as defined before. Figure 4.6(a) shows the distribution of latency elevation for the congested and uncongested paths separately and the difference in latency elevation between them. We can see that the congested paths have much more significant latency elevation than the uncongested ones. The average latency elevation is 24ms for the congested paths and 3ms for the uncongested ones. This confirms that Congi is capable of detecting significant latency imbalance between LB paths. Moreover, about 15% of congested paths have latency elevation higher than 40ms. We would expect that these congested paths experience severe throughput drop and that the corresponding uncongested paths have much higher throughput on average.

Imbalance in Latency Variation Besides latency elevation, we also want to understand the imbalance in latency variation, critical to interactive applications [79]. Specifically, we focus on latency spikes, which may be transient but of significant magnitude. We define *latency span* as in [105], which is the difference between the 90-th percentile sample and the minimum. Figure 4.6(b) shows the distribution of latency span for all the congested and uncongested paths. Note how the uncongested paths have much more stable latency than the congested ones. The figure shows that 16% of congested paths have latency span greater than 80ms, meaning that for 16% of congested paths, 10% of samples could experience latency spikes 80ms higher than the minimum path latency. The latency spikes are unlikely due to ICMP responses traversing a relatively slow path because Congi ensures that the congested and uncongested paths both have the same return path (§4.2.6). We also compare the standard deviation of latency samples between congested and uncongested paths. The average standard deviation is 14ms for the congested paths and only 1.3ms for the uncongested paths.

Imbalance in Latency Mean We have seen that congested paths have much higher latency elevation than uncongested paths, but this does not necessarily mean that the mean latency of a congested path would be higher than that of its corresponding uncongested path. Figure 4.6(c) shows the distribution of the difference in mean latency between congested and uncongested paths. About 10% of DC-destination pairs have an uncongested path with a higher mean latency than the congested one because due to a higher minimum latency. This implies that a shorter path is not always the better choice especially when it is congested. This figure also shows that about 10% of pairs have a mean latency difference greater than 80ms, where the congested paths of these pairs have a mean latency span of 102ms and a mean latency elevation of 59ms.

Table 4.5: Prevalence of short imbalance

Detection method	Google Cloud			Amazon AWS			Microsoft Azure			Alibaba Cloud		
	London	Tokyo	HK*	London	Tokyo	Cali	London	Tokyo	SP*	London	Tokyo	Shanghai
Interleaved	24%	36%	40%	30%	38%	37%	43%	42%	39%	36%	36%	6.6%
Non-interleaved	43%	62%	71%	64%	59%	58%	65%	62%	63%	58%	57%	29%

* HK = Hongkong, SP = Sao Paulo

4.4.4 Prevalence of Short Imbalance

As mentioned in §4.4.1, Congi achieves fast search of short imbalances at the cost of accuracy. To better characterize these short imbalances, we design a separate experiment as follows. The method used in this experiment to detect short imbalances is wasteful of resources and not very efficient. We employ it here to better understand short imbalances but do not otherwise use it as part of "production" Congi.

Experiment Design: Challenges in Measuring Short Imbalance To reduce the false positive rate of short imbalance detection, we use the interleaved sampling method originally employed in §4.4.1 to detect long imbalance. Given a fixed probe sending rate and a minimum number of samples required for each path, the more LB paths we probe under this method, the longer the detection will take. To short imbalances before they abate, we have to keep the number of LB paths probed to the minimum.

To figure out the minimum number of LB paths to probe, we consider the number of LB paths probed in our previous experiment on measuring long imbalances in §4.4.2. We found that short imbalances can be detected in nearly half the cases by probing only two LB paths and if a third path is added, short imbalances can be detected in about two thirds of the cases. Unfortunately, adding a third path also makes the method less sensitive to short imbalances. Whereas we can detect imbalances lasting as short as 24s when 2 paths are used, adding a third path lessens the sensitivity to 36s, i.e., only imbalances lasting 36s or longer would be detectable. In this experiment, we probe two LB paths in parallel to detect short imbalances lasting at least 24s.

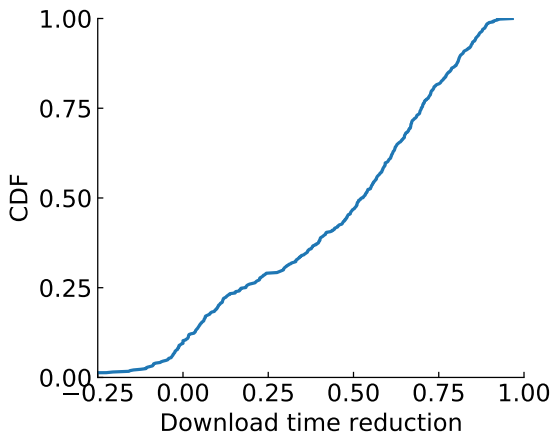


Figure 4.7: Impact of congestion imbalance on web page load

Data Collection and Experiment Results From each of our VPs (used in §4.4.2), we alternately probe two LB paths in parallel to 1M random IPv4 addresses and collect 12 samples for each LB path. Then, for each pair, we feed the samples of LB paths to Congi's classifiers to detect if short imbalance exists. Table 4.5 shows the percentage of short imbalance seen by our VP in each DC. We can see that congestion imbalance detected using interleaved sampling is much smaller than that using non-interleaved sampling, and should reflect more closely the actual percentage of short imbalances. Overall, except for Alibaba Shanghai, short imbalances are significant and prevalent in all DCs.

4.5 Impact on Applications

Since congestion affects all performance metrics (throughput, latency, and packet loss) of a path, we would expect that congestion imbalance has an impact on a wide range of applications. We use web page load as an example to show how application performance would be affected.

4.5.1 Web Page Load

When loading a web page, a client browser resolves the dependency between objects and issues a series of HTTP requests to download the objects following the dependency chain. The total download time is determined by each individual request on the longest dependency chain. To understand how congestion imbalance impacts the completion time of individual requests, we use Congi to monitor congestion imbalance to servers hosting the Alexa top-1000 websites. We find the largest object in a web page using the Chrome web browser's Remote Debugging Protocol (RDP). Since congestion imbalance may not persist, we download the target object immediately after short imbalance is detected. The object is fetched with HTTP requests along the congested and uncongested paths and is downloaded twice for each path, where the first download is to warm the cache and the second download is used to obtain the download times. All downloads are done serially, to avoid self interference.

We denote t_1 and t_2 as the respective download time for using the congested and uncongested paths, and measure the reduction in download time as $(t_1 - t_2)/t_1$. Note that the download time reduction could be negative if using the uncongested path results in a larger download time than using the congested path. Figure 4.7 shows the distribution of download time reduction for 500 downloads from a node in the campus network, where there is sufficient bandwidth for the access link. We can see that the median download time reduction is as high as 50% under congestion imbalance.

4.6 Discussion

Our measurement methodology and results have several limitations.

Limited to Wired Networks Congi assumes latency to be stable for uncongested paths and uses latency variability to detect congestion. However, as latency in wireless networks, e.g., cellular, is inherently varying, Congi is prone to make false inference. In our experi-

ments, we use Congi to probe the last-hop routers of end hosts to exclude potential wireless access links. It is interesting to extend the measurement of congestion imbalance to include the access networks, which are typically considered to be the bottleneck for throughput.

Unverified Results for Large-Scale Experiments We are able to verify Congi in terms of throughput and packet loss imbalance between our VPs and NDT servers, but cannot do the same for our large-scale experiments measuring imbalance to random Internet addresses. We thus interpret our large-scale measurement results as providing a coarse estimation of congestion imbalance. A more accurate large-scale experiment can be done by content providers that can directly infer throughput imbalance from the traffic between their servers and clients.

No Practical Solutions Implemented Although we estimate the impact of congestion imbalance on web page load, no practical solution is implemented. As future work, it is appealing to optimize path selection or multi-path transport layer protocols, e.g., Multi-Path TCP (MP-TCP), by considering congestion imbalance.

4.7 Summary

In this chapter, we took the first step to measure congestion imbalance at scale. We presented Congi, a network prober that uses SVM classifiers to detect congestion imbalance with a very small number of samples. Congi was verified in terms of its capabilities of detecting throughput, packet loss and latency imbalance among LB paths. We used Congi to measure congestion imbalance from our VPs to Internet-wide addresses and found that short-term congestion imbalance is prevalent in the Internet. We also used Congi to guide web page load and found a significant reduction in download time.

CHAPTER V

Related Work

This chapter presents the prior work related to the characterization and improvement of global load balancing, latency imbalance, and congestion imbalance respectively.

5.1 Client Aggregations for Global Load Balancing

Existing aggregations generally aggregate clients based on several attributes: (1) geographic locations, (2) topological proximity, (3) LDNSs, or (4) fixed-size prefixes. In geoclustering, clients with geographic proximity are aggregated into geoclusters [16]. By considering BGP prefixes as a natural group of Internet clients, geoclusters are obtained by splitting BGP prefixes into smaller prefixes until clients in prefixes have consensus on their geographic locations. Aggregating clients by topological proximity can be conducted at the prefix level or router level. Clients in the same BGP prefix largely have identical Internet routing. In anycast CDNs, clients are in the same BGP prefix are redirected to the same server and the nearest server is the one with the shortest BGP routing path to clients [17]. In iPlane, clients are aggregated into BGP prefixes to construct a topological map for the Internet [106]. Moreover, in latency-based client redirection, the latency from the representative client in a BGP prefix to a server is used as the latency from all clients in the prefix to the server [4]. The geographic locality of BGP prefixes has also been studied [7]. Recently, Lee and Spring [26] developed a method to aggregate /24s based on the last-hop routers

or non-hierarchical relationships. However, since the method relies on topology discovery, which makes the aggregates hard to cover the entire Internet address space.

Aggregating clients by LDNSs is widely used in DNS-based CDNs [107, 4], where clients using the same LDNS are redirected to the same server. As the DNS infrastructure evolves, the number of clients using remote DNS services grows, causing the mismatch problem between clients and LDNSs [108, 21]. To solve this problem, end-user mapping directly uses the subnets of clients included in the EDNS queries to locate clients on the Internet [5]. However, end-user mapping requires the mapping system estimate the performance from servers directly to client aggregations[107]. Thus, a proper aggregation is crucial to both the scalability and accuracy of the mapping system. Using a heuristic method, Chen et al. [5] suggested that /20 IP blocks are a good tradeoff between scalability and accuracy. Our work is also related to those studying whether /24s are a good aggregation unit. Krishnan *et al.* studied the similarity between individual addresses in /24 prefixes in terms of geographic co-locality [27]. Lee and Spring studied the similarity between individual addresses in /24 prefixes in terms of topological proximity [26]. Both works confirm that /24 prefixes are not necessarily the smallest aggregation units.

5.2 Latency Imbalance Among LB Paths

Since Augustin *et al.* presented the problems of *classic traceroute* under load balancing in 2006 [2], extensive efforts have been made to measure LB paths and study their impacts (e.g., [38, 9]). However, most of the efforts focus on the topology difference between LB paths, leaving the performance difference under-explored. In this dissertation, we characterize the latency difference between LB paths from a cloud-centric view and evaluate its impact on applications.

Discovering Load-Balanced Paths *Paris Traceroute* [2] is the most popular tool to identify diamonds and enumerate LB paths between a source and a destination via the Multi-

path Detection Alogrithm (MDA) [109]. To incentivize the deployment of Paris Traceroute and its MDA, Vermeulen *et al.* developed *MDA-Lite Paris Traceroute* incurring less network overhead [9]. Moreover, Almeida *et al.* extended MDA to measure load balancing in IPv6 networks [110] and Vanaubel *et al.* developed a method capable of discovering load balancing in MPLS tunnels [75]. Other than IP-level paths, load balancing was also studied at the AS level [8, 111]. This dissertation differs from these prior works in that we focus on the performance side of LB paths. We use the topology of LB paths to discover diamonds as in [8, 9] and further extend the discovery of topologically invisible diamonds by observing their impact on latency imbalance between LB paths (§3.5.4.2).

Latency Difference Between LB Paths In 2007, Augustin *et al.* [8] conducted the first study on latency imbalance between LB paths for 22K source-destination pairs and only 2% were found to have significant latency imbalance. To avoid the latency variability due to latency imbalance in latency measurement, *Tokyo ping* was developed as a replacement to `ping` [38]. Although both works are closely related to ours in that they found latency imbalance between LB paths and attempted to figure out the causes, our work [98] differs in several aspects. First, we measure latency imbalance both in the cloud and from the cloud to the public Internet, while they have vantage points (VPs) and destinations all in the public Internet. Although virtual machines (VMs) were carefully excluded as sources in [38], we observe almost no impact of using VMs in the cloud as VPs on measurement accuracy (3.5.4). Second, their approaches to measure latency imbalance are either not scalable due to requiring path enumeration [8] or not systematic in choosing sample size [38]. We propose a lightweight approach to measure latency imbalance, which requires no path information and provides accuracy guarantees. Third, we evaluate the impact of latency imbalance on application performance and propose potential solutions. Our work also differs from [38] in that 1) we do not exclude last-hop routers from probing and observe no significant influence of slow-path ICMP generation on measured latency (§3.5.4), and that

2) we use UDP probes, though having less reachability than ICMP [112], to reflect more closely the latency imbalance experienced by the Internet traffic.

Applications Latency imbalance implies that path selection may affect application performance. Wu *et al.* proposed to send multiple requests to explore the multi-path capability of load balancing to reduce latency inflation, where the first received request is considered taking the best path [92]. We evaluate how latency imbalance affects application performance on three applications: delay-based geolocation [84, 42], NTP [51] and VoIP [40]. Latency imbalance has impacts on latency-based client direction for global load balancing in content delivery networks [1, 113]. Latency imbalance could also affect many emerging applications, e.g., networked VR/AR [54], low-latency gaming [55] and industrial IoT [10]. Moreover, latency imbalance also makes us rethink the design of load balancing algorithms (e.g., hashing-based schemes [114]).

5.3 Congestion Imbalance Among LB Paths

Latency Difference Between LB Paths Different from the works measuring the difference in the minimum path latency between LB paths [8, 98], we here focus on the difference in inflated latency under congestion imbalance. We further study the imbalance in throughput and packet loss incurred by congestion imbalance, which disappears with the congestion imbalance and is thus hard to capture.

Latency-based Congestion Detection To detect congestion imbalance, we need to first find congestion. We are interested in an effective way of detecting congestion in the Internet. Latency inflation during TCP sessions has been used to infer congestion for TCP congestion control since TCP Dual and Vegas decades ago [115, 116]. In latency-based congestion control, the path latency is updated almost every round-trip time to infer transient congestion [115, 117]. Although our interest is not to detect transient congestion that

requires high-frequency latency samples, the idea of using latency inflation as an indicator to congestion is also applicable to detecting long-term congestion. In 2014, Luckie *et al.* [96] proposed the time-series latency probes (TSLP) method that detects persistent inter-domain congestion by observing elevated latency in latency time series collected over a long period of time. After that, the TSLP method was used to study the impact of congestion on the African IXPs in [118] and was further improved and validated in [95]. Our work also uses latency probes to detect congestion, but differs in two respects. First, we conduct a comparative study of several latency-based metrics for congestion detection to choose the best metric. Second, we further propose to detect congestion with a very small number of samples rather than long-term latency time series, which greatly improves the scalability and enables congestion detection at scale. Note that since our goal is to characterize congestion imbalance at scale, we want to detect paths that collectively have throughput drop and packet loss. It is not our goal to ensure that every detection is accurate and rigorously validate if the performance degradation is indeed due to congestion as in [95]. Although throughput drop and packet loss, as a result of congestion, can provide partial validation, it is unclear that how large a throughput drop can be interpreted as congestion [94]. A complete validation of inferred congestion requires feedback from the network operators [95].

Throughput Estimation and Modeling Our work is also related with those that use network probes to estimate available bandwidth or throughput. Available bandwidth estimation techniques send a sequence of probes and use the change either in inter-probe time or between the sending and receiving rates to estimate the available bandwidth [13, 119]. They generally have at least one of the following drawbacks preventing them from being used to measure available bandwidth at scale: 1) control is required at both the sender and the receiver sides, e.g., Pathload [13] and Spruce [14]; 2) performance varies for different network settings and there is no guarantee of accuracy [120] and that 3) estimation may incur high measurement overhead or require high resolution on inter-probe time [120].

Similar to available bandwidth estimation, there are extensive research efforts modeling TCP throughput with latency and packet loss [121]. He *et al.* [122] show that using latency and packet loss measured before a TCP session to predict TCP throughput could result in large errors and provide significantly less accuracy than using those metrics measured during the TCP session. A recent work [123] greatly improved the modeling accuracy by incorporating the congestion window size in the model, which however limits its usage to throughput prediction during TCP sessions. Considering these drawbacks and limitations, none of the works above provides a viable light-weight solution to characterize throughput imbalance between LB paths at scale with network probing.

Applications Performance imbalance between LB paths implies that there is one path better than another. In 2014, Wu *et al.* [92] observed that the LB paths from the PlatnetLab nodes to DCs could experience different levels of latency inflation and demonstrated the benefits of leveraging load balancing for latency-sensitive applications. Our work on congestion imbalance can extend the benefits to applications sensitive to throughput and packet loss. There are also many existing works addressing the performance imbalance issue in DCs, where it is clear that harnessing congestion imbalance improves performance [3, 93].

CHAPTER VI

Contributions and Future Works

This chapter concludes the dissertation. We highlight the key contributions and limitations of this dissertation and discuss potential future directions.

6.1 Key Contributions

This dissertation evaluates and improves Internet load balancing, i.e., global and path-level load balancing, with large-scale latency measurements. It supports the following thesis: *large-scale latency measurements, either passively or actively collected, are key to the evaluation and improvement of Internet load balancing; we can use them not only to systematically characterize and understand the performance issues, but also to develop data-driven approaches.* Specifically, it makes the following key contributions.

A Data-driven Client Aggregation for Global Load balancing AP-atoms are the first data-driven client aggregation designed for global load balancing. AP-atoms are learned from the large-scale latency measurements readily available for the content or service providers and thus incur no extra measurement overheads. The design of AP-atoms is inspired by the key finding in a comparative study of existing client aggregations: aggregating clients by attributes rather than performance is the culprit for the poor performance of existing client aggregations. The comparative study also uses the large-scale latency mea-

surements for the performance evaluation of existing client aggregations. As AP-atoms are generated from data streams continuously collected over time, they are adaptive to changing network conditions.

A Cloud-Centric View of Latency Imbalance Among LB Paths This dissertation is the first to evaluate latency imbalance from the perspective of public clouds. Latency imbalance, previously deemed insignificant, is found to be both significant and prevalent from the clouds to the public Internet. Considering that passive large-scale measurements are generally hard to obtain, we use network probing to actively collect latency measurements. We develop Flipr, a network prober that can efficiently measure latency imbalance between a source and a destination such that we can easily scale our experiments to measure from our VPs to Internet-scale addresses. We present a global view of latency imbalance and show that latency imbalance is stable over time and affects a wide range of applications, especially latency-sensitive applications.

Measuring Congestion Imbalance Among LB Paths at Scale This dissertation is the first to evaluate congestion imbalance at scale in the Internet. In contrast to latency imbalance, which are stable over time, congestion imbalance occurs with congestion that is typically short-lived. To detect congestion imbalance quickly and scalably, we develop Congi that uses SVM classifiers to detect congestion with a very small number of samples. We use Congi to present a global view of congestion imbalance from our geographically-diverse VPs to Internet-wide addresses. We confirm that congestion imbalance is also prevalent in the Internet and has significant impacts on latency- and throughput-sensitive applications.

6.2 Limitations

As with other data-driven approaches, our results are affected by the data we have. AP-atoms have better accuracy for client aggregations that have higher data density. The

measurement of latency imbalance is biased towards destinations with stable flow latencies because we are likely to discard destinations with unstable latencies. Congi is limited to measuring congestion imbalance in wired networks (or Internet core networks) where latency of uncongested paths is stable. In contrast, latency in wireless networks, e.g., cellular, is typically varying and causes Congi to easily make false inference.

6.3 Future Work

In this dissertation, we focus on using large-scale latency measurements to evaluate the performance issues of Internet load balancing and their impacts on applications. Apart from AP-atoms, no practical solutions are implemented to solve latency and congestion imbalance. Further, we take an end-to-end approach to characterize latency and congestion imbalance. The root causes are still not fully understood. Considering the importance of the performance imbalance issue, we would expect the following directions worth further exploring.

Understanding Diamonds that Cause Latency and Congestion Imbalance As mentioned, diamonds are the causes for latency and congestion imbalance. This dissertation has discussed the simplest form of diamonds in Chapter III, but there are still many questions left unanswered, e.g., how to discover diamonds of more complex structures, how prevalent diamonds are in the Internet, and how to locate these diamonds. Answering these questions may require developing more advanced probing tools that can see into networks currently invisible to our probers, e.g., invisible MPLS networks [77]. Further, if access to the physical infrastructure is available, we can delve into the root causes for congestion imbalance, e.g., router misconfigurations and poorly-designed load balancing algorithms. The ability to pinpoint diamonds in the Internet could help network operators better debug and design their networks.

Applying More Advanced Machine Learning Techniques The machine learning techniques we used in this dissertation are very basic ones for pattern recognition and classification problems. We expect that more advanced machine learning techniques would be able to achieve higher accuracy and be more powerful in taking large volumes of data streams as input for real-time monitoring and management. It is also interesting to study how to automate the learning process, e.g., using feature engineering to automatically extract features for prediction and using reinforcement learning to gradually learn the best decisions. Moreover, we could use machine learning techniques to push the limit of existing probers. Existing probers, e.g., Zmap [124] and Yarrp [47], are mostly used to discover active addresses or topology, which is unrelated to performance. When used to measure performance, e.g., Scamper [125], the sending rate is controlled at a very low level to avoid self-induced queueing [95]. It would be interesting to develop a prober that is aware of self-induced errors and meanwhile sends probes at its maximum rate. Such a prober can greatly accelerate large-scale measurement campaigns and enable real-time monitoring of networks at scale.

Solving Latency and Congestion Imbalance at Different Layers We can explore potential solutions for latency and congestion imbalance at different layers. At the network layer, we can revisit existing load balancing algorithms in terms of latency and congestion imbalance and try to figure out better designs. We can improve routing algorithms to be aware of congestion imbalance. To avoid routing instability, routes can be updated at a large time scale to mitigate persistent inter-domain congestion [95]. Further, in SDN-based networks, network operators can control paths by considering congestion imbalance at finer time granularity. At the transport layer, we can design end-to-end approach to tackle congestion imbalance. More specifically, we can optimize transport layer protocols to choose paths in a smarter way that is aware of latency and congestion imbalance or design multipath protocols to adaptively shift load across paths. As demonstrated by our experimen-

tal results, there is a significant performance difference between using the congested and uncongested paths. This implies that it is promising to unleash the power of multi-path transport layer protocols, e.g., multi-path TCP or QUIC, to tackle congestion imbalance and that the performance of throughput-sensitive applications, e.g., video streaming, would significantly benefit from considering congestion imbalance. If all the solutions above are not viable due to challenges in deployment, application-layer solutions can be researched. Sending redundant requests can explore LB paths, and analyzing measured performance data of paths can detect congestion.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Fangfei Chen, Ramesh K. Sitaraman, and Marcelo Torres. End-user mapping: Next generation request routing for content delivery. In *SIGCOMM*, 2015.

[2] Brice Augustin, Xavier Cuvellier, Benjamin Orgogozo, Fabien Viger, Timur Friedman, Matthieu Latapy, Clémence Magnien, and Renata Teixeira. Avoiding traceroute anomalies with Paris Traceroute. In *IMC*, 2006.

[3] Naga Katta, Aditi Ghag, Mukesh Hira, Isaac Keslassy, Aran Bergman, Changhoon Kim, and Jennifer Rexford. Clove: Congestion-aware load balancing at the virtual edge. In *CoNEXT*, 2017.

[4] Rupa Krishnan, Harsha V. Madhyastha, Sridhar Srinivasan, Sushant Jain, Arvind Krishnamurthy, Thomas Anderson, and Jie Gao. Moving beyond end-to-end path information to optimize CDN performance. In *IMC*, pages 190–201, 2009.

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

[6] Michael J. Freedman, Mythili Vutukuru, Nick Feamster, and Hari Balakrishnan. Geographic locality of IP prefixes. In *IMC*, 2005.

[7] Michael J. Freedman, Mythili Vutukuru, Nick Feamster, and Hari Balakrishnan. Geographic locality of IP prefixes. In *USENIX*, pages 153–158, 2005.

[8] Brice Augustin, Timur Friedman, and Renata Teixeira. Measuring load-balanced paths in the Internet. In *IMC*, 2007.

[9] Kevin Vermeulen, Stephen D. Strowes, Olivier Fourmaux, and Timur Friedman. Multilevel MDA-Lite Paris Traceroute. In *IMC*, 2018.

[10] Imtiaz Parvez, Ali Rahmati, Ismail Guvenc, Arif I. Sarwat, and Huaiyu Dai. A survey on low latency towards 5G: RAN, core network and caching solutions. *IEEE Communications Surveys & Tutorials*, 20(4):3098–3130, 2018.

[11] Frank Dabek, Russ Cox, Frans Kaashoek, and Robert Morris. Vivaldi: A decentralized network coordinate system. In *SIGCOMM CCR*, pages 15–26, 2004.

[12] Jonathan Ledlie, Paul Gardner, and Margo I. Seltzer. Network coordinates in the wild. In *NSDI*, volume 7, pages 299–311, 2007.

[13] Manish Jain and Constantinos Dovrolis. End-to-end available bandwidth: Measurement methodology, dynamics, and relation with TCP throughput. In *SIGCOMM CCR*, 2002.

[14] Jacob Strauss, Dina Katabi, and Frans Kaashoek. A measurement study of available bandwidth estimation tools. In *IMC*, 2003.

[15] Our tool and dataset. `https://github.com/yibopi/latency-imbalance`.

[16] Venkata N. Padmanabhan and Lakshminarayanan Subramanian. An investigation of geographic mapping techniques for Internet hosts. In *SIGCOMM*, pages 173–185, 2001.

[17] Matt Calder, Ashley Flavel, Ethan Katz-Bassett, Ratul Mahajan, and Jitendra Padhye. Analyzing the performance of anycast CDN. In *IMC*, pages 531–537, 2015.

[18] Citrix. NetScaler Global Server Load Balancer. `https://docs.citrix.com/en-us/netscaler/12/global-server-load-balancing.html`.

[19] Hao Jiang and Constantinos Dovrolis. Passive estimation of TCP round-trip times. In *SIGCOMM*, 2002.

[20] Haowen Tang, Fangming Liu, Guobin Shen, and Chuanxiong Guo. UniDrive: Synergize multiple consumer cloud storage services. In *Proceedings of the 16th Annual Middleware Conference*, pages 137–148, 2015.

[21] John S. Otto, Mario A. Sánchez, John P. Rula, and Fabián E. Bustamante. Content delivery and the natural evolution of DNS. In *IMC*, 2012.

[22] Cedexis. Radar crowd sourcing. `https://www.cedexis.com/technology/`.

[23] CAIDA. The CAIDA AS Relationships Dataset, <May 2015>. `http://www.caida.org/data/as-relationships/`.

[24] IP2Location. Geolocate IP Address Location using IP2Location. `http://www.ip2location.com`.

[25] Bruce M. Maggs and Ramesh K. Sitaraman. Algorithmic nuggets in content delivery. In *SIGCOMM CCR*, pages 52–66, 2015.

[26] Youndo Lee and Neil Spring. Identifying and aggregating homogeneous IPv4 /24 blocks with Hobbit. In *IMC*, pages 151–165, 2016.

[27] Manaf Gharaibeh, Han Zhang, Chritos Papaopoulos, and John Heidemann. Assessing co-locality of IP blocks. In *IEEE Global Internet Symposium*, 2016.

[28] Daniel Freedman and Pavel Kisilev. Fast mean shift by compact density representation. In *CVPR*, pages 1818–1825, 2009.

[29] Ivan Selesnick. Total variation denoising (an MM algorithm). `https://goo.gl/rjBpG`, 2014.

[30] Xin Jin and Jiawei Han. Quality threshold clustering. *Encyclopedia of Machine Learning, Springer*, pages 820–820, 2011.

[31] Anthony Danalis, Collin McCurdy, and Jeffrey S. Vetter. Efficient quality threshold clustering for parallel architectures. In *Parallel and Distributed Processing Symposium*, pages 1068–1079, 2012.

[32] Tsunemasa Hayashi and Toshiaki Miyazaki. High-speed table lookup engine for IPv6 longest prefix match. In *GLOBECOM*, pages 1576–1581, 1999.

[33] USC CDN coverage. `http://usc-nsl.github.io/cdn-coverage`.

[34] Matt Calder, Xun Fan, Zi Hu, Ethan Katz-Bassett, John Heidemann, and Ramesh Govindan. Mapping the expansion of Google's serving infrastructure. In *IMC*, pages 313–326, 2013.

[35] Ratul Mahajan, David Wetherall, and Tom Anderson. Understanding BGP misconfiguration. In *SIGCOMM CCR*, pages 3–16, 2002.

[36] Sipat Triukose, Sebastien Ardon, Anirban Mahanti, and Aaditeshwar Seth. Geolocating ip addresses in cellular data networks. In *PAM*, pages 158–167, 2012.

[37] Jakub Czyz, Mark Allman, Jing Zhang, Scott Iekel-Johnson, Eric Osterweil, and Michael Bailey. Measuring IPv6 adoption. In *SIGCOMM*, 2014.

[38] Sandeep Kumar Singh, Tamal Das, and Admela Jukan. From Paris to Tokyo: On the suitability of ping to measure latency. In *IMC*, Barcelona, Spain, 2013. ACMPress.

[39] Mark Gondree and Zachary NJ Peterson. Geolocation of data in the cloud. In *Proceedings of the third ACM conference on Data and application security and privacy*, pages 25–36, 2013.

[40] Junchen Jiang et al. Via: Improving internet telephony call quality using predictive relay selection. In *SIGCOMM*, 2016.

[41] Zhiruo Cao, Zheng Wang, and Ellen Zegura. Performance of hashing-based schemes for Internet load balancing. In *INFOCOM*, pages 332–341, 2000.

[42] Zachary Weinberg, Shinyoung Cho, Nicolas Christin, Vyas Sekar, and Phillipa Gill. How to catch when proxies lie: Verifying the physical locations of network proxies with active geolocation. In *IMC*, pages 203–217, 2018.

[43] Yilong Geng, Shiyu Liu, Zi Yin, Ashish Naik, Balaji Prabhakar, Mendel Rosenblum, and Amin Vahdat. Exploiting a natural network effect for scalable, fine-grained clock synchronization. In *NSDI*, pages 81–94, 2018.

[44] Ethan Katz-Bassett, Harsha V. Madhyastha, Vijay Kumar Adhikari, Colin Scott, Justine Sherry, Peter Van Wesep, Thomas E. Anderson, and Arvind Krishnamurthy. Reverse traceroute. In *NSDI*, 2010.

[45] MDA-Lite repository. `https://gitlab.planet-lab.eu/cartography/multilevel-mda-lite`.

[46] Abhinav Pathak, Himabindu Pucha, Ying Zhang, Y. Charlie Hu, and Z. Morley Mao. A measurement study of Internet delay asymmetry. In *PAM*, 2008.

[47] Robert Beverly. Yarrp'ing the Internet: Randomized high-speed active topology discovery. In *IMC*, pages 413–420, 2016.

[48] Perry R. Hinton. Statistics explained. Routledge, 2014.

[49] Christophe Leys, Christophe Ley, Olivier Klein, Philippe Bernard, and Laurent Licata. Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median. *Journal of Experimental Social Psychology*, 49(4):764–766, 2013.

[50] Ki Suh Lee, Han Wang, Vishal Shrivastav, and Hakim Weatherspoon. Globally synchronized time via datacenter networks. In *SIGCOMM*, 2016.

[51] Cristina D. Murta, Pedro R. Torres Jr, and Prasant Mohapatra. Qrpp1-4: Characterizing quality of time and topology in a time synchronization network. In *Globecom*, pages 1–5, 2006.

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

[53] Ip latency statistics. https://enterprise.verizon.com/terms/latency/, July 2019.

[54] Mohammed S. Elbamby, Cristina Perfecto, Mehdi Bennis, and Klaus Doppler. Toward low-latency and ultra-reliable virtual reality. *IEEE Network*, 32(2):78–84, 2018.

[55] Roy D. Yates, Mehrnaz Tavan, Yi Hu, and Dipankar Raychaudhuri. Timely cloud gaming. In *INFOCOM*, 2017.

[56] Sadjad Fouladi, John Emmons, Emre Orbay, Catherine Wu, Riad S. Wahby, and Keith Winstein. Salsify: low-latency network video through tighter integration between a video codec and a transport protocol. In *NSDI*, 2018.

[57] CAIDA. Routeviews prefix to AS mappings dataset for IPv4 and IPv6. https://www.caida.org/data/routing/routeviews-prefix2as.xml, Sep 2018.

[58] Timothy Wood, Prashant J. Shenoy, Alexandre Gerber, Jacobus E. van der Merwe, and Kadangode K. Ramakrishnan. The case for enterprise-ready virtual private clouds. In *HotCloud*, 2009.

[59] Sathiya Kumaran Mani, Ramakrishnan Durairajan, Paul Barford, and Joel Sommers. An architecture for IoT clock synchronization. In *Proceedings of the 8th International Conference on the Internet of Things*, 2018.

[60] Alexander Marder, Matthew Luckie, Amogh Dhamdhere, Bradley Huffaker, kc claffy, and Jonathan M. Smith. Pushing the boundaries with bdrmapit: Mapping router ownership at Internet scale. In *IMC*, 2018.

[61] CAIDA. The CAIDA Internet Topology Data Kit. `https://www.caida.org/data/internet-topology-data-kit`, August 2018.

[62] AFRINIC Extended Allocation and Assignment Reports. `ftp://ftp.afrinic.net/pub/stats/afrinic/`, 2019.

[63] RIPE Extended Allocation and Assignment Reports. `ftp://ftp.afrinic.net/pub/stats/ripencc/`, 2019.

[64] LACNIC Extended Allocation and Assignment Reports. `ftp://ftp.afrinic.net/pub/stats/lacnic/`, 2019.

[65] APNIC Extended Allocation and Assignment Reports. `ftp://ftp.afrinic.net/pub/stats/apnic/`, 2019.

[66] ARIN Extended Allocation and Assignment Reports. `ftp://ftp.afrinic.net/pub/stats/arin/`, 2019.

[67] Euro-IX IXP Directory. `https://www.euro-ix.net/tools/ixp-directory`, 2019.

[68] PeeringDB. `https://peeringdb.com/api`, 2019.

[69] Packet Clearing House: Internet Exchange Directory. `https://www.pch.net/applications/ixpdir/menu_download.php`, 2019.

[70] CAIDA. The CAIDA AS Relationships Dataset. https://www.caida.org/data/as-relationships/, Feb 2019.

[71] Google cloud network service tiers. `https://cloud.google.com/network-tiers#tab1`.

[72] Matthew Luckie, Amogh Dhamdhere, kc Claffy, and David Murrell. Measured impact of crooked traceroute. In *CCR*, pages 14–21, 2011.

[73] Ramesh Govindan and Vern Paxson. Estimating router ICMP generation delays. In *PAM*, 2002.

[74] Yves Vanaubel, Pascal Mérindol, Jean-Jacques Pansiot, and Benoit Donnet. Through the wormhole: Tracking invisible MPLS tunnels. In *IMC*, 2017.

[75] Yves Vanaubel, Pascal Mérindol, Jean-Jacques Pansiot, and Benoit Donnet. MPLS under the microscope: Revealing actual transit path diversity. In *IMC*, 2015.

[76] Joel Sommers, Paul Barford, and Brian Eriksson. On the prevalence and characteristics of MPLS deployments in the open Internet. In *IMC*, 2011.

[77] Benoit Donnet, Matthew Luckie, Pascal Mérindol, and Jean-Jacques Pansiot. Revealing MPLS tunnels obscured from traceroute. In *SIGCOMM CCR*, 2012.

[78] Abhinav Pathak, Ming Zhang, Y. Charlie Hu, Ratul Mahajan, and Dave Maltz. Latency inflation with MPLS-based traffic engineering. In *IMC*, 2011.

[79] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Mohan Nanduri, and Roger Wattenhofer. Achieving high utilization with software-driven WAN. In *SIGCOMM CCR*, 2013.

[80] Sushant Jain et al. B4: Experience with a globally-deployed software defined WAN. In *SIGCOMM*, pages 3–14, 2013.

[81] Ramakrishnan Durairajan, Sathiya Kumaran Mani, Joel Sommers, and Paul Barford. Time's Forgotten: Using NTP to understand Internet latency. In *HotNets*, 2015.

[82] Sathiya Kumaran Mani, Ramakrishnan Durairajan, Paul Barford, and Joel Sommers. Mntp: enhancing time synchronization for mobile devices. In *In Proceedings of the 2016 Internet Measurement Conference*, 2016.

[83] Amazon AWS IoT. `https://aws.amazon.com/iot/`.

[84] Bamba Gueye, Artur Ziviani, Mark Crovella, and Serge Fdida. Constraint-based geolocation of Internet hosts. *TON*, 14(6):1219–1232, 2006.

[85] Planetlab locations. `https://www.planet-lab.org/db/pub/sites.php`.

[86] iplane project dataset. `http://web.eecs.umich.edu/~harshavm/iplane/iplane_logs/data/`.

[87] Karyn Benson, Rafael Dowsley, and Hovav Shacham. Do you know where your cloud files are? In *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, pages 73–82, 2011.

[88] Yang Xu, Chenguang Yu, Jingjiang Li, and Yong Liu. Video telephony for end-consumers: Measurement study of Google+, iChat, and Skype. In *IMC*, pages 371–384, 2012.

[89] G.107. *The E-Model, a computational model for user in transmission planning*, 2017. `https://www.itu.int/rec/T-REC-G.107`.

[90] Haiyong Xie and Yang Richard Yang. A measurement-based study of the skype peer-to-peer VoIP performance. In *IPTPS*, 2012.

[91] 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*, 2017.

[92] Zhe Wu, Curtis Yu, and Harsha V. Madhyastha. CosTLO: Cost-effective redundancy for lower latency variance on cloud storage services. In *NSDI*, 2015.

[93] Mohammad Alizadeh *et al.* CONGA: Distributed congestion-aware load balancing for datacenters. In *SIGCOMM*, 2014.

[94] Srikanth Sundaresan, Xiaohong Deng, Yun Feng, Danny Lee, and Amogh Dhamdhere. Challenges in inferring Internet congestion using throughput measurements. In *IMC*, 2017.

[95] 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 *SIGCOMM*, 2018.

[96] Matthew Luckie, Amogh Dhamdhere, David Clark, Bradley Huffaker, and kc Claffy. Challenges in inferring Internet interdomain congestion. In *IMC*, 2014.

[97] M-Lab. NDT (Network Diagnostic Tool). https://www.measurementlab.net/tests/ndt/.

[98] Yibo Pi, Sugih Jamin, Peter Danzig, and Feng Qian. Latency imbalance among Internet load-balanced paths: A cloud-centric view. In *SIGMETRICS*, 2019.

[99] Bob Briscoe et al. Reducing Internet latency: A survey of techniques and their merits. *IEEE Communications Surveys & Tutorials*, 2014.

[100] Charles Truong, Laurent Oudre, and Nicolas Vayatis. Selective review of offline change point detection methods. *Signal Processing*, 2020.

[101] Charles Truong, Laurent Oudre, and Nicolas Vayatis. ruptures: change point detection in Python. arXiv preprint arXiv:1801.00826, 2018.

[102] Hang Guo and John Heidemann. Detecting ICMP rate limiting in the Internet. In *PAM*, pages 3–17, 2018.

[103] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: a library for support vector machines. *ACM transactions on intelligent systems and technology*, 2011.

[104] Ganesh R. Naik and Dinesh Kant Kumar. Twin SVM for gesture classification using the surface electromyogram. *IEEE Transactions on Information Technology in Biomedicine*, 2009.

[105] Toke Høiland-Jørgensen, Bengt Ahlgren, Per Hurtig, and Anna Brunstrom. Measuring latency variation in the Internet. In *CoNEXT*, 2016.

[106] Harsha V. Madhyastha, Tomas Isdal, Michael Piatek, Colin Dixon, Thomas Anderson, Arvind Krishnamurthy, and Arun Venkataramani. iplane: An information plane for distributed services. In *OSDI*, pages 367–380, 2006.

[107] Erik Nygren, Ramesh K. Sitaraman, and Jennifer Sun. The Akamai network: a platform for high-performance internet applications. In *ACM SIGOPS Operating Systems Review*, pages 2–19, 2010.

[108] Cheng Huang, Ivan Batanov, and Jin Li. A practical solution to the client-LDNS mismatch problem. In *SIGCOMM CCR*, 2012.

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

[110] Rafael Almeida, Osvaldo Fonseca, Elverton Fazzion, Dorgival Guedes, Wagner Meira, and Ítalo Cunha. A characterization of load balancing on the IPv6 Internet. In *PAM*, 2017.

[111] Eric Elena, Jean-Louis Rougier, and Stefano Secci. Characterisation of AS-level path deviations and multipath in Internet routing. In *6th EURO-NGI Conference on Next Generation Internet*, 2010.

[112] Young Hyun Luckie, Matthew and Bradley Huffaker. Traceroute probe method and forward IP path inference. In *SIGCOMM*, 2008.

[113] Yibo Pi, Sugih Jamin, Peter Danzig, and Jacob Shaha. Ap-atoms: A high-accuracy data-driven client aggregation for global load balancing. *IEEE/ACM Transactions on Networking*, 2018.

[114] Zhiruo Cao, Zheng Wang, and Ellen Zegura. Performance of hashing-based schemes for Internet load balancing. In *INFOCOM*, 2000.

[115] Lawrence S. Brakmo, Sean W. O'Malley, and Larry L. Peterson. TCP Vegas: New techniques for congestion detection and avoidance. In *Proceedings of the conference on Communications architectures, protocols and applications*, 1994.

[116] Belma Turkovic, Fernando A. Kuipers, and Steve Uhlig. Fifty shades of congestion control: A performance and interactions evaluation. In *arXiv preprint arXiv:1903.03852*, 2019.

[117] Venkat Arun and Hari Balakrishnan. Copa: Practical delay-based congestion control for the Internet. In *NSDI*, 2018.

[118] Rodérick Fanou, Francisco Valera, and Amogh Dhamdhere. Investigating the causes of congestion on the African IXP substrate. In *IMC*, 2017.

[119] Ningning Hu and Peter Steenkiste. Evaluation and characterization of available bandwidth probing techniques. *IEEE JSAC*, 2003.

[120] Alok Shriram, Margaret Murray, Young Hyun, Nevil Brownlee, Andre Broido, and Marina Fomenkov. Comparison of public end-to-end bandwidth estimation tools on high-speed links. In *International Workshop on Passive and Active Network Measurement*, 2005.

[121] Jitendra Padhye, Victor Firoiu, Don Towsley, and Jim Kurose. Modeling TCP throughput: A simple model and its empirical validation. In *SIGCOMM*, 1998.

[122] Qi He, Constantine Dovrolis, and Mostafa Ammar. On the predictability of large transfer TCP throughput. In *SIGCOMM*, 2005.

[123] Yi Cao, Javad Nejati, Aruna Balasubramanian, and Anshul Gandhi. Econ: Modeling the network to improve application performance. In *IMC*, 2019.

[124] Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. ZMap: Fast Internet-wide scanning and its security applications. In *USENIX Security*, 2013.

[125] Matthew Luckie. ZMap: Scamper: a scalable and extensible packet prober for active measurement of the Internet. In *IMC*, 2010.