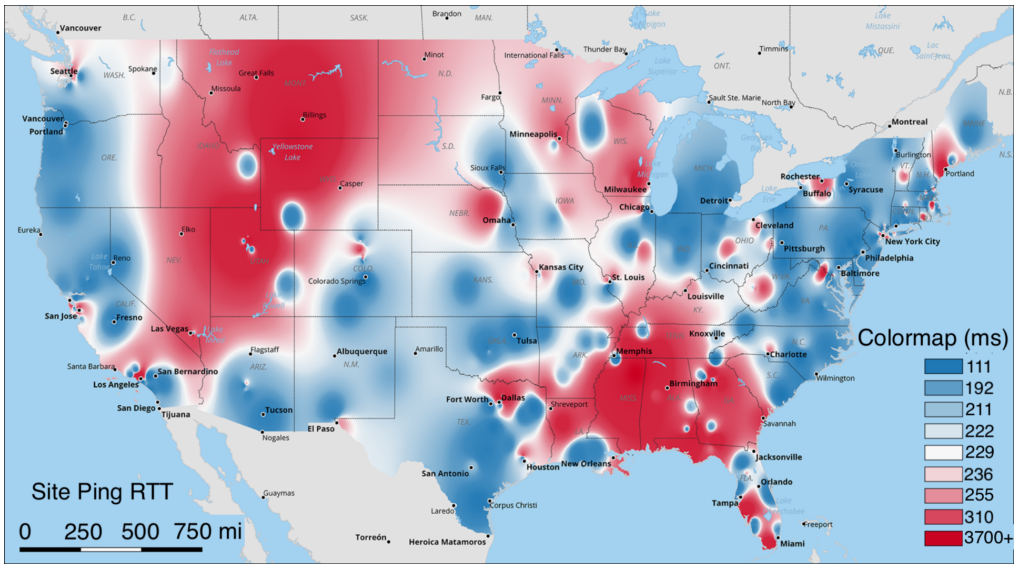
Evaluating and Mapping Internet Connectivity in the United States



Samuel Goldman
Evan Goldstein
Christopher Myers
David Vollum

Advised by: Professor Craig Wills

Worcester Polytechnic Institute
Department of Computer Science

MQP-CEW-2001
March 2019

Abstract

We evaluated Internet connectivity in the United States, drawn from different definitions of connectivity and different methods of analysis. Using DNS cache manipulation, traceroutes, and a crowdsourced site ping method we identify patterns in connectivity that correspond to higher population or coastal regions of the US. We analyze the data for quality strengths and shortcomings, establish connectivity heatmaps, state rankings, and statistical measures of the data. We give comparative analyses of the three methods and present suggestions for future work building off this report.

Contents

List of Figures

List of Code Snippets

List of Tables

1

Introduction

Internet access is an increasingly important part of the American economy and everyday life. Common tasks like applying for a job, keeping in touch with friends, and education all require Internet connectivity. Major technology firms are often in the public spotlight, and common Internet services can be found everywhere, such as music and video streaming, e-books, and shopping. In 2018 an industry group comprised of major technology firms estimated that the "internet sector" of the economy alone represented $2.1 trillion a year, or about 10% of the US economy [30].

Unfortunately, not all parts of the US are as well connected as others, even by measures from simple personal anecdotes. For example, in rural areas the best connection frequently is not good enough to stream a movie, while driving just 45 minutes to a more urban area will yield a connection an order of magnitude better. Subjective measures like this are common everywhere across the US, but there is little scientifically-rigorous or complete data available. For instance, the Federal Communications Commission (FCC) has a map of estimated broadband deployment [7], but simple broadband deployment is not necessarily a good measure of how well connected people in those areas actually are.

To address these problems, we set out to gather and analyze as much data as possible on how well connected Americans are to the Internet, using various means and measures. Our hypothesis is simple: although there may be some regional variations or "spottiness," there is a relationship between your location in the US and what sort of Internet service you can expect. We hypothesized that areas near each other are likely to have similar connectivity to one another, and that these similarities will form large-scale trends that should be visible on a map, interpretable by non-technical readers.

The end goal of our project was to find if such relationships exist, and if so, to conduct analyses on the data to make the differences between areas of the US clear. An important quality of our work is that it should be scientifically rigorous and statistically valid – that is, we should avoid systemic error and account for random error in our calculations – so a great deal of effort was expended on ensuring the validity of our results.

The reminder of the report is organized into chapters as follows:

In Chapter 2 we present background information on the fundamentals of the Internet, information relevant to our methods, and research conducted on prior works and attempts at measuring connectivity.

In Chapter 3 we rely upon collected background research to define metrics for Internet connectivity. These are not limited to traditional metrics used by consumers (such as speed), instead taking a more expansive approach.

Chapter 4 describes a brief overview of our three main methods for collecting & analyzing data on Internet connectivity. We also present a brief overview of the statistical methods used, and an explanation of methods that were considered but ultimately rejected.

Chapters 5 to 7 present detailed methods of the design, implementation, and analysis for each of our three methods for collecting data. Their analyses differ since each data set is somewhat different, but the fundamentals (e.g. definitions of connectivity) remain largely the same.

Chapter 8 contains our comparative analysis of the results of the three data analyses. We present overall conclusions drawn from the data, in both assertions we are confident of and matters we are uncertain of.

Finally, in Chapter 9 we suggest ideas for future projects to follow up on, drawing from shortcomings noted in our methods and the unpursued methods explained in Chapter 4.

2

Background

This chapter presents an overview of the of foundations the Internet and elements critical to understanding how our methods and analyses work. Section 2.1 covers the basics of how Internet the Internet is organized, while Section 2.3 provides an example of a common metric that consumers use to test their Internet access and Section 2.2 explores how traceroutes can be used to analyze a network connection's path. Section 2.4 explains how various forms of caching, while Section 2.6 gives an overview of caching's close cousin, the content delivery network (CDN). Section 2.7 presents an overview of the methods used to assign a location to an internet protocol (IP) address, which is critical to all parts of this report. Finally, Section 2.8 details some past attempts to understand Internet connectivity in the US.

2.1 Internet Architecture

The Internet is comprised of networks spread across the globe, connected together by high speed links and data centers collectively referred to as the "**backbone**." Although the networks are distributed, they do not form a mesh network. Instead, data is generally routed through hierarchical networks. These networks start at the small, local level, and progress upwards (as needed, depending on the internet service provider (ISP)'s architecture) through increasingly larger networks as data is routed to its final destination. Data packets eventually repeat the process in reverse as they approach their destination server.

Since the Internet has a hierarchical structure, end users are prone to experiencing a bottleneck that is the networks they are directly connected to: their ISP, its parent network, and so on. As a result Internet connections vary in all ways across the US, with some areas receiving over 50 megabits per second (Mbps) while others receive less than 1 according to our own experiences.

2.2 Traceroutes

A traceroute is a method for determining which servers are along a packet's route, and determining the round trip time (RTT) to each of them. Most traceroute programs work

using internet control message protocol (ICMP) but in theory anything that uses IP works.

Traceroutes hinge on the time-to-live (TTL) field of IP packets, an eight-bit field that is decremented by one every time the packet is processed [28]. Typically the TTL is set reasonably high so packets can take many hops to reach their destination, but low enough that if undeliverable they will eventually be discarded. Upon discard, most servers respond with an ICMP packet to the original host indicating TTL expiry. This message is received by the machine running the traceroute, which uses the sender's IP address and knowledge of the TTL of the packet it sent out to place the server somewhere along the route. This process is visualized in Fig. 2.1.

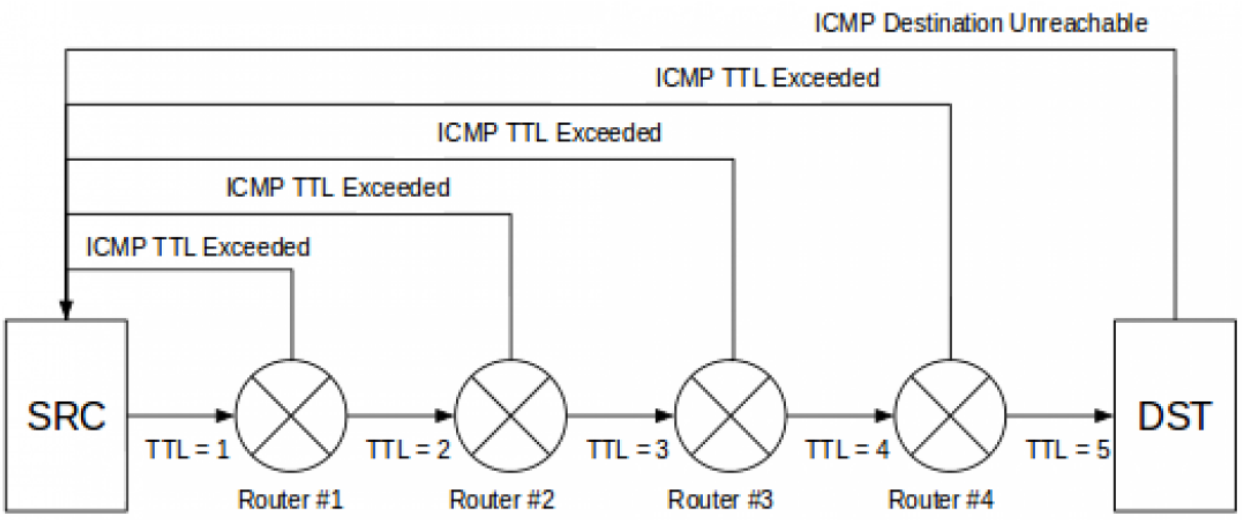


Figure 2.1: Diagram of how traceroutes work

The mechanics of a traceroute program are then a simple procedure. With $i = 1$ to start, an ICMP-based traceroute program follows the below:

1. Send an ICMP ping with TTL $= i$ to the destination server.
2. If an ICMP TTL-expired packet is received, mark the sender as hop i along the route. If a timeout is reached, skip.
3. If an ICMP ping response is received (i.e. the destination was reached), mark the destination server as the i-th hop and note the time it took to receive the packet.
4. Increment i by one and repeat from step 1.

This process produces a list of servers along the route and their associated RTTs. A sample output from the aptly-named `traceroute` utility[1] is shown in snippet 2.1. The `* * *` instances on lines 2 and 10 indicate that no response was received from these hops, so `traceroute` proceeded without them.

[1] `traceroute` and its variations can be found on virtually every system made in recent memory. Linux distributions and Windows machines ship with command-line tools `traceroute` and `tracert` respectively, while MacOS has its own dedicated GUI application as a system tool.

```
 1  192.168.1.1  16.529 ms
 2  * * *
 3  96.34.83.9  26.158 ms
 4  96.34.84.212  30.452 ms
 5  96.34.2.142  31.084 ms
 6  96.34.0.51  31.012 ms
 7  96.34.0.137  27.184 ms
 8  96.34.3.89  23.049 ms
 9  96.34.148.35  28.703 ms
10  * * *
11  108.170.246.33  26.405 ms
12  108.170.246.34  24.091 ms
13  172.217.164.142 24.720 ms
```

Snippet 2.1: Truncated output from `traceroute google.com`

2.3 Speed Tests

Speed tests are common measures used by consumers to figure out how "good" their Internet connection is. The idea is simple – connect to a website and try to transfer as much content as possible within some timeframe. The amount of data transferred can be used to calculate a data transfer speed in Mbps, where higher is better. Examples include Ookla's aptly-named speedtest.net, or Netflix's fast.com.

In theory speed tests are a sound idea, since speed of data transfer is immediately obvious to a consumer. Connection speed dictates everything from how long it takes pages to load, to how long videos have to buffer, to how fast you can download files. In practice, though, speed testing is challenging and often inaccurate. In order to accurately measure speed, you must have a server receiving and sending test data with at *least* the same speed as the summed connections of all of its users at any one time, posing an immediate performance challenge. Second, the measured connection is only the one between you and whatever data center the speed test server is located in, which may not represent your overall connectivity. Finally, ISPs are doubtless familiar with speed test sites and have a serious motive to bias connections in favor of them, to trick their customers into thinking they have better Internet than they actually do [31].

As a result of these combined factors, we decided against using speed tests as any method for assessing Internet connectivity.

2.4 Caching

RFC 7234 describes caching as the process of storing "cacheable responses in order to reduce the response time and network bandwidth consumption on future, equivalent requests" [8]. Caches save frequently-used resources for a given amount of time to reduce the load on upstream resources, from the origin server to Internet infrastructure that provides service along the way. Several components of the Internet may contain caches:

the user's browser, CDNs (see Section 2.6), DNS servers, and others [25]. These all work together to free up Internet bandwidth and provide a smoother experience to end users.

Most (if not all) web browsers implement hypertext transfer protocol (HTTP) caching, saving commonly-used website resources locally [13]. For example, if users make frequent requests to `amazon.com`, a cache between Amazon's servers and their users may store a copy of Amazon's logo as it does not change frequently. When `amazon.com` then requests the logo, the browser's cache responds with the copy, preventing the request from going all of the way to Amazon's servers. This reduces network load that would normally be caused by the request. Of course, sometimes resources do change, so an origin server can give cacheable resources an expiration time after which the cache should validate the freshness resource before fulfilling the request [8].

Outside of the browser, web resources are often cached by CDNs. These networks are set up by companies that operate major websites and are intended to move the delivery of frequently accessed resources closer to end users, both in network and geographic terms. CDNs are discussed in more detail in Section 2.6.

Beyond caching resources that the end user sees and interacts with, other things, like DNS results, use caching as well. As discussed in more detail in section 2.5, recursive DNS servers retrieve requests from other DNS servers and often cache them for responding to future requests [23]. Similar to the expiration time in HTTP caching, DNS caching has a TTL that forces the DNS resolver to request a fresh answer.

2.5 Domain Name System

DNS, a key component of Internet infrastructure and connectivity, is a hierarchical system responsible for converting domain names to IP addresses. Domain names are human-readable names used to identify servers and websites. They are easier to remember than IP addresses, and can point to more than one address depending on geographic location or load-balancing constraints to provide higher-performance access to websites for end-users. Domain names contain one or more parts called *labels*, separated by dots. The rightmost label is the top level domain (TLD) (e.g. `com`, `org`, `net`, etc.). The DNS hierarchy tree subdivides into "zones," with each zone containing one or more domain names and sub-domains. Databases of domain names are maintained by DNS Name Servers, which resolve domains to IP addresses (Fig. 2.2). There are two types of name servers: authoritative and recursive.

Beyond caching resources that the end user sees and interacts with, other things, like DNS results, use caching as well. As discussed in more detail in section 2.5, recursive DNS servers retrieve requests from other DNS servers and often cache them for responding to future requests [23]. Similar to the expiration time in HTTP caching, DNS caching has a TTL that forces the DNS resolver to request a fresh answer.

2.5.1 Authoritative Servers

Authoritative servers are the primary source for the domain names within a given zone. When querying for any domain, the answer will ultimately come from the authoritative

server for that domain. A DNS client can query an authoritative server directly, but it is more common that an authoritative server will be queried by a recursive server on behalf of a DNS client.

2.5.2 Recursive Servers

Recursive DNS servers work to find an IP address for a client, so that the client does not have to do the leg work of searching through other DNS servers [4]. DNS resolvers use a predetermined upstream recursive DNS server, such as one provided by an ISP [27]. This server then checks its cache and, if it has a valid answer, returns it. Otherwise, the server makes a series of iterative queries in order to find the requested name.

Take for example, `www.google.com`. If the recursive server has no knowledge of any parts of the uniform resource locator (URL), it will first query a pre-configured root DNS server for the authoritative server for the `.com` TLD. It will then query that server for `google.com`, then query the server provided from that request for `www.google.com` itself. At this point, the iterative component of the lookup is complete. At each stage of this process, the recursive server caches the results of its query, and assuming the TTL has not expired, will use that cached value instead of making a fresh query. Finally, the recursive server then returns the IP address it located, completing the recursive request from the user.

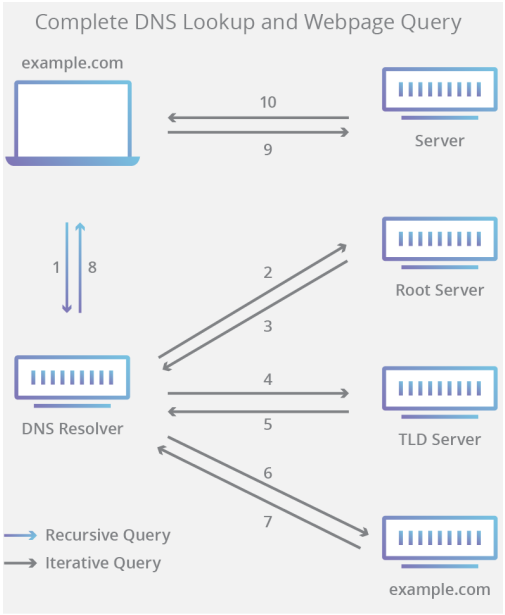


Figure 2.2: Diagram of DNS resolution [3]

2.5.3 Public Recursive DNS Servers

An important part of this project involves public recursive DNS servers. These servers are DNS servers configured to respond to requests from anyone. Whereas most ISP servers do not advertise their DNS servers to non-customers, public servers do. Some organizations, like Google and CloudFlare, provide public DNS services because they believe doing so improves the browsing experience for end users [11]. These provide the public with DNS options outside of their ISP and provide this project with an important set of geographically diverse servers.

2.6 Content Delivery Networks (CDNs)

CDNs are used by many popular websites across the Internet to deliver content quickly and efficiently to their end users. CDNs are made of data centers distributed across the world, often located near large population centers or elements of the Internet backbone.

Some CDNs are even run by an ISP directly to help alleviate strain on their infrastructure. Content owners, such as Reddit, pay CDN providers to host their content on the CDN's servers around the world to serve content more efficiently to users. One of the drawbacks of CDNs are that they can make it more difficult to update content stored within. As a result they are often used for static content that is infrequently updated, such as videos for Netflix or user content that is posted to Reddit.

2.7 IP Address Geolocation & Reverse Geocoding

Geolocation services for this project are provided by two sources: Texas A&M University's reverse geocoding application programming interface (API), and the MaxMind GeoIP2 database.

The MaxMind database is a set of binary files and an software development kit (SDK) designed to estimate the location (with varying resolution [19]) of an IP address, provided by the company MaxMind. This process is referred to as *IP geolocation*. The database s imperfect[2] but is regarded as best-in-class and is suitable for our use. Unfortunately the database is proprietary, so we have no information on how exactly MaxMind assembles and verifies it.

The Texas A&M API provides a reverse geocoding service that converts a set of existing coordinates into a city, state, zipcode, or other information. This is accomplished using open reference data sets (used in reverse; the normal function is to map city/state/zipcode into coordinates) and algorithms to efficiently query the database.

2.8 Prior Work

There have been several past attempts at evaluating US Internet connectivity, including a previous Worcester Polytechnic Institute (WPI) major qualifying project (MQP).

2.8.1 "The Internet Connected Project"

In 2018, another MQP was run at WPI, also with the goal of mapping the Internet connectivity across the US. They used traceroutes from Worcester, Massachusetts to top websites, and also performed DNS cache manipulation to collect their data. They had mixed success collecting data, but they were ultimately able to produce a map of all their DNS data interpolated to cover the entire US, shown in Fig. 2.3. They did draw any conclusions on the best or worst states [6].

The concept of DNS cache manipulation was ultimately used in our own research in evaluating and ranking Internet connectivity, discussed in Chapter 7.

[2]Inaccuracies are less a result of quality of data and more a result of the fact that IP address allocation does not follow a meaningful geographic pattern.

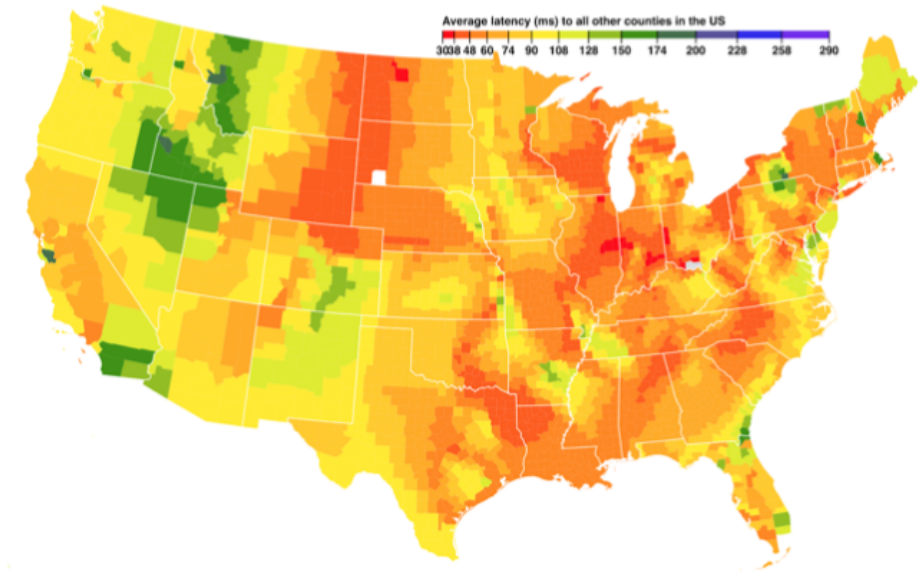


Figure 2.3: Interpolated DNS map from “The Internet Connected Project”[6]

2.8.2 Physical Mapping of Fiber-Optic Networks in the United States

In 2015 researchers at the University of Wisconsin (Madison) mapped locations of fiber backbone within the US, attempting to understand how the backbone was influenced by existing infrastructure (e.g. railroads and highways). They found strong correlation between the location of fiber lines and the locations of major roads built in the mid 20th century. This result is significant for Internet connectivity because it further highlights that cities that were well connected physically during the industrial revolution continue to be the best connected [5].

2.9 Summary

In summary, we covered the basic architecture of the Internet and how it can be studied through traceroutes or speed test, some common technologies that users will encounter like caching and the domain system, and some techniques relevant to our project like IP address geolocation and reverse geocoding. As part of our investigations we also covered a pair of notable prior works, namely a past WPI MQP and a project that mapped fiber-optic networks in the US.

All of this information was essential for the completion of our project and will be of use in understanding later chapters.

3

Definitions of Internet Connectivity

To measure Internet connectivity it is important to concretely define Internet connectivity To that end we have put together a list of possible metrics for defining it.

3.1 RTT to Everywhere

RTT is a measure of how long a packet's round trip between source and destination is. It has a major effect on latency, the quality of streaming applications such as videos, video calls, voice over Internet protocol (VoIP) calls, and multiplayer gaming. Lower RTTs indicate a better connection and a more responsive Internet experience.

By measuring a location's average RTT to any other location, we can roughly determine that connection's average quality. For example, we might find that a particular source has an average RTT of 50 ms while another has an average 150 ms, so we would say the second has a worse connection than the first.

Normalized by Distance Unnormalized RTT is a valid metric on its own but when averaged across many data points it is difficult to pin a meaningful number on. For instance, if a user lives in San Francisco, CA and has a measured RTT to Sacramento, CA of 15 ms but an RTT of 100 ms to Boston, MA (~3,000 miles away), between the two the user has an average RTT of 57.5 ms. However, 57.5 ms has no particular meaning to a user because it does not actually show up in the data. That is to say, 57.5 ms does correspond to any real measurement point, and it fails to accurately represent both the Sacramento RTT and the Boston RTT.

A possible solution is to normalize RTTs by distance, calculated by dividing RTT between source and destination by the distance in kilometers between them. This results in a measurement measured in ms/km, which is naturally a small number. This can be thought of as a measurement of infrastructure quality.

3.2 RTT to Regional Locations

It may be useful to group RTT values by region, likely either by distance or national borders, based on the principle that an Internet user may be more likely connect to a server in the same region than to one in another region. This measure of raw RTT shares the same properties as raw RTT from Section 3.1, except with fewer high-RTT destinations influencing averages.

Normalized by Distance Just like raw global RTTs, regional RTTs can be normalized by distance. Regional RTT normalized by distance is a good measure of a region's connectivity within itself, could be a better indicator of typical Internet connectivity for residents of that region.

3.3 Aggregate RTT to /24 Prefixes

A /24 prefix is the first 3 bytes of a internet protocol version 4 (IPv4) address, where the subnet mask (when expanded) is represented as 255.255.255.0. This leaves the last byte of the address free to vary, so a /24 prefix encompasses up to 256 individual addresses.

Aggregate RTT to /24 prefixes collects data for only one node per /24 prefix, assuming that the other 255 addresses are accurately represented by the single data point from their prefix. This analysis would demonstrate average connectivity between /24 networks, similar in a way to analyzing connectivity between counties or zip codes. In doing so, it would characterize the overall connectivity of that network. Additionally, we could link /24 networks to geographic areas and show which areas have the best and worst /24 prefixes.

3.4 RTT to Top Websites

Five websites (Google, Facebook, Youtube, Yahoo, and Amazon) dominate Internet traffic in the US with over 30% of the traffic share. 67% of US Internet users visit Google to search the Internet and 68% use Facebook for social media[32]. These websites (among others) play a major role in Internet connectivity and usability for a large portion of Internet users in the United States. Therefore, measuring connectivity to these websites provides a window into user experience of the Internet for a geographic location.

Beyond direct usage of top websites, major websites adopting content delivery networks (CDNS) contributes to Internet consolidation. According to the Internet Society, 87.5% of the top 1000 websites in 2018 used CDNs to speed up the delivery of their content. Of these 1000 sites, Amazon Cloudfront and Akamai provide CDN services to 474 websites. The Internet Society states that four services (Dyn, Akamai, Amazon Web Services (AWS), and Cloudflare) serve an estimated 50% of the top 1000 `.com`, `.net`, and `.org` domains [35].

The reality of a more centralized Internet is that, as the web grows ever more consolidated around these cloud services, the ability to speedily connect to any IP address becomes less of an issue for everyday users. From this vantage point, a better Internet

connection is one that provides faster load times to the top content providers, regardless of their location. This metric is similar to the RTT measurements described in 3.1, but is more focused on the end user's use case.

Normalized by Distance While users might only care about end result and how it impacts their ability to browse the web, normalizing the data by distance to the website data center could provide valuable information regarding regional infrastructure. A region showing high latency to top websites and a high ms/km rating might have room to improve, while a high latency region with low ms/km might not be able to improve. Network architects and companies looking to expand could use this data to prioritize locations, while everyday users could use it to determine if their area has the potential to improve.

3.5 Aggregate Regional RTT to Other Regions

This metric considers average connectivity measurements (such as RTT) in a political region to all other regions. These regions could be U.S. counties, zip codes, census blocks, or something else. Such aggregation does not necessarily follow the nature of network architecture, but this aggregation would highlight certain potential inequalities on political boundaries. Whatever these inequalities might be, focusing on these regions in this way would be useful to local authorities and constituents looking to improve Internet access in their areas.

3.6 Advertised speeds per region

One factor that can often dictate "internet connectivity" is the available "advertised" broadband connection speeds in a given region. It would provide an idea of both the infrastructure in the area and what the ISP can successfully market. Advertised connection speed per region provides a strong metric on what average consumers can access across the US. Comparing the available advertised speeds between regions can show relative infrastructure differences.

Maximum The maximum available connection speed across an area gives an idea of what connections are possible. This metric can be used if cost is not considered and all that is desired is the best possible connection.

Minimum The minimum connection speed available provides a strong summary of what would be considered to be the most accessible to every American.

Average The average advertised connection speed offered by broadband providers in the US provides a general metric for available speed across the US. The metric would be the average of all of the connections offered for a region.

3.7 RTT to Internet Backbone

For all non-local traffic – likely the majority, since major data centers are sparse compared to residential locations – packets will inevitably pass through the backbone en route to their destination. Backbone RTT is then a measurement of how long it takes for a user to reach the nearest backbone entry point.

Normalized by distance Normalizing RTT to backbone by distance is as useful as in other RTT methods, since it gives a better picture of infrastructure quality and avoids outlier biasing. The effect of normalization will likely be diminished, however, since RTT to backbone is technically a form of regional grouping.

3.8 Data Cap by Region

A data cap on broadband connections could reveal limited bandwidth in an area, and an attempt by the ISP to limit how much traffic consumers are generating. This is not a direct comparison, but possibly a correlation.

3.9 IPv6 Availability

Officially established as an Internet standard in 2017, internet protocol version 6 (IPv6) expands IPv4 to 128 bits in order to address the exhaustion of 32-bit IPv4 addresses. However, deployment of network hardware that supports IPv6 has been slow. According to Google, as of September 2019, the US has only reached 36.4% IPv6 adoption [12]. While this is not an immediate problem, determining where IPv6 has not been implemented in the US might show which regions are being prioritized. Overall, this definition of connectivity is a form of future Internet connectivity.

3.10 Connection Stability & Risk of Disconnection

Another way of looking at future connectivity involves determining how at risk a region is of becoming disconnected from the Internet. A community or region may have sufficient connection now, but if that connection is reliant on a single point of failure (or any degree of failure lower than the rest of the population) in the network, its future connectivity is at risk. Another form of this, although less technical, would involve whether a government entity is capable and willing to sever or curtail Internet availability.

3.11 Summary

We presented many possible metrics for Internet connectivity. Some metrics are based on technical measures (such as RTT to /24 prefixes) while others are more closely re-

lated to user experience or advertised metrics. We also explored some other more exotic connectivity metrics, such as data caps, IPv6 availability, or RTT to backbone.

4

Methods

Given the diversity in the possible definitions of Internet connectivity defined in Chapter 3 and the number of factors that determine internet connection quality, we saw a need for multiple methods of data collection and analysis. With multiple methods of data collection and multiple resulting sets of data, conclusion can be drawn by comparing the different sets of data and looking for matching trends. This section provides an overview of the methods we pursued and some that we considered but chose not to pursue.

4.1 Measurement Methods

We chose three methods for measuring Internet connectivity as described in Chapter 3: mass traceroute data analysis, crowd-sourced "site ping" data, and DNS cache manipulation. These methods were pursued in parallel in hopes that by the end of our project, their results could be compared.

RTT to Everywhere: Traceroute Analysis Two different organizations, Réseaux IP Européens (RIPE) (through its Atlas project) and Center for Applied Internet Data Analysis (CAIDA) have large, distributed networks of devices that run traceroutes to every part of the Internet around the clock. This data is publicly available for download, and totals in tens of terabytes of data. Analysis of the data can reveal useful information about connectivity and networks in the US using the normalized RTT-to-everywhere metric described in Section 3.1.

Data collection and analysis of this method is described in Chapter 5.

RTT to Top Websites: Site Ping The site ping method is a web-based, crowd-sourced approach that involves a user's web browser attempting to download a small asset from popular websites and measuring how long it takes. This method allows us to estimate how long it takes for a user to interact with a popular website, a measure of Internet connectivity described in Section 3.4.

This method is detailed in Chapter 6.

Aggregate RTT by Region: DNS Cache Manipulation The DNS cache manipulation method is a continuation from the prior MQP [6]. It uses a set of geographically diverse recursive and authoritative DNS servers. By measuring latency to the recursive DNS server and then forcing it to go to a specific authoritative server, we can measure the RTT from the location of the recursive server to the location of the authoritative server. This provides data for the aggregate regional approach described in Section 3.2.

This method is explored in more detail in Chapter 7.

4.2 Measurement Methods Not Pursued

The below section details methods that were considered or attempted, but ultimately not pursued. We leave this section here as both as advice to future researchers on what methods to try, and as a warning on what to avoid.

4.2.1 Road trip

Before the CAIDA and RIPE Atlas data was uncovered, an idea for gathering everywhere-to-everywhere data was conceived: take an enormous road trip across the US, collecting data all along the way. The idea is simple enough in theory and in practice. Just assemble a list of destinations and run constant traceroutes against them on the journey, mapping them to global positioning system (GPS) coordinates along the way.

By the end of the trip there would be traceroutes from every point along the route to all the different destinations in the list, many times over. This would yield a significant amount of data. With two people in one car taking shifts in driving (or alternately, staying in hotels for 7-8 hours at a time), it was estimated that the entire US could be roughly circumnavigated in about 7 days.

Fortunately the CAIDA and RIPE Atlas data sets were discovered well before any serious planning was underway. The idea was immediately nixed, since it turns out that nobody actually wants to spend a week in a car.

4.2.2 Network Time Protocol

Network Time Protocol (NTP) is a protocol designed to synchronize computer clocks around the world. Per the specification for the third version, NTP is designed to "maintain accuracy and robustness" despite being implemented on "unreliable" networks with "dispersive delays" [21]. As part of this protocol, the delay, or delta, between the client and server is calculated [22]. Given the precise nature of this protocol, it would be ideal for this project, which is focused on measuring the network time between geographic locations. The only requirements would be a geographically diverse set of NTP servers willing to provide the calculated delay values to their peers. Luckily, both parts of this requirement are met – mostly.

The NTP specification details a server "mode 6", that allows for "remote control queries", providing a way for remote management of certain aspects of the server [14]. One of the

these queries, the `peers` command, prompts the server to return a list of its peers, [1] along with calculated statistics for these peers – including the delay. Figure 4.1 is an example of such a command using the `ntpq` tool (with some fields removed for formatting).

```
>ntpq -c peers 127.0.0.1
     remote           refid      delay   offset  jitter
==============================================================
*ntp1.wpi.edu    130.215.32.36   0.851   -0.076   0.134
+ntp2.wpi.edu    130.215.32.36   0.646   -0.366   0.182
+ntp3.wpi.edu    130.215.144.33  1.376    0.711   0.261
```

Figure 4.1: Sample NTP Mode 6 `peers` command output

As the example shows, the `peers` command response includes both delay and jitter calculations for each peer. Coupled with IP geolocation, requesting the peers from a list of mode 6 NTP servers would be a straightforward way of getting point to point measurements. And there is no shortage of NTP servers with mode 6 enabled in the US: according to the ShadowServer Foundation, which conducts period scans for such servers, as of January 22nd, 2020 there are 522,415 such servers in the country [36]. Additionally, as Fig. 4.2 shows, they are distributed across most of the country.

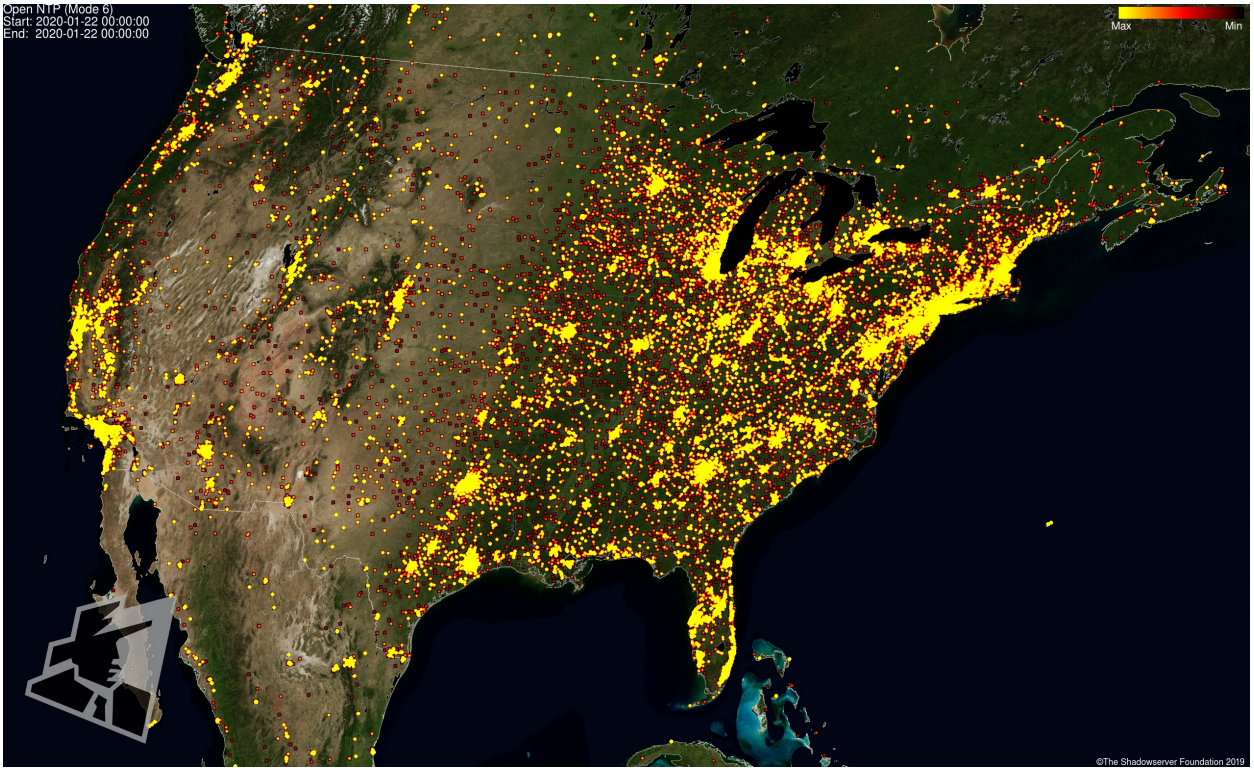


Figure 4.2: Distribution of NTP Mode 6 Servers in the US [36]

In theory, using these servers and conducting a simple survey of the delay times to each of their peers would be an ideal method for this project. Unfortunately, one of the

[1] Peers are other NTP servers that a given server is configured use for synchronization.

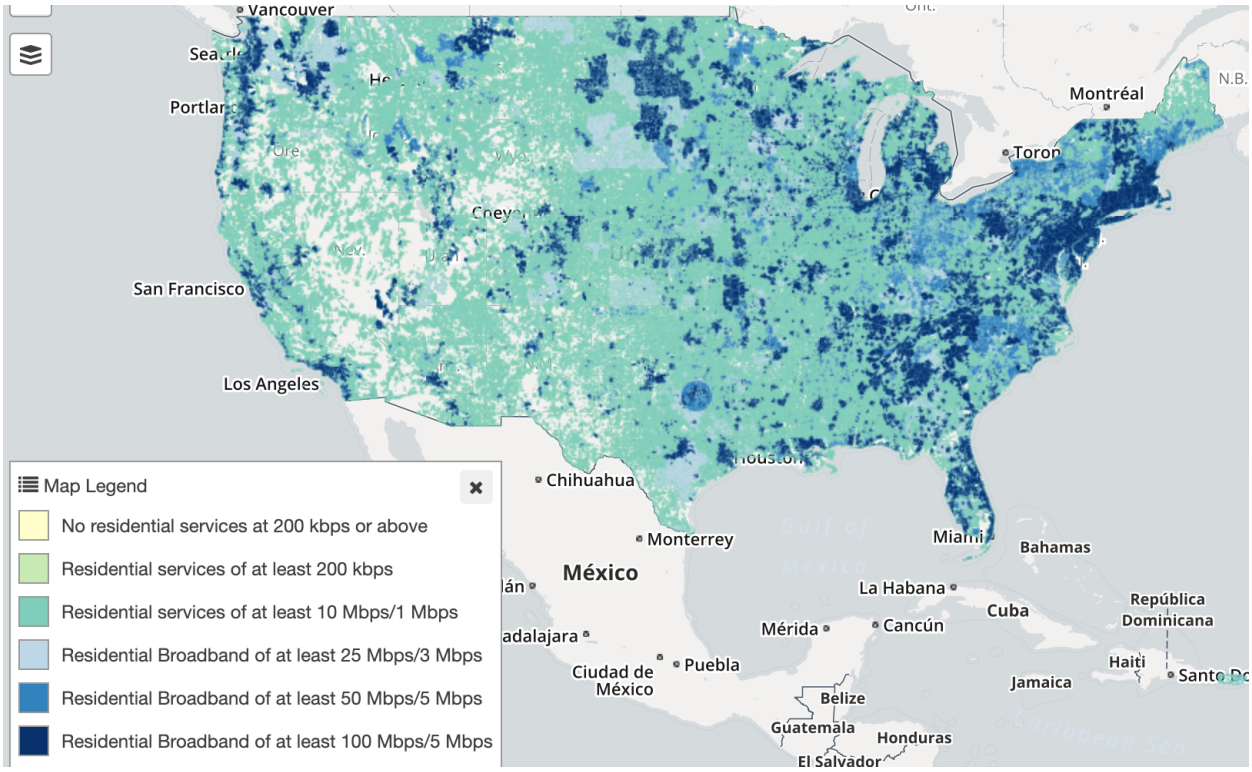


Figure 4.3: Maximum available connection speed across the US

commands included in mode 6 makes the servers vulnerable to being exploited for use in amplified distributed denial of service (DDoS) attacks [37]. Thus, despite performing frequent scans for mode 6 servers, the ShadowServer Foundation does not publish a list of such servers, since doing so would pose a security risk. We found no other source of potential servers and proceeded to attempt our own scan of potential IP address ranges. While we found some potential servers, many lacked any peers and the search was time intensive. Future work may include this method, provided a list of potential servers is available or more time can be dedicated to scanning for them.

4.2.3 ISP Mapping

The FCC maintains a data set of broadband Internet deployment across the US [7]. The data set is composed of a list of census blocks, and contains all of the providers that are within the census block, as well as the maximum, minimum, and advertised speed offered by each provider; this map is shown in Fig. 4.3. Unfortunately, the data set only considers residential broadband connections, which may not be representative of what is actually *used* in the region. Nonetheless, broadband connection availability could provide valuable insight into what connections are available to the average American.

4.2.4 Data Collection via the Postal Service

One data source we considered was to utilize the United States Postal Service (USPS) to send a small cellular connected device such as a phone around the US via ground shipping. As it traveled it would continuously run traceroutes to many other IP addresses that are located across the US. This would give us data along the route there and back. Unfortunately, it is not possible to send a package somewhere and then get it back easily without someone to receive it on the other end. However, assuming this problem could be solved it would be a viable method.

4.3 Statistical Methods

This chapter covers statistical methods used in all methods for processing data. Additionally, since a list of states as ranked by Internet connectivity may be of interest for a quick overview of our results, we present some analysis techniques used for generating them. Unfortunately complications in the data (such as the nature of aggregation by arbitrary political boundaries) make this process not as simple as conducting a sort.

4.3.1 Z-Score Filtering

To filter out statistical outliers from our data sets we choose use a technique known as z-score filtering. Z-score filtering works by calculating the standard deviation of a data set and then removing values that are greater then a set number of standard deviations away from the mean. The z-score is number of standard deviations away as data point needs to be for it to be considered an outlier. For this report, we choose to use a z-score value of 2, and therefor 95% of the data will be preserved.

4.3.2 Kruskal-Wallis test

Regardless of the method of data collection used, if aggregating by state the data will inevitably become a list of data points for each possible state. However, states are *massive* regions with varying populations, infrastructure, etc. and there was not an established relationship between states and Internet connectivity (which would normally make the data much cleaner and easier to analyze) prior to this project. Within any state there may be a large amount of variation in the data and a potentially complex distribution. For this reason a proper statistical test is needed.

The requirements are simple: a non-parametric (i.e. does not assume a normal distribution) test that determines if a ranking of two or more categorical variables is impossible, on variables with unequal sample sizes. The chosen test that meets these requirements is the Kruskal-Wallis *H* test, also known as a one-way analysis of variance (ANOVA) on ranks. When used on a data set, the result is a *H* value that, assuming a chi-squared distribution, can be used to calculate a p value [17]. In this case p should be interpreted as the probability that all the given samples come from the same distribution, i.e. that we cannot tell a difference between them. For a more concrete example, if we run the Kruskal-Wallis test on samples for the states of California and Tennessee and obtain $p = 0.75$, there's a

75% probability that the two states' values come from the same distribution. This report uses the $p < 0.05$ level for its analyses, so in this case the two states would be deemed indistinguishable.

4.3.3 Distinguishability Graphs

The Kruskal-Wallis test cannot tell us if a ranking of 3+ variables is possible, only that one sample dominates the others [17]. So, in a sample of 51 categorical variables[2] the p value may be low, but not all states can be directly compared. An important property of an ordered list is that any two values should be comparable to all those before and after it in a meaningful way. A traditional sorting algorithm running on means or medians cannot be applied, as there is no way to control whether it will try to compare states that the Kruskal-Wallis test says cannot be distinguished at the $p < 0.05$ level.

To visualize this we developed the concept of a distinguishability graph. Briefly, states can be interpreted as vertices on a graph, and pairwise comparisons that are valid or invalid based on the Kruskal-Wallis test can be visualized as edges. With some processing from the SciPy toolkit, Pandas, and visualization + arrangement done by NetworkX or D3 [2, 15, 20, 39], we can generate graphs showing these relationships and relevant attributes.

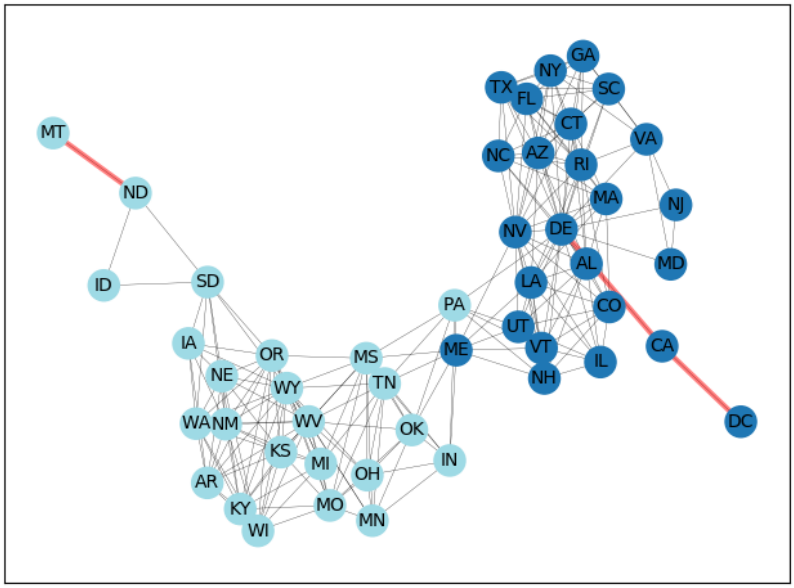


Figure 5.15: Indistinguishability graph (at $p \geq 0.05$) for pairwise state comparisons (repeated from page 40)

Figure 5.15 shows an example of such a graph, for pairwise state comparisons. Each edge between states represents a comparison that *cannot* be made (making this an **in**distinguishability graph), red-highlighted edges are bridges, and node colors correspond to the community that node belongs to. On this particular graph there are no disjoint subgraphs, so

[2] 50 states + the District of Columbia.

ranking by clusters of states is not possible. In graphs where their are disjoint subgraphs, however, ranking by clustered states would be possible.

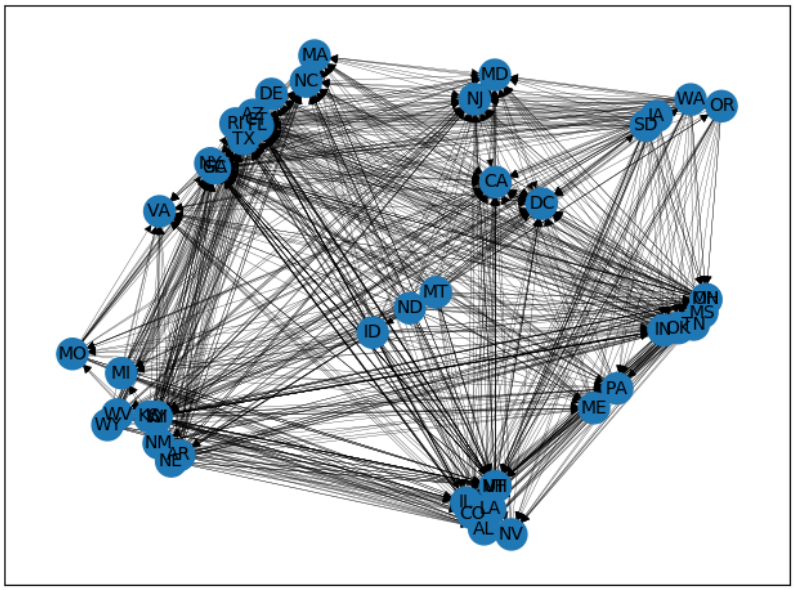


Figure 5.16: Distinguishability graph (at $p < 0.05$) for pairwise state comparisons (repeated from page 40)

Figure 5.16 shows an example of a distinguishability graph at the $p < 0.05$ level; since edges here represent comparisons between states that *are* distinguishable, this graph is drawn as a directed graph. The direction of the edge follows the order of which state has better connectivity (the ancestor of a node has better connectivity). Topological sorting thus opens up new possibilities for the ranking of states *without* relying on traditional sorting algorithms.

4.3.4 Topologically-sorted state rankings

A graph of states that are distinguishable is most naturally represented as a directed graph. When conducting comparisons it's possible to calculate a ratio between the states based on the fraction of connectivity quality that the worse state has to the second (a value ranging from 0-1). These ratios can be used as weights along edges between states, allowing a topological sort of the graph to be conducted. Edges with weights that are higher are explored first (since they correspond to states that are closer to being equal). The result may not be entirely intuitive at first and undoubtedly has some oddities from the unusual sorting method, but may be useful in addition to a simple mean/median-based sort.

One drawback of the topological sort method is that it implicitly compares states that according to the Kruskal-Wallis test cannot actually be compared. For instance, if you have $CA \rightarrow MA$ and $CA \rightarrow TX$, but the comparison between MA and TX is not supported by the data, the topological sort method must choose between them somehow. That choosing process is an implicit comparison, making the topological sort method a rough guess at state rankings at best.

4.4 Summary

In summary, we covered the basic architecture of the Internet and how it can be studied through traceroutes or speed test, some common technologies that users will encounter like caching and the domain system, and some techniques relevant to our project like IP address geolocation and reverse geocoding. As part of our investigations we also covered a pair of notable prior works, namely a past WPI MQP and a project that mapped fiber-optic networks in the US.

All of this information was essential for the completion of our project and will be of use in understanding later chapters.

5

RTT to Everywhere: Traceroute Analysis

As discussed in Section 3.1, one measure of Internet connectivity is an everything to everything approach that collects the RTT between many devices in a region. Collection of such data requires either moving one device to many places, or a distributed network of devices. In either case the device(s) would ping as many different networked devices as they can find.

5.1 Design

Fortunately, such projects exist. Two organizations, CAIDA and RIPE, maintain projects that do almost exactly that. CAIDA and RIPE (the latter through its "Atlas" project) have networks of thousands of small devices, typically Raspberry Pis or similar, that scan vast swathes of the Internet, constantly running traceroutes (see Section 2.2). For example, CAIDA's project involves a technique they call "prefix probing" where their network tries to run a traceroute to at least one device in every /24 prefix. Together these networks have generated terabytes of data over many years, all of which is publicly available.[1]

5.1.1 Direct Ping Calculation

Since a traceroute is really just a series of pings, and a traceroute output reports the RTTs for all of them, CAIDA and RIPE Atlas traceroute data can be used for the everything-to-everything RTT approach. The technique is simple: for every hop in each traceroute, record the source, the destination, and the RTT. IP address geolocation can be used to determine source and destination coordinates, and the haversine formula can be used to find a distance between them (formula 5.1). We refer to this technique as *direct ping calculation*.

$$d = 2r \arcsin \sqrt{\sin^2\left(\frac{\rho_2 - \rho_1}{2}\right) + \cos\rho_1 \cos\rho_2 \sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)} \tag{5.1}$$

Formula 5.1: Haversine formula for distance; ρ_1, ρ_2 and λ_1, λ_2 are latitude/longitude respectively for the two points in radians, and r is the radius of the Earth at 6,371 km.

Figure 5.1: Diagram of indirect traceroute ping calculation

5.1.2 Indirect Ping Calculation

A crude calculation of the RTT between each individual server in a traceroute can also be performed without directly sending pings between them. Figure 5.1 shows roughly how this process works. Red lines indicate the time between servers that we *want* to measure, while black lines indicate data that the server running the traceroute can actually give us. By subtracting a server's RTT from the RTT of the server just behind it, we can estimate the RTT directly between the two. The same technique applies to any two arbitrary pairs of hops in the traceroute log, although a sanity check to guard against negative RTTs (caused by jitter along connections compounded by the subtraction operation) is needed.

This method results in almost double the amount of ping data per traceroute, since if you have a traceroute $A \to B \to C \to D$, you have data for not only $A \to B, A \to C$, and $A \to D$, but you can also calculate $B \to C, B \to D$, and $C \to D$. More formally, for a traceroute of n hops, you can extract $2n - 2$ RTTs. We refer to this technique as *indirect ping calculation*, and can similarly involve distance calculations provided by formula 5.1.

5.2 Implementation

We collected as much CAIDA and RIPE Atlas data (from 2018-2019) as could store using the common tool `wget` (and later an http-accelerator program `axel`). RIPE Atlas data is stored in Javascript Object Notation (JSON) format while CAIDA developed its own binary format "WARTS" for binary storage. While the code for reading it is open source, a toolkit including a warts-to-JSON converter is available for download[2] and once converted the

[1]CAIDA's prefix probing data can be found at https://www.caida.org/data/active/ipv4_prefix_probing_dataset.xml. The RIPE Atlas data set may be found at https://data-store.ripe.net/datasets/atlas-daily-dumps

[2]http://www.caida.org/tools/measurement/scamper

resulting JSON file is similar to the RIPE Atlas format. The total volume of data processed is estimated at 5-10 terabytes.

5.2.1 Data ETL Pipeline

The pipeline was implemented as a bash script that operated in three stages. This process is highly parallelizable and so the aptly-named `parallel` tool [33] was used to process 8-16 files at a time. A command-line interface (CLI) program was written in C++, called the "traceroute hopper" while in development, capable of parsing JSON files and performing ping calculation for the extract-transform-load (ETL) process listed below. C++ was chosen for its performance[3] and availability of performance JSON parsing libraries [34].

Extraction

RIPE Atlas distributes data in compressed (gzip) format, each file of which contains a single 10 GB JSON file. CAIDA distributes files in compressed WARTS format, which each expand to 3 GB JSON files when extracted and converted. Since 10 GB files are unwieldy and the file format permitted it, RIPE Atlas files were split into chunks of 100,000 traceroutes each – about 3,000 files total. Each file was fed to a bash script that encompassed the entire extraction and conversion process, in addition to running the traceroute hopper.

Internally, the traceroute hopper maps a traceroute file into memory with a read-ahead flag set. This forces the operating system into loading the entire file into memory at once so future reads to the file never hit the disk, freeing up disk usage for other processes (such as the database). After loading, the program reads through the file line-by-line, as both RIPE Atlas and CAIDA JSON files are composed of thousands of JSON objects per file, one per line. This method is shown in snippet 5.1.

```
1  // Map file into memory for faster processing
2  struct stat st{};
3  stat(args.inputs[fileNum], &st);
4  int fd = open(args.inputs[fileNum], O_RDONLY, 0);
5  void* data = mmap(0, st.st_size, PROT_READ, MAP_PRIVATE | MAP_POPULATE, fd, 0);
6  std::stringstream file;
7  file.rdbuf()->pubsetbuf(static_cast<char *>(data), st.st_size);
8
9  // Loop over each individual traceroute and process it
10 vector<tuple<string, float>> rawHops(50);
11 while(getline(file, line)) {
12   Document traceroute;
13   traceroute.Parse(line.c_str())
14   // ...
15 }
```

Snippet 5.1: Traceroute hopper JSON loading

[3] Python was originally used but known to be slow; switching to C++ yielded a 100% performance boost, saving several days of processing time at a cost of a few hours of development.

Transformation

The next step in the process is to perform specific processing on CAIDA and RIPE Atlas JSON formats. The principle layouts of both are nearly identical, but the structure is different and CAIDA files require DNS lookups on sources.[4] Since it would be wasteful to construct two entirely separate parsers, the traceroutes are converted to an intermediate format first. The CLI requires a flag for whether the file is of CAIDA origin or RIPE atlas origin to distinguish between the two. The entire sequence for converting to an intermediate format is shown in snippet 5.2.

```
1  vector<tuple<string, float>> convertAtlas(const Document& traceroute) {
2    vector<tuple<string, float>> hops;
3    hops.reserve(traceroute["result"].Size());
4
5    const auto& hopsArray = traceroute["result"].GetArray();
6    for (const auto& hop : hopsArray) {
7      // Verification - not all hops have results
8      if (!hop.HasMember("result"))
9        continue;
10
11     string src = getHopSource(hop["result"].GetArray());
12     if (src.empty())
13       hops.emplace_back("", -1); // Error condition for this hop
14
15     hops.emplace_back(src, rttAverage(hop["result"].GetArray()));
16   }
17
18   return hops;
19 }
20
21 vector<tuple<string, float>> convertCaida(const Document& traceroute) {
22   vector<tuple<string, float>> hops;
23   hops.reserve(traceroute["hops"].GetArray().Size());
24
25   for (const auto& hop : traceroute["hops"].GetArray())
26     hops.emplace_back(hop["addr"].GetString(), hop["rtt"].GetFloat());
27
28   return hops;
29 }
```

Snippet 5.2: CAIDA and RIPE Atlas JSON pre-processing to intermediate format

The next stage involves actual ping calculation on traceroutes. The CLI accepts a flag for this too, allowing the user to enable indirect calculation. Alternatively the user can switch to “ping” mode, where only the RTT for the absolute endpoints of the traceroute are calculated. The relevant section of calculation code is shown in snippet 5.3.

[4] CAIDA files are stateful; most lines only contain a JSON object describing a traceroute, but those leave out the source IP address. At the top of each file (or sometimes, multiple times across each file) is a separate JSON object that contains information about the source server, and all traceroute entries that follow are sent from that source. The server is given as a hostname, so it needs DNS resolution to obtain an IP address.

```
if (args.ping_given) {
  // Ping mode - just do one entry, source to final destination
  float rtt = get<1>(rawHops[rawHops.size() - 1]);
  if (rtt < 0)
    continue; // Can't parse this traceroute
  hops.emplace_back(baseSrc, get<0>(rawHops[rawHops.size() - 1]), rtt, time, false);
} else {
  // Either direct or calculated mode
  int j = -1;
  for (auto& hop : rawHops) {
    j++;

    // Add a hop for the direct source-> hop entry
    if (get<1>(hop) < 0 || get<0>(hop).empty())
      continue; // Bad hop
    hops.emplace_back(baseSrc, get<0>(hop), get<1>(hop), time, false);

    // If we're not on calculate mode OR we're at the first hop, skip. Processing the
    ↪  first hop would be
    // redundant since direct mode already picks it up.
    if (!args.calculate_given || j == 0)
      continue;

    // Make sure the last hop is valid, otherwise we can't calculate an rtt
    tuple<string, float> lastHop = rawHops[j - 1];
    if (get<0>(lastHop).empty() || get<1>(lastHop) < 0)
      continue;

    hops.emplace_back(get<0>(lastHop), get<0>(hop), get<1>(hop) - get<1>(lastHop),
    ↪  time, true);
  }
}
```

Snippet 5.3: Direct, indirect, and ping mode calculation

Load

Once the traceroute hopper reaches its buffer capacity, it dumps the contents out to a PostgreSQL database. This is performed as a streamed operation through *libpqxx*, the official C++ client library for PostgreSQL [38].

PostgreSQL was chosen for its performance, advanced features, and in particular its ability to generate spatial indices. Performing basic statistical analyses and fast joins was also a sought-after feature.

Post-processing: Geolocation

Once all data was collected in the database it was possible to sort out unique IP addresses, each of which was fed into a geolocation library (see Section 2.7) provided by MaxMind to estimate the location of the machine the IP address belongs to. This step was delayed

until after the ETL process for performance reasons. After processing was completed, the database contained ~71 billion individual RTT measurements.

5.2.2 Data Cleaning & Filtering

The data required some cleaning before it could be considered viable for serious analysis, since not all measurements or servers returned results consistent with nearby neighbors, and RTT-based data is vulnerable to influence from outliers. Since threshold filters (ex. removal of all points above x ms) risk biasing data, simple z-score filtering was selected as the main filtering method. Filtering was performed at two levels: during aggregation, and after aggregation by IP address pair.

Aggregating and filtering per IP address pair

For each IP address pair there are potentially many measurements. A pair may have 50 perfectly good measurements, for instance, but one measurement in the tens of thousands of milliseconds that should be discarded. For each IP address pair the standard deviation was calculated and each measurement for that pair was z-scored; points that exceeded 2.0 (absolute value) were discarded. Since this was performed at the raw data level (operating on ~71 billion rows) it was integrated as part of the IP-pair aggregation query. This query may be found in snippet 5.4. This process also involved assigning locations to the endpoints of each of the IP address pairs using a prior-calculated table.

Filtering IP pair outliers

Some IP address pairs consistently performed poorly no matter the filtering at the individual measurement level, with RTT values that far exceed the mean for the entire pool of address pairs. Since these values are also likely to influence results in undesirable ways, they were filtered out using the same z-score method. At this point the data was on the order of a few hundred million rows and was thus suitable for export to more traditional data processing tools, so from this point on filtering was conducted with the Python *pandas* library [20]. After all filtering and aggregating was complete, there were ~230 million data points.

Snippet 5.5 shows some of the code responsible for filtering out bad IP address pairs. The `df = [expression]` format is repetitive but intended for improved readability; there are better ways of organizing *pandas* code. The code accomplishes several things at once. Line-by-line:

1. Filter the data by direct or indirect ping calculation. `args.indirect` parameratizes this for CLI use. At the same time, filter to IP address pairs where the distance between endpoints is greater than 0 (indicates geolocation imprecision) and the RTT is greater than 0 (indicates timing error).
2. Filter IP address pairs by the z-score of their `rtt_avg` field, or the mean of the RTT for that address pair.
3. Calculate a primitive "connectivity" value as milliseconds per kilometer, for all IP pairs.

```sql
CREATE MATERIALIZED VIEW hops_aggregate_view AS (
  SELECT
    agg.src,
    agg.dst,
    agg.indirect,
    src_loc.coord as src_loc,
    dst_loc.coord as dst_loc,
    haversine_distance(src_loc.coord[0], src_loc.coord[1], dst_loc.coord[0],
    ↪  dst_loc.coord[1]) AS distance,
    agg.rtt_avg,
    agg.rtt_stdev,
    agg.rtt_range,
    agg.measurements
  FROM (
      SELECT hops.src,
         hops.dst,
         hops.indirect,
         AVG(RTT)        AS rtt_avg,
         STDDEV_SAMP(RTT)    AS rtt_stdev,
         MAX(RTT) - MIN(RTT)   AS rtt_range,
         COUNT(*)        AS measurements
      FROM hops
           INNER JOIN hops_stats hs
                ON hops.src = hs.src
                 AND hops.dst = hs.dst
                 AND hs.stddev_samp != 0
                 AND hops.indirect = hs.indirect
                 AND ABS((hops.rtt - hs.avg) / hs.stddev_samp) <= 2
      GROUP BY (hops.dst, hops.src, hops.indirect)
    ) agg
      INNER JOIN locations src_loc ON agg.src = src_loc.ip
      INNER JOIN locations dst_loc ON agg.dst = dst_loc.ip
)
```

Snippet 5.4: Aggregation and base filtering SQL query

4. Z-score filtering for IP pairs based on connectivity, i.e. throw out all pairs whose "connectivity" is too far one way or the other.

```python
df = df[(df["indirect"] == (1 if args.indirect else 0)) & (df["distance"] >
↪  0) & (df["rtt_avg"] > 0)]
df = df[np.abs(stats.zscore(df["rtt_avg"])) <= 2.0]
df["connectivity"] = df["rtt_avg"] / df["distance"]
df = df[np.abs(stats.zscore(df["connectivity"])) <= 2.0]
```

Snippet 5.5: Pandas filtering of IP address pairs

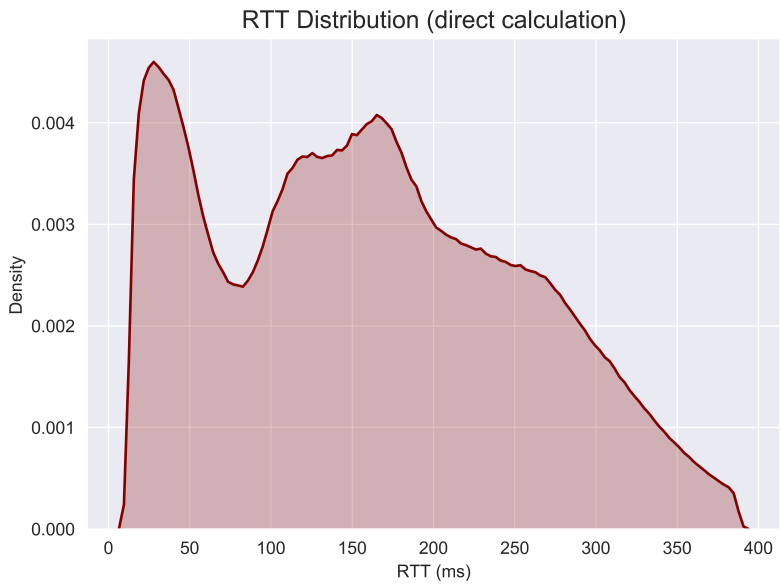


Figure 5.2: RTT distribution, direct ping calculation

5.3 Analysis

The following sections describe analysis of the CAIDA and RIPE Atlas data in a natural progression from analysis of data quality, to basic connectivity analysis, and finally to geographic plotting and geographic information system (GIS) tooling.

5.3.1 Data Quality

To determine the quality of the data, we made a series of kernel density estimation (KDE) charts with the Python library *seaborn* [41]. Since histograms are vulnerable to binning effects and cumulative distribution charts tend to be less intuitive to read, distributions in this report are presented as () charts. Briefly, these work by drawing a Gaussian distribution around each point of data, summing all distributions together, and normalizing so the area under the curve is equal to 1. The y axis, then, does not represent a real value, instead only a probability density. KDE charts contrast cumulative distribution function (CDF) charts which can be used to more easily extract median, percentiles, etc.; however, the point of the charts here is more to show clustering than anything, which KDE charts excel at intuitively presenting.

The most immediately useful distribution is that of the RTT between IP pairs, shown for direct-calculated RTTs in Fig. 5.2. The distribution appears weakly bimodal, which we hypothesize is due to the global nature of RIPE Atlas and CAIDA's individual measurement networks. The leftmost peak corresponds to measurements to a device that shares a land mass with the device performing the traceroute, while the rightmost peak corresponds to a combination of devices on a different land mass and devices with lower-performing connections. Figure 5.3 appears to confirm this hypothesis, since it too is similarly bimodal.

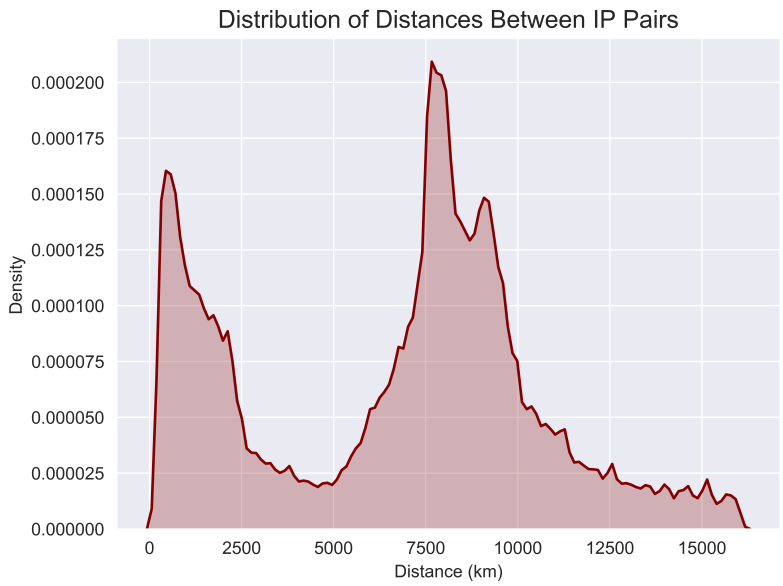


Figure 5.3: Address pair distance distribution

Figure 5.4 shows the distribution of RTTs calculated using the indirect ping calculation method, of which a calculated ~29% are below zero – an impossible value. Since a significant fraction of the data points are completely impossible it was decided that this data was too unreliable for further analysis. The remainder of the data analyses in this section are based on the direct ping calculation method only.

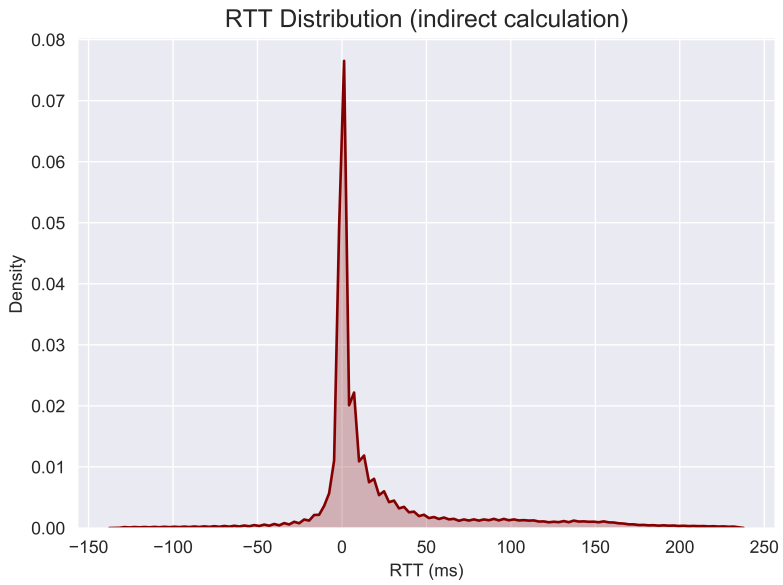


Figure 5.4: Distribution of RTT between IP pairs, indirect ping calculation

To further assess data quality we turned to measures of the data spread for each data point. Figure 5.5 shows the distribution of measurement counts between each IP address

pair, showing that although most pairs had on the order of 1-20 measurements, a sizeable fraction had more than that, and there were even some in the 500+ measurements range. This effect is likely a result of the way RIPE Atlas and CAIDA nodes are networked. A node's local gateway would always show up on a traceroute (unless configured to not respond to pings), as would common paths through a node's ISP, so these IP addresses are measured extremely frequently.

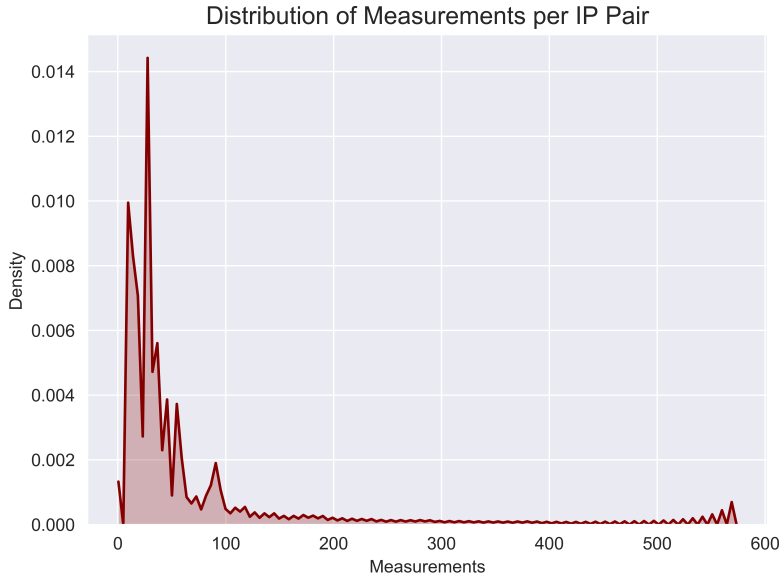


Figure 5.5: Distribution of measurements count for each address pair

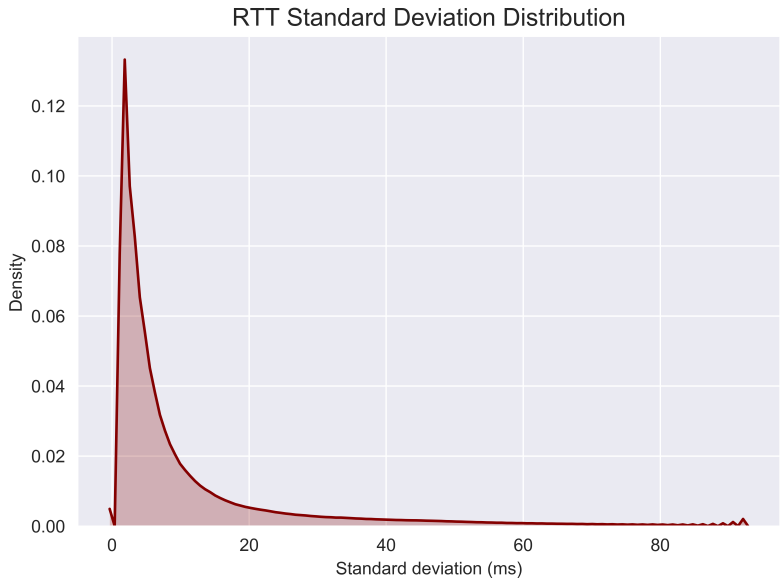


Figure 5.6: Distribution of address pair standard deviations

The first measure of data quality used is standard deviation. Figure 5.6 shows the distribution of standard deviations across all measured IP address pairs (for charting purposes, pairs with only one measurement were interpreted as 0 standard deviation). The chart shows an incredibly smooth curve where the overwhelming majority of pairs have standard deviations well below 20 ms.

Since standard deviations are all relative, we next calculated a coefficient of variation (CV) for each IP address pair as a measurement of data quality. Figure 5.7 shows the distribution of CVs for all IP address pairs, with the majority of CVs below 0.1 – in other words, excellent-quality data with low spread for each pair.

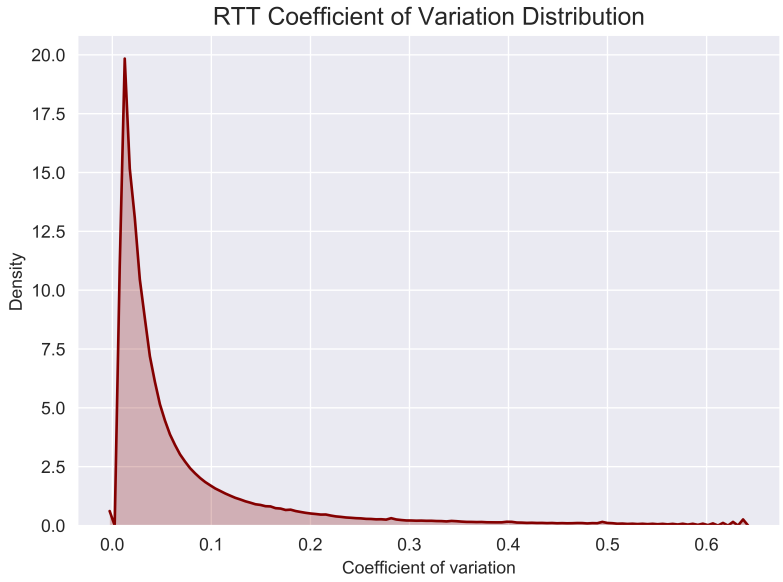


Figure 5.7: Distribution of address pair coefficients of variation

5.3.2 Primitive Connectivity Analyses

Each data point comprises an IP address pair, but each destination IP address – that is, each IP address that a CAIDA or RIPE Atlas node ran a measurement against – appears many times, at least once for every node that ran a measurement against it. Averaging these will not work (it makes little sense to average together measurements from a server in Boston with a server in Moscow against a server in New York, since the Moscow measurement node will naturally report a much higher RTT), so normalization is needed. The first method used was simple normalization by distance, which returns values in ms/km. This may be unconventional, but milliseconds and kilometers are natural units for RTTs and distance, respectively.

The ms/km distribution shown in Fig. 5.8 is uninformative on its own, but it does demonstrate an important feature. Normalization succeeds in removing the bimodality of the RTT distribution shown in Fig. 5.2 without removing *all* the spread of the data. This both further affirms the earlier hypothesis about the cause of the bimodality, and gives cause to believe that geographic charting may yield interesting results.

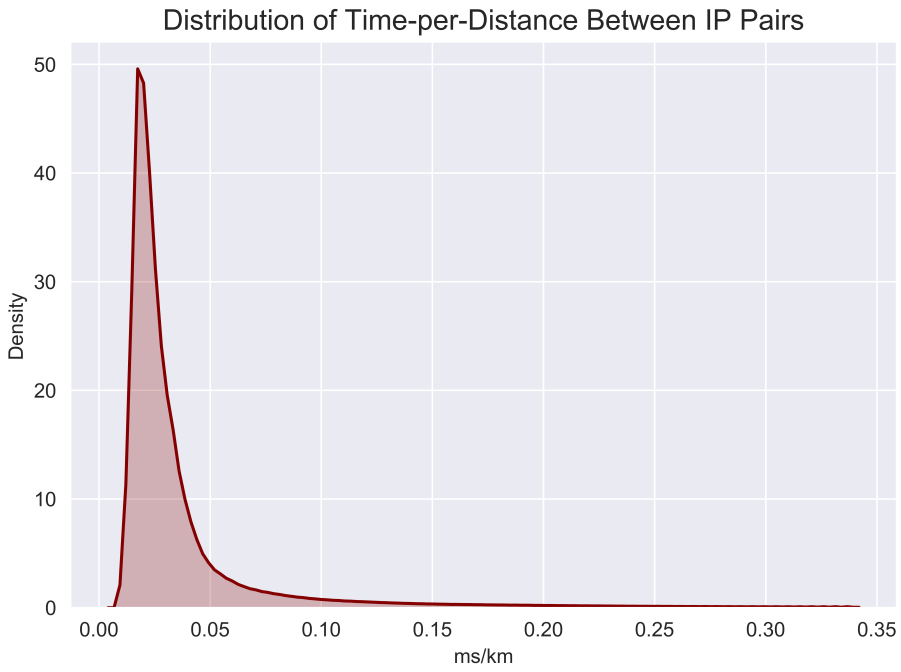


Figure 5.8: ms/km connectivities distribution

Unfortunately this metric is challenging to chart with any color scale. The majority of values are between 0.0 and 0.05 ms/km but values an order of magnitude higher must also be charted, and a log scale fails to capture important but relatively small variances between areas. To solve this, we devised a new metric based on efficiency relative to the speed of light.

The speed of light is 299.79246 km/ms, so the theoretical minimum RTT between two points ~300 km apart is ~2 ms – one ms one way, and another on the return trip. All telecommunications happen over electromagnetic mediums,[5] be it fibre optic cables or copper wires, so communications always have a transmission delay proportional to the speed of light. If the RTT was higher than two milliseconds, there must logically be some loss in speed somewhere in the network, whether that means poor infrastructure or wiring that does not follow a straight line to its target – either way, an inefficiency. The smaller the RTT, the higher the efficiency, and vice versa. This has the desirable quality that extreme outliers are always between zero and one regardless of how high the RTT is. The formula for speed-of-light-efficiency based on RTTs is shown in formula 5.2.

$$E = \frac{2d}{t \times 299.79246} \tag{5.2}$$

Formula 5.2: Speed-of-light efficiency; E is efficiency as a scalar from 0-1, d is distance in kilometers, and t is the RTT in milliseconds.

[5]With the possible exception of internet protocol over avian carrier (IPoAC), which has been successfully demonstrated [1, 40]

Speed-of-light efficiency can also be thought of as a scalar multiplied by c, the speed of light – it's the equivalent "speed" of a ping. With this improved normalization scheme to work with, the subtle differences and patterns like those in Fig. 5.9 can be seen in the data when mapped. Also important is that this normalization method provides another means of filtering data – anything above 1.0 efficiency can be removed since it violates the laws of physics by exceeding the speed of light.

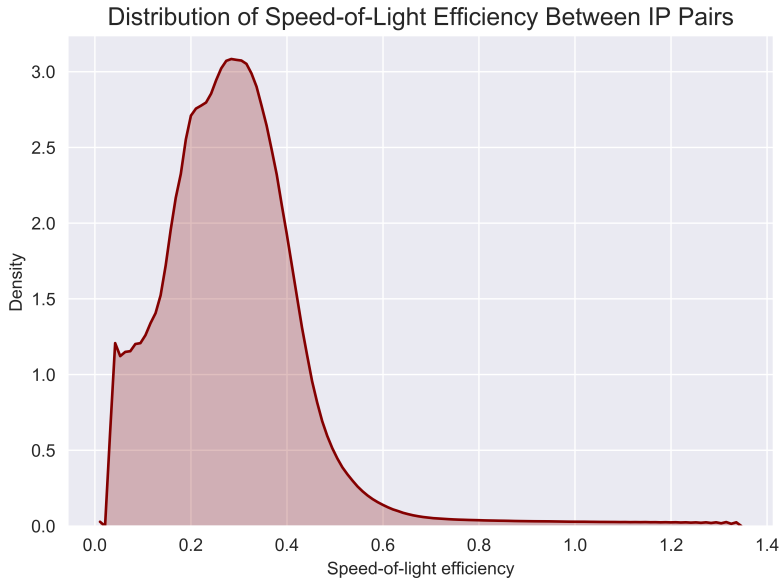


Figure 5.9: Speed-of-light efficiencies distribution

5.3.3 Mapping

The simplest way of mapping a set of points on a coordinate plane with values attached to each of them is a simple scatterplot, but with massive amounts of unevenly distributed data it becomes tough to visualize and draw conclusions. For example, in Fig. 5.10 we see a point for *every single pinged device* in the CAIDA and RIPE Atlas data set that was collected and analyzed – at least, those in the US.

Some simple relationships can be inferred with moderate difficulty, like better Internet connections near the coast or possibly major cities, but otherwise this map is only good at confirming that geolocation of IP addresses works. Many areas simply do not have measurements either, and those that appear covered look that way because the dots for each measurement were inflated for visual effect. If they were more accurately represented to-scale as single pixels, the map would be sparse.

To solve this problem the map needs some interpolation to fill in the gaps and make the data easier to understand. Ideally someone looking at the map should be able to point to a spot on the map and get an estimate for connectivity at that location, even if there was not a measurement at precisely that location. However, even after z-score filtering, aggregation by source-destination pairs, and more filtering, there were still millions of data points within the US alone. To that end we tried some unusual techniques for further

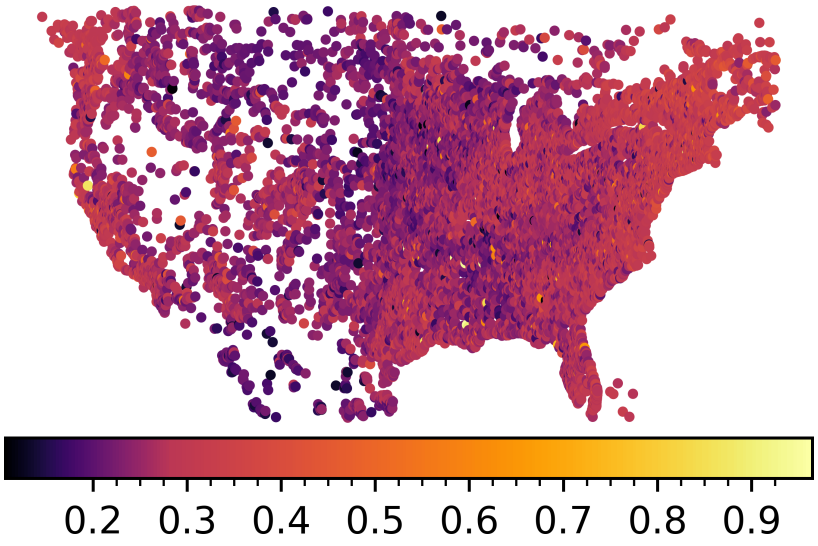


Figure 5.10: Traceroute scatterplot, colored by speed-of light-efficiency

aggregating the data together, in ways that were more manageable for visualization tools. It was also decided that we should not lose resolution as the density of points increased. For example, drawing a grid of static boxes on top of the map and grouping using those would not do because there may be a city with thousands of points in one, and a rural area with only a few points in another, but they'd both take up the same area on the map.

Quadtree grouping

One of the first techniques tried for grouping data together was a technique we dubbed "quadtree grouping". The technique is adapted from methods used to optimize collision detection in 2D games where the screen is cut into four boxes based on the number of entities in each quadrant, then each box is cut into four more with the same metric, and so on until some threshold with an optimal number of entities per box is reached. This process was performed on the data set here and adjusted for different parameters like maximum tree depth, maximum nodes per box, etc.

Figure 5.11 shows a plot generated using this technique, with brighter areas corresponding to the higher RTT-per-km metric, indicating worse connectivity. Smaller boxes tend to denote areas of higher population density, as they were areas the algorithm needed to subdivide the most. This technique appears to show a pattern in connectivity, with the East coast having overall better connectivity, but it's also visually difficult to read and difficult to interpret the data from – it's displeasing to the eye.. The only way of assigning a single point to each quad is based on its center,[6] but this results in points at odd positions, even out in the ocean.

[6]Technically it's possible to use some form of clustering within each quad to find an area with the highest density to pin the point on, but at that point you may as well just use clustering on the whole map.

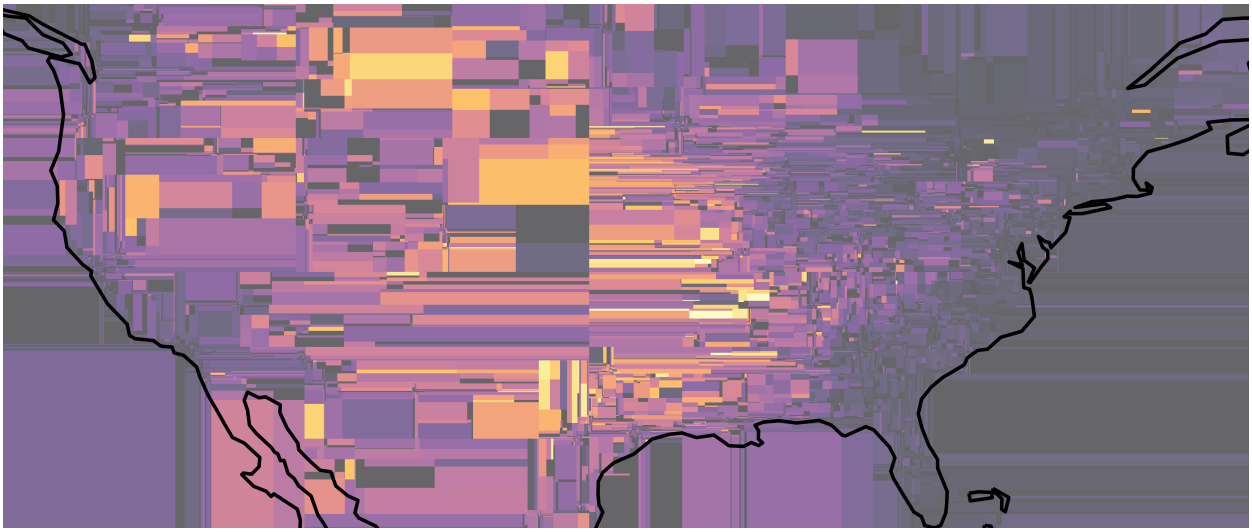


Figure 5.11: Traceroute ms/km quadplot, generated by quadtree grouping

Nearest-neighbor interpolation

At this point it was discovered that, due to the fairly low resolution of IP address geolocation, many measurements from different IP address pairs share the same coordinates. This made even further grouping by location possible, down to only ~55,000 data points. This is much more manageable to visualize, so the next interpolation method tried was simple nearest-neighbor interpolation (which prior to grouping would have been infeasible).

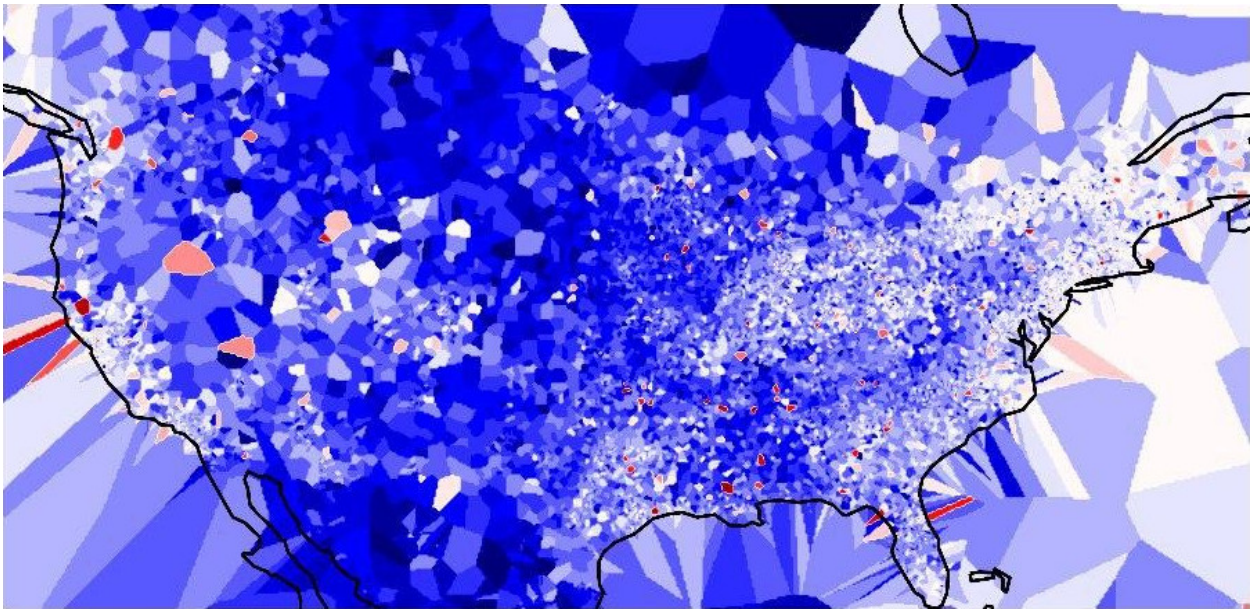


Figure 5.12: Traceroute speed-of-light efficiency nearest-neighbor diagram

Figure 5.12 shows a nearest-neighbor plot, otherwise known as a Voronoi diagram [18], of the CAIDA and RIPE Atlas data combined, generated with Python *matplotlib* and

scikit-learn [16, 39]. The colormap is a divergent linear colormap, where blue is worse-than-average and red is better-than-average efficiency. This does an arguably better job of presenting the data than quadtree grouping, but it's visually noisy and suffers from expanded influence of points in sparse areas. For instance, consider the large splotch in Utah, which corresponds to areas arounhd Salt Lake City. While it's likely accurate that the city has far better Internet connectivity than its desert surroundings, it is likely inaccurate to say that the areas within the few hundred mile radius shown on the map have the same level of connectivity.

Linear interpolation

Since nearest-neighbor interpolation proved insufficient, the next method tried was linear interpolation. Instead of assuming that all points nearer to a given data point than any other have the same value as the data point does, linear interpolation assigns values at each point between the different data points according to a simple linear scaling model. This has the effect of making the map look far less discrete.

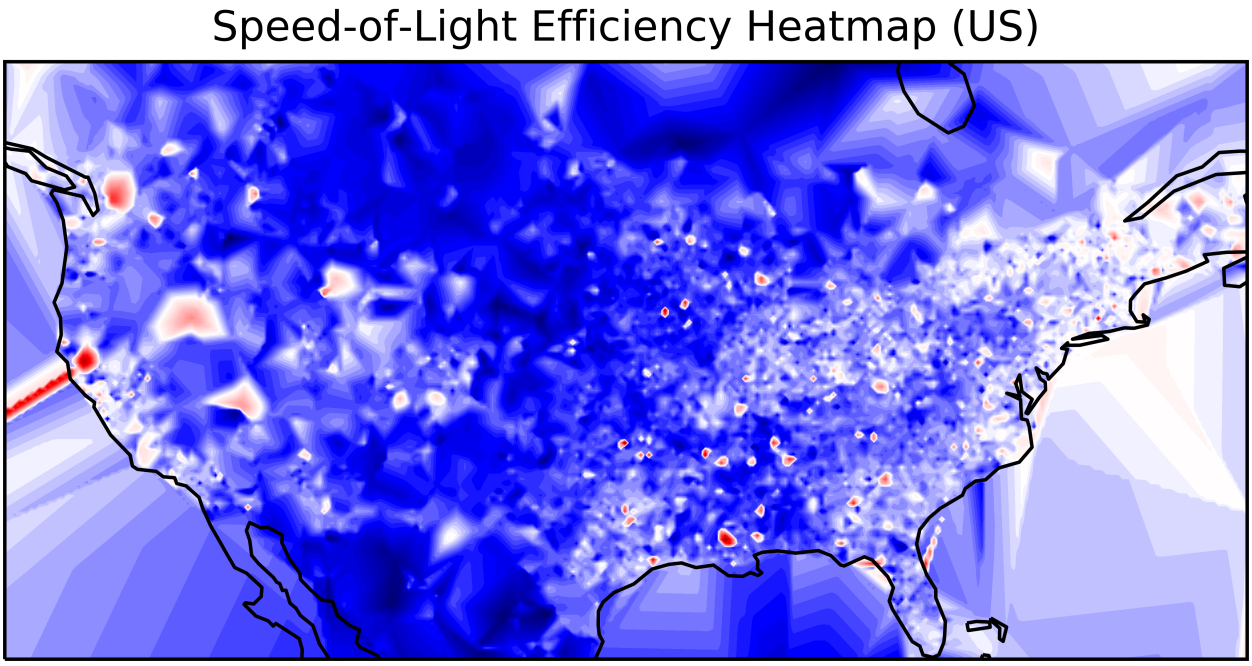


Figure 5.13: Linear-interpolated traceroute speed-of-light efficiency heatmap

Figure 5.13 uses the same divergent colormap as Fig. 5.12 but with linear interpolation instead, showing a far smoother map that preserves the same patterns. It does not look as discrete but it still has an angular affect to it, with sharp angles showing through where Voronoi cell bounds and intersections previously were.

Inverse Distance Weighting

Inverse distance weighting (IDW) is a method for interpolating point data, operating under the assumption that areas closer together are more likely to be similar. The influence of

any given point on the interpolated data falls off with the inverse of the distance to that point, producing much smoother, much more easily understood (and likely more accurate) results.

Figure 5.14 shows the data as interpolated with IDW and touched up to include state lines, major cities, and a key. This is what we consider to be the final heatmap generated from the CAIDA & RIPE Atlas data. It roughly confirms our expectations that the more densely-populated coasts would have have better connectivity on average while adequately highlighting large-scale trends.

Interestingly, no matter how the data is visualized it always has a great deal of variance from one area to another – that is, the map looks "splotchy". Loosely speaking, this is also what we would expect to see, since people tend to report a great deal of subjective difference in Internet quality from one area to another.

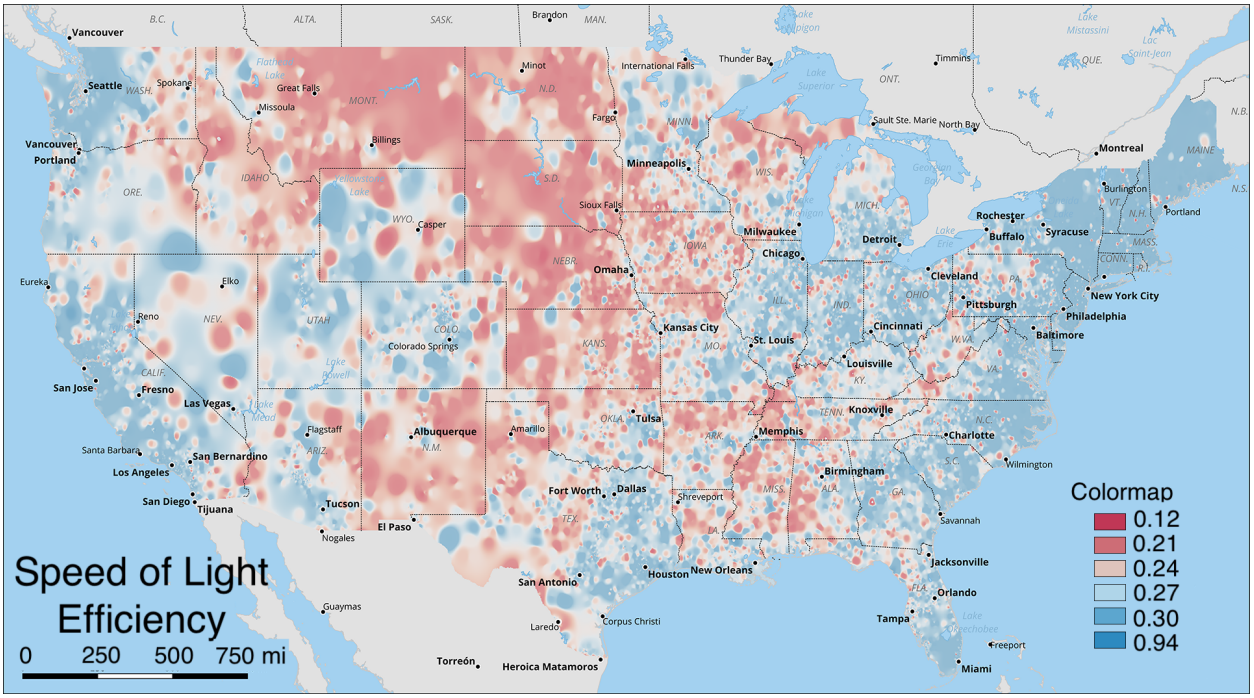


Figure 5.14: Inverse distance weighting traceroute speed-of-light efficiency heatmap

5.4 State Rankings

Using the statistical and (in)distinguishability graph methods described in Section 4.3, we were able to develop both graph visualizations of state distinguishability and a possible ranking of the states.

Figure 5.15 shows an indistinguishability graph of comparisons between the states (previously seen in Section 4.3), where node colors correspond to communities and bold red lines correspond to bridges.[7] As noted earlier, this graph has no disjoint subgraphs, so

[7]Bridges can be thought of as indicators of comparison quality; any state on the end of a bridge can be compared to all but one of the other states.

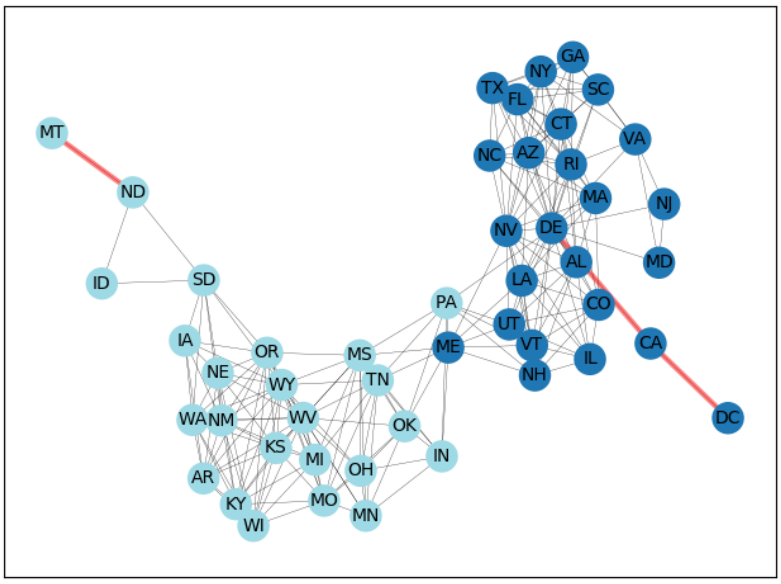


Figure 5.15: Indistinguishability graph of traceroute data as aggregated by state

ranking by clusters is not possible. Figure 5.16 shows the corresponding distinguishability graph, which is much more connected, indicating this data is a good candidate for the topological sort method.

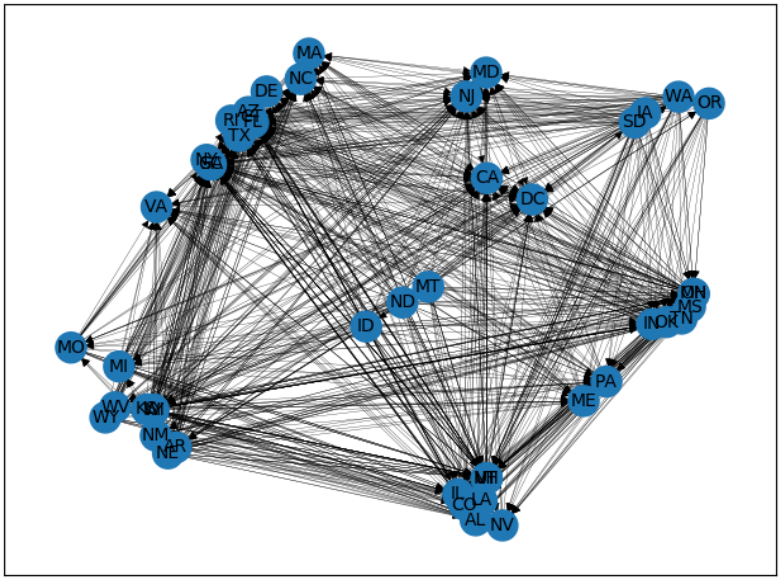


Figure 5.16: Distinguishability graph of traceroute data as aggregated by state

When sorted we obtain a list of 49 states (including the District of Columbia; Hawaii and Alaska were unreachable); two states were not reachable by a maximal topological sort (described in Section 4.3.4). Table 5.1 shows this ranking, which roughly confirms what we would expect based on prior notions and the heatmaps. Since the topological sort

Table 5.1: CAIDA+Atlas topologically sorted state rankings

Rank	State	Rank	State	Rank	State	Rank	State	Rank	State
1	DC	**11**	TX	**21**	LA	**31**	TN	**41**	WY
2	DE	**12**	AL	**22**	NV	**32**	MN	**42**	NE
3	MD	**13**	CT	**23**	PA	**33**	OH	**43**	NM
4	CA	**14**	RI	**24**	VT	**34**	WI	**44**	OR
5	NJ	**15**	NC	**25**	NH	**35**	KY	**45**	IA
6	NY	**16**	MA	**26**	UT	**36**	KS	**46**	WA
7	SC	**17**	AZ	**27**	IN	**37**	MO	**47**	ID
8	GA	**18**	IL	**28**	WV	**38**	MI	**48**	MT
9	VA	**19**	CO	**29**	MS	**39**	AR	**49**	ND
10	FL	**20**	ME	**30**	OK	**40**	SD		

method is based on a graph interpretation and not a traditional sort of means or medians, those values are not shown alongside the states.

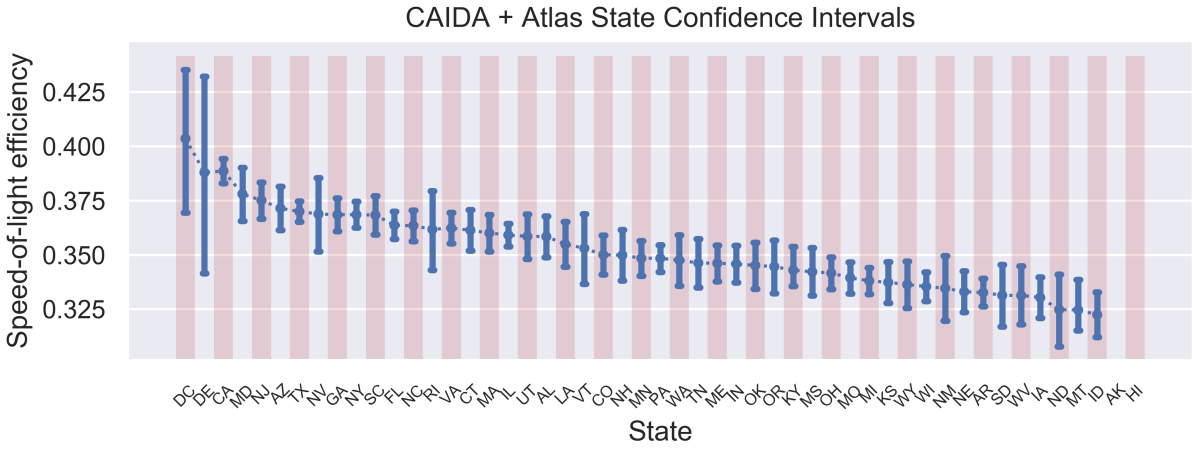


Figure 5.17: Traceroute confidence intervals for rankings; higher is better

Figure 5.17 shows an alternate way of displaying state ranks, using the mean of each state for comparisons. Since data for each state does not follow a normal distribution (see Section 8.1 for a variety of KDEs on the subject) the bootstrapping method was used to calculate confidence intervals. Loosely speaking, for each state the "true" mean could be anywhere between the upper and lower bounds of the intervals. Table 5.2 shows the specific values for each of these points.

The confidence intervals for each state's mean are large, indicating that there are few meaningful comparisons to be made here. The same conclusion can be drawn from the indistinguishability graph in Fig. 5.15; there are no disjoint subgraphs so a ranking by groups of states cannot even be drawn. The topological sorting results shown in Table 5.1 are interesting but they have the fundamental flaw of topological sorts as described in Section 4.3.4: the sorting relies on implicit comparisons that cannot be made according to

Table 5.2: Traceroute state confidence intervals

State	Min	Mean	Max	State	Min	Mean	Max
DC	0.369	0.406	0.435	**WA**	0.336	0.348	0.359
DE	0.342	0.390	0.432	**TN**	0.335	0.347	0.357
CA	0.383	0.389	0.394	**ME**	0.338	0.346	0.354
MD	0.366	0.378	0.390	**IN**	0.337	0.346	0.354
NJ	0.367	0.375	0.383	**OK**	0.334	0.345	0.356
AZ	0.361	0.372	0.381	**OR**	0.332	0.345	0.357
TX	0.365	0.370	0.375	**KY**	0.332	0.343	0.354
NV	0.352	0.369	0.385	**MS**	0.331	0.342	0.353
GA	0.361	0.369	0.376	**OH**	0.334	0.342	0.349
NY	0.363	0.369	0.375	**MO**	0.332	0.340	0.347
SC	0.359	0.369	0.377	**MI**	0.332	0.338	0.344
FL	0.357	0.364	0.370	**KS**	0.328	0.337	0.347
NC	0.356	0.364	0.370	**WY**	0.326	0.337	0.347
RI	0.343	0.363	0.379	**WI**	0.329	0.336	0.342
VA	0.355	0.362	0.369	**NM**	0.320	0.335	0.350
CT	0.352	0.361	0.371	**NE**	0.324	0.333	0.343
MA	0.352	0.360	0.368	**AR**	0.326	0.333	0.339
IL	0.354	0.359	0.364	**SD**	0.317	0.332	0.346
UT	0.348	0.359	0.369	**WV**	0.318	0.331	0.345
AL	0.349	0.359	0.368	**IA**	0.321	0.331	0.340
LA	0.345	0.355	0.365	**ND**	0.308	0.326	0.341
VT	0.337	0.354	0.369	**MT**	0.310	0.325	0.339
CO	0.341	0.350	0.359	**ID**	0.312	0.322	0.333
NH	0.338	0.350	0.362	**AK**			
MN	0.340	0.349	0.356	**HI**			
PA	0.342	0.349	0.355				

this data set.

The underlying problem is not with variance in the data itself (as shown by Fig. 5.7 and Fig. 5.6 the data spread is low, proving quality data), but in the aggregation method. As visualized in Fig. 5.14 there is simply too much variation within a state, prohibiting a statistically-valid ranking of all 50 of them. We can make simple assertions where confidence intervals do not overlap, or between two states where $p < 0.05$, but unfortunately nothing beyond that.

The conclusions that we *can* draw on a state level are that, loosely speaking, states that are more populated or more urban tend to have better Internet than those are are less populated or more rural on average. The District of Columbia, being only a city and also the seat of the US federal government, has the best Internet. On the other hand, sparsely populated states like Montana or North Dakota rank at absolute last. To get an accurate idea of Internet connectivity, the only statistically-valid choice is to use aggregation by a narrower area, or a continuous interpolation method like the IDW heatmap in Fig. 5.14.

5.5 Summary

The analysis of traceroute data from CAIDA and RIPE Atlas provided a massive quantity of data. During the process we developed a program that could process it all and store it in a database of ~71 billion rows, which we were able to analyze. We devised a new method for normalizing by distance alongside several unique methods of plotting the data, before ultimately settling on an inverse distance weighting (IDW) heatmap.

We attempted to perform statistical analyses on the data set to determine if ranking states by Internet connectivity was possible, but unfortunately the data did not support such a ranking. We managed to generate a confidence intervals chart and associated table, however, that allow us to at least compare the extreme ends of the data.

6

RTT to Top Websites: Site Ping

A common use for the Internet is accessing web sites. Connection speed to different sites often vary based on a variety of factors. In this chapter we attempt to measure end users connection speeds to top websites. This metric of Internet connectivity is discussed further in Section 3.4.

6.1 Design

The "site ping" site was developed as a tool for crowd-sourced Internet speed testing. The site attempts to measure RTT between an end user and multiple websites in the top 50 Alexa list. The Alexa top 50 list was chosen as a source for sites since it represents a significant fraction of sites that users are most likely to use. The complete list of the sites used by the site may be found in appendix C.

6.1.1 RTT Measurement

There is no standardized protocol for measuring the RTT to an arbitrary web server using only client-side browser technologies. Our solution was to request objects from web servers and measure the time taken for the client to receive them. Due to cross-origin resource sharing (CORS) restrictions,[1] the only way for client-side Javascript to request external objects is using the Hypertext Markup Language (HTML) `img` tag, which is limited to requesting image resources.

Transmission control protocol (TCP), the underlying transport protocol used for HTTP, performs a handshake between server and client when establishing a connection. This handshake requires additional packets to be exchanged, adding time to the overall transaction. In addition, modern websites use hypertext transfer protocol secure (HTTPS), which adds transport layer security (TLS) encryption, and a TLS handshake following the TCP

[1] CORS is a security policy implemented by browsers, designed to guard against cross-site scripting attacks by preventing scripts from loading resources or sending data to another domain. For instance, if a malicious user injects a script into another user's view of `foo.com`, that script would *only* be able to send or receive data from `foo.com`, and not `bar.com`.

handshake. As a result, simply measuring the time it takes to request an object includes much more then just the time to transfer the data.

To work around this, the website makes two requests for an object from a web server. The first request includes the HTTP `keep-alive` parameter, which keeps the TCP connection open after the transaction is complete. The second request is timed, and because the connection was kept open the timing results do not include the overhead of the TCP and TLS handshakes.

Because our goal is to measure RTT, we want data received by the image tag to be transmitted in a single packet – any more would incur additional delays from extra TCP packets and acknowledgements. We wrote a plugin for the Google Chrome browser that uses the Chrome developer tools API to find the smallest image loaded by each of the websites we wanted to test.

6.1.2 Geolocation

Site ping creates and displays a map of the US with colored data points representing users. The color of the points corresponds to RTT, and the point's location is that of the user. Texas A&M's API is used to determine a user's city in cases where the user's device reports a location (e.g. a smartphone) in geographic coordinates. Otherwise, IP addresses are stored for offline processing using the MaxMind database.

6.1.3 Connection Type Reporting

Since it is important to compare RTTs for comparable devices (e.g. wired (Ethernet), WiFi, or cellular), site ping makes a best-effort attempt to determine each user's connection type. This is done using the "network information" browser API. This experimental API is only enabled by default in Google Chrome and Opera but can be manually enabled on Mozilla Firefox. On mobile devices it is only available on Android browsers, and not iOS.

6.1.4 Displaying Results to the User

The site displays a map to the user. This allows users to see not only real-time results of their test when their device reports a location, but also compare their results to others. This was intended to create appeal for users.

6.2 Implementation

To measure Internet connectivity from a users browser we had to to develop ways to collect data via a user's browser, and find ways of distributing the site to users.

6.2.1 Data Collection

JavaScript does not have an easy or well-supported method for client-side browsers to measure Internet connectivity. After researching alternatives to the network information

API, we settled on the idea of timing how long it takes for the client to load small images. Early prototypes were based on a similar tool called PingJS [9]. To collect data, the site dynamically inserts HTML `img` tags pointing to a number of small resources, each owned by one of the top 45 sites in the US. The site records the time prior to `img` tag insertion and again upon completion of loading; this method is shown in snippet 6.1. Once resources from all 45 sites have been loaded, the client packages the data into a JSON object and sends it via web socket to the server, which stores the data in a MongoDB database [24].

```
function ping(url) {
  return new Promise(function (resolve, reject) {
    const start = (new Date()).getTime();
    const response = function () {
      let delta = ((new Date()).getTime() - start);
      resolve(delta);
    };
    request_image(url).then(response).catch(reason => reject(reason));
  });
}
```

Snippet 6.1: JavaScript “ping” function

6.2.2 Selection of Loaded Resources

Initially we decided to load favicons as our images. Favicons are small images that browsers use to display an icon for the site, usually in the address bar. These images work well because they were easy to find and every website had them. Unfortunately they often varied greatly in size. Ideally all of the images loaded would have been under 1500 bytes so that they would fit in a single TCP packet. To get the consistency and small image size needed, we developed an extension for the Chrome web browser that visits each of the top 45 sites and loads all of the images available, choosing and outputting the smallest one; a complete list may be found in appendix C. Unfortunately, a few of the chosen images were in a file format that was not readable by some browsers and as a result fewer data points were received from those sites.

6.2.3 Site Ping Front End

For the website itself we used the JavaScript Library `D3.js` (“D3”) to draw maps. D3 is a library specifically designed for visualization of large data sets [2]. To draw the states themselves, D3 loads a JSON file containing a definition of the map and renders it as an scalable vector graphics (SVG) image on the screen. To draw points for each city, collected data is passed into D3. D3 then converts the longitude and latitude to coordinates on the screen and uses a color gradient based on RTT to determine the color of the dot. An example of such a map is displayed in Fig. 6.1.

Initially, we used the absolute minimum and maximum values of the data to determine the limits of the gradient, but we found that color gradients would often be skewed towards

outliers. To solve this problem, we switched to using the inter-quartile range (IQR) of the data set for the color gradient. All of the points outside this range are displayed as pure red or pure white. This eliminates outlier effects while staying fast enough to be performed on the server or in the browser, as opposed to slower methods like z-score filtering.

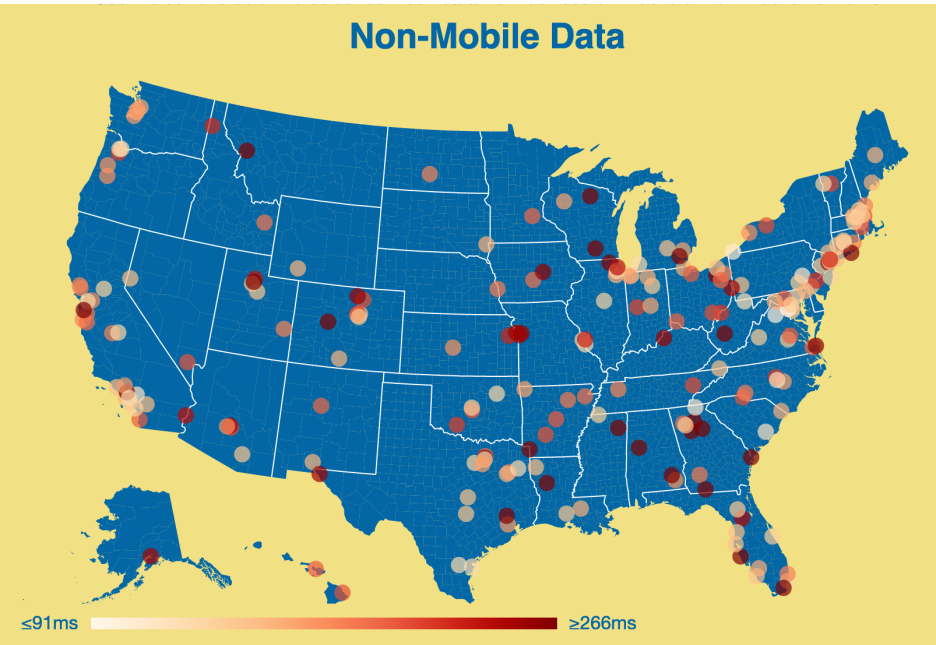


Figure 6.1: Site ping city view

6.2.4 Back End and Database

When a client sends data to the server, the server uses geolocation to assign the data a latitude and longitude. All of this information is stored in the database, and additionally is sent via web sockets to all of the other connected clients so the site map can update in real time. For the back-end database, we chose to use MongoDB to host all of our data [24], which is queried by the server when a client connects. We chose MongoDB because of its simple and free cloud hosting platform (known as Atlas), as well as the ease of integration with NodeJS (our chosen server platform) [26]. The backend web server was a NodeJS server hosted on an AWS Amazon Elastic Compute Cloud (EC2).

6.2.5 Distribution

We used multiple methods to distribute the site to people across the US. These methods included posting to social media, using crowd sourcing, and word of mouth. To track site usage and gauge the success of these methods, we used Google Analytics, a website analytics tool. Analytics uses embedded JavaScript that gathers information about the user's browser and activity. The script can also read other cookies and metadata to determine where the user came from and how long they remained on the site. Finally, the collected information is encoded into the metadata of a request for a one pixel image [10].

Figure 6.2: Facebook site visits over time (2019)

Reddit

The site was first properly distributed on Reddit, a popular message board and content aggregation site. We posted in two communities, /r/dataisbeautiful (14.2 million readers) and /r/samplesize (121k readers). Unfortunately, posting on Reddit proved unsuccessful and gained only around 10 views. Our posts failed to gain traction on either of these subreddits and thus few people saw them.

Facebook

One of our most successful methods for distributing the site was through Facebook. Initially, one of our members posted it on his own personal page, after which the site received a few views. Soon afterwards, a family member posted the link into the WPI parents' Facebook group, resulting 125 views across the US. Finally, a group member posted the link to four of the WPI undergraduate Facebook groups, resulting in a further 70 views of the site. We intentionally waited to post to the WPI undergraduates Facebook group until winter break started, hoping to maximize geographic diversity. The timeline of visits to the site originating from Facebook is displayed in Fig. 6.2. Despite this distribution, geographic diversity was limited to areas where family lives or WPI undergraduates are typically from – Colorado and the Northeast, respectively.

Amazon Mechanical Turk

To reach people in states lacking data, we turned to Amazon's Mechanical Turk service. Mechanical Turk is a crowd sourcing platform that allows people to pay others ("workers") to complete small online tasks ("HITs"[2]). We paid 1-50 cents for users in states that we had little data for, with higher incentives for states with sparser results. When someone selected our task, they were directed to a special on the site ping website. After they began data collection and waited for two cycles worth of data, they were given a token which they copied and pasted into Mechanical Turk to ensure that they actually completed the task. The tokens were generated using a JSON Web Token (JWT) module for NodeJS. Once the worker submitted the task, an AWS Lambda (with the same server-side secret used to generate the token), validated the token and approved payment. Using Mechanical Turk resulted in an additional 221 visits to the site, bringing the total of covered states up to 47. Figure 6.3 shows the nationwide distribution of Mechanical Turk users. Note that this

[2]Human Intelligence Tasks

includes users that visited the site but did not complete the task (including completing it incorrectly and returning it).

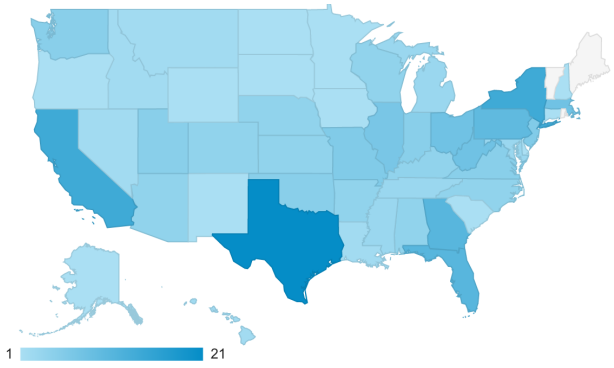


Figure 6.3: Involvement from Mechanical Turk by state

6.3 Analysis

The site ping website yielded data across the entire US. We had to find ways of visualizing the data and analyzing trends.

6.3.1 Initial Site Ping Analysis

The first analyses was conducted live, directly on the site for users to see on the aforementioned D3-powered map view (see Fig. 6.4). This map allowed us to both come to early conclusions about the data, and allowed participants to be able to see the data in real time. This map is calculated by averaging all of the cities in the state and then using the IQR to scale the color scheme.

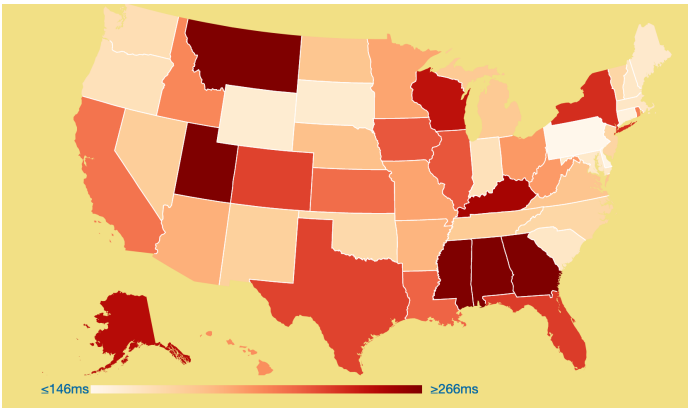


Figure 6.4: State view chloropleth map

Data was aggregated server-side by state and city to rank them from best and worst and display the rankings on the site. Every new data point caused an update in the analyses –

both for our own uses and to help users understand how their state performs. An example of these rankings is shown in Fig. 6.5.

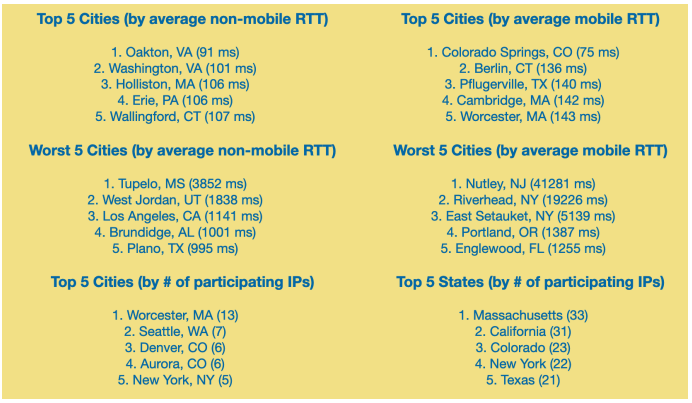


Figure 6.5: Site ping live-updating state rankings

6.3.2 Data Quality

The first measure of data quality tried was standard deviation. Figure 6.6 shows the per-location distribution of these values, which follow a fairly smooth curve.

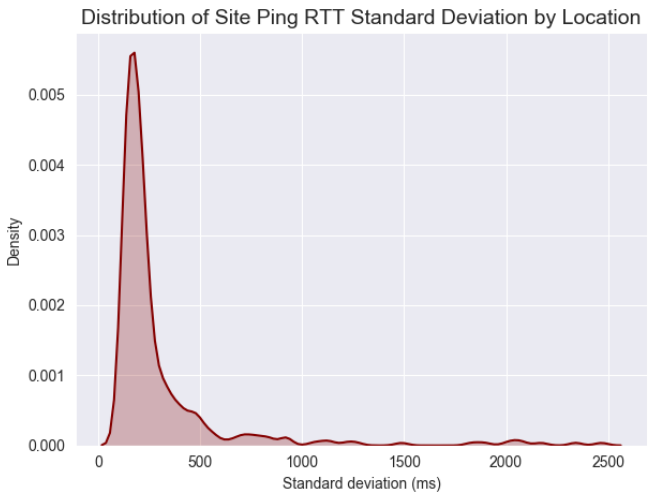


Figure 6.6: Distribution of site ping standard deviations by location

The overwhelming majority of points have standard deviations of around 150 milliseconds. This is a fairly high standard deviation, most likely caused by the small number of data points for each location.

Next, we calculated the CV for each location, shown in Fig. 6.7. CVs are dimensionless values that can be judged independent of the original source, making them useful for gauging the spread of any data set. The lower the CV, the lower the spread of the data and the better the quality. Most of the states are around 1, meaning there is significant variance

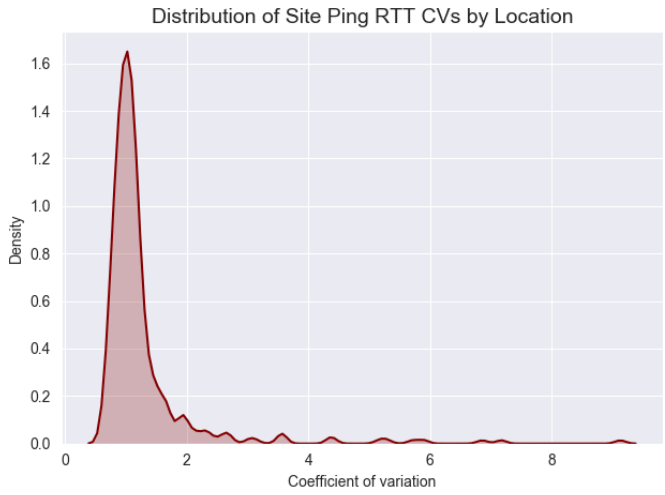


Figure 6.7: Distribution of coefficients of variation by location

in the data. Again, this is most likely related to the relatively small number of data points that exist for each location.

6.3.3 Filtering

To remove outliers in our data we used a simple z-score filter with z-score of two. More info on z-score filtering can be found in Section 4.3.1. When the data was filtered, we found that it brought the resulting value standard deviation from 511 to 175 while still keeping 95% of the data points.

We also filtered out all of the data that came from mobile devices. This was due to the unpredictability of it and the fact that we could not tell if they were connected to a cellular connection or a broadband connection.

Evaluating State Rankings

To examine whether or not it was statistically allowable to aggregate the data by state, we used the Kruskal-Wallis test, as described in Section 4.3. From there we grouped the states into cliques based on whether or not they could be compared with all the other states. Figure 6.8 shows a CDF of all of the groupings of states.

Table 6.1 only provides a rough estimate of Internet Connectivity. It was created based on the median RTT of all of the data points within a state after z-score filtering the entire data set. Median was chosen because we felt it gives the best representation of the state while not being highly influenced by outliers. While this is not a comprehensive rank, it does provided a strong high level view of Internet connectivity across the US. As expected, the more densely populated areas have better connectivity, like DC and Delaware. More rural areas, like Alaska and Montana have worse connectivity. North Dakota was filtered out with z-score filtering.

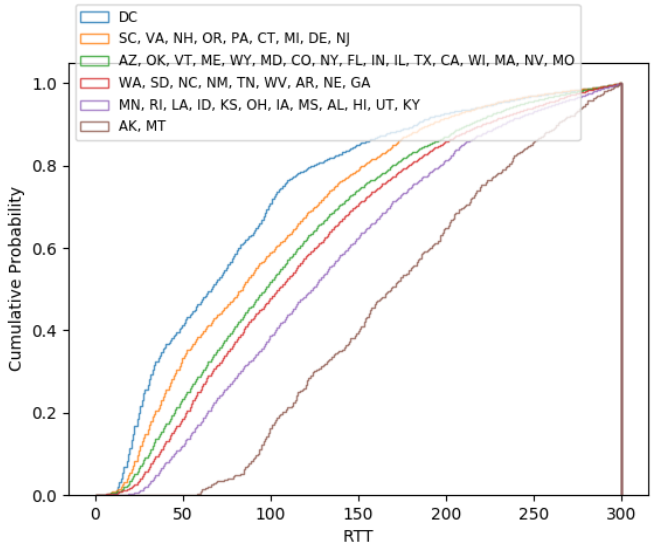


Figure 6.8: Continuous distribution function of state rankings

Table 6.1: Site ping state rankings

Rank	State	(ms)	Rank	State	(ms)	Rank	State	(ms)
1	DC	73.0	17	WY	108.0	35	WA	129.0
2	DE	75.5	19	FL	109.0	36	MN	130.0
3	CT	76.0	19	IL	109.0	37	NM	132.0
4	NH	80.0	21	CO	110.0	38	AL	133.0
5	NJ	83.0	22	WI	113.5	39	OH	134.0
6	PA	85.0	23	WV	115.0	40	MS	140.0
7	MI	88.0	23	MO	115.0	41	KS	142.5
8	SC	89.0	25	TX	116.0	42	UT	148.0
9	VA	90.0	26	SD	116.5	43	IA	153.0
10	OK	98.0	27	AR	117.0	44	HI	157.0
11	MD	100.0	27	CA	117.0	45	ID	162.5
12	NY	101.0	29	NV	118.0	46	RI	164.0
13	MA	102.0	30	AZ	119.0	47	LA	169.0
14	VT	104.0	31	GA	120.0	48	KY	179.5
15	IN	105.0	32	NC	121.0	49	MT	197.5
16	OR	106.0	32	TN	121.0	50	AK	217.0
17	ME	108.0	34	NE	122.0	–	ND	–

6.3.4 Results

CDNs

When searching for the small image files we discovered that the vast majority of websites use a CDN for serving their content, including both JavaScript as well as static content. As such, our ping results are a reflection of a user's connection to each CDN. To determine

if there was a strong correlation between user location and speed to a given CDN we attempted to determine the CDN each site was using through ICMP pings and reverse DNS lookups, then mapped the pings for a given CDN (including pings to all the sites that use that CDN). We found that the RTT to a given CDN tended to be similar across a given area.

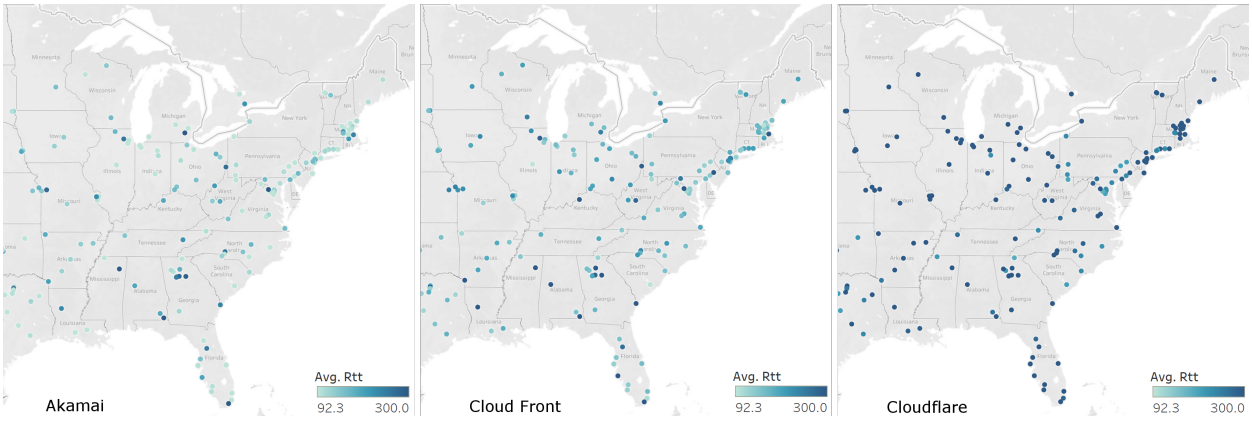


Figure 6.9: Pings to sites on the Akamai CDN

Final Heat Map

The final heat map produced from the site ping data shows clear trends. The coastal and central areas have significantly lower ping times than the southeast and northwest. In states with high ping times there are pockets of low ping times around major cities, such as in Tampa, Jacksonville, Orlando, and Miami, Florida; Knoxville and Memphis, Tennessee; and Colorado Springs, Colorado. There are some areas with surprising, and somewhat dubious results. For example, New York City, New York; Seattle, Washington; and Los Angeles, California all appear to be in pockets of poor RTT surrounded by good RTT.

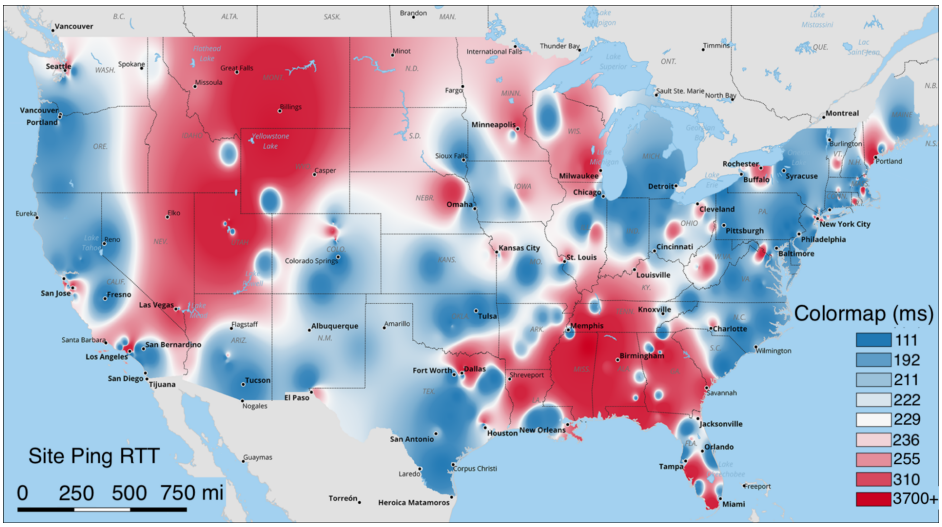


Figure 6.10: Final heat map

6.4 Summary

Using our Site Ping technique we successfully gathered data from many points across the US including at least one point from all 50 states. The data we collected provided a view into the connectivity that Americans get to the part of the internet they use most, websites.

Overall we found that the coastal and central regions tend have significantly lower ping times then those in southeast and northwest. While it was difficult to statistically differentiate between states in the middle of the ranking, those on the ends were clearly defined. Over all District of Colombia (DC) proved to have strong connectivity. More rural states, such as Alaska and Montana had the worst connectivity.

7

Aggregate RTT by Region: DNS Cache Manipulation

This chapter focuses on evaluating Internet connectivity based on the "RTT to Regional Locations" (Section 3.2), with a focus on measuring DNS infrastructure connectivity as a proxy for overall connectivity. DNS is a crucial component of the Internet. By measuring a region's ability to resolve domain names using servers in other regions, we can evaluate how well the first region is connected to the rest of the country. To accomplish this, we used a method called DNS cache manipulation and over 900 DNS servers spread across the United States and measured the RTT between them. The following sections detail the design, implementation, and results of this method.

7.1 Design

The DNS cache manipulation method is largely derived from the method used in "The Internet Connected Project," a 2018 MQP preceding this one [6]. The method makes use of two sets of geographically diverse DNS servers: one list each of recursive and authoritative servers. The basic principle rests on the concept of making requests directly to each of the recursive servers that *must* make their way to the authoritative ones and deducing the time between the two. As DNS includes no mechanism for reporting the time taken between two servers, DNS cache manipulation must rely on the RTT from when the local request is dispatched to when an answer is received.

7.1.1 Collection Stages

Collecting data via DNS cache manipulation takes six stages of processing: three for pre-processing and three to make the actual measurements. The preparatory stages, server confirmation, reliability checks, and geolocation, eliminate servers that are not appropriate to include in the study.

Figure 7.1 shows the three stages required for data collection: priming the DNS cache, measuring latency to the recursive DNS server, and measuring the lookup time. We will now discuss all six stages in more detail.

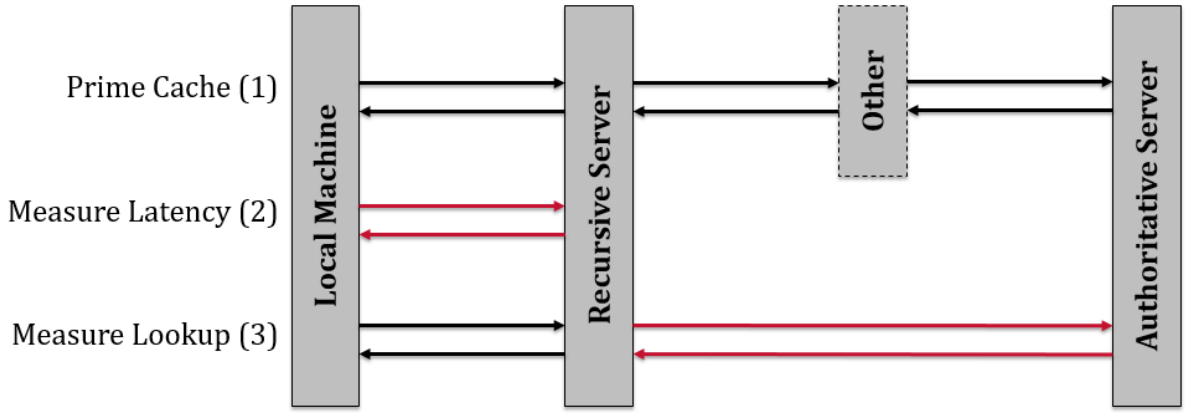


Figure 7.1: DNS cache manipulation stages

Confirmation of Server Status and Type

Prior to running any tests, two lists of candidate recursive and authoritative servers are fed into tools that confirm their status as usable servers. For recursive servers, this process confirms that they are indeed public recursive servers. For authoritative candidates, it verifies that the server provides an answer for the domain it is expected to.

Testing Reliability

Before actually using any of the servers for testing, we tested their reliability – i.e. how consistent their response times were under constant parameters. After making repeated requests to each server, that in theory should have similar response times, we filter servers based on the CV for the measured response times. This eliminated any servers with high variation from the testing. For recursive servers, priming the cache (Section 7.1.1) and then making repeated requests for the primed value served as the repeated requests. For authoritative servers, this was accomplished by making repeated requests for a random subdomain that the servers are authorities for through the WPI recursive DNS server. This server is close enough to the machine we ran the tests on that the latency was minimal. The justification for running the filtering prior to testing is that each server included in the test increased the time required to run the entire test suite exponentially, as each recursive server was paired with each authoritative server.

IP Geolocation

Just as in Chapter 5 and Chapter 6, the DNS cache manipulation used IP address geolocation provided by MaxMind to determine the location of the servers under test. We collected both state and coordinate information for each server. If a server could not be located using the database, it was discarded from the test at a later point.

Priming the DNS Cache

The first step of the actual testing involves priming the target recursive server's cache. Say the target recursive server's IP address is `1.2.3.4` and the target authoritative server provides answers for `example.com`. By making a DNS query to `1.2.3.4` for `example.com`, we get the target DNS server to cache the answer to the query.

Measuring Latency

After the target recursive server has cached the answer for the query for `example.com`, it will use that cached answer to respond to subsequent requests without making any iterative requests. We can then make another request, identical to the one in the priming stage, and time it. To get a latency measurement, we do this several times back to back and take the lowest value, which indicates the best possible latency between the machine the measurements are being taken on and the recursive server.

Measuring the Total RTT

The final step in the process is to measure the RTT between the local machine and the target authoritative server and subtract the latency to get the final result. To do this, we make a query to the recursive server for `random.example.com`, where `random` is a randomly generated sub-domain that is unlikely to actually exist on the targeted domain. This random sub-domain forces the recursive server to get an answer. Because it already knows the authoritative server for `example.com`, it can skip the process of querying the root and `.com` servers and make an immediate query to the server for `example.com`. We measure the time for this entire process and then subtract the previously measured latency, leaving only the time between the target recursive server and the target authoritative server, which is recorded.

7.2 Implementation

This section details the implementation of the methodology designed in the Section 7.1. The implementation utilizes the command line utility `dig` to make DNS requests, Python to wrap around dig to make parsing its output easier, a bash script, and the `parallel` command line utility to enable parallel processing.

7.2.1 Tools and Setup

The entirety of the data collection for DNS cache manipulation took place on two WPI servers: `ccc.wpi.edu` and `rambo.wpi.edu`.

At its core, this method used the `dig` utility for making DNS requests - this is the same tool used in the prior project [6]. The template command used for all stages of testing is shown in snippet 7.1.

Notably, this command template does not include the `+noall` flag, unlike the prior project [6]. By omitting this flag in our version, we preserve the status and records actually

```
>dig [@<target server>] <target_domain> time=2 tries=1 +stats
```

Snippet 7.1: Template `dig` command

produced by the recursive DNS server. snippet 7.2 is an example of such output (modified slightly for formatting). This output includes the status flag `NXDOMAIN` which would be omitted with the `+notall` parameter. This status flag confirms that the random subdomain for a given domain does indeed not resolve to anything - this is important for the testing because if it did, the recursive server may have had a cache for that subdomain, invalidating the lookup time check. Additionally, this version of the command include the `ANSWER` and `AUTHORITY` sections (depending on the lookup) which aid in validating the type of server and the responses they give. These too would be left out with the `+noall` flag.

```
>dig @8.8.8.8 doesnotexist.wpi.edu +time=2 +tries=1 +stats

; «» DiG 9.11.3-1ubuntu1.11-Ubuntu «» @8.8.8.8
doesnotexist.wpi.edu +time=2 +tries=1 +stats
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 31928
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 1,
ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;doesnotexist.wpi.edu.          IN      A

;; AUTHORITY SECTION:
wpi.edu.               1799    IN      SOA     adns1.wpi.edu.
netops.wpi.edu. 2010473291 3600 600 1209600 3600

;; Query time: 88 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Thu Feb 06 00:12:43 STD 2020
;; MSG SIZE  rcvd: 98
```

Snippet 7.2: Generic `dig` output

Instead of running the `dig` tool directly via a bash script, we wrapped it in a Python script for ease of parsing. This utility itself utilized the `subprocess` module to make `dig` calls.

As mentioned, each stage in this method was implemented using Python, separating into several scripts. These were all brought together with a bash script and the `parallel` utility

[33], which allowed for parallelization, greatly reducing the time required for testing.

```
>./parallel -a test_pairs.csv -colsep , -header '.*\n'
    -progress -eta -jobs $JOB_COUNT
    ./run_test.py {1} {2} {3} $TEST_TRY_COUNT $TIMEOUT
    » results/test_results.csv
```

Snippet 7.3: Sample `parallel` command

Snippet 7.3 shows the parallel command used to run the primary portion of the test. It uses a comma separated value (CSV) listing the pairs of recursive and authoritative DNS servers and pipes them into the Python script `run_test.py` before outputting the results. This is done with `JOBCOUNT` parallel jobs, which was configured to 32 jobs for each batch.

7.2.2 Candidate DNS server lists

Initial lists of both types of DNS servers were provided to us by our advisor, Professor Craig Wills. After performing initial verification of these servers, we augmented the list by searching through a list of '.gov' domains filtered by the states we needed to fill in gaps for. In total, these lists included 535 authoritative servers and 1225 recursive servers (prior to final filtering). Note that the final list of recursive servers did not include any located in the state of Rhode Island despite an extensive search for one.

7.2.3 Pre-processing Stages

These stages ran prior to any actual data collection in an effort to streamline and shorten that collection processes. Each ran as one or more separate steps in the overall script, in the order presented.

Confirmation of Server Status and Type

The code in snippet 7.4 is what we used to confirm that an expected authoritative server was indeed an authoritative server. The request for the domain tied to the server was directed directly to the its IP address. If there was a result (i.e. `dig` output a valid result), the status of that result was `NOERROR` (the server was able to generate a response for the given domain) and the `AUTHORITY` section of the response was non-zero in length, the server was confirmed as a authoritative for the expected domain.

```
result is not None
    and result.status == "NOERROR"
    and result.AUTHORITY > 0
```

Snippet 7.4: DNS authoritative confirmation

The code use to confirm recursive servers, was similar: there must be a result with an answer. However, it must also not have the phrase `Recursion requested but not available`. Additionally, it is allowed to have responses in the `ANSWER` section instead of just in the `AUTHORITY` section.

Testing Reliability

Snippet 7.5 is part of the code used to verify the reliability of candidate authoritative servers. The code was passed a domain and IP address for the authoritative server under tests. For a given number of trials, configured to 20 for every batch, the query time for a request made directly to the authoritative server for a domain we have already confirmed it was an authority for, was recorded [1]. After all results were recorded, the authoritative server's IP address was output if there were at least two successful results and the CV was less than or equal to the threshold, configured to 0.5 for all tests.

```
1     for _ in range(trials):
2         result = run_dig(domain=domain, target_server=target_ip)
3         if <result is valid>:
4             results.append(result.query_time)
5     if len(results) >= 2 and stdev(results)/mean(results) <= max_cov:
6         print("{},{}".format(target_ip, generate_statistics(results)))
```

Snippet 7.5: Authoritative server reliability measuirng (modified for formatting)

Similarly, the code for testing the reliability of an authoritative server is similar. A domain for testing (`cnn.com` for all runs) and a recursive server IP address were passed in. First, the cache is primed to ensure the domain is cached in the server and then the query time for requesting that same domain from the server is recorded for a given number of trials (again, configured to 20 for each batch). Finally, if there were at least two results and the CV met the threshold (0.5 for all tests), the domain was output as reliable. This is essentially making repeated latency measurements and ensuring that the results are consistent.

Note that both checks also output other statistics alongside the servers. These included the measurement mean, median, standard deviation, and variance. This gave us the option to do further analysis on server reliability, which we opted not to pursue.

7.2.4 Data Collection Stages

These three stages are all contained within a single script and are run back-to-back-to-back with each other. The two main parameters for the script are a single IP address, `recursive_ip`, corresponding to a recursive server and a domain name, `authoritative_domain`, and IP address, `authoritative_ip`, corresponding to the an authoritative server.

Priming the DNS Cache

Priming the DNS cache was performed by making a single DNS query to `recursive_ip` for `authoritative_domain`. This request did not include any randomized subdomain and occurred immediately prior to latency measurement.

[1] For a definition of a valid result, see Section 7.2.3 and snippet 7.4

Measuring Latency

Snippet 7.6 shows the code that we used to measure the latency between the testing location and the recursive DNS server under test. For all tests, `tries = 10`, meaning that for each test for a given server pair, there were 10 latency measurements. Of the measurements, we kept the lowest one, as this indicated the shortest observed latency.

```
[run_dig(domain=authoritative_domain,
         target_server=recursive_ip,
         time=TIME_LIMIT)
    for _ in range(tries)]
```

Snippet 7.6: DNS latency measuring

The authoritative domain is provided without any subdomain, as we know that, due to the cache priming stage, the domain is cached in the recursive server.

Measuring Lookup RTT

Similar to the snippet for measuring latency, snippet 7.7 shows the snippet for measuring the total lookup time. Notably, a 12 character, randomized subdomain is added to the target authoritative server's domain. Once again, the number of observations was configured to 10 for all trials, and the minimum observed value was kept.

```
1 [run_dig(domain="{}.{}".format(rand_subdomain(), authoritative_domain),
2          target_server=recursive_ip,
3          time=TIME_LIMIT)
4      for _ in range(tries)]
```

Snippet 7.7: DNS lookup RTT measuring

To calculate the final RTT observed in a given test for a given pair of servers, we simply subtracted the lowest latency measurement from the lowest round-trip measurement to get the best possible RTT between the recursive DNS server and the authoritative DNS server.

7.3 Collection Results

Using the implementation outlined in the previous section, we ran four batches of data collection. This section provides an overview of the results of that collection, including issues we encountered and potential future mitigations for these issues.

7.3.1 Server Reliability Filtering

After filtering authoritative and recursive DNS servers for testing reliability (max coefficient of variation in repeated tests = 0.75), we were left with 387 authoritative servers across

50 states and the District of Columbia and 654 recursive servers across 49 states and the District of Columbia. We could not locate an open recursive DNS server, even unreliable, in the state of Rhode Island. Figure 7.2 and Fig. 7.3 show the locations of the final server lists for authoritative servers and recursive servers respectively.

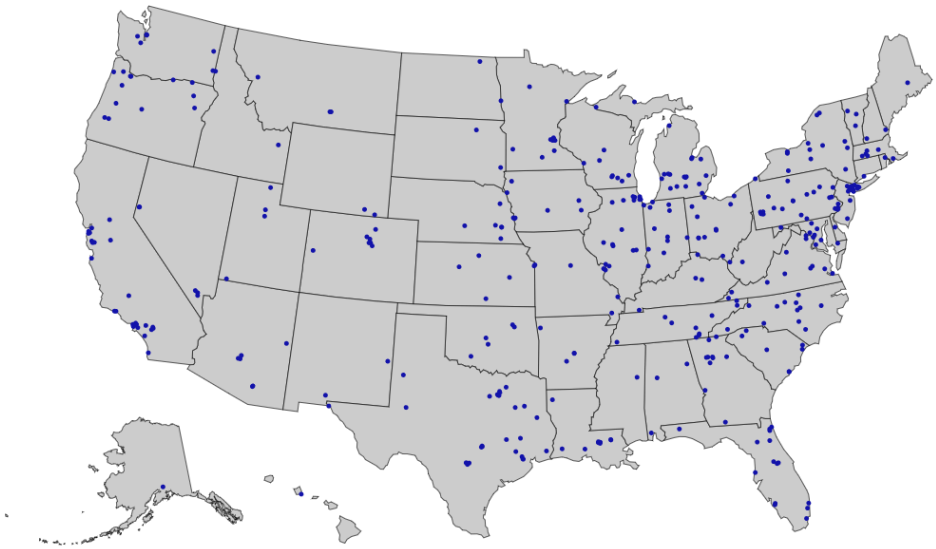


Figure 7.2: Map of authoritative DNS servers

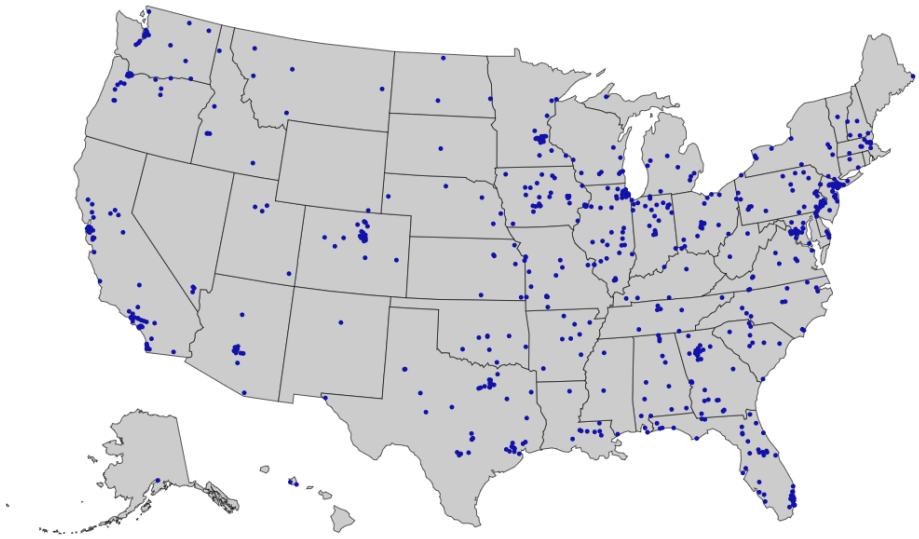


Figure 7.3: Map of recursive DNS servers

7.3.2 Data Collection

Data collection was done in two main batches, along with two supplemental ones to add additional servers and increase geographical coverage. All runs used `cnn.com` as

the domain used to confirm candidate server's status as recursive servers and to test the recursive server's reliability.

Table 7.1: Overview of DNS batch runs

#	Trial Count	Timeout	Records	Runtime	Notes
1	5	2s	1,831,495	62h	
2	10	3s	3,763,311	244h	
3	10	3s	4,920	9m	Sup. for VT rec.
4	10	3s	7,320	11m	Sup. for HI auth.
		Total	5,607,046	306 hours	

Again, as Table 7.1 shows, the vast majority of data was collected over the course of two collection runs, #1 and #2. The first run only conducted five trials for each authoritative-recursive server pair and limited the timeout to for each `dig` command to two seconds, whereas subsequent runs ran with ten trials and a three second timeout. There are two reasons for this. First, the initial batch was run on `ccc.wpi.edu`, which has more limitations than the machine used for subsequent runs, `rambo.wpi.edu`. Additionally, that run was the first major step up from the more limited development test runs and we wanted to minimize time lost in the event something went wrong. On subsequent tests, we had more confidence in the collection scripts and were able to let it run for longer unsupervised.

Batch #3 was run with a limited number of recursive servers located only in Vermont (with the full authoritative server list) while batch #4 ran with a limited number of authoritative servers located only in Hawaii (with the full recursive server list). These were run after we discovered that we lacked data of the given type for these states.

We ran into one issue with batch #4: the collection scripts randomize the order of all pairs under test to minimize the repetitive requests to the same DNS servers in too short of a time period. However, in both supplemental tests, there were only small numbers of recursive servers (for #3) or authoritative servers (for #4). In batch #4, this resulted in repetitive requests to the same authoritative server in a short period of time, which in turn caused WPI's IT system to believe we were using a DNS tunnel. The use of such a tunnel is prohibited under WPI's Acceptable Use Policy and resulted in a temporary suspension of network privileges.

7.4 Analysis

With the data we collected, we conducted two major approaches at analysis. The first approach treated every measurement for a recursive server in a give state as equal and gave each state a score based on these measurements aggregated together. The second method first aggregated measurements within one state based on the authoritative state the measurement was made with, which also gave us the opportunity to weight connections to states differently.

7.4.1 Distance Normalization

One of the initial steps in processing the data collected involved adding an additional field for RTT normalized by the distance between the points being connected. As mentioned earlier in this report, by normalizing for distance, we can form another measurement that allows for analysis of connectivity as related to the *ideal* speed of connection, the speed of light.

Additionally, calculating the speed of the connection, not just the total RTT, allowed us to filter out measurements that were faster than the speed of light. Such measurements were likely the result of anomalous latency readings, causing the calculated RTT to be far lower than physically possible. In total, this initial filter eliminated 418,625 measurements (7.5%) from the total dataset.

7.4.2 Data Characterization

Figure 7.4 shows the distribution of the median true RTT values for all locations. It shows that the data takes on a bimodal distribution with peaks around 45ms and 68ms with a heavy bias towards the former.

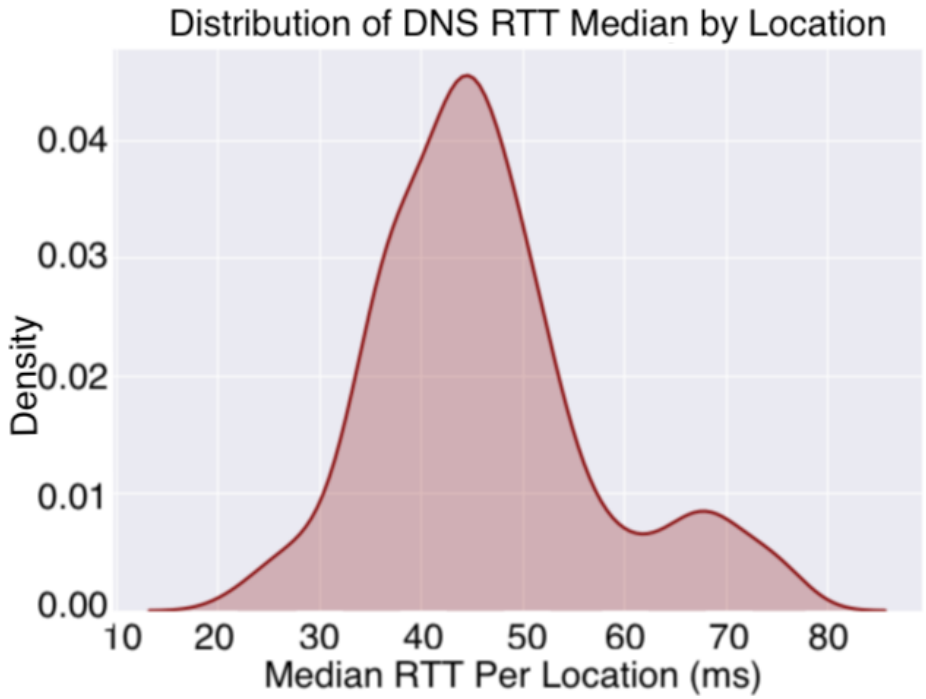


Figure 7.4: DNS RTT median distribution

Normalized RTT

After normalizing by distance, the bimodality all but disappears, as Fig. 7.5 shows. In this chart, the distribution pears around 30km/ms.

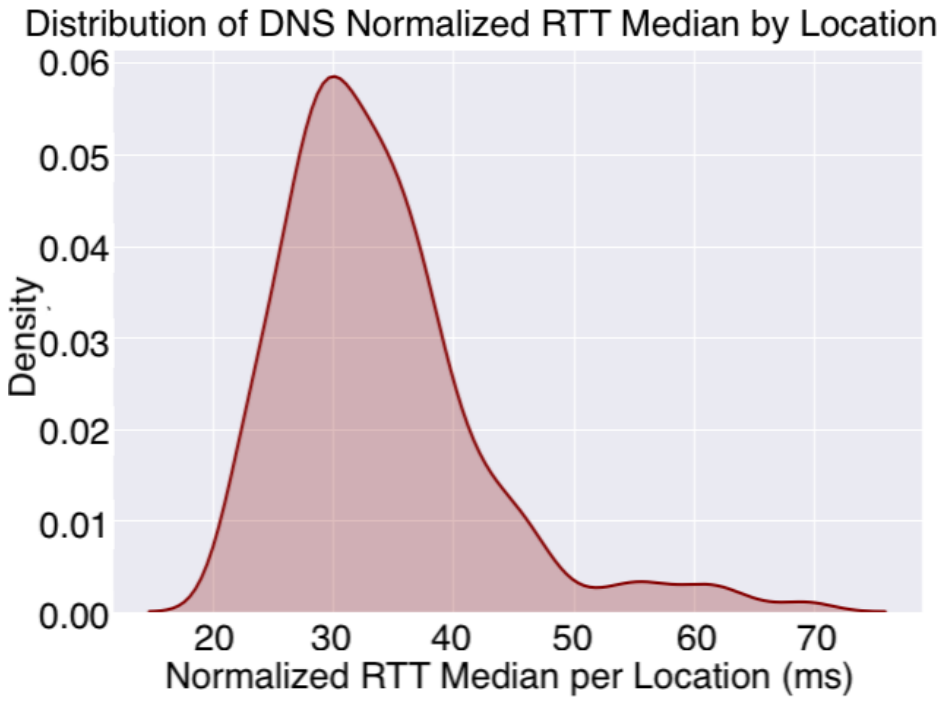


Figure 7.5: DNS normalized RTT median distribution

7.4.3 Heat Maps

Figure 7.6 shows a heatmap based on true DNS RTT measurements. This map shows that a significant portion of Western US has poor DNS RTTs. In contrast to the West, a majority of the the East Coast performs better. As we discuss later, this is likely due to the higher quantity of authoritative servers present in the East. Another highlight is that the South is rather inconsistent: Mississippi is probably the most homogeneous in being bad, but Alabama, Georgia, Louisiana, Tennessee, Missouri, and even eastern Texas are rather "splotchy", for lack of a better term.

Figure 7.7 shows a similar heatmap, but for distance normalized DNS RTT data. In stark contrast to the prior map, the West coast displays significantly better performance - likely due to the distance to eastern authoritative servers being cancelled out. The Midwest and mid-Atlantic regions fare much worse, while the South is far more consistent in its poor results. The two main regions that remain consistent between the two maps are the North East and Colorado/Wyoming.

7.4.4 Aggregation by Server Pairs

Since there we multiple data points for each recursive-authoritative server pair, the first step in aggregating the data was to aggregate by these server pairs. To do this, we removed points within each pair that had a z-score greater than a given threshold `z` as these were considered outliers within their pairs. We then computed the CV of each pair with the remaining values and discarded any pairs with a result greater than a given threshold `v` - these pairs did not have consistent measurements and were considered unreliable.

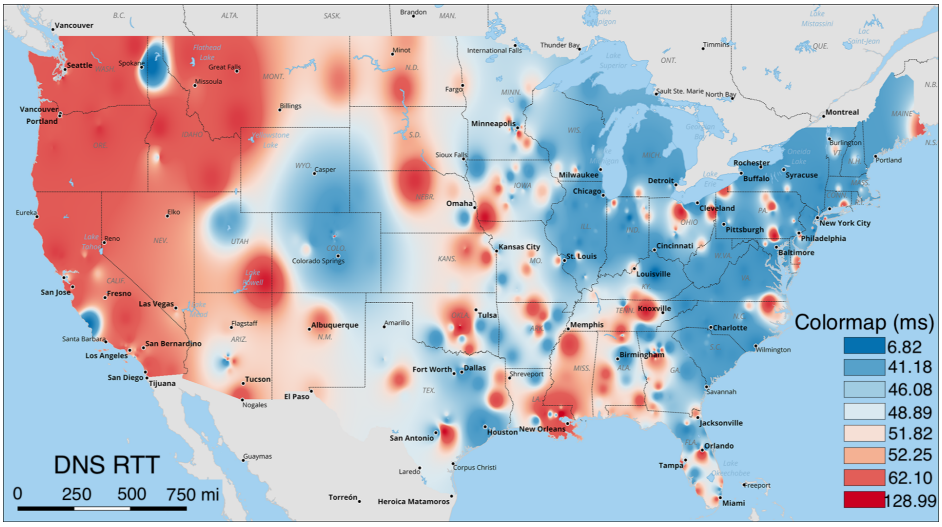


Figure 7.6: DNS true RTT heatmap

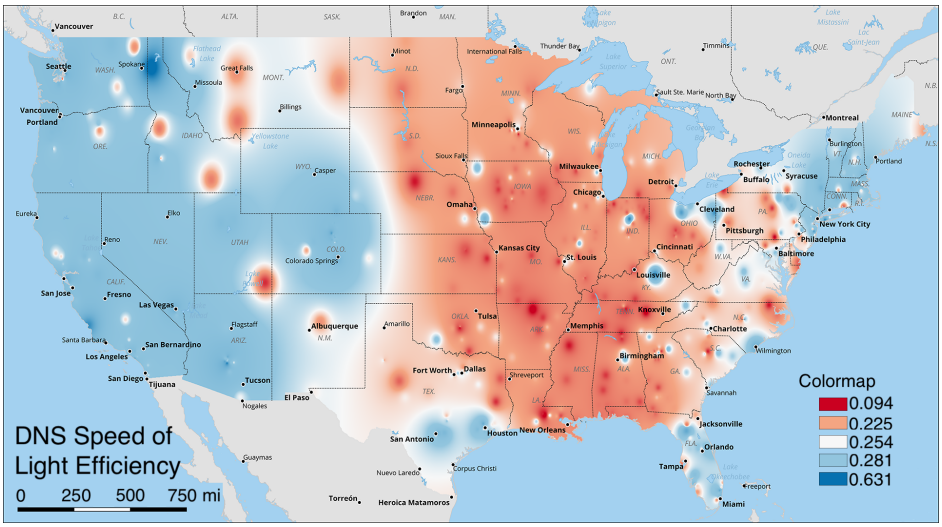


Figure 7.7: DNS normalized RTT heatmap

We chose to proceed with `v=1.0` and `z=2`, which left the vast majority of pairs in the data set excluding only the most inconsistent measurements. The reasoning behind this is that we are not just interested in stable connections - so we left potentially unstable or inconsistent measurements in the data set.

7.4.5 Aggregating Pairs by Recursive Server State

One option to further aggregate the pair data is to group it by the location of the recursive server in each pair. This results in a list of RTTs, either true or normalized by distance, that can be aggregated into a single value for each state. This is one way of making comparisons between states.

Table 7.2: DNS Z-score filtering

	True RTT records dropped	**Normalized RTT records dropped**
z=3	92,553/5,188,421 (1.8%)	63,008/5,188,421 (1.2%)
z=2	269,651/5,188,421 (5.2%)	237,353/5,188,421 (4.6%)
z=1	1,362,184/5,188,421 (26.3%)	1,389,031/5,188,421 (26.8%)

Table 7.3: True RTT DNS pair CV filtering

	z=3 (n=381,333)	z=2 (n=381,333)	z=1 (n=381,333)
v=0.05	193,099 (50.6%)	167,618 (44.0%)	92,799 (24.3%)
v=0.10	111,294 (29.2%)	92,023 (24.1%)	40,889 (10.7%)
v=0.20	52,239 (13.7%)	40,353 (10.6%)	13,262 (3.5%)
v=0.50	9,330 (2.4%)	6,333 (1.7%)	1,809 (0.5%)
v=1.00	1,741 (0.5%)	766 (0.2%)	176 (∼0.0%)

Table 7.4: Normalized RTT DNS pair CV filtering

	z=3 (n=382,963)	z=2 (n=382,963)	z=1 (n=382,963)
v=0.05	192,522 (50.3%)	166,514 (43.5%)	87,383 (22.8%)
v=0.10	109,493 (28.6%)	89,906 (23.5%)	35,553 (9.3%)
v=0.20	48,414 (12.6%)	37,623 (9.8%)	10,579 (2.8%)
v=0.50	6,304 (1.6%)	4,570 (1.2%)	1,314 (0.3%)
v=1.00	1,038 (0.3%)	415 (0.1%)	126 (∼0.0%)

Initial State Rankings

Table 7.5 and Table 7.6 show the results of ranking states by the median of each measurement in that state. There are two different rankings, one each for the true RTT and the distance normalized RTT, based on data filtered with z=2 and v=1.0. As with all DNS data, there is no ranking for Rhode Island.

Notably, the two tables indicate that Wyoming ranks in the top five for both rankings. No other state appears performs so consistently well. Additionally, the rankings show that while Hawaii and Alaska rank near or at the bottom for true RTT, both states appear near the top of the normalized ranking. This shows that while the some states, Alaska and Hawaii in particular, may be inherently unconnected from the rest of the United States, the quality of their connection could be relatively high and merely dominated by sheer distances.

Evaluating State Rankings

To determine the validity of the rankings proposed above we used the Kruskal-Wallis test to determine whether there was evidence that datasets for different states came from different distributions. If we cannot reject the null hypothesis that they are from the same distribution, we cannot rank them in distinct positions. After running Kurskals between the datasets for each of the 49 states and DC, we used a p value threshold of 0.05 to determine

Table 7.5: DNS state rankings – recursive aggregation only – true RTT

Rank	State	(ms)	Rank	State	(ms)	Rank	State	(ms)
1	WY	30.00	19	IN	43.00	34	NV	53.00
2	VA	31.00	19	GA	43.00	38	ID	54.00
2	DC	31.00	21	MN	45.00	39	SD	56.00
4	WV	32.00	22	OK	46.00	39	KS	56.00
4	NY	32.00	22	MO	46.00	41	ND	59.00
6	NJ	33.00	22	LA	46.00	42	CA	59.50
7	MD	34.00	22	AR	46.00	43	NM	60.00
8	SC	34.50	22	CT	46.00	43	HI	60.00
9	CO	38.00	27	FL	47.00	45	MS	62.00
10	NH	38.50	28	IA	48.00	46	OR	65.00
11	IL	39.00	29	KY	49.00	47	WA	67.00
12	MI	40.00	30	NE	49.50	48	ME	67.50
12	PA	40.00	31	UT	50.00	49	MT	71.00
12	WI	40.00	32	VT	51.00	50	AK	99.25
15	NC	41.00	33	TN	52.00	–	RI	–
16	TX	42.00	34	AL	53.00			
16	MA	42.00	34	AZ	53.00			
16	OH	42.00	34	DE	53.00			

if we could differentiate two states from each other.

For true RTT, Fig. 7.8 shows a graph with a node for each of the 49 states and DC and a link between each pair of nodes where the Kruskal-Wallis test between the two yielded a p value greater than 0.05, indicating that there is no evidence of a difference between the two states. Each the y-position for each state in the graph is dictated by its ranking in the the initial ranking table (ties were broken up arbitrarily for readability and the x-position is arbitrary).

Note that the top seven states in the true RTT ranking in Table 7.5, WY, VA, DC, WV, NY, NJ, and MD, form a distinct sub graph. Similarly, the states ranked #9-12, IL, CO, NH, and MI, form another distinct sub graph. This indicates that while the states that make up these two groups cannot be distinguished from one another (i.e. there is no evidence that Wyoming is better than Virginia), we can state the the entirety of the first group is better than the entirety of the second group.

Similar to the groups at the top of the ranking, there are distinct groups at the bottom. Kruskals identified Alaska, #50, as entirely different. However, the six preceding states ranked #43(t) to #49, HI, MS, OR, WA, ME, and MT, are a distinct sub graph, as are the three states preceding them, ND, NM, and HI. Note that in the initial ranking, New Mexico and Hawaii are tied for #43, but are identified as distinct using Kruskals.

Also notable is Louisiana: the state ranks #22(t), but is labeled as indistinguishable from Arizona, #34(t), and Nevada, #34(t).

Figure 7.9 maps the each distinct sub graph to a separate color (graphs are numbered by in the order of their highest ranked consituent state). This highlights some patterns and

Table 7.6: DNS state rankings – recursive aggregation only – normalized RTT

Rank	State	(ms)	Rank	State	(ms)
1	HI	91.00	27	MI	31.54
2	WY	54.47	28	WI	30.61
3	AK	47.35	29	WV	30.61
4	CA	45.46	30	NM	30.35
5	OR	43.87	31	MN	30.00
6	NV	43.68	32	ME	29.79
7	CO	43.61	33	GA	29.59
8	NY	43.04	34	ND	29.41
9	UT	42.72	35	LA	29.33
10	ID	41.97	36	OK	28.91
11	WA	41.82	37	SD	28.63
12	NH	41.66	38	NE	28.28
13	AZ	41.03	39	IL	28.12
14	NJ	40.85	40	DE	27.56
15	MA	40.65	41	MO	26.41
16	VA	37.06	42	AR	26.19
17	FL	37.01	43	OH	25.40
18	TX	36.80	44	IA	25.03
19	MD	36.44	45	AL	24.45
20	DC	35.82	46	IN	24.40
21	SC	34.60	47	KS	22.87
22	CT	33.56	48	KY	21.89
23	PA	33.32	49	TN	21.55
24	NC	32.43	50	MS	20.44
25	MT	31.99	–	RI	–
26	VT	31.63			

some oddities. For example, seven of the top eight states, represented by groups 1 and 2, are located on the East Coast, with Wyoming being the odd state out. The Midwest and the western Mountain region are dominated by states in groups 7 and 8. The Great Lakes region consists primarily of states in groups 3 and four. The West Coast and Hawaii did not fair as well as the East Coast with California, Oregon, and Washington (and Hawaii) in groups 9 and 10. Meanwhile, the South is a bit of a "hodge-podge" of groups, indicating a lack of consistency there.

Finally, for true RTT, Fig. 7.10 shows bootstrapped confidence intervals for each state. This data is also show in Table 7.7 With a few exceptions, bounds are fairly tight, indicating high confidence that the ranking listed in Table 7.5 is reliable. One of the notable exceptions is Wyoming, which has a much higher confidence interval than its neighbors in the "good" RTT region. One possible cause for this may be the fact that Wyoming only had one recursive server in it. However, this was also the case for other servers that did not exhibit the behavior that Wyoming did.

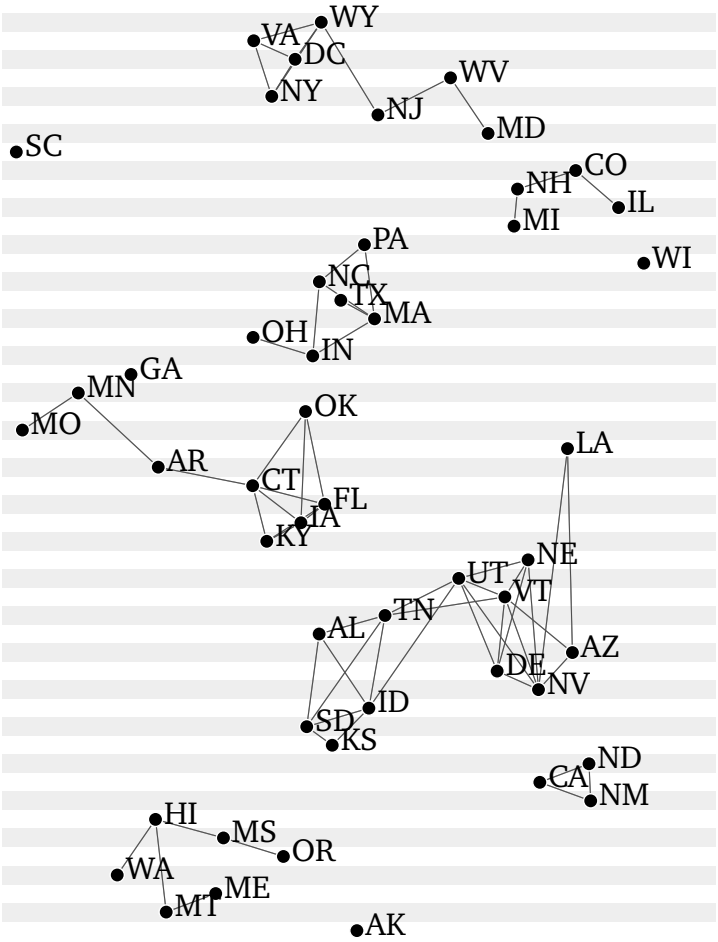


Figure 7.8: DNS true RTT indistinguishability graph

Similar to the graph for true RTT, Fig. 7.11 is a graph of states in the normalized RTT ranking that are indistinguishable. This graph includes a much larger and messier middle group consisting of 27 states. Hawaii, ranked #1 is an independent node, as is California at #4, with Alaska and Wyoming forming a pair between these two. At the bottom, Mississippi is distinctly alone. The second largest distinct subgroup consists of 10 states spanning from #5 to #15, with Utah distinct from all of them, but ranked #9.

Overall, Fig. 7.11 shows that while there are more completely distinct states in the normalized rankings, the states in the middle of the pack are dominated by a single large cluster.

Mapping the distinct sub graphs for normalized DNS RTT shows a majority of the large, mid ranking sub graph is on the East Coast and in the Midwest, with some states in the Mountain region. The West Coast is comprised entirely of states in groups 3 and 4, which also contain some states in the North East.

7.4.6 Aggregation by Authoritative State then Recursive State

One shortcoming of aggregating pairs directly by the location of the recursive server is that this inherently weights the data by the number of authoritative servers in a single state.

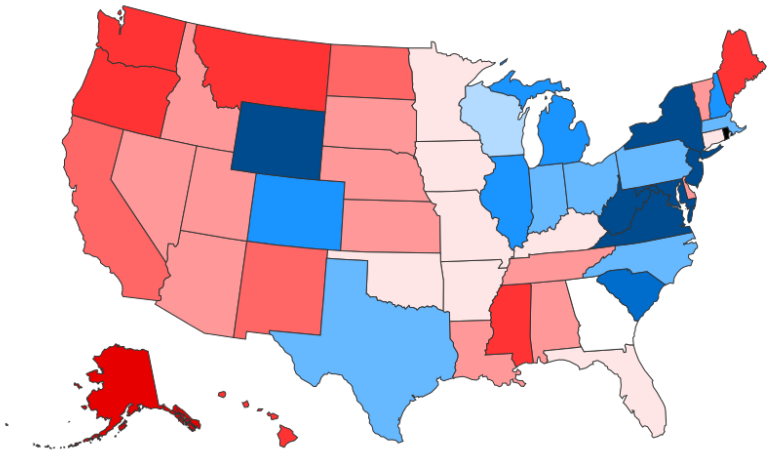


Figure 7.9: DNS true RTT state groupings

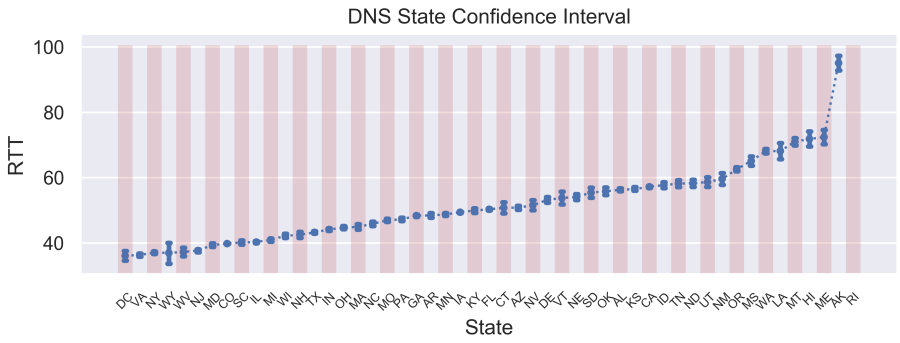


Figure 7.10: DNS true RTT confidence intervals

To compensate, this analysis approach first groups and aggregates by recursive server and authoritative server location.

Given the presence of authoritative servers in every state, this creates a list of 51 measurements for each state with a recursive server. For example, all measurements between a recursive server in California and an authoritative server in Massachusetts are aggregated into a single measurement (using median) for California. This is then done for every other state with measurements from California, giving California a 51 measurement dataset, and then repeated for each state with a recursive server.

To get a final value for a given state, each of these 51 measurements is averaged. Back to the example of California, this gives each state with measurements from California an equal weight in the final value for California.

However, prior to averaging the 51 measurements for each state, we had the option of applying weights to the values. Again with the California example, we could apply a weight to the aggregated measurement between it and Massachusetts based on Massachusetts' population, gross domestic product (GDP), or other metric. For example, Massachusetts has a high population than Wyoming, so with a population based weighting scheme, the measurement to Massachusetts is valued higher than the measurement to Wyoming.

Table 7.7: DNS state confidence intervals

State	Min	Mean	Max	State	Min	Mean	Max
DC	34.642	36.158	37.623	**CT**	49.122	50.823	52.518
VA	35.895	36.371	36.857	**AZ**	50.234	50.845	51.453
NY	36.709	37.096	37.467	**NV**	50.059	51.591	53.151
WY	33.712	37.217	40.108	**DE**	52.377	53.235	54.102
WV	36.010	37.287	38.587	**VT**	51.896	53.826	55.739
NJ	37.185	37.731	38.283	**NE**	53.199	54.082	54.971
MD	38.893	39.424	39.947	**SD**	53.899	55.469	57.001
CO	39.608	39.857	40.109	**OK**	54.724	55.873	57.010
SC	39.557	40.206	40.838	**AL**	55.873	56.326	56.781
IL	40.013	40.327	40.628	**KS**	55.873	56.531	57.184
MI	40.472	41.030	41.576	**CA**	56.905	57.239	57.566
WI	41.583	42.239	42.883	**ID**	56.847	57.748	58.656
NH	41.600	42.546	43.488	**TN**	57.146	58.225	59.291
TX	42.937	43.300	43.650	**ND**	57.222	58.318	59.386
IN	43.768	44.178	44.568	**UT**	57.256	58.678	60.129
OH	44.259	44.736	45.215	**NM**	57.846	59.624	61.423
MA	44.148	44.992	45.813	**OR**	62.018	62.600	63.187
NC	45.220	45.919	46.607	**MS**	63.705	65.195	66.525
MO	46.541	47.006	47.441	**WA**	67.500	68.164	68.820
PA	46.746	47.296	47.820	**LA**	65.691	68.208	70.634
GA	48.040	48.400	48.765	**MT**	69.931	71.042	72.139
AR	47.793	48.487	49.185	**HI**	69.600	71.892	74.191
MN	48.377	48.778	49.184	**ME**	70.274	72.446	74.624
IA	49.175	49.460	49.734	**AK**	92.841	95.128	97.335
KY	49.230	49.947	50.675	**RI**	–	–	–
FL	50.038	50.353	50.662				

We chose to proceed with unweighted and population weighted aggregations. Weighting by population gives higher value to connections with states that have greater populations. Essentially, this creates a proxy for how well connected people in one state are to all other people in the United States.

State Rankings

Table 7.8, Table 7.9, Table 7.10, and Table 7.11 show rankings based on this aggregation method, unweighted and population weighted, for both true DNS RTT and normalized DNS RTT. There are some shared patterns with the previous aggregation rankings: Wyoming ranks highly in all four, Alaska and Hawaii are at the bottom in both true RTT metrics but rise to top when looking at the normalized rankings. Under true RTT, all weighted RTT values are lower than the values at the same rank in the unweighted column. This indicates states with higher populations have generally lower values than their counterparts, driving down the weighted metrics.

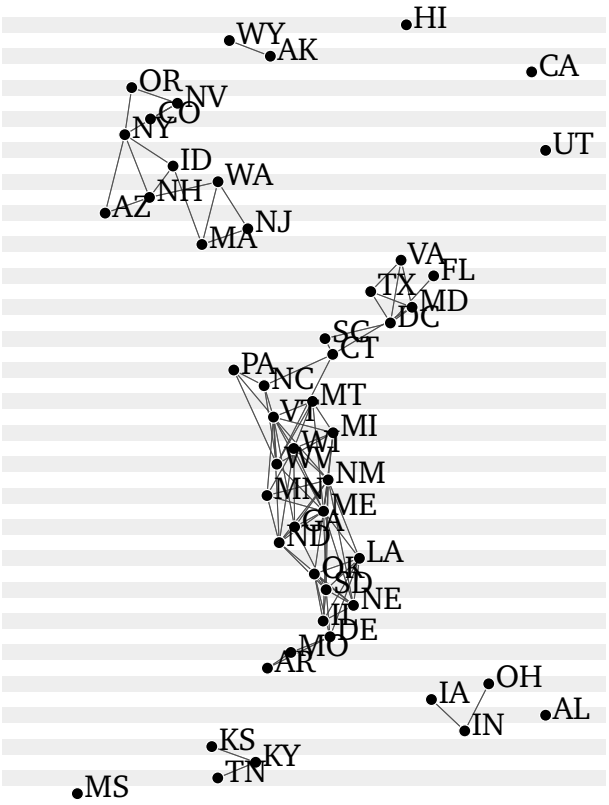


Figure 7.11: DNS normalized RTT indistinguishable states graph

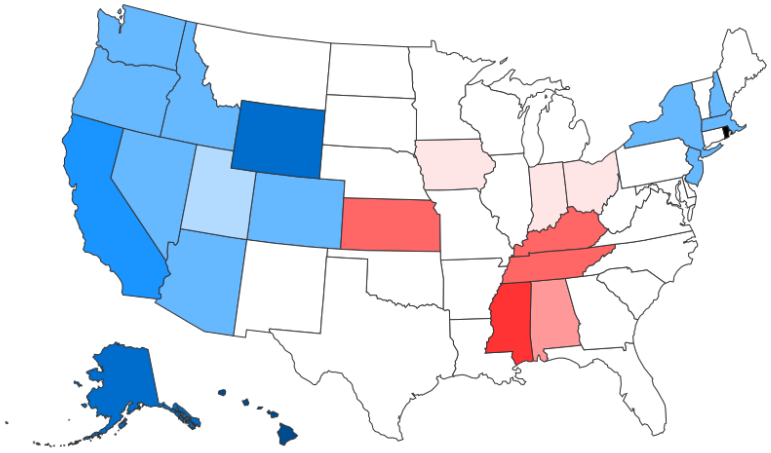


Figure 7.12: Normalized RTT indistinguishability graph

Evaluating State Rankings

As with the previous aggregation method, we began evaluating this approach by using the Kruskal-Wallis test. States were compared against each other using the 51 measurement dataset created by aggregating by the location of authoritative servers.

The results of this test indicate high unreliability for all four of these rankings. In each,

Table 7.8: DNS state rankings – with authoritative aggregation – true RTT – unweighted

Rank	State	(ms)	Rank	State	(ms)
1	VA	39.1	27	CT	52.7
2	NY	39.2	28	IA	53.2
3	WY	39.5	29	NE	53.5
4	NJ	40.0	30	VT	53.6
5	WV	40.3	31	FL	53.7
5	DC	40.3	32	ID	54.0
7	MD	40.7	33	AZ	54.3
8	CO	41.5	34	NV	55.4
9	SC	42.0	35	KY	56.0
10	MI	43.4	36	SD	58.7
11	IL	43.7	37	TN	59.3
12	WI	44.5	38	CA	60.1
13	PA	45.4	39	AL	60.7
14	NH	45.9	40	NM	60.9
14	NC	45.9	41	DE	61.4
16	TX	46.9	42	KS	61.8
17	OH	47.0	43	OR	63.2
18	UT	48.0	44	ND	64.6
19	MN	48.1	45	WA	67.3
20	IN	48.3	46	MS	67.4
21	MA	48.4	47	HI	69.7
22	AR	49.9	48	MT	71.9
23	GA	50.1	49	ME	77.6
24	OK	50.6	50	AK	100.4
25	MO	51.0	–	RI	–
26	LA	52.3			

the large subgraph of states that are indistinguishable from each other is comprised of 49 or 50 states, which means that that a standard ranking based on the aggregated value is too unreliable.

As a result, we continued by determining how many states each state is distinctly better than for each ranking. For example, if California is ranked above Wyoming and the p-value comparing the two is less than or equal to 0.05, California is considered to be distinctly better than Wyoming. On the other hand, if California is also ranked above Massachusetts but the p-value between the two is greater than 0.05, California does not gain credit for being distinctly higher than Massachusetts. States initially ranked above California would not be considered. Table 7.12, Table 7.13, Table 7.14, and Table 7.15 show the results of this analysis.

Perhaps most notable is that for the normalized RTT data weighted by population, over half of states were not distinctly better than a single other state. Overall, the unweighted methodologies fared better at distinguishing states at the top and bottom.

Table 7.9: DNS state rankings – with authoritative aggregation – true RTT – population weighted

Rank	State	(ms)	Rank	State	(ms)
1	WY	35.7	26	MO	48.7
2	VA	37.0	28	FL	49.1
3	NY	37.2	29	IA	51.1
4	MD	37.6	30	NE	51.2
5	SC	37.7	31	VT	52.6
6	DC	37.9	32	NV	52.9
7	NJ	38.5	33	KY	53.2
8	WV	38.9	34	AZ	53.4
9	CO	40.0	35	ID	53.7
10	IL	41.5	36	TN	54.7
11	WI	42.3	37	AL	56.2
12	MI	42.4	38	CA	56.9
12	NC	42.4	39	DE	57.1
14	TX	43.1	40	KS	57.9
15	PA	43.2	41	SD	58.3
16	NH	43.9	42	NM	58.8
17	OH	45.1	43	ND	61.7
18	GA	45.9	44	OR	63.4
19	MA	46.1	45	MS	63.9
20	MN	46.3	46	HI	65.3
20	IN	46.3	47	WA	66.8
22	OK	46.6	48	MT	70.7
23	AR	46.9	49	ME	75.5
24	UT	47.9	50	AK	99.1
24	LA	47.9	–	RI	–
26	CT	48.7			

And again, Wyoming ranked highly in all tables - its worst rank is #8 for unweighted normalized RTT. Alaska and Hawaii rank low in the true RTT metrics but are at the top for normalized.

Figure 7.13 and Fig. 7.14, show maps of the results in the preceding tables, where states with darker green coloring are distinctly better than more states than states that are colored grey. In Fig. 7.13, which show the true RTT unweighted and weighted maps, East Coast states, and to a lesser degree Midwestern states, perform prominently better, with Colorado and Wyoming as the primary exceptions. When weights are applied, the only states that remain clearly above the rest are a handful of East Coast states and Wyoming.

On the other hand, the normalized maps, Fig. 7.14 show Western states performing far better than the rest of the country, although some on the East Coast do fairly well. However, much of that performance is lost when weighted by population, where Western states are far more muted. One possible explanation for this is that while the normalized

Table 7.10: DNS state rankings – with authoritative aggregation – normalized RTT – unweighted

Rank	State	(ms)	Rank	State	(ms)
1	HI	104.5	27	LA	31.6
2	WY	77.3	28	MI	31.5
3	AK	46.4	29	NM	30.9
4	CA	45.8	30	GA	30.8
5	NV	45.0	31	MN	29.8
6	ID	44.3	31	WI	29.8
6	UT	44.3	33	OK	29.7
8	AZ	44.2	34	IL	29.2
9	OR	43.2	35	MT	29.0
10	CO	41.7	36	AR	28.1
11	NY	41.1	37	NE	27.9
12	WA	41.0	38	SD	27.7
13	DC	39.1	38	ME	27.7
14	NJ	38.8	40	OH	27.1
15	VA	38.0	41	ND	26.4
16	NH	37.8	41	MO	26.4
17	FL	37.7	43	AL	26.0
18	TX	37.6	44	IN	25.7
19	MD	37.5	45	DE	25.4
20	SC	36.3	46	IA	24.8
21	MA	35.9	47	KY	23.1
22	CT	34.4	48	MS	22.9
23	WV	33.6	49	TN	22.4
24	NC	33.0	49	KS	22.4
25	PA	32.2	–	RI	–
26	VT	31.9			

metric removes the distance barrier for the West, these states are also a great distance from the denser East Coast.

Overall, these maps show that there are few meaningful conclusions that we can draw from weighting states by population: when doing so, states are far too indistinguishable from each other. However, the unweighted maps show that even if we cannot draw conclusions about specific states when eliminating the implicit DNS weighting, we can show that, for true RTT, the East Coast and the Midwest tend to perform better as a region, as does the West Coast for normalized RTT.

Table 7.11: DNS state rankings – with authoritative aggregation – normalized RTT – population weighted

Rank	State	(ms)	Rank	State	(ms)
1	HI	106.5	26	MN	32.7
2	WY	80.1	28	MI	32.2
3	AK	48.9	29	WI	31.8
4	ID	43.7	30	LA	30.8
5	CO	43.3	30	NM	30.8
6	UT	43.0	32	MT	30.6
7	CA	42.0	33	OK	30.5
8	OR	41.8	34	IL	30.1
9	NY	41.6	35	NE	29.8
9	NV	41.6	35	GA	29.8
11	WA	40.6	37	SD	29.6
12	NH	40.4	38	ND	29.5
13	NJ	39.1	39	ME	29.4
13	AZ	39.1	40	AR	28.2
15	MA	37.9	41	OH	27.4
16	MD	37.7	42	DE	27.3
17	DC	37.6	43	MO	27.2
18	VA	37.0	44	IN	26.4
19	TX	36.8	45	IA	26.2
20	FL	36.3	46	AL	25.2
21	CT	36.2	47	KS	24.1
21	SC	36.2	48	KY	23.4
23	WV	32.9	49	TN	23.0
23	PA	32.9	50	MS	22.0
25	VT	32.8	–	RI	–
26	NC	32.7			

Table 7.12: DNS authoritative aggregation – Number of states better than – true RTT, unweighted

Rank	State	(ms)	Rank	State	(ms)
1	VA	40	26	CT	15
2	WY	39	26	FL	15
3	NY	36	29	IA	14
3	WV	36	29	NE	14
3	DC	36	31	VT	12
6	NJ	35	31	ID	12
6	MD	35	31	KY	12
8	SC	34	34	AZ	10
9	CO	32	35	TN	9
9	MI	32	36	NV	8
11	IL	31	36	SD	8
12	WI	30	36	AL	8
13	PA	28	39	KS	6
13	NC	28	39	DE	6
15	NH	26	41	NM	5
16	OH	23	42	CA	4
17	TX	22	43	ND	3
18	MN	21	43	MS	3
18	IN	21	45	OR	2
20	MA	20	46	WA	1
20	GA	20	46	HI	1
22	UT	19	46	MT	1
22	AR	19	46	ME	1
24	OK	18	50	AK	0
25	MO	17	–	RI	–
26	LA	15			

Table 7.13: DNS authoritative aggregation – number of states better than – true RTT, population weighted

Rank	State	(ms)	Rank	State	(ms)
1	WY	24	22	UT	2
1	NY	24	22	AR	2
3	VA	23	22	OK	2
3	DC	23	22	CT	2
5	MD	21	31	CA	1
6	NJ	18	31	OR	1
7	WV	16	31	ND	1
8	SC	13	31	NM	1
9	MI	9	31	SD	1
10	CO	8	31	KS	1
10	IL	8	31	DE	1
12	WI	7	31	TN	1
12	NH	7	31	ID	1
14	PA	6	31	AZ	1
14	NC	6	31	KY	1
16	OH	5	31	NV	1
17	TX	4	31	NE	1
18	GA	3	31	AL	1
18	MA	3	45	MT	0
18	MN	3	45	WA	0
18	IN	3	45	ME	0
22	FL	2	45	MS	0
22	VT	2	45	HI	0
22	IA	2	45	AK	0
22	MO	2	–	RI	–
22	LA	2			

Table 7.14: DNS authoritative aggregation – number of states better than – normalized RTT, unweighted

Rank	State	(ms)	Rank	State	(ms)
1	HI	49	27	WV	9
2	AK	42	28	GA	7
3	CA	38	29	NM	6
3	ID	38	29	MN	6
5	NV	36	29	WI	6
5	UT	36	32	MT	5
7	OR	31	32	OK	5
8	WY	30	32	VT	5
9	AZ	28	35	ND	4
9	CO	28	35	ME	4
9	NY	28	35	SD	4
9	WA	28	35	AR	4
13	TX	26	35	IL	4
13	FL	26	40	NE	3
13	NH	26	41	MO	1
16	NJ	25	42	KY	0
17	DC	24	42	MS	0
18	MD	22	42	IA	0
18	MA	22	42	TN	0
18	VA	22	42	IN	0
21	SC	20	42	AL	0
22	NC	12	42	OH	0
23	CT	11	42	DE	0
23	MI	11	42	KS	0
23	LA	11	–	RI	–
26	PA	10			

Table 7.15: DNS authoritative aggregation – number of states better than – normalized RTT, population Weighted

Rank	State	(ms)	Rank	State	(ms)
1	HI	49	25	KY	0
2	AK	28	25	IA	0
3	WY	20	25	ND	0
4	CA	18	25	IN	0
5	NV	15	25	MO	0
6	OR	12	25	DE	0
7	UT	11	25	OH	0
8	ID	10	25	AR	0
8	WA	10	25	ME	0
8	AZ	10	25	SD	0
11	CO	9	25	VT	0
11	NY	9	25	NE	0
11	FL	9	25	IL	0
11	TX	9	25	OK	0
15	DC	8	25	MT	0
16	VA	7	25	NM	0
16	NH	7	25	LA	0
18	NJ	6	25	WI	0
18	MD	6	25	MN	0
20	MA	4	25	TN	0
20	SC	4	25	PA	0
22	MI	1	25	CT	0
22	NC	1	25	GA	0
22	WV	1	25	MS	0
25	KS	0	–	RI	–
25	AL	0			

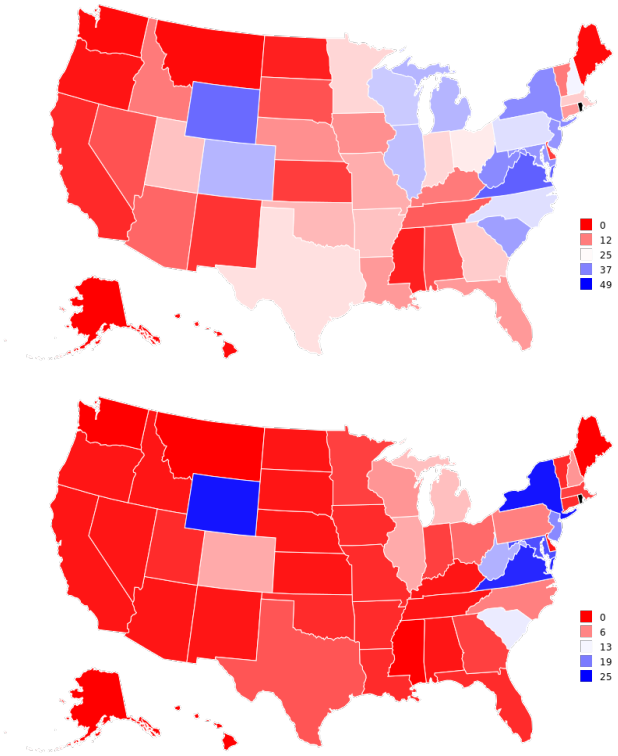


Figure 7.13: DNS true RTT unweighted and population weighted “better than” map

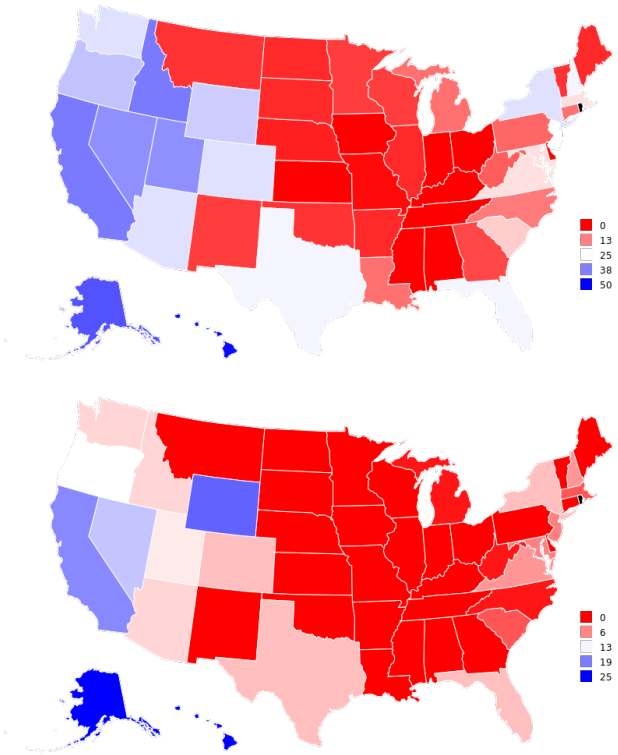


Figure 7.14: DNS normalized RTT unweighted and population weighted “better than” map

7.5 Summary

One of the general conclusions that we can draw from the DNS cache manipulation method is that western states fair better when the data is normalized by distance, while states on the East Coast are better with the true RTT metrics. This makes intuitive sense: the Eastern US has a higher population density than the west, as well as a higher density of DNS servers. By simple proximity it the East coast is more connected to more of the US. When you take out that factor, the West coast shines, indicating higher overall quality of DNS connectivity to other states in the United States.

8

Comparative Analyses

Since chapters Chapters 5 to 7 cover three distinctly different methods with different metrics and analyses for each, this chapter is dedicated to a comparative analysis of the results from each. Here, we aim to determine to what extent the different methods report similar results and present highlighted similarities and differences.

8.1 Data Distributions Across States

This section focuses on the differences in the distribution of measured Internet connectivity as a means of demonstrating how the different methods vary in their results. Building off the KDE charts from other chapters, we can filter to each state and plot the distributions of each of the methods in order to compare and contrast them.

Because the different methods use different metrics — speed-of-light efficiency for traceroutes, time to load data from the top 50 website for site ping, and RTT between DNS server for DNS cache manipulation — there has to be some re-scaling of an x axis to show a proper distribution. Traceroute data is reported as a fractionless scalar from 0-1, while site ping and DNS cache manipulation reports data in units of milliseconds; if graphed together without adjustment, one or the other would not even be visible. Consequently, all charts in this section are based on simple normalization by maximum value recorded across the US. The DNS and site ping x axes have been inverted for ease of understanding, so higher along the x axis should always be interpreted as better.

As there are 51 charts total only a few are listed here, intended to provide the best evidence for the different observations about the distributions:

Site ping is the most extreme Across all KDE charts for all graphs, data from traceroutes and DNS cache manipulation tend to agree with each other (although spread differs). However, site ping consistently reports values near the best possible value on these normalized axes, as demonstrated in Fig. 8.1. The peak for traceroute data (referred to as just CAIDA on the charts) is sharper than DNS but close to its median value. In comparison, the peak for site ping is incredibly sharp and narrow, and focused over the 0.95-1.0 range.

This should not be taken as an indicator that site ping reports consistently good quality, but that there are less common instances where it reports *exceptionally* poor. The fact that its distribution is centered over high values is an artifact of normalizing by maximum value (and inverting) – it means that somewhere to the far left, there's a small group of data points with poor connectivities, just not enough to be seen on the KDE charts. Keep in mind, this is *after* z-score filtering to eliminate outliers.

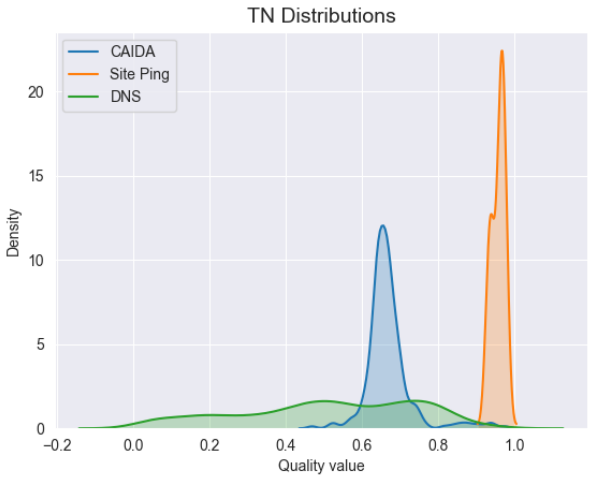


Figure 8.1: Tennessee data distributions

Put more concisely: the worst connectivity reported by site ping are dramatically worse than the worst of either DNS or CAIDA data, even though on average it reports good connectivity.

DNS data has higher spread than traceroute data In all charts, DNS cache manipulation data shows a much larger spread than traceroute data does. For instance, in Fig. 8.2 we see consistent centering around the same point (approximately 0.65-0.7) for *all* measures[1], with traceroute and site ping showing roughly the same spread, but with DNS much more spread out.

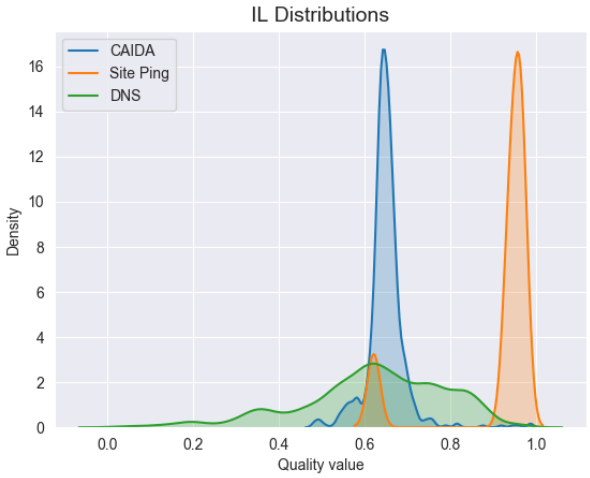


Figure 8.2: Illinois data distributions

There's a fairly simple explanation for this, having to do with the way data was collected. As described in Section 7.1.1 the DNS method involves taking several measurements and performing subtraction operations between them to determine a final value. Since each measurement has a margin of error (e.g. ± 5 ms), and error adds up with each addition or subtraction operation, it's only natural to see a wide distribution of data.

The downside to this, of course, is that aggregate measurements – like those displayed in the state distributions charts – are vulnerable to a wide spread. This makes it more statistically

[1]Interestingly we see a smaller peak for site ping data at that point too. This helps confirm the prior theory that there is a small subset of poor-connectivity site ping data points.

challenging to draw comparisons between states, for instance, as represented in various indistinguishability graphs across Chapters 5 to 7.

Site ping is more likely to be bimodal; traceroute is typically unimodal Although not all KDE charts show it as strongly, site ping distributions are far more likely to be strongly bimodal. In contrast, traceroute data is almost always unimodal, although with occasional ripples on the outer edges of the distribution. Figure 8.3 is a good example of this behavior.

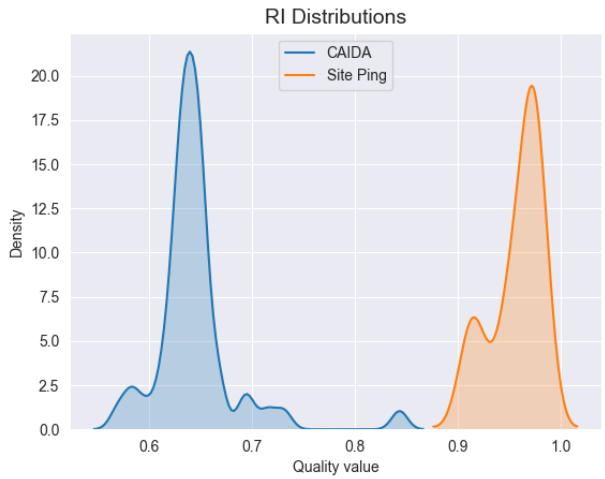


Figure 8.3: Rhode Island data distributions

The reasons for this are not entirely clear. Figure 5.2, the global KDE chart for all traceroute data in Section 5.3, shows an unusual, *very* weakly bimodal distribution, but not an especially spread-out one. It may be that the occasional leftward peaks seen in site ping data area again due to the hypothesized greater-extremes from earlier before, but it's difficult to quantify.

8.2 State Ranking Comparisons

Another way of examining the differences between methods is to conduct a test on the correlations between the state rankings produced by each method. Since all methods have different distinguishability graphs, with different valid and invalid comparisons, it's difficult to analyze the similarities between two sets of state rankings directly.

Logically, in any strict ordering, the number of elements that an element in a list is greater than will correspond directly to its position in the list. That is, if an element is #40 in a list of 50, it should have a "greater-than" count of 10. By counting all statistically valid comparisons where one state has better connectivity than another we can establish a statistically valid method of measuring the correlation between the different methods. That is, if a state is position #40 according to one method, it should be position #40 in the other two if the methods are perfectly correlated.

Table 8.1: States-better-than correlations between methods

	Traceroute	Site Ping	DNS	DNS (Normalized)
Traceroute	1.00	0.43	0.37	0.30
Site Ping	0.43	1.00	0.21	0.30
DNS	0.37	0.21	1.00	0.21
DNS (Normalized)	0.30	0.30	0.21	1.00

Table 8.1 shows a table of each method's correlation to every other method using this technique, including DNS with and without normalization (traceroute is always normalized, site ping never is). The highest correlation displayed is 0.43, between the traceroute and site ping methods, while the lowest is DNS to normalized DNS at 0.21. Regardless, all of these values are low, suggesting that all methods yield different rankings. Since they're not *too* low, however, it's reasonable to assume that certain parts of the rankings are similar, e.g. the tail ends of the rankings. Across all three methods, DC had the best connectivity and Montana connectivity scores the worst connectivity.

8.3 Analysis

After three distinctly different methods – some overlapping on connectivity metrics and some not – and comparative analyses to determine similarities and differences between the results, there are some broad conclusions that we can draw.

8.3.1 Areas with Superior Connectivity

Table 8.2: Top 3 states for each data source

Rank	Traceroutes	Site Ping	DNS
1	DC	DC	DC
2	DE	DE	VA
3	CA	CT	NY

Washington DC Across all three methods, the area with the best Internet connectivity by far is Washington DC. DC consistently ranks highest among the states[2], even with serious contenders such as the coastal states taken into consideration. This is shown in Table 8.2, which draws its rankings from Tables 5.2, 6.1 and 7.7.

As discussed in other chapters, this actually makes perfect sense. All heatmaps indicate that areas with higher populations, especially cities, have better Internet connectivity on average. Washington DC is a zone that is *only* a city, so it's effectively a single point of good Internet connectivity without rural areas to weigh it down in the rankings. It's also the seat of the US federal government, complete with the Pentagon just across a river, making the entire area one enormous governmental hub where massive information flow is a necessity. With this in mind, it's no wonder the city has the best Internet connectivity.

Regions of Dense Population Areas of increased population density almost universally have better Internet connectivity, which confirms expectations based on economic factors. ISPs are more likely to expand into and prioritize areas with more paying customers, so those areas receive better infrastructure, more upgrades, and so on, ultimately leading to better connectivity.

[2]Or more accurately *administrative areas* considering DC is not a state, but for our purposes it's acceptable to lump DC in with the states.

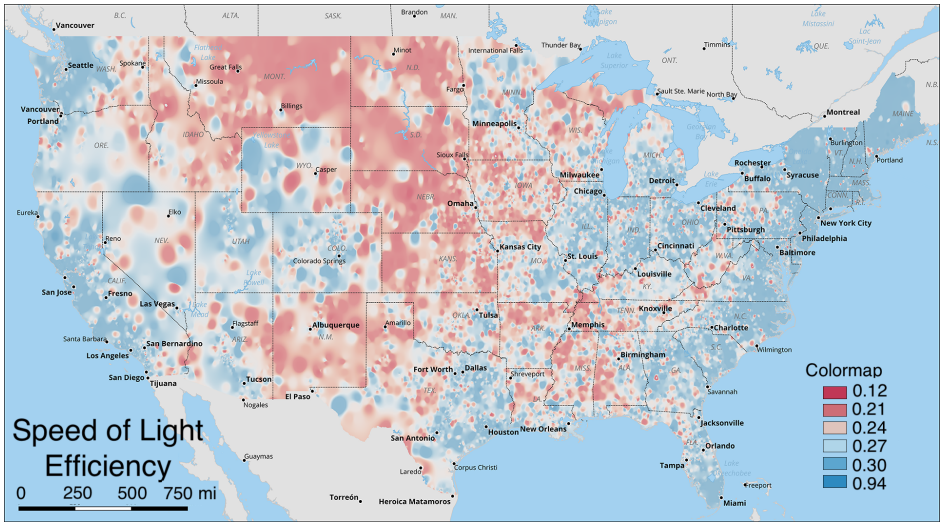


Figure 5.14: Traceroute speed-of-light efficiency heatmap (repeated from page 39)

Coastal Regions Broadly speaking, both the West coast and the East coast perform better than the central US according to all metrics. This falls in line with the prior conclusion about regions of dense population: the coasts are more densely populated, therefore they have better Internet connectivity.

Both of these effects are demonstrated in Fig. 5.14.

8.3.2 Areas with Poor Connectivity

The Deep South States part of the "deep south", in particular Alabama, Arkansas, Louisiana, Mississippi, and to a lesser extent Tennessee, universally have poor Internet. Although there are others with worse Internet, this region shows a strong unity in having poor connectivity. The reason for this is unknown, but we hypothesize it involves economic or political factors of the region.

The Central Northwest North states leaning central-North, such as Idaho, the Dakotas, and Montana have been observed to have poor connectivity across all three methods. Montana in particular is a region of egregiously poor Internet, consistently at or near the absolute worst connectivity by every metric.

Both of these effects are demonstrated in Fig. 6.10.

8.3.3 Anomalies

Wyoming Wyoming, the least populous state in the nation, is rather anomalous. Under CAIDA/Atlas, it ranks near the bottom, site ping data shows it as being upper/middle, while with DNS cache manipulation, it is consistently in or near the top five. While we are not certain, these discrepancies are likely do to the relative lack of data we have for state.

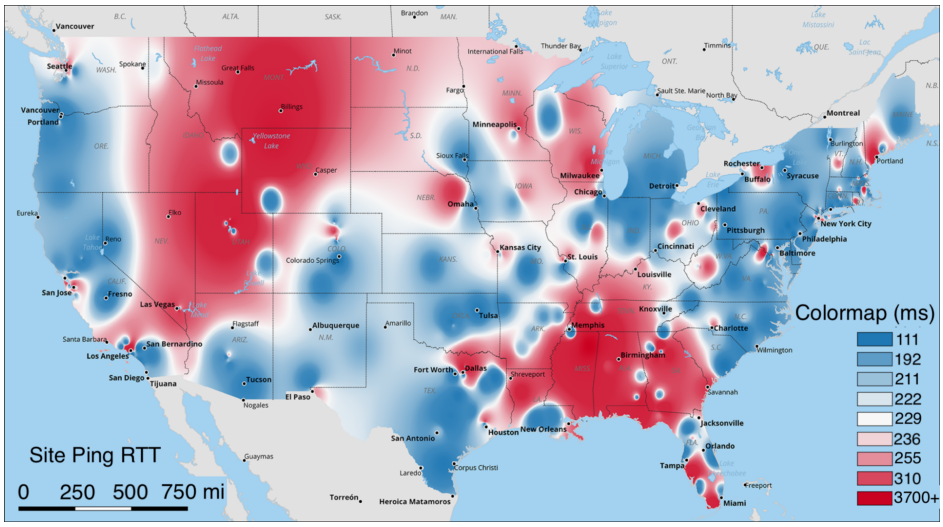


Figure 6.10: Site ping heatmap (repeated from page 53)

Alaska Alaska is an anomaly simply because we lack data for it. It turns out that gathering Internet connectivity data on Alaska is very difficult, even when offering high incentives for workers on Amazon's Mechanical Turk.

8.3.4 Rankings

Unfortunately, beyond the broad, region-based conclusions drawn here, devising a statistically-valid ranking of all states is challenging. There are cluster-based approaches that can be used where applicable, or topological sort methods that can give a rough estimate, but nothing concrete. We can confidently assert that select states like California or Washington DC are near the top, and states like Montana are near the bottom, but the non-extremes of the states are difficult to distinguish between.

8.4 Summary

The takeaway from this analysis has to be that, although there were definitely some similarities, the different methods yielded different results. Each of state KDE charts indicates differences on a state level while the per-method KDE charts in earlier chapters indicate differently shaped distributions. Although the DNS and traceroute methods seem to agree with each other more than either do with the site ping data, they're still different enough to make it impossible to call them the same.

In a way, this makes sense. Consider that all three methods collect fundamentally different kinds of data: traceroutes (as processed here) collect data related to infrastructure quality, site ping assesses web page loading delays, and DNS collects time for DNS queries to be resolved. At a pure surface level these have select elements in common (such as a bottlenecked connection from a user to their ISP will naturally degrade all three), but beyond that it's only logical that they should report different results.

9

Future Work

As we progressed with our data collection and analysis, we realized some ways in which our methods were flawed, or could be improved should another group continue this research in the future. This section outlines how we would approaching methods differently as well as other methods we think are worth pursuing in the future.

9.1 Improved Site Ping Data Collection

One potential issue with the way we selected the images to load for site ping is that we did not consider where the images were hosted. Most websites utilize a CDN to serve their content, and most of the image files used for the "ping" were, in fact, located on a CDN. The ping results are therefore more of a measure of the user's connection to that particular CDN than to the site as a whole. A revised Site Ping app could measure ping times for one object from each domain a site loads content from and take an average.

9.2 More Accurate IP Address Geolocation

One of the things that could improve the quality of the data collected is more accurate IP address geolocation. One way this could be done would be the use of more then one IP geolocation service and flag data points where the services do not agree as questionable. Another way geolocation could be improved would be through the use of up to date of a constantly updating database. IP addresses to move around, even across state lines, based on how the ISP decides to allocate them. Using a continuously updated database would allow for each data point to be logged exactly where it is.

9.3 More recursive and authoritative servers for each state

One of the issues with the DNS cache manipulation method is that it lacked a recursive DNS server in Rhode Island. Additionally, several states had only a single server of a given

type. Therefore, future iterations of this method could expand the geographic diversity of servers under test.

9.4 Backbone Analysis

One of the methods discussed in Section 4.2 is "backbone analysis". To briefly summarize, the idea was to to examine every indirectly calculated hop (Section 5.3) from the CAIDA and RIPE atlas dataset and consider those as edges on a graph, where the endpoints of the hop are vertices on the graph. A *massive* graph could be established in this way, giving rise to numerous possibilities for analysis based in graph theory.

Using graph theory methods it would be possible to identify nodes that are part of the Internet backbone, and measure connectivity to *those* instead. This provides a far more centralized region of the Internet to measure connectivity to, which may improve the quality of the analyses if implemented. It would also give insight on what fraction of traffic passes through the backbone, which could be another interesting metric for connectivity.

9.5 Surveys & Subjective User Experience

Although described in Chapter 2, we ultimately did not pursue methods of collecting data on consumer-oriented statistics like cost, advertised speeds, data caps, or connection stability. These are harder to gather data on using purely technical means – there is no series of servers we can query like in the DNS method, and there are no databanks of this sort of information available for scraping. To gather real data on the subject it would be necessary to communicate with actual users instead of just their browsers.

This implies a survey of some kind, asking users to provide quantitative information like how much they pay for Internet, how fast their Internet seems to be, or what their data cap is. Although more challenging to analyze, gathering data on qualitative measures like apparent connection stability may also be useful, although less reliable. These are methods that we did not consider at the start of our project, but that future researchers may be interested in pursuing.

9.6 IPv6 Availability

As mentioned in Chapter 3 it might be useful to measure the availability of IPv6 by region, along with mitigating measures like IPv6-over-IPv4 tunnels. IPv4 address exhaustion is becoming more and more urgent, with the RIPE Network Coordination Center recording exhaustion of their pool in Europe on 25 November 2019 [29]. As difficulty of obtaining an IPv4 address increases and network infrastructure is strained further, future researchers will likely want to examine IPv6 availability as a metric for whether Internet connections will even be possible in a region.

9.7 Summary

While the methods we explored and the analysis we performed were highly informative, there is still much room for future work. A number of key points in our methods could use improved accuracy or more data points, while some other connectivity metrics and analysis techniques could be applied to the data collected during this project. We also rehashed the potential usefulness of collecting data for metrics that we chose not to pursue, such as IPv6 availability.

10

Conclusion

Our research at the beginning of this project demonstrated that there is no single definition of connectivity. Some metrics, such as aggregation by /24 prefixes, may focus on network infrastructure, others on measured user experience, and yet others on connectivity advertised to users. And these metrics can further focus on types of connectivity: the efficiency of network in an area can show where improvements can be made, while the absolute performance indicates which locations are most connected in the physical world.

In total we pursued three different definitions of connectivity using three different methods: using traceroute data from CAIDA and RIPE Atlas, we gathered data about RTT from everywhere to everywhere; with a crowd sourcing website, we collected data about RTTs to the top websites in the United States; and leveraging DNS caching, we measured aggregate RTT between geographic regions. Each of these produced different results ranging from surprising to expected. For example, Washington DC and areas with regions of dense population generally performed better across all metrics. This conforms to expectations: ISPs are likely to prioritize better service for regions with more paying customers. Meanwhile, the deep south did not perform as well, possibly due to economic or political factors. One anomaly that stood out is Wyoming: the least populated state performed poorly in CAIDA/Atlas, middle to upper range using the crowd sourced site ping method, and consistently in or near the top five for the DNS cache manipulation.

While we conducted extensive research into Internet connectivity in the United States, the nature of this project is that there will always be more to do. As IPv6 becomes increasingly necessary, the availability of networks capable of handling the protocol could shed light on a region's future connectivity. Beyond that, backbone analysis would take the infrastructure based metrics further in exploring connectivity to the core components of the Internet. On the other hand, surveying users and exploring subjective user experience would delve deeper into the day-to-day experiences of everyday Internet users. And of course, our methods could always be improved on. Taking the location of the resources being served into account would improve the site ping method, while DNS cache manipulation could be made better by increasing the number of DNS servers being used.

In the end, we conducted in-depth analysis into three different definitions of Internet connectivity and, as expected, developed at least three different evaluations. No two definitions of connectivity will every agree completely and this report serves as evidence to

that. While we might be able to make some generalizations, everywhere has its strengths and weaknesses (some more than others) and at the end of the day, we leave it up to the reader of this report to determine which is most applicable to their problem.

A

Github Repositories

Table A.1 is a list of repositories created for this MQP, available at: https://github.com/Internet-Connectivity-MQP-2019.

Name	Purpose
data	Collection of datasets for this project.
DigForPy	Python wrapper for the `dig` utility. Used by DNS_Testing and Website Resolution.
DNS_Analytics	Scripts used to analyze the DNS cache manipulation data.
DNS_Testing	Scripts for collecting DNS cache manipulation data.
fp	Precursor project for SitePing. Created for CS4241 Webware.
JavaScript-Ping-Prototype	Proof of concept code for the Site Ping methodology.
MTurkValidator	Python script used to validate Mechanical Turk submissions.
SitePing	Collection site for the Site Ping method.
SitePingDataAnalysis	Data analysis scripts for the Site Ping data.
small-file-finder	Chrome extension to find small objects for use in the Site ping method.
tracerouter	Proof of concept code for constantly running traceroutes on a phone for the road trip concept.
traceroute-hopper-cpp	Traceroute hopper, ported from Python to C++ for better performance.
traceroute-processing-scripts	Traceroute analysis & processing scripts for CAIDA, RIPE Atlas, and more.
WebsiteResolution	Tool for using the list of recursive DNS servers to determine where websites resolve to from across the country.

Table A.1: List of Github repositories

B

Select Database Design

Some database optimization schemes were used to store CAIDA and RIPE Atlas traceroute/ping data in a performance-friendly way. These are reproduced below for possible future use and to clarify on the database design. The below shows the table, partition, and index definitions for our raw data tables for CAIDA and RIPE Atlas data. Note that it was somewhat naively assumed that IPv6 addresses would represent a relatively small portion of the dataset.

```
CREATE TABLE hops (
    src         INET,
    dst         INET,
    rtt         REAL
);
CREATE INDEX src_index ON he USING HASH(src);
CREATE INDEX dst_src_BRIN_index ON hops USING brin(dst, src);

CREATE TABLE h0 PARTITION OF hops FOR VALUES FROM ('0.0.0.0') TO ('16.255.255.255');
CREATE TABLE h1 PARTITION OF hops FOR VALUES FROM ('16.255.255.255') TO
 ↪  ('32.255.255.255');
CREATE TABLE h2 PARTITION OF hops FOR VALUES FROM ('32.255.255.255') TO
 ↪  ('48.255.255.255');
CREATE TABLE h3 PARTITION OF hops FOR VALUES FROM ('48.255.255.255') TO
 ↪  ('64.255.255.255');
CREATE TABLE h4 PARTITION OF hops FOR VALUES FROM ('64.255.255.255') TO
 ↪  ('80.255.255.255');
CREATE TABLE h5 PARTITION OF hops FOR VALUES FROM ('80.255.255.255') TO
 ↪  ('96.255.255.255');
CREATE TABLE h6 PARTITION OF hops FOR VALUES FROM ('96.255.255.255') TO
 ↪  ('112.255.255.255');
CREATE TABLE h7 PARTITION OF hops FOR VALUES FROM ('112.255.255.255') TO
 ↪  ('128.255.255.255');
CREATE TABLE h8 PARTITION OF hops FOR VALUES FROM ('128.255.255.255') TO
 ↪  ('144.255.255.255');
CREATE TABLE h9 PARTITION OF hops FOR VALUES FROM ('144.255.255.255') TO
 ↪  ('160.255.255.255');
CREATE TABLE hA PARTITION OF hops FOR VALUES FROM ('160.255.255.255') TO
 ↪  ('176.255.255.255');
```

```
20  CREATE TABLE hB PARTITION OF hops FOR VALUES FROM ('176.255.255.255') TO
    ↪  ('192.255.255.255');
21  CREATE TABLE hC PARTITION OF hops FOR VALUES FROM ('192.255.255.255') TO
    ↪  ('208.255.255.255');
22  CREATE TABLE hD PARTITION OF hops FOR VALUES FROM ('208.255.255.255') TO
    ↪  ('224.255.255.255');
23  CREATE TABLE hE PARTITION OF hops FOR VALUES FROM ('224.255.255.255') TO
    ↪  ('240.255.255.255');
24  CREATE TABLE hF PARTITION OF hops FOR VALUES FROM ('240.255.255.255') TO
    ↪  ('255.255.255.255');
25  CREATE TABLE hZ PARTITION OF hops FOR VALUES FROM ('::') TO
    ↪  ('FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF');
```

C

List of Sites and Web Resources

The below list identifies the sites in the US that were used for the site ping website:

Site	Rank	Resource
Google	1	Resource
Youtube	2	Resource
Amazon	3	Resource
Yahoo	4	Resource
Facebook	5	Resource
Reddit	6	Resource
Wikipedia	7	Resource
Ebay	8	Resource
Office	9	Resource
Bing	10	Resource
Netflix	11	Resource
ESPN	12	Resource
Salesforce	13	Resource
Live	14	Resource
Instructure	15	Resource
Chase	16	Resource
Apple	17	Resource
Instagram	18	Resource
CNN	20	Resource
Dropbox	21	Resource
Tmall	22	Resource
LinkedIn	23	Resource
Twitter	24	Resource
Twitch	25	Resource
Microsoft	27	Resource
Shopify	28	Resource
NYTimes	29	Resource
Walmart	31	Resource
Pornhub	32	Resource
Adobe	33	Resource
IMDb	35	Resource
Stack Overflow	36	Resource

Continued on next page

Site	Rank	Resource
AWS	37	Resource
Sohu	38	Resource
QQ	39	Resource
Indeed	40	Resource
Zillow	41	Resource
Wellsfargo	42	Resource
Spotify	43	Resource
MSN	44	Resource
Imgur	45	Resource
Yelp	47	Resource
Taobao	48	Resource
Etsy	49	Resource
Hulu	50	Resource

Acronyms

Notation	Description	Page List
ANOVA	analysis of variance	19, *Glossary:* ANOVA
API	application programming interface	8, 45, 46
AWS	Amazon Web Services	11, 47, 48
CAIDA	Center for Applied Internet Data Analysis	15, 16, 23–26, 30, 32, 33, 35, 37, 39, 43, 84, 85, 91, 93, 96
CDF	cumulative distribution function	30, 51
CDN	content delivery network	3, 6–8, 11, 52, 53, 90, *Glossary:* CDN
CLI	command-line interface	25, 26, 28
CORS	cross-origin resource sharing	44
CSV	comma separated value	59
CV	coefficient of variation	33, 50, 56, 60, 65, *Glossary:* CV
DDoS	distributed denial of service	18
DNS	domain name system	i, 6–8, 15, 16, 26, 53, 55–63, 65, 67, 70, 72, 76, 83–85, 87–91, 93, 95
EC2	Amazon Elastic Compute Cloud	47, *Glossary:* EC2
ETL	extract-transform-load	25, 28, *Glossary:* ETL
FCC	Federal Communications Commission	1, 18
GDP	gross domestic product	71
GIS	geographic information system	30
GPS	global positioning system	16
HTML	Hypertext Markup Language	44, 46
HTTP	hypertext transfer protocol	6, 44, 45
HTTPS	hypertext transfer protocol secure	44
ICMP	internet control message protocol	4, 53, *Glossary:* ICMP
IDW	inverse distance weighting	38, 39, 42, 43
IP	internet protocol	3, 4, 6–9, 11, 17–19, 22, 23, 26–33, 37, 56, 57, 59, 60, 90
IPoAC	internet protocol over avian carrier	34
IPv4	internet protocol version 4	11, 13, 91

Notation	Description	Page List
IPv6	internet protocol version 6	13, 14, 91–93
IQR	inter-quartile range	47, 49
ISP	internet service provider	3, 5, 7, 8, 12, 13, 32, 87, 89, 90, 93, *Glossary:* ISP
JSON	Javascript Object Notation	24–26, 46, 48, 101
JWT	JSON Web Token	48
KDE	kernel density estimation	30, 41, 84–86, 89, *Glossary:* KDE
Mbps	megabits per second	3, 5
MQP	major qualifying project	8, 9, 16, 22, 55
NTP	Network Time Protocol	16, 17
RIPE	Réseaux IP Européens	15, 16, 23–26, 30, 32, 33, 35, 37, 39, 43, 91, 93, 96
RTT	round trip time	3, 4, 10–16, 23, 24, 26, 28, 30, 31, 33, 34, 36, 44–46, 53, 55, 57, 61, 64–70, 72, 74–76, 83, 84, 93
SDK	software development kit	8
SVG	scalable vector graphics	46
TCP	transmission control protocol	44–46
TLD	top level domain	6, 7
TLS	transport layer security	44, 45
TTL	time-to-live	4, 6, 7
URL	uniform resource locator	7
US	United States	v, 1, 3, 8, 9, 11–13, 15–19, 22, 35, 42, 45–49, 83, 84, 87, 88, 98
USPS	United States Postal Service	19
VoIP	voice over Internet protocol	10
WPI	Worcester Polytechnic Institute	8, 9, 22, 48, 56, 57

Glossary

ANOVA Analysis of variance is a class of statistical models and methods for estimation, used to analyze the difference between the means of a sample. Although there are many types, they all fundamentally calculate the probability (a p value) that two population means are equal.

CDN A content distribution network, sometimes called a Content Delivery Network, is a network of proxy servers that form a kind of cache used to enhance delivery of content to Internet users. Although helpful for Internet users, they complicate measurements of connectivity to websites *actually* connecting to the site's servers.

CV Coefficients of variation, or relative standard deviations, are defined as the ratio of the absolute value of the mean of a variable divided by its standard deviation: $\frac{|\mu|}{\sigma}$. CVs are dimensionless values that can be judged independent of the original source, making them useful for gauging the spread of any data set. The lower the CV, the lower the spread of the data and the better the quality.

EC2 EC2 is a service from Amazon Web Services that provides virtual machines in the cloud for general-purpose or task optimized work. EC2 can be configured for different performance and pricing classes, as well as complex auto-scaling schemes or virtual private cloud setups.

ETL Extract-transform-load is a generic procedure for extracting data, transforming it into a more useful format, and loading it into a large volume storage system, such as a database. The term closer describes an architecture rather than a specific algorithm.

ICMP Internet Control Message Protocol is a protocol designed for error reporting and other utility purposes across the Internet, typically used most by routers and other intermediary devices.

ISP An Internet service provider is typically the "last mile" organization that provides a user with Internet access — otherwise known as a *tier 3* ISP. They are distinct from tier 2 and tier 1 ISPs which are responsible for much of the Internet backbone, although ISP corporations may operate on multiple tiers. Common ISPs in the US include AT&T, Comcast, and Verizon.

KDE Kernel density estimation is a technique for estimating the probability density function of a variable. Briefly, KDEs work by processing each measurement of a variable as

if it was at the center of some given probability density function, e.g. a gaussian curve. These curves are then summed together to form one curve and normalized so the area underneath the curve is equal to 1. A KDE chart can be read in the same way as a histogram can, but the y axis corresponds to a density instead of an absolute value. The advantage to using a KDE over a histogram is that KDEs are not vulnerable to binning effects (from choosing the wrong bin size) while histograms are.

Bibliography

[1] Bergen Linux User Group, "The informal report from the RFC 1149 event", Bergen, Norway, Tech. Rep., 2001. [Online]. Available: https://www.blug.linux.no/rfc1149/writeup/.

[2] M. Bostock, V. Ogievetsky, and J. Heer, "D3: Data-Driven Documents", *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 12, pp. 2301–2309, Dec. 2011, ISSN: 1077-2626. DOI: 10.1109/TVCG.2011.185. [Online]. Available: http://ieeexplore.ieee.org/document/6064996/.

[3] Cloudflare, *What Is DNS?*, 2020. [Online]. Available: https://www.cloudflare.com/learning/dns/what-is-dns/.

[4] ——, *What Is Recursive DNS?* [Online]. Available: https://www.cloudflare.com/learning/dns/what-is-recursive-dns/.

[5] R. Durairajan, P. Barford, J. Sommers, and W. Willinger, "InterTubes: A Study of the US Long-haul Fiber-optic Infrastructure", in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication - SIGCOMM '15*, New York, New York, USA: ACM Press, 2015, pp. 565–578, ISBN: 9781450335423. DOI: 10.1145/2785956.2787499. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2785956.2787499.

[6] J. Fakult and K. Levasseur, "The Internet Connected Project", Worcester Polytechnic Institute, Worcester, Massachussetts, Tech. Rep., 2019, pp. 1–40.

[7] Federal Communications Commission, *Fixed Broadband Deployment Map*. [Online]. Available: https://broadbandmap.fcc.gov.

[8] R. T. Fielding, M. Nottingham, and J. Reschke, *Hypertext Transfer Protocol (HTTP/1.1): Caching*, R. Fielding, M. Nottingham, and J. Reschke, Eds., RFC 7234, Jun. 2014. DOI: 10.17487/rfc7234. [Online]. Available: https://rfc-editor.org/rfc/rfc7234.txt%20https://www.rfc-editor.org/info/rfc7234.

[9] J. Frederic, *PingJS*, 2015. [Online]. Available: https://github.com/jdfreder/pingjs.

[10] Google, *Google Analytics Tracking Code Overview*. [Online]. Available: https://developers.google.com/analytics/resources/concepts/gaConceptsTrackingOverview.

[11] ——, *Introduction to Google Public DNS*. [Online]. Available: https://developers.google.com/speed/public-dns/docs/intro.

[12] ——, *IPv6 Statistics*, 2019. [Online]. Available: https://www.google.com/intl/en/ipv6/statistics.html#tab=per-country-ipv6-adoption.

[13] I. Grigorik, *HTTP Caching*, 2019. [Online]. Available: https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/http-caching.

[14] B. Haberman, *Control Messages Protocol for Use with Network Time Protocol Version 4*, RFC Historic, 2019. [Online]. Available: https://tools.ietf.org/html/draft-ietf-ntp-mode-6-cmds-07.

[15] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring Network Structure, Dynamics, and Function using NetworkX", in *Proceedings of the 7th Python in Science Conference*, G. Varoquaux, T. Vaught, and J. Millman, Eds., Pasadena, CA USA, 2008, pp. 11–15.

[16] J. D. Hunter, "Matplotlib: A 2D Graphics Environment", *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007, ISSN: 1521-9615. DOI: 10.1109/MCSE.2007.55. [Online]. Available: http://ieeexplore.ieee.org/document/4160265/.

[17] W. H. Kruskal and W. A. Wallis, "Use of Ranks in One-Criterion Variance Analysis", *Journal of the American Statistical Association*, vol. 47, no. 260, pp. 583–621, Dec. 1952, ISSN: 0162-1459. DOI: 10.1080/01621459.1952.10483441. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/01621459.1952.10483441.

[18] A. Malhotra, *Love thy k – nearest neighbors*, 2017. [Online]. Available: https://towardsdatascience.com/love-thy-k-nearest-neighbors-58932fa66a08.

[19] MaxMind, *GeoIP2 City Accuracy*, 2020. [Online]. Available: https://www.maxmind.com/en/geoip2-city-accuracy-comparison?country=United+States&resolution=50.

[20] W. McKinney, "Data Structures for Statistical Computing in Python", in *Proceedings of the 9th Python in Science Conference*, S. van der Walt and J. Millman, Eds., Austin, Texas, 2010, p. 51.

[21] D. Mills, *Network Time Protocol (Version 3) Specification, Implementation and Analysis*, RFC 1305, Mar. 1992. DOI: 10.17487/rfc1305. [Online]. Available: https://tools.ietf.org/html/rfc1305%20https://www.rfc-editor.org/info/rfc1305.

[22] D. Mills, J. Burbank, and W. Kasch, *Network Time Protocol Version 4: Protocol and Algorithms Specification*, J. Martin, Ed., RFC 5905, Jun. 2010. DOI: 10.17487/rfc5905. [Online]. Available: https://tools.ietf.org/html/rfc5905%20https://www.rfc-editor.org/info/rfc5905.

[23] P. Mockapetris, *Domain names - implementation and specification*, RFC 791, Nov. 1987. DOI: 10.17487/rfc1035. [Online]. Available: https://www.rfc-editor.org/rfc/rfc1035.txt%20https://www.rfc-editor.org/info/rfc1035.

[24] MongoDB, *MongoDB*, 2019. [Online]. Available: www.mongodb.com.

[25] Mozilla Foundation, *HTTP Caching*, 2019. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Caching.

[26] OpenJS Foundation, *Node.js*, 2019. [Online]. Available: nodejs.org.

[27] Oracle, *Recursive DNS: How Does It Work?*, 2010. [Online]. Available: https://dyn.com/blog/recursive-dns-how-does-it-work/.

[28] J. Postel, *Internet Protocol*, RFC 791, Sep. 1981. DOI: 10.17487/rfc0791. [Online]. Available: https://rfc-editor.org/rfc/rfc791.txt%20https://www.rfc-editor.org/info/rfc0791.

[29] Reseaux IP Européens Network Coordination Centre, *What is IPv4 Run Out?*, 2019. [Online]. Available: https://www.ripe.net/manage-ips-and-asns/ipv4/ipv4-run-out.

[30] D. Shepardson, *Internet sector contributes $2.1 trillion to U.S. economy: industry group*, London, UK, Sep. 2019. [Online]. Available: https://www.reuters.com/article/us-usa-internet-economy/internet-sector-contributes-2-1-trillion-to-u-s-economy-industry-group-idUSKBN1WB2QB.

[31] *Speedtest.net and Comcast*, 2012. [Online]. Available: http://truxtertech.com/speedtest-net-and-comcast/.

[32] A. Tachalova, *This Is What They Search For: The Most Popular US Industries & Traffic Shares*, 2017. [Online]. Available: https://moz.com/blog/most-popular-us-industries-traffic-shares.

[33] O. Tange, *GNU Parallel*, Fredericksberg, Denmark, 2011. DOI: 10.5281/zenodo.16303. [Online]. Available: https://www.gnu.org/software/parallel/.

[34] Tencent, *RapidJSON*, 2016. [Online]. Available: https://rapidjson.org/.

[35] The Internet Society, "Consolidation in the Internet Economy", Reston, VA, Tech. Rep., 2019, pp. 1–75.

[36] The Shadowserver Foundation, *NTP Version (Mode 6) Scanning Project*. [Online]. Available: https://scan.shadowserver.org/ntpversion/.

[37] US Department of Homeland Security, *NTP Amplification Attacks Using CVE-2013-5211*, 2014. [Online]. Available: https://www.us-cert.gov/ncas/alerts/TA14-013A.

[38] J. Vermeulen, *LibPQXX*, 2019. [Online]. Available: https://github.com/jtv/libpqxx.

[39] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, . Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, and P. van Mulbregt, "SciPy 1.0: fundamental algorithms for scientific computing in Python", *Nature Methods*, Feb. 2020, ISSN: 1548-7091. DOI: 10.1038/s41592-019-0686-2. [Online]. Available: http://www.nature.com/articles/s41592-019-0686-2.

[40] D. Waitzman, *Standard for the transmission of IP datagrams on avian carriers*, RFC 1149, Apr. 1990. DOI: 10.17487/rfc1149. [Online]. Available: https://www.rfc-editor.org/info/rfc1149.

[41] M. Waskom, *Seaborn*, Sep. 2020. DOI: 10.5281/zenodo.883859. [Online]. Available: https://github.com/mwaskom/seaborn.