MQP-CEW-1903

The Internet Connected Project

Submitted on January 22, 2019

Submitted by:

Jacob Fakult

Kenneth Levasseur

Advised by:

Dr. Craig Wills

Abstract

As the Internet becomes more crucial to the world's economy, determining relative connectivity to a given location becomes more important. Associating a connectivity metric with a physical location can be of use to organizations looking to optimize their Internet traffic. Dynamic routing protocols and ambiguous mappings between an IP address and a real host make the task non-trivial. The Internet connected project investigates the effectiveness of DNS cache manipulation and traceroute as a means of mapping point-to-point Internet connectivity from specific geographic locations in the United States.

1 - Introduction

The Internet Connected project aims to determine a metric of network connectivity between a geographically diverse set of hosts within the United States. These data would have a broad range of uses from businesses looking to host networked services in optimal locations, to ISPs looking to capitalize on building infrastructure in under-connected regions. Other organizations have attempted to produce the same data via placement of remotely managed hosts at known physical locations whereas we focus on measurements from a single physical location (Worcester, MA). Thus we must rely on hosts and protocols that allow us to exert as much control as is practical over the endpoints. Selection of endpoints is important; we should look to measure traffic to hosts that represent locations that real Internet traffic would follow for a given region. Our definition of connectivity is constrained to the relative latency between other regions in the U.S., and not related to bandwidth. Several approaches were evaluated to this end, with DNS (Domain Name System) cache manipulation and traceroute mapping found most promising. The approaches were implemented separately, and the results show the effectiveness of each approach. The idea for this project was expanded from a previous study called the Geoconnected project]. The original project measured expected travel times from one county to another based on public roads, transit and air travel [22]. The inspiration for both projects draw on the fundamental importance of understanding how we are connected in a spread out world.

The remainder of the report is organized as follows. In Chapter 2, the background, we discuss the technical concepts needed to understand the project's implementation as well as alternate paths we explored in order to accomplish the project's goal. In Chapter 3 we discuss the approaches that we selected and implemented for this project. In Chapters 4 and 5 we discuss the implementation of the DNS Cache Manipulation and Traceroute approaches respectively, and provide rationalization and discussion for each design choice. Chapter 6 compares the results of the two methods. Finally, Chapters 7 provides conclusions and future work.

2- Background

The Internet is a complex system composed of thousands of organizations with their own autonomy that must be connected in a standardized manner. Likewise, there are thousands of protocols that nodes in a network may use, whether it is connecting to to a device next door or streaming from a service across the world. In attempting to gather accurate metrics of latency from Internet hosts, and subsequently extract geographic information about the capabilities of the Internet, it is important to understand how traffic is routed and what causes latency.

2.1 - Internet Protocols

Border Gateway Protocol (BGP) is a routing protocol widely used on the Internet that allows large groups of routers to act as a single autonomous system, and for organizations to preferentially advertise routes or accept routes into these systems [15]. The Border Gateway Protocol will thus control the path that each packet travels from source to destination. The routers can preference network related metrics, i.e. least number of hops, or they can factor in geographical, monetary, or political metrics. From a single point of origin on a single Internet Service Provider (ISP), preferences are transparent; there is no way to tell what routes a remote host receives on the reverse path to the origin. Preferences are in constant flux as organizations respond to outages or sub-optimal latency. Routes can also change when ISPs respond to “route leaks” [17], or the advertisement of illegitimate routes not present in the autonomous system; BGP itself has no validation for the existence of an advertised route. Organizations responding to outages, sub-optimal latency, or route leaking creates variation in any latency data collected at two different points in time.

When considering tools that can be used to monitor latency, it is important to evaluate the protocols each tool relies on, and what considerations must be made when treating the values they produce as accurate. Transmission Control Protocol (TCP) makes up the majority of Internet traffic [1], however to evaluate its latency, a remote server must be configured to respond via the protocol. TCP is connection-oriented and capable of recovering from packet loss by resending data. Latency measuring tools must take such retransmissions into account to prevent time due to packet loss being added to a measurement. User Datagram Protocol(UDP) in comparison is connectionless, and the sender is not required to handle any data loss via retransmission. Servers configured to receive UDP packets will not necessarily produce a response; the protocol has no handshake or other exchange requirements. Both protocols require either a remote-controlled host or a server running an application that can be manipulated to reveal latency metrics. Conversely, Internet Control Message Protocol (ICMP) is a packet protocol that is implemented in the networking stack of most operating systems, used for sending information and control signals between hosts. The functionality and message types of this protocol are standardized and not application-dependent; if ICMP is not blocked by a firewall, it is likely that communication over the protocol can occur between two hosts. The protocol functions similar to UDP in that there is no retransmission of lost packets, so there is no need to account for unwanted additional latency. These features make ICMP an ideal protocol to monitor latency, with a few caveats.

Tools that operate over ICMP, such as traceroute [19] and ping [13], are useful as they enable us to communicate with a larger range of destination hosts. ICMP on routers must be considered with caution, however, as inbound ICMP traffic is often deprioritized from other protocols, and given limited bandwidth[7]. Transit traffic that is destined for hosts beyond the router is forwarded using ASICs(Application-specific integrated circuits). These circuits pass traffic faster than the device's CPU, which handles traffic destined to the router itself along with other OS functions. Tools that measure latency via ICMP packets directed to routers may not produce measurements consistent with real Internet traffic, and will skew the data to higher latency values.

2.2 - Internet Mapping

Other attempts to produce maps of Internet hosts with connectivity metrics have made use of remotely accessible endpoints from which data can be retrieved. Réseaux IP Européens NCC(RIPE), an organization that handles IP address allocation in continental Europe, has produced a publicly available atlas of endpoints and their relative connectivity, along with their online or offline status relative to the service provider[3]. RIPE distributes "probes", small network connected devices that reside on an organization or individual's network and report back to RIPE. While the organization operates primarily in the European Union, their map does show a presence of at least one probe per state in the U.S.

The Federal Communications Commission (FCC) has produced national data on bandwidth through similar means, using "measurement clients" in the homes of a selection of ISP customers and "measurement servers" residing elsewhere to determine upload and download speeds [11].

2.3 - IP Address Geolocation

One of our goals is to identify as much information about U.S. Internet traffic metrics from a single location, without distributing remote hosts. Since we are unable to place physical hosts ourselves, we require a means of determining the physical location of an Internet host. While the American Registry for Internet Numbers(ARIN) provides a physical location for the organization attached to each block of IP addresses, the information is entered manually and not enforced; each address may move or correspond to hosts outside the given region. Large organizations may use state or airport code identifiers on publicly accessible infrastructure in creating aliases for their devices[18]. However, these naming conventions are still inconsistent and manually entered.

Anycast addressing creates difficulty for geolocation, where a single IP address can be preferentially routed to two or more different physical hosts depending on the source of traffic (or other factors). The BGP protocol allows such a behavior because only the preferred route is shared to avoid conflict. Organizations that operate over large geographical regions can place several hosts around a region, and route traffic traffic only to the closest one for a given source.

Despite these setbacks, using a geolocation service for IP address lookup is an ideal solution for our purposes. Services often advertise relationships with large ISPs that enable them to gather data about address location and anycast mappings that would normally be unavailable to a client. Many offer APIs that allow for bulk lookups, making them ideal for the programmatic data collection involved in this project. The service IPStack fits these criteria, and was selected for use in both DNS cache manipulation and traceroute mapping implementations.

IPStack, among other similar services, are not without fault; if an address cannot be located, often a point in rural Kansas, representing the geographic center of the U.S., will be returned [10]. Bad data points, once identified, can be filtered out as is described in the methodology sections for both DNS cache manipulation and traceroute mapping.

2.4 - Mapping via TOR

In attempting to relate IP addresses to physical locations, other services we could use to provide similar data were investigated. The TOR(The Onion Router) client is a web browser that passes web traffic to a server using a technique called “onion routing.”[12] When we will discuss “nodes” in this context, each is a host running TOR software configured to pass traffic, not necessarily a router in the general definition. Traffic enters the network via a publicly available entry node, is encrypted, and then passed through the client relay nodes. Once it reaches an exit node, the traffic is decrypted and passed to the destination web server.

Initially, it seems possible to find a list of desired exit nodes and pass a ping or other measurement through the node to a known address. We could measure the latency between the exit node and our desired host. However, the highest granularity of exit node that can be requested by the TOR client is destination country. Further, nodes will only pass TCP traffic. These limitations make TOR less than ideal for precise mapping or using a large set of uncontrolled hosts.

2.5 - Background Summary

Despite the difficulties outlined above, with the use of proper data sets, accurate geolocation data, and careful planning, the task can be completed. We took great care to prune out and manage the inevitable inconsistencies that arise with a project such as this. Chapter 3 discusses the background necessary to understand each approach taken; Chapter 4 will describe the solutions we implemented to combat the problems.

3 - Approaches

Two approaches to Internet mapping were identified as promising and explored separately. DNS cache manipulation involves measuring latency via the communication between two remote DNS servers. Traceroute mapping involves measuring the latency between a single point of origin and routers discovered along a path to an endpoint. Due to the limitations discussed previously, TOR as a means of mapping Internet traffic was not used. In succeeding chapters, we describe how these approaches are employed on this project.

3.1 - DNS Cache Manipulation

The Domain Name System (DNS) exists solely as a translation mechanism between spoken language and physical addresses. The Domain Name System is composed of many hierarchical “phonebook” servers. These so-called “nameservers” translate a domain name to an IP address. Much like a name in a phone book, a domain name, such as google.com, acts as an easy-to-remember name that corresponds to the physical address of that web page. A browser will find this physical address by querying a DNS server. The Internet only knows how to send data between IP addresses, not domain names. Thus the DNS system is used as a necessary translation service since humans easily remember domain names but have more difficulty remembering IP addresses. In order to understand how we utilized DNS servers to estimate traffic latency between arbitrary points on the Internet, we must first describe the mechanics of the hierarchical nature of the DNS resolution process. This process is visually demonstrated in Figure 3.1.

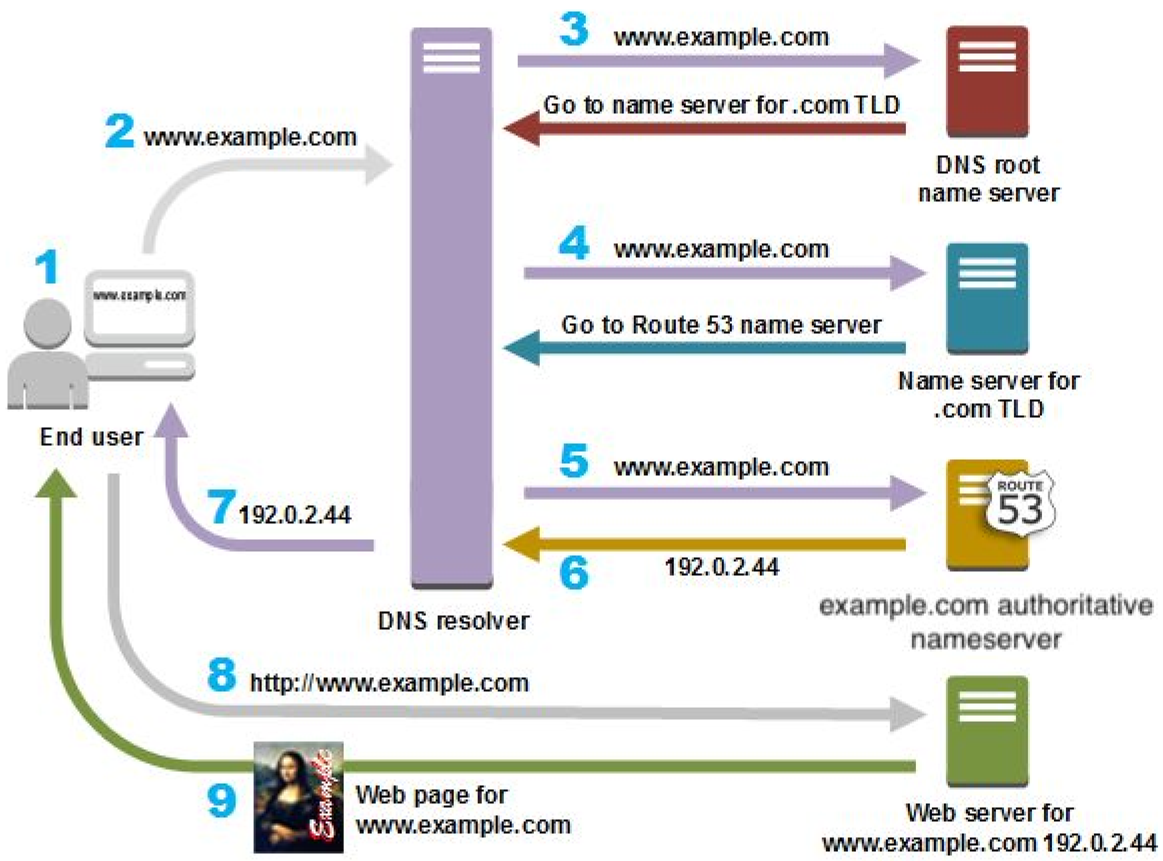


Figure 3.1. The step-by-step breakdown of a single DNS query

There are two types of DNS servers: recursive nameservers also known as resolver or local nameservers, and authoritative nameservers. When a browser performs a DNS lookup it will ask a recursive nameserver for help. The recursive nameserver will then begin the process of finding the IP address by performing a reverse zone lookup. The hierarchy of DNS is demonstrated while performing the reverse zone lookup. Let us examine the typical process to perform a lookup on “www.example.com.” (notice the trailing dot). The first action taken by the recursive nameserver is to check its cache to see if www.example.com has previously been answered. Assuming the cache lookup fails, the recursive nameserver will query a root DNS server. The root server looks at the top level of the reversed domain, the “.com” part. It will respond with a list of authoritative nameservers who contain IP information on ony address ending with “.com”. The recursive nameserver will then choose an authoritative server based on its admin policy, and query for the next zone: “example.com”. The response from the “.com” authoritative servers will contain a list of nameservers that contain address information on “example.com”. The process will repeat, traversing in the reverse direction of the domain name, from .com to example.com to www.example.com (thus the reverse zone lookup) until a nameserver that is responsible for managing the domain to IP mapping of the site is found. At that point the recursive nameserver will discover the IP address that the user is interested in and return it to the client’s browser, which then directly communicates with the target server, www.example.com, using the corresponding IP address. It is worth noting that all nameservers are capable of caching results for time specified by the administrator of the server. In fact any device from the browser to the root server is capable of caching such responses.

The hierarchical nature of DNS, though seemingly tedious and involved, works incredibly well for a number reasons. One such reason is that the hierarchy allows for geographical distribution and master slave servers. Thus DNS servers can be placed closer to end users, distributing load and reducing response time. Further, if one DNS server fails, a duplicate backup servers will replace it, causing relatively little issues. A second benefit of the hierarchy is that each nameserver only has to keep its own records. Otherwise every DNS server would have to contain the full copy of every domain to IP conversion causing a plethora of technical problems for implementation, as well as the hassle of maintaining immense lists.

3.2 - Traceroute

Traceroute is a tool that observes the path traffic takes to a given host. Operating systems implement the traceroute tool differently. The Windows version sends an ICMP echo request to the endpoint, while the Unix implementation sends a UDP packet to a high numbered port considered “unlikely” to be hosting an application on the endpoint. However, both implementations have the same results. For both implementations, a packet will only reach its destination if the receiving host allows it. Whether or not a host is only blocking ICMP or blocking all unnecessary ports, some packets may be considered unnecessary and simply get dropped. The traceroute approach only records data with the routers that respond, making it compatible with either implementation of the traceroute tool. Due to the existing knowledge base on Linux systems for project members, the UNIX implementation was chosen.

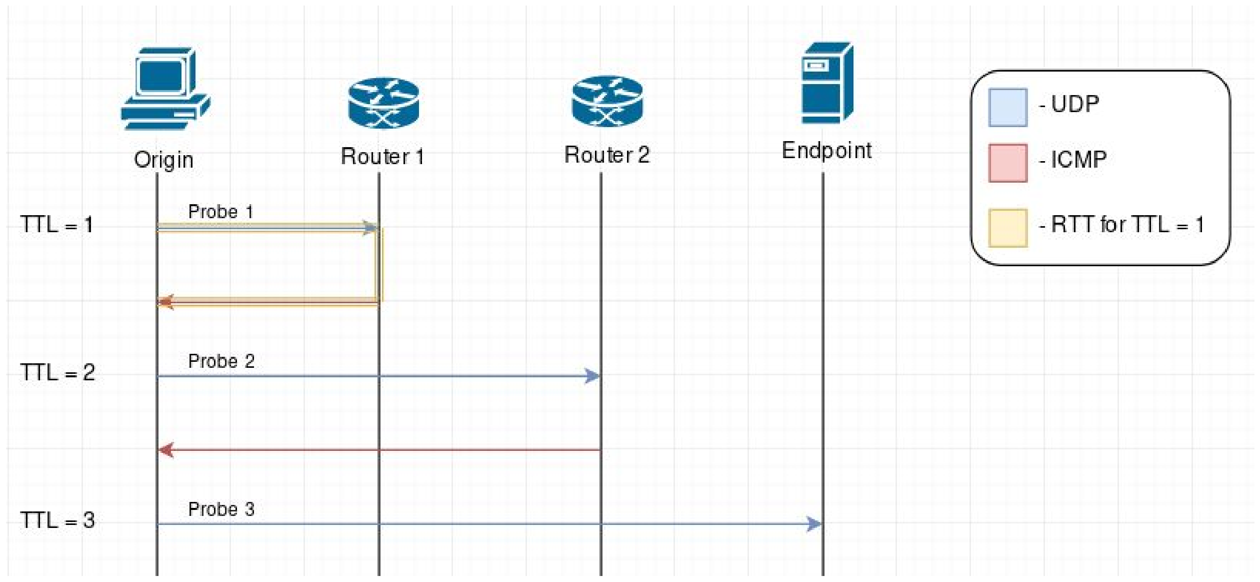


Figure 3.2 - Traceroute mechanism(UNIX)

Traceroute determines a path by sending packets with increasingly large TTL(Time to Live) values. The TTL value designates the maximum number of hops, or routers, that the packet should pass through before it is dropped; this behavior is shown in Figure 3.2. By default, three packets are sent for each TTL value up to a default maximum of 30 hops, or until it reaches the host, whichever occurs first. If the packet does reach the host, it is directed to a random port. If the target port is blocked, no response will be generated from the host itself. For each router or "hop" the packet encounters, the TTL value is decreased by 1. When a router decreases the

TTL value to 0, it responds with a “TTL Exceeded" message via ICMP, and drops the packet. Since the initial packet is identified by the router as transit traffic, we partially avoid the delay from interacting with the control plane. However, because the response occurs via ICMP, we still incur delay on the return trip, and thus cannot treat the resulting latency as completely reflective of transit latency.

Using traceroute presents problems relevant to mapping a route and finding latency. As discussed previously, traffic between two Internet hosts may not follow the same route every time. Since the "TTL Exceeded" response only provides data about the last hop on that particular instance of the route, we cannot be sure that the hops used in the previous instance are still valid. In fact, the Unix implementation of traceroute will denote this information when showing each of the three packets sent per TTL value as seen below:

```
 9  ae47.bb02.ord1.tfbnw.net (204.15.20.161)  34.384 ms 96-34-148-35.static.unas.mo.charter.com (96.34.148.35)  49.172 ms  47.886 ms
```

Here, the first packet reached 204.15.20.161 with a latency of 34.384ms, while the second and third packets reached 96.34.148.35 with latencies 49.172ms and 47.886ms. The latency value provided on the last hop is a Round-Trip-Time (RTT) which represents the time between sending the UDP packet and receiving the ICMP response, as shown in Figure 3.2. Simply halving the RTT value is not sufficient to describe the forward leg of the path; traffic may take a different route for each leg. These limitations, while not all solvable, were taken into consideration when implementing traceroute mapping.

3.3 - Approaches Summary

DNS queries and traceroute are powerful tools because they can acquire data from specific nodes. A DNS request can pass through a specific nameserver. Given sufficient knowledge about the geographical location it is simple to then measure point to point latencies. Traceroute will provide a list of unknown middle-point routers that can be manipulated to obtain geographical data, as well as in between time measurements. Both approaches, although different in method, show promise in their ability to perform the functions necessary to complete this project

4 - DNS Cache Manipulation

To ensure a pure network environment, all data collection processes were run on a Fedora linux esxi virtual machine with 8 cores and 6GB of memory, residing on a Charter residential Internet connection. The server is located in Worcester, Massachusetts and has access to 100Mbps down/10Mbps up bandwidth.

4.1 - Methodology: DNS Cache Manipulation

In the following sections we will demonstrate how to utilize the mechanics of DNS queries to direct traffic between arbitrary nameserver nodes and measure the latency between them. In the latency measurements, the recursive nameserver acts as the “from” node, and the destination authoritative nameserver will be the “to” node. The client sends its DNS query to a recursive nameserver, which subsequently communicates with authoritative nameservers. We exploit this mechanism to take latency measurements. If the recursive nameserver is configured to accept requests from clients outside its local network, then we can send a DNS query to it from our testing server in Worcester, Massachusetts. If we know the geographic location of the recursive DNS server and destination authoritative nameserver, then we can subtract the latency from our client server to the local DNS from the total time of the request, thus giving the time between the recursive DNS server and the authoritative nameserver.

This is the basis for which we generate latency data using DNS queries. We store the data in a two dimensional grid where the value in each cell shows our calculated latency from the corresponding recursive DNS to the authoritative DNS. To implement our latency measurements we used the Unix built-in “dig” tool, a command line tool that performs DNS queries and returns responses as well as latency [6]. It also includes a feature that directs queries through a specific recursive nameserver. The general command format used can be seen in the code snippet below. The “+noall” tells the nameserver to remove nearly all overhead data. The “+time=4” means that the server will wait for at most 4 seconds before timing out. The “+tries=2” means that if there is a timeout, the server will try one more time. Finally, the “+stats” tells the dig to just return the response time statistics.

```
> dig @8.8.8.8 $subdomain.example.com +noall +time=4 +tries=2 +stats
```

To gather latency data across the Internet we first begin by assembling two lists: one for authoritative nameservers, and one for recursive nameservers. For the authoritative nameservers we worked with a collection of approximately 7700 college and university domain names [4]. We wrote several python scripts to collect their authoritative nameservers and run the data through an step-by-step process to create a distilled table containing domain name, authoritative nameserver, IP address, geographic coordinates, and county for each nameserver. The first step generates a list of nameservers for each school domain name using dig. The next step removed redundant authoritative nameservers that were used by multiple schools. The step after that uses ping to collect the IP addresses of each domain name and their authoritative nameservers. Once the IP data is collected, we used the IPStack API, as described in Section 2.3, to determine the geographic coordinates of each authoritative nameserver. After trimming the results based out of the United States or other invalid locations (refer to Section 2.3), we used the haversine formula to determine the nearest county to the data point [20]. Note that the nearest county may place the data point in the county that it actually exists in, rather it will put it in the county whose geographic center is nearest to the data point. The final step involved pruning the nameservers residing in the same county, in order to gather one data point per county. We decided to use servers at the county granularity for multiple reasons. One reason being that graphing data points at a county level is simple and provides easily understood information. A second reason was that the number of counties, approximately 3300, would give us a large dataset, but the data collection phase could be done in a reasonable amount of time, i.e less than a few days to complete. After distilling and finalizing that list we ended up with less than one server per county, only finding 787 authoritative nameservers in unique counties. We used two methods to collect the list of recursive nameservers. We started by checking to see if the authoritative nameservers collected in the previous step acted as recursive nameservers as well. We checked this by querying the finalized list of authoritative nameservers and as if they were recursive, and comparing their responses. In the end, it turned out that only about 12 schools had public authoritative nameservers that also acted as recursive nameservers. We later supplemented that list with a list of nameservers found from multiple online lists [14]. After distilling the list in a similar multi-step process to the one described above, we ended up with 405 recursive nameservers.

Using these two datasets, we began to perform "to-from" lookups, from the recursive nameservers to the authoritative nameservers. Each latency measurement consisted of three phases: the cache priming phase, the calibration phase, and the measurement phase. The purpose of the priming phase, seen in the code snippet below, was to groom the cache of the recursive DNS server to ensure that the target server previously exists. The priming is run twice to ensure that the value is cached.

```
> dig @8.8.8.8 $subdomain.example.com +noall +time=4 +tries=2 +stats
> dig @8.8.8.8 $subdomain.example.com +noall +time=4 +tries=2 +stats
```

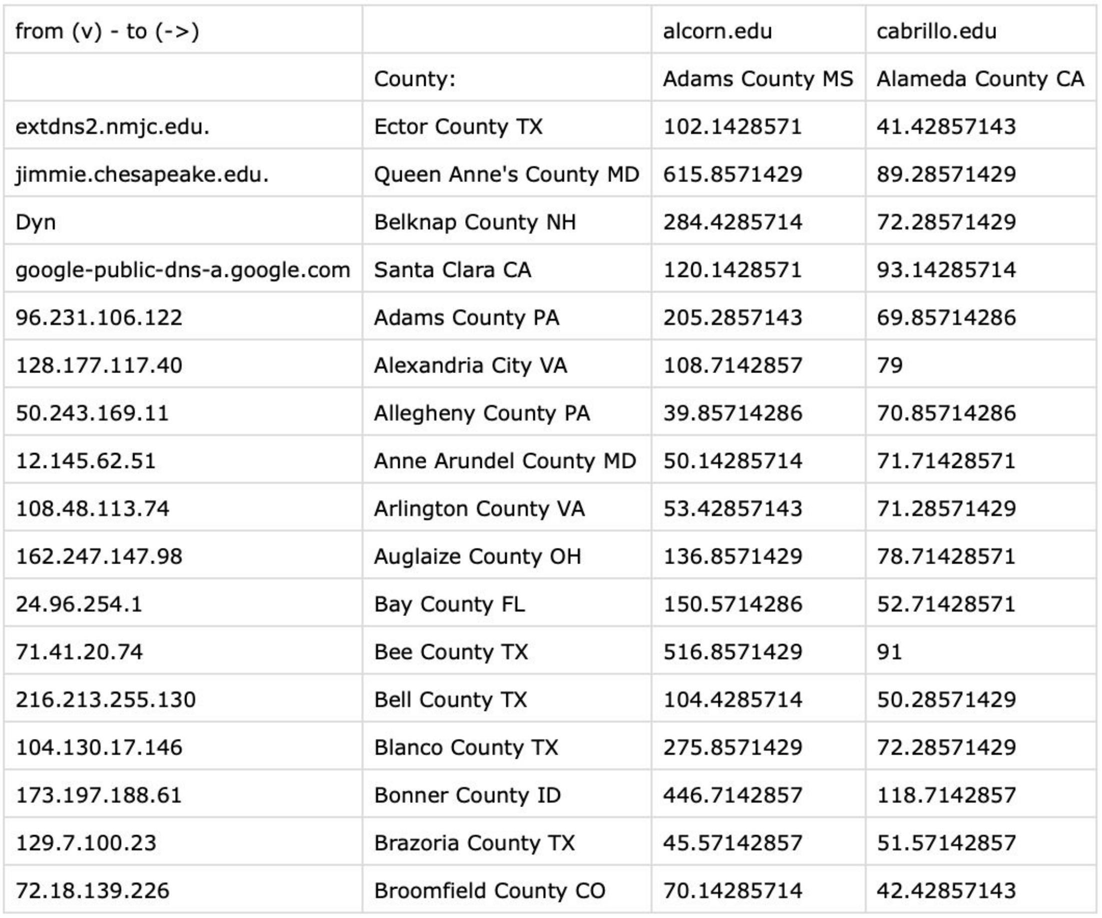
The calibration phase, shown below, sends eight DNS requests for the cached value, thereby allowing us to measure the latency from our server to the recursive nameserver. The grep and awk commands are simple text parsers that, in this case, remove all text except for the millisecond latency value given by dig.

```
> dig @8.8.8.8 $subdomain.example.com +noall +time=4 +tries=2 +stats | grep Query | awk '{print $4}'
#find and return the latency of the request in milliseconds
```

The measurement phase, seen in below, then sends eight more DNS requests to the recursive nameserver. However, these requests query a random subdomain of the target server, that is unlikely to have been asked before, and will thus not be cached at the recursive nameserver. The value we associate with the domain we then chose to be the median of the measured values.

```
> dig @8.8.8.8 $random_subdomain.example.com +noall +time=4 +tries=2 +stats | grep Query | awk
'{print $4}' #find and return the latency of the request in milliseconds
```

Thus we can measure response time as the RTT value of the path from our server, to the recursive nameserver, and finally to the authoritative nameserver, and back. By subtracting the result of the calibration phase from the measurement phase, we can determine the latency between the recursive nameserver and the target authoritative nameserver. Using this mechanism we can measure latency between any recursive and authoritative nameservers on the Internet. We stored our results in a two-dimensional matrix, a sample can be seen below in Figure 4.1. The value in each cell represents the measured latency from the county in the row to the county in the column. It is important to note that generating these matrices requires millions of lookups. On our sample 400 by 787 node dataset, we are sending nearly 5.5 million dig queries, or a runtime of about 60-80 hours. Fortunately, as network requests such as dig are I/O bound tasks, they require little CPU time, making it an easy opportunity to parallelize the tasks. After testing the system, we found that running somewhere between 20-25 threads provided an optimal balance of a steady CPU load while using less then ~60% of the available bandwidth in our experiment environment. We found that using about 60% of bandwidth available accounted for random network fluxuations. Other optimizations included giving lookups a maximum number of attempts before assuming that one of the servers is down, and skipping those lookups.



from (v) - to (->)		alcorn.edu	cabrillo.edu
	County:	Adams County MS	Alameda County CA
extdns2.nmjc.edu.	Ector County TX	102.1428571	41.42857143
jimmie.chesapeake.edu.	Queen Anne's County MD	615.8571429	89.28571429
Dyn	Belknap County NH	284.4285714	72.28571429
google-public-dns-a.google.com	Santa Clara CA	120.1428571	93.14285714
96.231.106.122	Adams County PA	205.2857143	69.85714286
128.177.117.40	Alexandria City VA	108.7142857	79
50.243.169.11	Allegheny County PA	39.85714286	70.85714286
12.145.62.51	Anne Arundel County MD	50.14285714	71.71428571
108.48.113.74	Arlington County VA	53.42857143	71.28571429
162.247.147.98	Auglaize County OH	136.8571429	78.71428571
24.96.254.1	Bay County FL	150.5714286	52.71428571
71.41.20.74	Bee County TX	516.8571429	91
216.213.255.130	Bell County TX	104.4285714	50.28571429
104.130.17.146	Blanco County TX	275.8571429	72.28571429
173.197.188.61	Bonner County ID	446.7142857	118.7142857
129.7.100.23	Brazoria County TX	45.57142857	51.57142857
72.18.139.226	Broomfield County CO	70.14285714	42.42857143

Figure 4.1. Sample DNS Matrix output.

Once the matrix is generated, we use the Javascript D3 library to plot the data on a map. Because we only collect data points for the nameservers on our list, there are inevitably multiple counties that have no data associated with them. For the sake of appearance we perform a linear interpolation across the map in order to show expected latency data, however, we realize that these interpolated values are estimates, and do not necessarily reflect the latency measurements we might find in each specific county. To interpolate the data we implemented the numpy GridData function, which creates a three-dimensional interpolated grid given a set of (x, y, z) value pairs, with longitude being x, latitude being y, and the measured latency being z [15]. We then found the projected latency value at the coordinates of each county, and plotted those values. Naturally, we show two graphs for each dataset, one with interpolated values and one without. We can also generate a map for each county with measured latency values, showing the latency from that county to every other county. For counties without measured latencies, we could attempt to estimate interpolated values based on its neighbors, but this data would not be of much use due to its lack of real data measurements. We also generate a graph showing the average latency from one county to every other county, seen in Figure 4.2.

4.2 - Results: DNS Cache Manipulation

Using our list of 405 recursive DNS servers and 787 authoritative DNS servers we generated a 405x787 matrix of latency values between each pair. The graph in Figure 4.6 shows, for each recursive DNS server, the average latency to all authoritative servers.

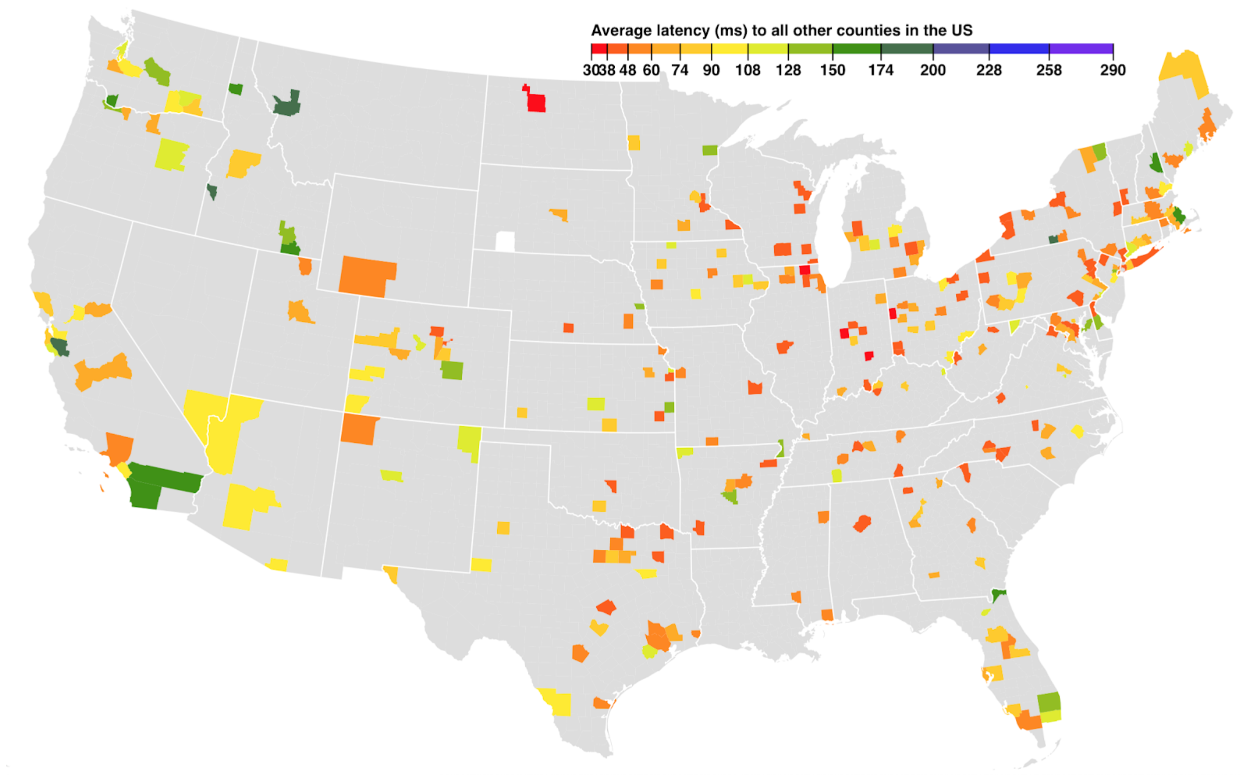


Figure 4.2. A map showing the average latency to all other authoritative nameservers from each recursive nameserver.

The figure clearly displays the geographical distribution of the recursive nameservers used in the test as well as their average latency to all other counties. The average latency refers to the mean time from a particular recursive nameserver to all other authoritative nameservers. For example, in Yolo County, California, our calculated value of 73.81 ms means that the average latency from Yolo County to all 787 authoritative nameservers was measured to be 73.81 ms. Interestingly, the geographical center point of all collected nameservers lies in eastern Missouri, close to the mean population center of the contiguous United States. To help better visualize this dataset across the entirety of the United States, we interpolated values from the above Figure onto the entire country, seen in Figure 4.3. As mentioned previously, only the counties with data measurements taken from them are accurate in Figure 4.3. All interpolated values are mathematically estimated, and not necessarily accurate. There are, however, some benefits to interpolating our data in such a way.

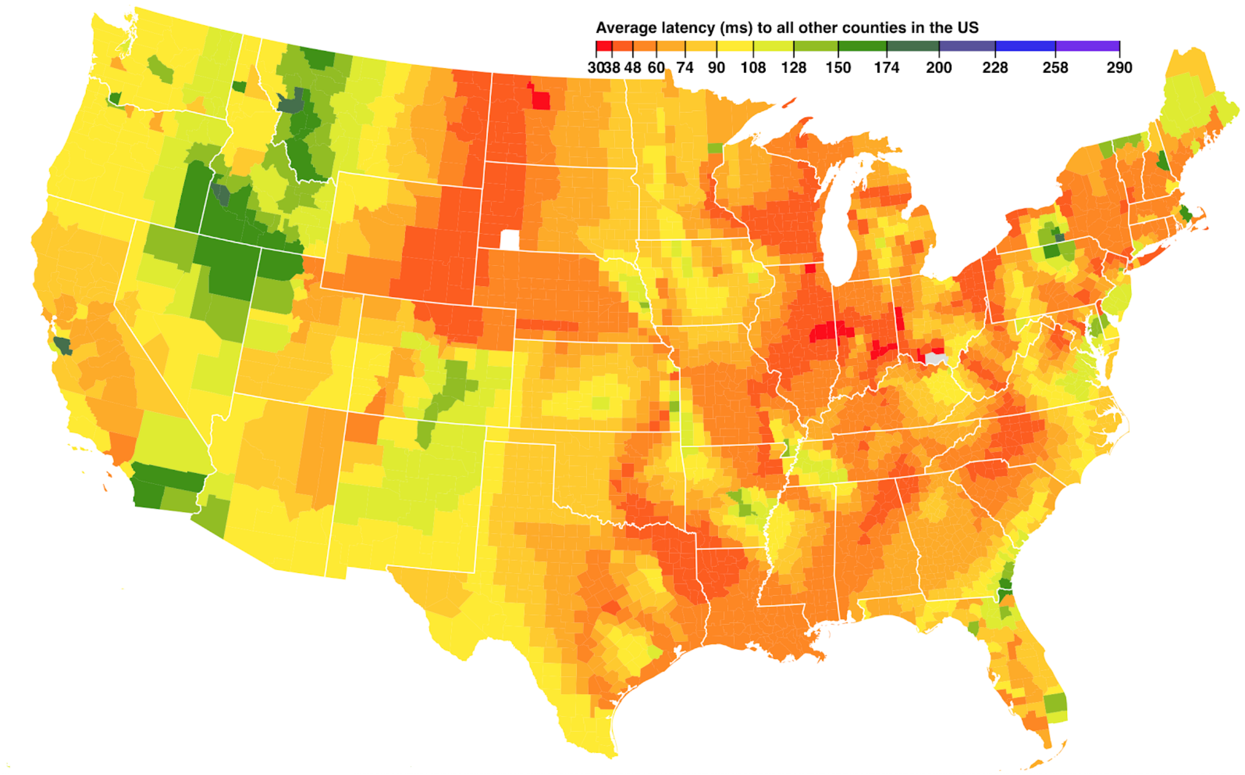


Figure 4.3. The values from the above map were taken and interpolated

This Figure demonstrates the connectivity of various regions. Many major cities can be observed as having lower latencies in their connectivity. Places such as Denver, Colorado, Dallas, Texas, Raleigh, North Carolina, Cleveland, Ohio, Buffalo and Manhattan, New York, and their surrounding areas show low latency values measured. More remote regions such as Montana, Nevada, Idaho, rural New York and northern Maine clearly reside in locations with slower connectivity. Some unexpected outliers appear in these graphs, such as the low latency measurements between Wyoming and North Dakota, or the high latency measured in the San Diego County region. Explanations for outliers will be explained in the discussion section.

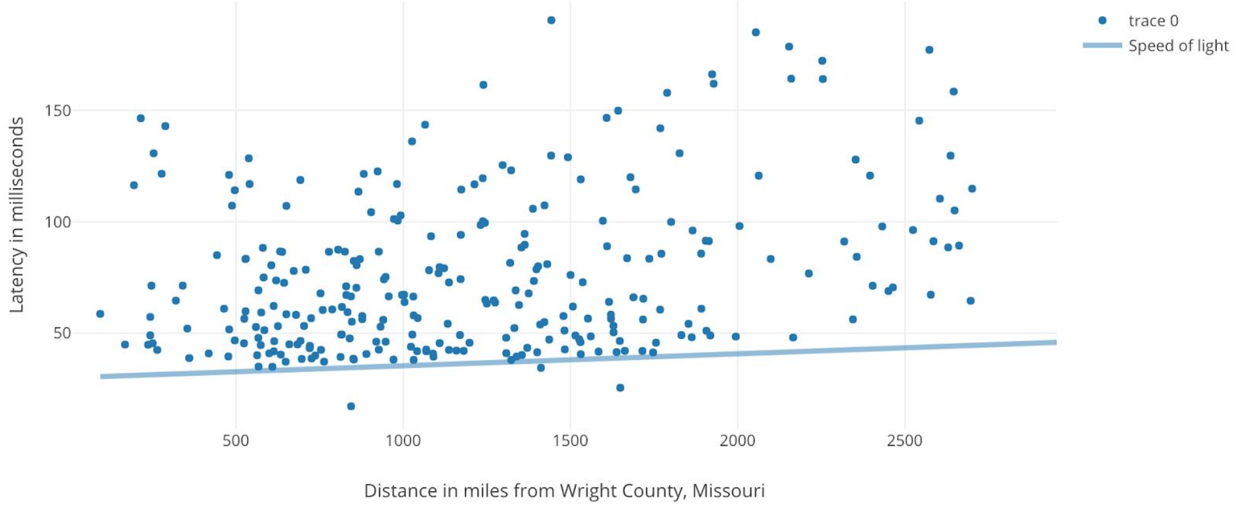


Figure 4.4. Latency increases (Y-axis, milliseconds) as distance to the mean population center of the United States increases (X-axis, miles)

Figure 4.4 demonstrates our latency measurements in terms of distance. As distance increases, we observe a “connectivity cone” shape starting to form. Other than the sparse outliers in the top left, the points all lie within the connectivity cone. The points on the bottom line, an approximation of the speed of light, translated on the Y-axis to fit the bottom of the cone, clearly demonstrate close connectivity to backhaul fiber. As distance increases, these points tend to increase in latency at a rate nearly proportional to the speed of light in a fiber optic cable. For example, Niagara County, New York (data point: 1300 miles, 39ms latency), sits right outside of Buffalo, New York. It can be surmised that Buffalo particular location must be connected near a backhaul fiber network. In fact, Buffalo is a particularly well connected hub not only with regard to Internet connectivity, but also roadway and railway infrastructure [8]. Figure 4.4 shows an estimate of the approximate connectivity of each point based on how far it lies above the speed of light line. In Figure 4.4 below we see a number of outliers in the top left portion. Figure 4.5 shows those outlier points plotted on a map.

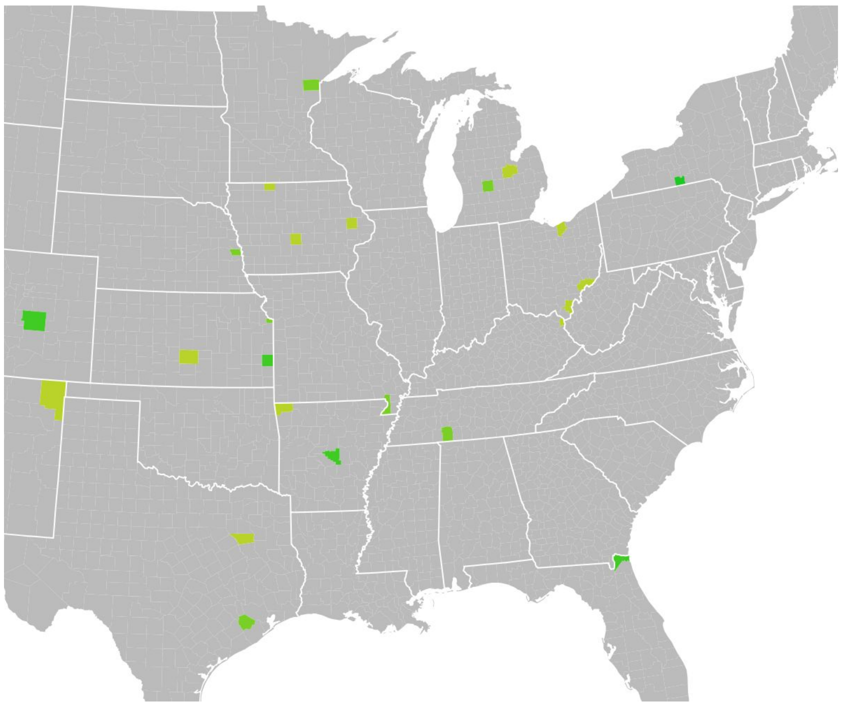


Figure 4.5. A graph of the outlier nodes from Figure 4.4

Overall there does not appear to be any interesting pattern coming from the outliers shown. It can thus be inferred that these nameservers have higher latencies due to poorer connection or another independant issue. In Figures 4.6 and 4.7 below, we attempted to plot the data in terms of closeness to large cities. Unexpectedly, we could not find strong correlation between the outlier data points based on their location.

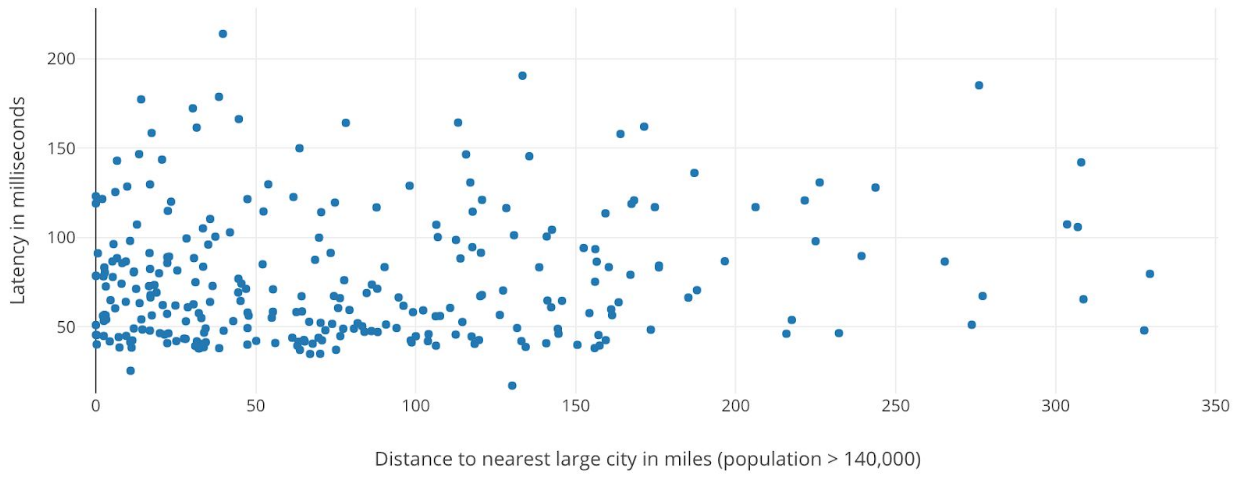


Figure 4.6. How latency measurements change as the data point is physically closer to a large city

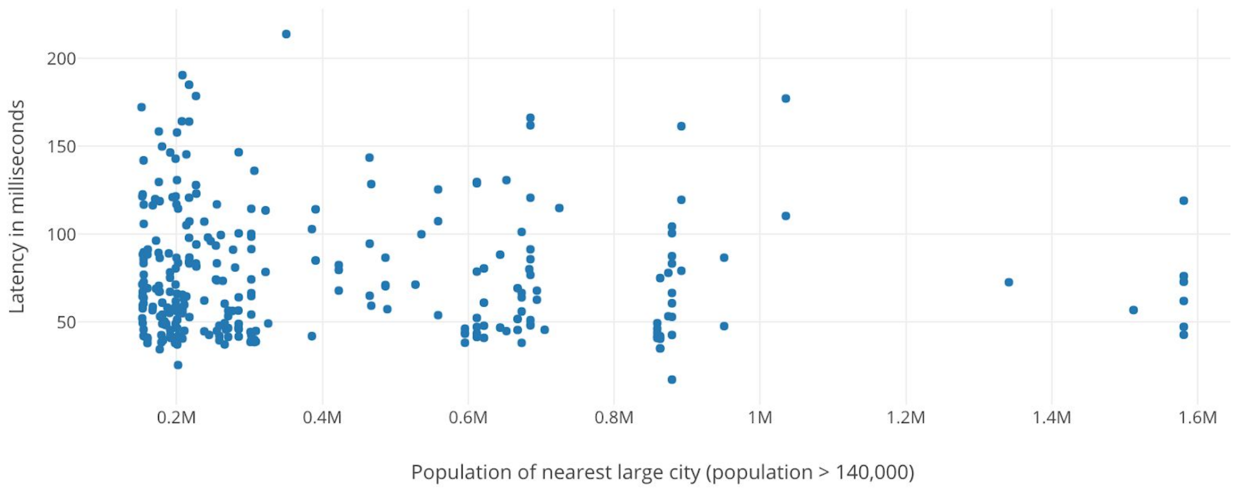


Figure 4.7. How latency measurements change compared to the size of the nearest large city

We chose 140,000 as the lower limit of our definition of a "large city" because approximately 300 U.S cities have a population of 140,000 or more, the number of cities cited in the article by the census bureau. We would have loved to see some stronger correlation in Figures 4.6 and 4.7. It is possible that our data was just inaccurate enough to provide clear patterns in these particular graphs. Figure 4.7 shows what appears to be a slight decrease in latency as population of the nearest city rises, however the data are too weak to give any certainty to the claim.

4.3 - Discussion: DNS Cache Manipulation

The data collected via the DNS nameserver method closely resembles what we expected to see in many facets. In Figure 4.3 we saw a map of the United States with faster latencies tending to lie in densely populated areas. In Figure 4.3 we can reasonably agree with data points such as Buffalo New york having good connectivity speeds as discussed in the results section. We see unusual data in other places that we might not expect to see, such as the low latency measurements of the entire section between eastern Wyoming and North Dakota and the high latency measurements in San Diego County, California. Many of these irregularities can be explained because of a lack of data points in terms of both regional data point density and geographical point density. In the case of San Diego county, the DNS approach estimates an average latency of 150-170 milliseconds, while the traceroute method shows a near cross-country latency value of 80-100 milliseconds. The discrepancy in measured values can be attributed to any of the data inconsistencies noted in our discussion of the methods we used. If our controlled environment was not in error, then the outlier county shows untrustable data because we extrapolated the average latency for the entire county based on one recursive nameserver alone. It is possible that the nameserver in question may have one or more of the following: poor Internet connectivity, packets deprioritization from external networks, and inconsistent reporting timing metrics. The lack of sample server density per county easily skews the data we collect to represent the unique values of the data points we have gathered. In the end, we cannot recommend that the data we have collected through the DNS cache manipulation method be used as source for estimating latencies between regions of the United States. However, given a sufficiently large dataset of local and authoritative nameservers as well as extremely accurate geolocation measurements, the DNS cache manipulation method may be a viable way to measure latencies across the Internet from a single point of transmission.

4.4 - DNS Cache Manipulation Summary

The results of the DNS cache manipulation approach are very promising. The values returned showed a level of accuracy and consistency that we expected. Although it is believed that there were a few steps that reduced the precision of our measurements. The first being small nameserver datasets. The next being inaccurate geolocation information. And the last being collecting sample sizes of only eight queries for each nameserver. The data generated provides much insight into the state of the Internet geographically in terms of latency and closeness to backhaul fiber. These results could be cross referenced with information from ISPs to increase the accuracy. They can also be cross-referenced with the results of the traceroute approach whose data has different inferences.

5 - Traceroute

The traceroute-based mapping implementation operated on a dataset of 1000 U.S based university websites[4]. All scripts ran on the same host as specified for DNS cache manipulation. Data was acquired from an origin host in Worcester, MA on a Charter residential connection.

5.1 - Methodology: Traceroute

The dataset used for traceroute mapping is useful not only as a set of publicly accessible endpoints that can be expected to receive regular TCP traffic, but as a means of finding metrics on routers that would normally be uncontactable. Many routers block direct ICMP traffic altogether along with traffic to other ports that would normally be used by traceroute probes; the only way to measure their latency is through transit traffic that experiences an exceeded TTL.

A Java application was written to programmatically run traceroute over the dataset. For each hop, previously unseen routers were logged along with their RTT value. For routers that were encountered multiple times per hop, only the lowest RTT value was taken. As neither the forward or reverse path costs can be individually derived from a Round-Trip-Time alone, no modifications to the value are made.

The same routers are often encountered several times over the course of multiple traces. For each router entry, the endpoints which it routes to are recorded, and a count for each router is produced. New RTT values are only taken if they are lower than a previous instance. Further, it is not useful to run the program over the full set of 7700 hosts, as the number of new routers encountered drops over multiple runs.

A geolocation lookup was then performed on the resolved address of each router, using IPStack's web api. This service will return a point close to the U.S geographical center if it is unable to retrieve a valid latitude and longitude for the address. Addresses that resolve to this point are removed from the dataset.

From here, the haversine formula is used to approximate a distance between the origin and the latitude and longitude of the address. For the distance found, a lower bound for latency can be derived. This is twice the time required for light to travel along a fiber optic cable run from source to destination. The speed of light in a vacuum is 299,792,458m/s. Fiber used for long run over 2km, the majority of the makeup of our measurements, is primarily single mode. The refractive index for single mode fiber is 1.44475. Dividing our speed gives us:

$$\frac{299792458}{1.44475} = 207,110,506m/s$$

Or 128.69 miles per millisecond. Dividing each calculated distance by this value and multiplying the result by 2 gives us a minimum theoretical RTT for the router. If the observed latency is below this value, we can conclude the latitude and longitude produced by IPStack are likely incorrect, and the result can be discarded. While it is still possible that values above the minimum theoretical latency are derived from incorrect geolocations, we can reduce the amount of noise in our data by eliminating values below the minimum.

Attempts were made to make use of the path data collected by traceroute. Connections between routers were represented as edges in a graph, weighted by the RTT value. As multiple routers are often acquired per TTL value, this quickly creates a large number of variant paths that traffic could have taken. Certain paths were determined unlikely to have occurred, and dropped, using a method shown in Figure 5.1 below. The red dotted line shows a path that would have been removed between R1 and R4.

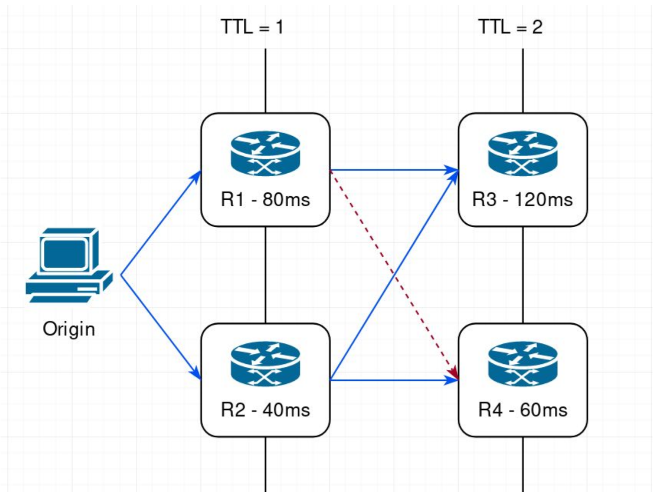


Figure 5.1 - Traceroute unlikely path removal

Since an RTT value for a given TTL contains the latency of the previous hop, paths where the RTT decreases are identified as unlikely. Graphing the data this way was eventually abandoned due to the untrustworthiness of the data; traceroute provides no confirmation that the routers traversed in a previous hop were traversed for the current hop.

5.2 - Results: Traceroute

Output addresses from the traceroute mapping application were fed into the FCC's area API[2] to produce an associated FIPS code. These codes are used here to identify a unique county in the United States, however they can also identify other geographic entities[20]. The data were then mapped along with the RTT values discovered for each address.

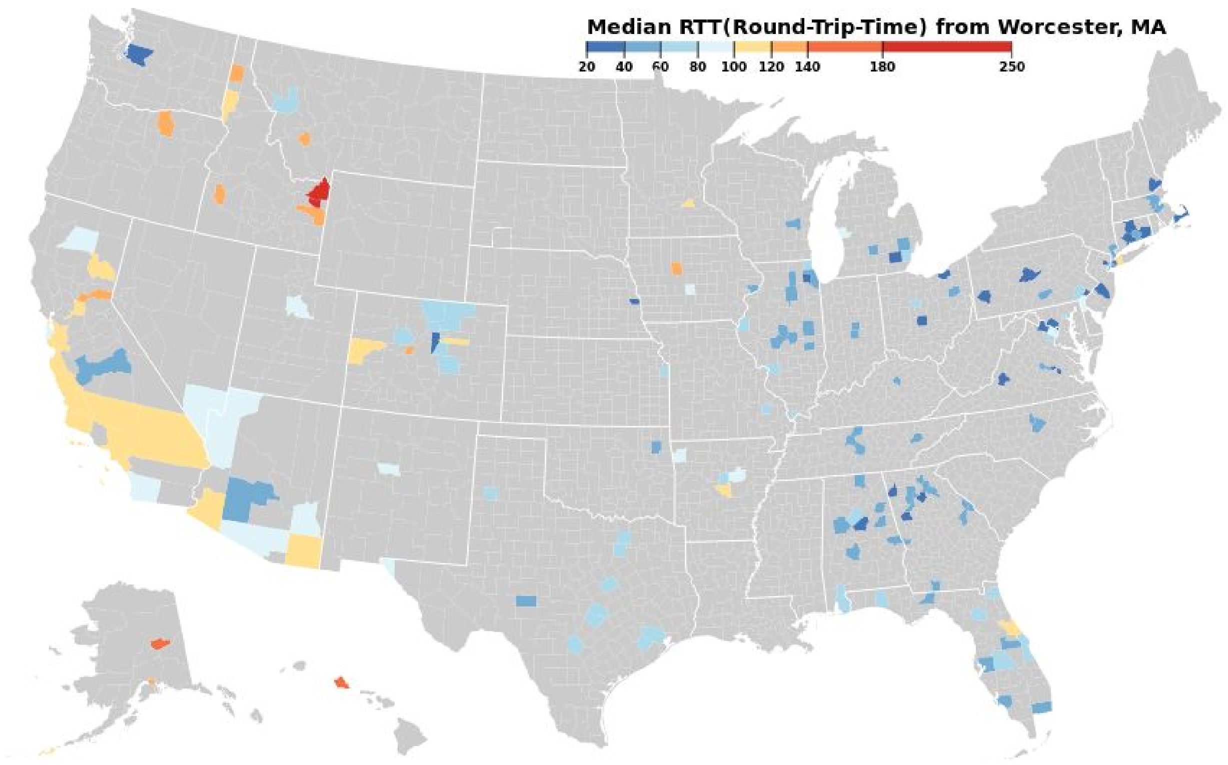


Figure 5.2 - Traceroute latency map

This map was produced using D3js with the additional TopoJSON block[5]. TopoJSON takes in a two column tab-separated values file mapping each FIPS code to a numerical value, in this case latency in milliseconds.

Several routers had particularly high latency for their region. In Volusia, Florida, a router owned by Level 3 Communications presented at 100.86ms, relative to sub 70ms values in surrounding counties. The device was encountered 3 times throughout all traces. An Amazon router in Webster, Iowa reported a latency of 121.9ms, and was only encountered once. Low latency outliers show similarly low frequency. A Comcast router in Knox, Tennessee reported 55.15ms, but was only encountered once. A Hurricane Electric router in Fresno, California reported 55.41ms with a single encounter. Based on the distribution of these outliers on the map, few are high latency and low distance; most stand out due to low latency at high distance. This could be an indication that the quality of long-distance network infrastructure is the deciding factor in a location's relative latency.

When plotting latency against distance we find what we might expect; as distance increases, so does latency(Figure 5.3). With the amount of variation present in this graph it can be difficult to assign a particular latency to a location. However, a few routers are encountered more often than others. The graph below plots distance against the number of endpoints that produces each router at least once on a trace. The graph is zoomed to a region showing Loudoun, VA, at approximately 420 miles from origin, with the shaded regions showing routers owned by the two largest organizations found in the region(Amazon and Charter.)

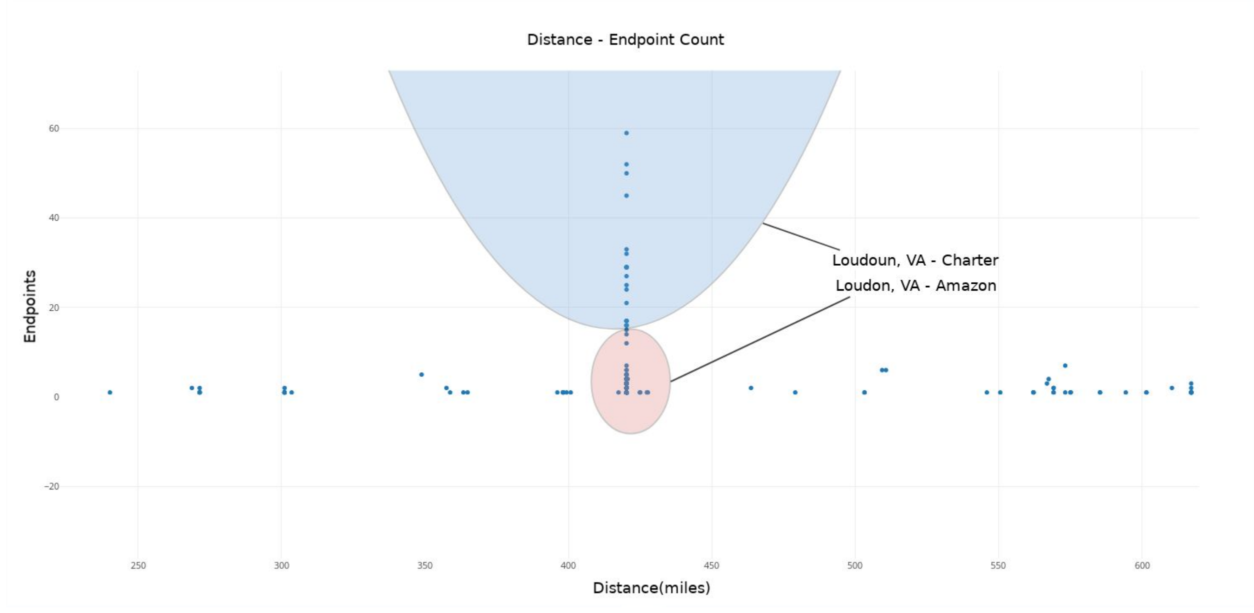
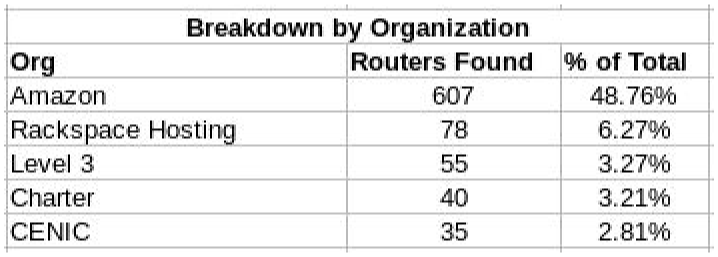


Figure 5.3 under zoom - Frequency of Charter/Amazon routers

For both organizations, it appears that only a few routers are absorbing the majority of traffic.

Organizations with the largest total number of routers found are shown below in Figure 5.4. Given that the endpoints used are web servers, it is unsurprising to see hosting providers among these organizations.



Breakdown by Organization		
Org	**Routers Found**	**% of Total**
Amazon	607	48.76%
Rackspace Hosting	78	6.27%
Level 3	55	3.27%
Charter	40	3.21%
CENIC	35	2.81%

Figure 5.4 - traceroute organization frequency

This data however does not take into account where the majority of traffic routes. While major routing hubs closer to the origin will receive more traffic, there are clear traffic preferences even when location is isolated. Below, the frequency of encounter for each block of Amazon and Charter routers can be seen. In the location Loudon, VA at a distance of 420.26 miles from our origin, a total of 169 routers owned by Charter and 33 owned by Amazon are found. Charter's exhibit a variation of between a minimum of 17 encounters and a maximum of 872 encounters. Amazon's routers show a variation of between 1 and 25 encounters. Further, when we compare latency to endpoint count in Loudoun for each organization(Figure 5.5 and 5.6):

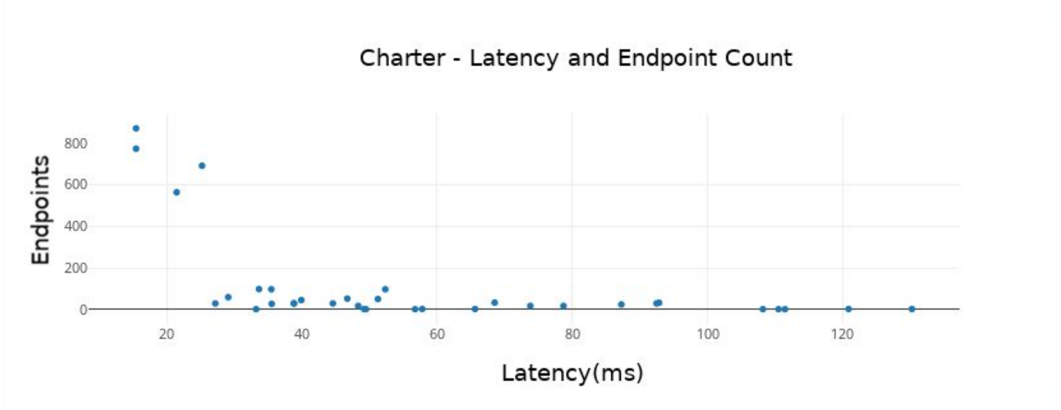


Figure 5.5 - Charter latency v. encounter frequency for Loudon, VA

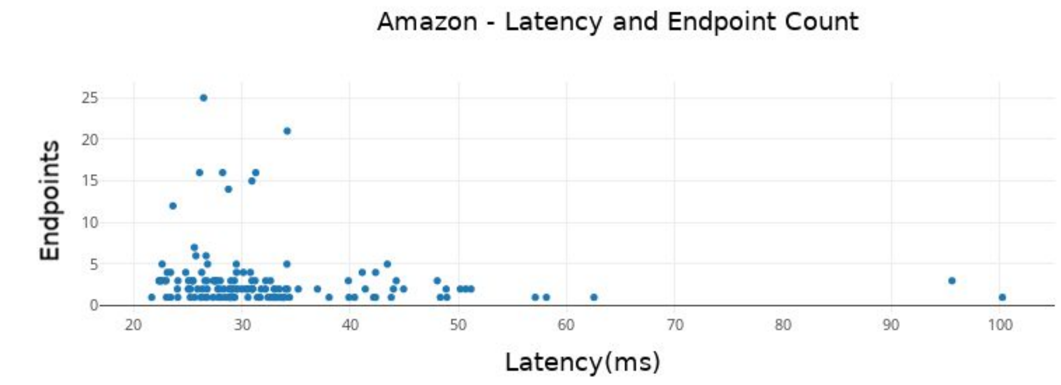


Figure 5.6 - Amazon latency v. encounter frequency for Loudon, VA

For Charter and ISPs in general, a possible explanation for this is that high frequency, low latency routers are ISP-owned infrastructure, while low frequency, high latency routers are other organizations that have bought service and addresses from Charter. These organizations would only receive traffic to their endpoints, in this case any university websites, and would likely have slower last-mile infrastructure compared to larger ISPs.

In King County, Washington, almost exclusively Amazon routers are found; 295 of the 302 devices are owned by the organization. These routers exhibit a variation of between 1 and 13 endpoints encountered. There does not appear to be a significant correlation here between frequency of encounter and latency in this region.

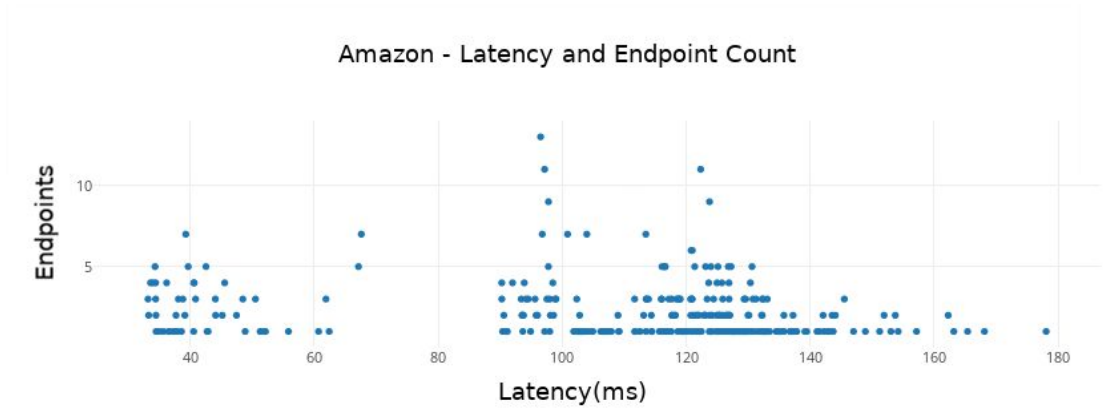
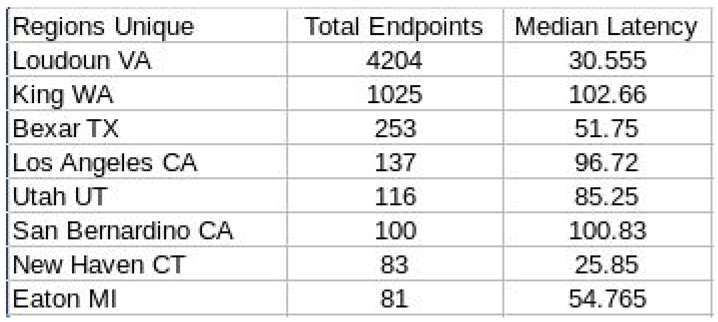


Figure 5.7 - Amazon latency v. encounter frequency for King, WA

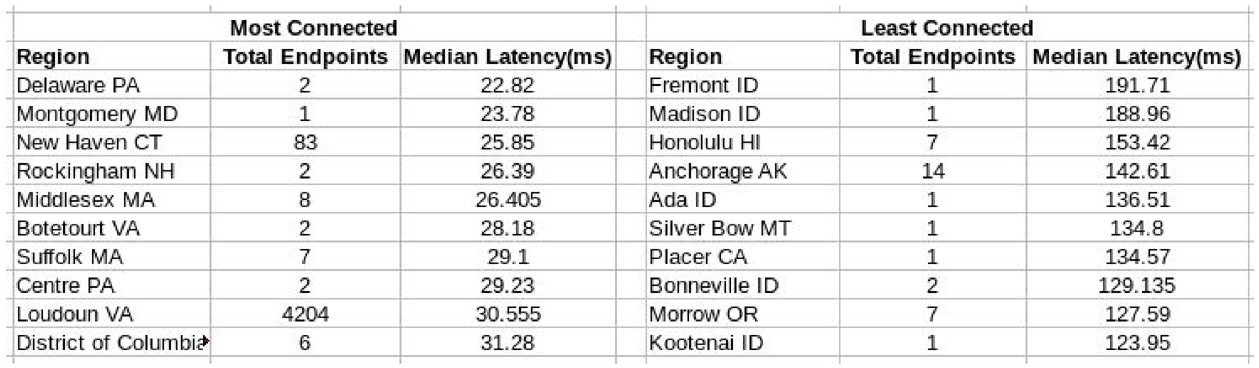
Figure 5.8 shows the most connected regions relative to the origin(Worcester, MA) by frequency(Total Endpoints). Note that this is the total number of endpoints routed over all routers in the region, and will contain duplicates across the dataset. Also shown is the median latency for that region.



Regions Unique	Total Endpoints	Median Latency
Loudoun VA	4204	30.555
King WA	1025	102.66
Bexar TX	253	51.75
Los Angeles CA	137	96.72
Utah UT	116	85.25
San Bernardino CA	100	100.83
New Haven CT	83	25.85
Eaton MI	81	54.765

Figure 5.8 - Most frequently encountered regions

Only two regions (Loudoun VA, New Haven CT) appear under the most connected regions by median latency.



Most Connected			Least Connected		
Region	Total Endpoints	Median Latency(ms)	Region	Total Endpoints	Median Latency(ms)
Delaware PA	2	22.82	Fremont ID	1	191.71
Montgomery MD	1	23.78	Madison ID	1	188.96
New Haven CT	83	25.85	Honolulu HI	7	153.42
Rockingham NH	2	26.39	Anchorage AK	14	142.61
Middlesex MA	8	26.405	Ada ID	1	136.51
Botetourt VA	2	28.18	Silver Bow MT	1	134.8
Suffolk MA	7	29.1	Placer CA	1	134.57
Centre PA	2	29.23	Bonneville ID	2	129.135
Loudoun VA	4204	30.555	Morrow OR	7	127.59
District of Columbia	6	31.28	Kootenai ID	1	123.95

Figure 5.9 - Most and least connected regions

5.3 - Traceroute Summary

Even though the traceroute method was restricted to a single point of origin and its path data deemed to inaccurate for use, the data produced insights on traffic flow and major ISP and hosting infrastructure. Router load balancing was also apparent, especially in the case of Amazon’s hub in Washington. The data suffered from limited interpolation potential between non-origin routers; this could be resolved by performing the method from multiple locations.

6 - Comparison

Both methods of mapping produced data that correlated increased distance with increased latency. However, data acquired via DNS cache manipulation results in generally lower latency values, as the method acquires data from multiple points of origin to multiple endpoints. This also makes the data more conducive to interpolation and determining network connectivity from any point in the geographical U.S. Traceroute data was acquired from a single point of origin on the east coast of the U.S.; this results in generally larger latency measurements with limited potential for interpolation.

The ability of DNS cache manipulation to perform multi-origin data collection may have also influenced outliers. Outliers found through this method were more often high latency and low distance, whereas traceroute discovered mostly low latency, high distance outliers.

Overall, DNS cache manipulation produced metrics more applicable to determining point-to-point connectivity between regions, while the traceroute data revealed primary hubs that Internet traffic follows.

7 - Conclusions

The data collected by both the DNS cache manipulation method and the traceroute method proved promising for their potential in gathering Internet latency metrics from a single location. The DNS cache manipulation demonstrates geographical regions of Internet “hotspots”, giving insight into general connectivity based on location. The traceroute data gives interesting insight into the structure of Internet routers and how Internet Service Provider manage and load balance them.

Both methods could benefit from improved geolocation accuracy and comparison of data collected at different points in time. Internet connectivity may be worse during peak activity. Traceroute suffered from an inability to make accurate assumptions from path data; a means of ensuring measurement probes traverse the same path each time is necessary.

For traceroute, future work should focus on a means of distributing the methodology used to acquire this data to other remote hosts, in order to determine the effect of asymmetric reverse routes. In addition, efforts should be made to determine more accurate ways of determining the geolocation of an IP address. The DNS cache manipulation method would greatly benefit from large datasets of recursive and authoritative nameservers.

8 - References

8.1 - Sources cited

1. "Analyzing UDP Usage in Internet Traffic." *CAIDA*, 2018, https://www.caida.org/research/traffic-analysis/tcpudpratio/
2. "Area API." *FCC*, https://geo.fcc.gov/api/census/#!/area/get_area
3. "Atlas Project." *RIPE NCC*, https://atlas.ripe.net/about/
4. College Scorecard Data. (n.d.). Retrieved September, 2018, from https://collegescorecard.ed.gov/data/
5. D3 TopoJSON https://bl.ocks.org/almccon/410b4eb5cad61402c354afba67a878b8
6. "Dig", *Linux Manual Page*, 2018, https://linux.die.net/man/1/dig
7. Dugal, Dave, et al. "Protecting the Router Control Plane." *IETF*, 2011, https://tools.ietf.org/html/rfc6192
8. Durairajan, Ramakrishnan, et al. "InterTubes." 2015, http://pages.cs.wisc.edu/~pb/tubes_final.pdf
9. "Geographic Terms and Concepts", *United States Census Bureau,* 2010 https://www.census.gov/geo/reference/gtc/gtc_codes.html#fips
10. Hill, K. (2017, July 24). How an internet mapping glitch turned a random Kansas farm into a digital hell. Retrieved October, 2018, from https://splinternews.com/how-an-internet-mapping-glitch-turned-a-random-kansas-f-1793856052

11. "Measuring Fixed Broadband." *FCC*, 2016, https://www.fcc.gov/reports-research/reports/measuring-broadband-america/measuring-fixed-broadband-report-2016
12. Onion Routing. (2005). Retrieved August, 2018, from https://www.onion-router.net/
13. "Ping", Linux Manual Page, *Linux Manual Page*, 2018, http://man7.org/linux/man-pages/man8/ping.8.html
14. Public DNS Information. (n.d.). Retrieved September, 2018, from https://public-dns.info/
15. Rekhter, Yakov, et al. "Application of the Border Gateway Protocol in the Internet.", *IETF*, 1995, https://tools.ietf.org/html/rfc1772
16. Scipy -- Interpolate.Griddata. (n.d.). Retrieved October, 2018, from https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.griddata.html
17. Sriram, Kotikalapudi, et al. "Problem Definition and Classification of BGP Route Leaks", *IETF*, 2016, https://tools.ietf.org/html/rfc7908
18. Steenbergen, Richard "Troubleshooting with Traceroute." 2009, https://www.nanog.org/meetings/nanog45/presentations/Sunday/RAS_traceroute_N45.pdf
19. "Traceroute", *Linux Manual Page*, 2018, http://man7.org/linux/man-pages/man8/traceroute.8.html
20. U.S. Gazetteer: 2010, 2000, and 1990. (2012, September 01). Retrieved September, 2018, from https://www.census.gov/geo/maps-data/data/gazetteer.html
21. Van Brummelen, Glen Robert (2013). Heavenly Mathematics: The Forgotten Art of Spherical Trigonometry. Princeton University Press. Retrieved October, 2018 from https://books.google.com/books?id=0BCCz8Sx5wkC&pg=PR7#v=onepage&q&f=false

22. Wills, C. (2017, June). Geographical Connectivity In the United States. Retrieved September, 2018, from https://web.cs.wpi.edu/~cew/papers/tr1701.pdf