CalcuLatency: Leveraging Cross-Layer Network Latency Measurements to Detect Proxy-Enabled Abuse

Reethika Ramesh, *University of Michigan;* Philipp Winter, *Independent;*
Sam Korman and Roya Ensafi, *University of Michigan*

https://www.usenix.org/conference/usenixsecurity24/presentation/ramesh

This paper is included in the Proceedings of the
33rd USENIX Security Symposium.

August 14–16, 2024 • Philadelphia, PA, USA

978-1-939133-44-1

Open access to the Proceedings of the
33rd USENIX Security Symposium
is sponsored by USENIX.

CalcuLatency: Leveraging Cross-Layer Network Latency Measurements to Detect Proxy-Enabled Abuse

Reethika Ramesh
University of Michigan

Philipp Winter
Independent

Sam Korman
University of Michigan

Roya Ensafi
University of Michigan

Abstract

Efforts from emerging technology companies aim to democratize the ad delivery ecosystem and build systems that are privacy-centric and even share ad revenue benefits with their users. Other providers offer remuneration for users on their platform for interacting with and making use of services. But these efforts may suffer from coordinated abuse efforts aiming to defraud them. Attackers can use VPNs and proxies to fabricate their geolocation and earn disproportionate rewards. Balancing proxy-enabled abuse-prevention techniques with a privacy-focused business model is a hard challenge. *Can service providers use minimal connection features to infer proxy use without jeopardizing user privacy?*

In this paper, we build and evaluate a solution, CalcuLatency, that incorporates various network latency measurement techniques and leverage the application-layer and network-layer differences in roundtrip-times when a user connects to the service using a proxy. We evaluate our four measurement techniques individually, and as an integrated system using a two-pronged evaluation. CalcuLatency is an easy-to-deploy, open-source solution that can serve as an inexpensive first-step to label proxies.

1 Introduction

Online advertising revenue in the U.S. reached over $209.7 billion in 2022, almost three times as much as the revenue from U.S. TV advertisements [1, 2]. Companies and service providers increasingly thrive on collecting, processing, and sharing large amounts of data on users, and monetize this data by enabling targeted ads and tracking. Reports have criticised leading companies such as Google and Facebook, calling them surveillance giants, and asserting that their business models are a threat to privacy and even to human rights [3].

With the growing awareness and demand for privacy among the general public, emerging tech companies are aiming to democratize the ad delivery ecosystem by building services that are privacy-focused and even share ad revenue with users. A plethora of companies offer ad and tracker-blocking for users regular browsing experience and provide alternative search engines geared toward privacy. Some also create solutions that promote user agency by allowing users to opt-in to viewing and earning rewards from privacy-preserving ads. On a technical level, such "ad reward networks" make use of browsing activity to generate ads but all the computational process occur locally on the user's device, without any data being exfiltrated to the companies. They then use privacy-preserving protocols to confirm ad event activity and reward users in cryptocurrency based on ad-providers and region-specific pricing. There are several companies that have similar rewards and provide remuneration to users on their platform for making use of their services [4, 5, 6, 7].

One critical threat to these novel types of reward networks is attackers who actively game the reward system by leveraging VPNs or proxies. These attackers fabricate their geolocation to gain access to more high-reward ads and falsify interaction activity to earn disproportionate rewards. Ensuring the long-term viability of these privacy-focused business models necessitates cost-effective and easily deployable techniques to differentiate regular users from those who use VPNs or proxies, for further monitoring for suspicious activity.

Balancing abuse-prevention techniques with rigorous privacy-requirements is a hard challenge, especially if the service provider seeks to uphold user privacy. The state of the art in generic VPN and proxy detection uses a series of heuristics developed by surveilling client traffic at large. Deployed by large platforms, these services use user data to build IP reputation metrics, generate user-specific metrics, and even use black box services that claim to detect "suspicious" users and IP addresses. But to get these metrics, the service provider will need to compromise on protecting the privacy of their users. In short, we seek to answer: *Can service providers build a system using minimum connection features, such as latency, to infer VPN or proxy use, without the need for data collection or jeopardizing user privacy?*

In this paper, we build and evaluate our solution to this question, CalcuLatency, that leverages cross-layer network latency

measurement techniques to differentiate users that are using a remote, long-distance VPN or proxy from those who are not. We leverage the fact that the application-layer latency when a user connects through a proxy is end-to-end (browser to server), whereas, any measurement on the network-layer only reaches the proxy, and the round-trip time (RTT) difference between the two can be a reliable indicator. To this end, we combine existing techniques such as WebSocket RTT, TCP handshake RTT, and ICMP ping, as well as implementing and evaluating a modified traceroute method (0trace), which has not been done before. 0trace conducts hop enumeration from within an *existing, established TCP connection* such as a web-socket session. Using CalcuLatency, we can even differentiate between network-layer and application-layer proxies.

To evaluate CalcuLatency, we evaluate each of the measurement techniques on their own and also evaluate the system as a whole by using a comprehensive (geo)diverse set of clients. The former is necessary to characterize and quantify, for each technique, the effects of network jitter and reliability of the methods. For the latter, we implement the system as a web service and conduct a two-pronged evaluation of CalcuLatency as a whole: (i) we perform an in-depth testbed evaluation where we maximize the number of VPN products, servers, protocols and browsers tested, from four different user geolocations; (ii) we expand this to include a real-world, crowdsourced evaluation to collect and analyze more diverse user geolocations. To this end, we rally user participation on Twitter and personal contacts, using the authors' accounts. We have participants from 37 different countries located in all (six) continents, 144 autonomous systems, and collect a large crowdsourced dataset.

We find empirically that a viable threshold to consider a particular client as a remote VPN or proxy connection is 50 milliseconds. In 98% of *all direct measurements* from both sets of evaluation, we find that the RTT difference is below this threshold of 50ms. Conversely, 89.1% of *all VPN measurements* in the testbed evaluation and 63.9% of the crowdsourced evaluation have an RTT difference of *above 50ms*. Through location analysis, we were able to attribute a majority of the remaining 10.9% and 36.1% VPN measurements to the fact that the VPN server and the user were located very close to each other.

Investigating the VPN measurements whose RTT difference was below 50ms, we find that in a majority of such cases (66.2%) the VPN server is located close to the user (within 650mi). This can be equated to straight line distance between Washington DC and Boston, MA (635mi) or the straight line distance between Mountain View, CA and San Diego, CA (685mi). Surprisingly, this indicates that we can reliably detect proxy use when a user and the VPN or proxy are just 650 miles apart or more.

Overall, we consider 50ms to be the RTT difference threshold for a connection to be labelled as a remote VPN or proxy, our system has a *false negative rate of 2.9%* (28/964) and a low *false positive rate of 0.95%* (2/210).

Adopting a more conservative analysis strategy, if we only consider measurements where ICMP ping is successful and 0trace reaches the client or its network (i.e. 96% of all measurements), we find that the RTT difference is below 50ms for the 210 direct measurements, and none were wrongly flagged using our system. These measurements include 137 unique user IPs from 84 different Autonomous Systems (ASes), from six different continents and over 34 different countries. We also reduce our false negatives to 2.77% (26/937).

While CalcuLatency cannot detect all proxy use especially if the user and VPN are close to each other, CalcuLatency is an easy-to-deploy, open-source solution that can serve as an inexpensive defense against proxy enabled abuse for server-side operators. We incorporate existing methods to calculate network latency and implement a modified traceroute method to overcome the challenges of our probes getting blocked due to stateful firewalls or NATs. Though this traceroute concept was first discussed over two decades ago, we are the first to implement, deploy in a real-world system, and evaluate its performance.

The following are our contributions in this paper: we propose a system, CalcuLatency, that combines existing building blocks and implements measurement techniques to help detect VPNs and proxies. We extensively evaluate our technique and elaborate on its calibration. We perform several controlled and crowdsourced experiments with our system deployed in real-networks and collect data to quantify thresholds and characterize the reliability of our techniques.

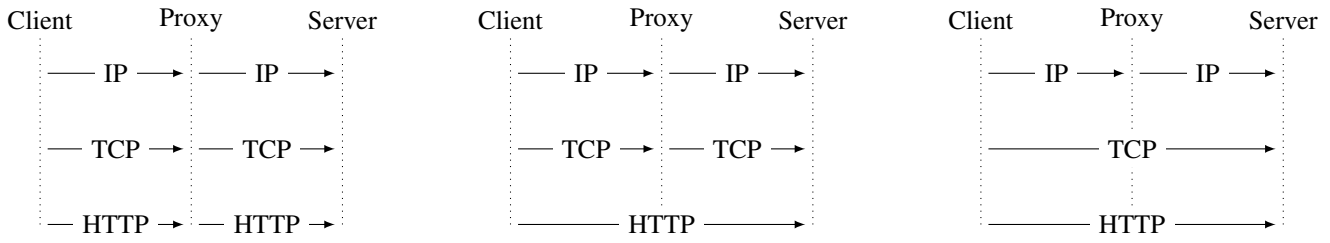
Roadmap: In Section 2, we discuss the background of our techniques and the architecture of proxies. In Section 3, we introduce our system architecture, followed by our techniques to measure roundtrip times, the implementation of CalcuLatency (§ 3.5), and our ethics discussion (§ 6). Next, we evaluate each of our building-blocks (§ 4), followed by our two-pronged evaluation of the system (§ 5.1, § 5.2). In § 7, we present our limitation, followed by related work (§ 8) and finally, we discuss CalcuLatency, its implications, and conclude in Section 9.

2 Background

Below, we discuss how network proxies differ in their architecture and we explain each of our measurement methods.

2.1 Architecture of Network Proxies

Figure 1 illustrates how a proxy's type affects the underlying protocol stack. We divide the protocol stack into three layers as per the OSI model: the application layer (HTTP), the transport layer (TCP), and the network layer (IP). Application-layer proxies terminate the application protocol, e.g., to inspect content for malware (Figure 1a). This includes Web



(a) Application-layer proxies, e.g., proxies that work via HTTP `CONNECT`.

(b) Transport- and Sessions-layer proxies, e.g., SOCKS, Tor, and SSH.

(c) Network-layer proxies, e.g., VPNs like OpenVPN and WireGuard.

Figure 1: **Architecture of Network Proxies**—We distinguish between three types of proxies that differ based on the layer at which they terminate client connections. Clients connecting to application-layer (1a) and transport-layer proxies (1b) first establish a TCP connection with the proxy and initiate the proxying using protocol specific messages. The proxies create new TCP connections to the web server, and then establish the tunnel. Whereas, network layer (1c) proxies authenticate the client and establish a tunnel, after which all packets including TCP SYN packets are encapsulated and proxied through the server.

servers that support the HTTP `CONNECT` method. Transport-layer proxies like Tor and SSH (both build on top of the SOCKS protocol) pass through the application protocol but terminate the client's TCP connection (cf. Figure 1b). Clients use SOCKS's signalling mechanism to tell the proxy what destination to connect to. Finally, network-layer proxies like OpenVPN or WireGuard perform NAT but pass through the client's TCP connection (Figure 1c).

2.2 The 0trace Technique

In a traceroute, a client sends multiple packets to a server, with each packet containing an incrementing time-to-live value (TTL) in the IP header. These packets are stateless "stray" packets, meaning that they do not belong to an open network connection. Stateful firewalls may reject such packets, terminating the traceroute.

In 2007, Zalewski invented a traceroute technique that can get past stateful firewalls [8]. This technique—called 0trace—creates trace packets that match the five-tuple of an already-established TCP connection. Unable to tell apart 0trace packets from packets belonging to this TCP connection, stateful firewalls will let 0trace's trace packets pass. 0trace achieves its goal by manipulating network packet headers and does so by crafting its own network packets using Linux's raw socket API. However, this techniques comes at the cost of potentially corrupting the TCP connection at the end: the receiving endpoint may terminate the TCP connection upon receiving an unexpected trace packet.

While 0trace's purpose is hop enumeration in the presence of firewalls, we use it for RTT measurements. The go-to technique for RTT measurements are ICMP echo requests (colloquially called "pings") but we found that many residential ISPs block pings (§ 3.4.1). While 0trace does not always work in such settings, it does allow for more accurate RTTs in the presence of firewalling as we show in Section 4.3.

2.3 The WebSocket API

The WebSocket protocol is an application layer protocol on top of TCP, which offers Web applications a bidirectional socket for communication. While distinct from HTTP, the WebSocket protocol is compatible with HTTP and uses the HTTP `Upgrade` header in its handshake. This allows Web servers to handle both HTTP and WebSocket connections on the same port. Once a WebSocket connection is established, client and server exchange binary data. Web applications can use WebSockets by taking advantage of the JavaScript WebSocket API that's supported by all modern browsers [9]. Later in this work, we use WebSockets to determine the application-layer round trip time between a client and server.

3 Method

Our aim is to measure round-trip times on the application, transport, and network-layers, and use a combination of these different measurements to reason about whether a particular connection is coming through a proxy server. We combine four cross-layer latency measurement techniques into a single software service that we call CalcuLatency. Our system integrates well into existing service provider infrastructure, and we create and publish a pipeline to analyze the collected data.

3.1 System Architecture and Assumptions

A typical client-proxy-service scenario consists of three entities: (i) a service provider that makes available one or more HTTP endpoints to its clients; (ii) clients that use the service provider's services; and (iii) proxy servers that some clients use to disguise their topological (i.e., IP address) and physical (i.e., their home country) location. While most clients are honest, there are some malicious ones that seek to defraud the service provider while using network proxies to disguise their

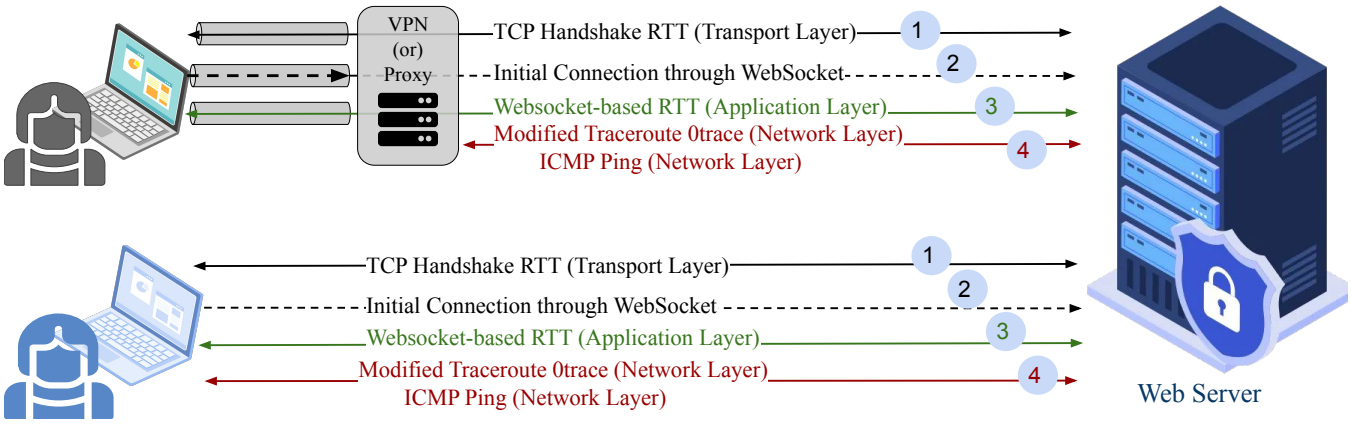


Figure 2: **Measurement Setup**—User connects to the Web server either via a VPN or Proxy (top) or directly (bottom). We illustrate the measurements done in both cases. The network layer measurements only reach the proxy server in case the user is using a proxy, but reaches all the way to the user's public IP if they are connecting directly.

location. For instance, an ad reward network or survey distribution service provides different rewards for geolocations, and users may seek to earn disproportionate rewards using proxies. Service providers need an inexpensive and practical solution that can tell apart clients that use remote proxies to mask their geolocation from those that do not, without user surveillance using blackbox detection techniques. We introduce CalcuLatency to fill this gap.

Broadly, our technique allows for the detection of the three proxy types illustrated in Figure 1. CalcuLatency's key insight is that the use of a proxy affects the layers in the OSI model differently. If we find a non-trivial difference between the application-layer RTT (Δ_{AL}) and the transport-layer RTT (Δ_{TL}) and/or the network-layer RTT (Δ_{NL}), we can conclude that the client is using a proxy server.

Consider a concrete example: A client in India is using a VPN server in Canada to make an HTTP connection to our CalcuLatency server in the U.S. Upon accepting the connection, our server now determines three types of round-trip times to the client, as shown in Figure 2. First, it upgrades the HTTP connection to WebSocket and sends several pings (using JavaScript) to determine the application-layer RTT. Next, our server inspects the (previously-recorded) TCP handshake to infer the transport-layer RTT. Finally, our server determines the network-layer RTT by sending ICMP echo requests and by running a 0trace measurement, which piggybacks onto the already-established HTTP connection.

Having determined all RTTs, our server notes that the WebSocket RTT is 250 ms—time taken for WebSocket pings to travel from the U.S. to Canada, to India, and back. The ICMP echo responses exhibit an RTT of only 35 ms—the time it takes to go from the U.S. to Canada, and back again. This difference manifests because the WebSocket ping was answered by *the client's browser in India* while the network-layer ping was answered by *the proxy in Canada*. Confronted with the RTT difference of $250 - 35 = 215$ ms, the service provider concludes that the client is using a proxy.

Assumptions: First, CalcuLatency requires an HTTP connection between the client and the server. We chose HTTP because of its ubiquity but other application-layer protocols work equally well. Second, the client's proxy must not be geographically close to the user. This assumption is reasonable in the case of service providers that seek to identify users who spoof their country of origin. Third, we assume that clients do not control the network behavior of the proxy and therefore cannot have it delay selected network packets. Instead, we assume that clients either use open proxies or rent them, e.g., as part of a VPN subscription. Even if the client does control the proxy, it is not guaranteed to evade CalcuLatency's detection as our 0trace component can estimate the RTT to the client using adjacent hops on the path that are outside the client's control.

While not universally applicable, we believe that these assumptions are both realistic and commonplace. What's more, CalcuLatency does not suffer from the shortcomings of existing, established proxy detection techniques like IP reputation blocklists that just use IP-to-Geolocation databases [10, 11, 12, 13].

3.2 Determining the Application-layer RTT

How can the service provider determine the application-layer RTT to the client? We chose WebSocket for this task: As of June 2023, WebSocket is supported by all major browsers [14], it is compatible with HTTP, and it is purpose-built for real-time communication, making it a natural choice for determining round-trip times. As mentioned above, CalcuLatency is

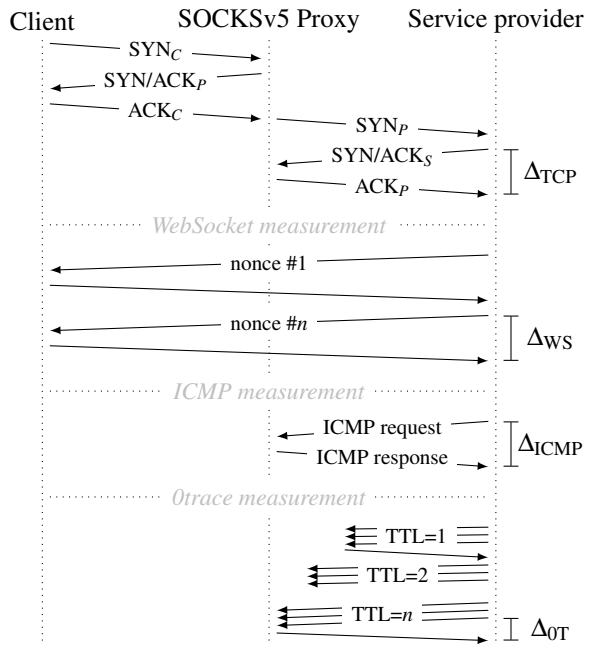


Figure 3: **Measurement Flow**—In this example, the client uses a SOCKSv5 proxy. The service provider determines four round-trip times (on three layers) to infer what kind of proxy the client uses.

not dependent on WebSocket and could employ alternatives, like WebRTC, XMPP, or HTTP page load times [15, § II.B].

Specifically, a client establishes a WebSocket connection with the service provider. The server then sends n WebSocket-based pings to the client as illustrated in Figure 3. Section 4.1 suggests that $n \geq 10$ is useful to make the measurement more robust against transient networking issues or asymmetric routing, both of which interfere with our measurement. In our CalcuLatency system, we send 100 WebSocket-based pings to the client. Upon receiving a WebSocket ping, the client's browser echoes back each ping to the server. Upon receiving the echo response, the server determines the WebSocket RTT on the Application Layer (Δ_{AL}) between itself and the client as follows: $\Delta_{AL} = \min\{\Delta_{AL_1}, ..., \Delta_{AL_n}\}$.

Importantly, our technique must work in an adversarial environment because malicious clients seek to disguise their use of a proxy. These clients control the server-provided JavaScript, and can therefore choose to either delay the echo or send an "anticipatory" echo before having received the corresponding request. The former is against a malicious client's interest because it would increase the odds of CalcuLatency concluding that the client is using a proxy. Malicious clients are therefore incentivized to respond as quickly as possible. To thwart the "anticipatory response" attack, we include a unique nonce in every echo request. The client has to embed the nonce in its echo response, preventing it from sending responses before having received the request.

3.3 Determining the Transport-layer RTT

We take advantage of the fact that in our setting, the client establishes a TCP connection with our server, meaning that we are in control of the TCP three-way handshake. We estimate the RTT between client and server by calculating the time difference between the server responding with a SYN/ACK segment and the client acknowledging receipt with an ACK segment, as discussed by Ding and Rabinovich [16]. We refer to this RTT as Δ_{TL}. Again, clients have no incentive to delay their ACK segment and are unable to send an "anticipatory" segment because of the difficulty of predicting TCP sequence numbers. We find that Δ_{TL} is reliable and straightforward to determine but we have *only a single sample* per connection, which makes this technique susceptible to transient conditions like network congestion. Indeed, Høiland-Jørgensen et al. [17] showed that "20% of the clients experience increased delays of more than about 80 ms, at least 5% of the time."

We considered taking advantage of the TCP timestamp option to measure RTT. We chose not to because TCP timestamps are not universally used [18, § 6] and often exhibit poor granularity—Veal et al. showed in their PAM'05 paper that the majority of 500 popular web servers used a granularity of either 10 ms or 100 ms [19, § 3].

3.4 Determining the Network-layer RTT

Finally, we focus on the lowest layer for which we determine the round-trip time: the network layer. Determining the RTT to an IP address is challenging because it is difficult to reliably get a remote network stack to respond to unsolicited IP packets. Prior work reported that it is increasingly rare for hosts to respond to ICMP echo request, or send a TCP RST segment in response to unsolicited SYN segments [20, § 4]. To maximize success, we draw on two separate techniques to estimate Δ_{NL}, our network-layer RTT: ICMP (see § 3.4.1) and 0trace (see § 3.4.2).

3.4.1 ICMP RTT

CalcuLatency sends five ICMP echo requests to the client. We refer to this measurement as Δ_{ICMP}. Prior work had found that some proxies don't answer to ICMP: 90% of VPN servers tested by Weinberg et al. [10] reportedly ignored ICMP requests. However, they only tested servers belonging to seven undisclosed VPN providers. Due to ICMP's potential usefulness to CalcuLatency, we revisit this topic by asking the following research question:

Do VPN servers respond to ICMP pings? To answer this question, we sent ICMP requests to IP addresses belonging to 80 VPN providers that past work studied [21, 11]. These VPN providers host servers that run various VPN protocols including OpenVPN, WireGuard, and other proprietary protocols. We augment our list of IP addresses with addresses present in the same /24 network as the VPN IP addresses, resulting in a

list of 492 unique IP addresses. We chose to ping adjacent addresses in CalcuLatency because a response from an adjacent address is likely to have a near-identical RTT to that of the actual proxy, which may not respond to ICMP requests. Our ICMP requests to these addresses resulted in 369 addresses (75%) that responded—a higher percentage than what past work found.

VPN servers appear likely to respond to ICMP requests but what about a random sample of the IPv4 population? To answer this question, we ran a ZMap scan [22] targeting a randomly-selected set of 1 million IP addresses. This resulted in 10.52% of IP addresses that responded to our ICMP requests. This is in line with prior work by Bano et al., which found that approximately 10% of IP addresses respond to ICMP pings [20, § 4.1]—making the odds of a response low.

We conclude that VPN servers are significantly more likely to respond to ICMP requests than randomly-selected IP addresses, which includes residential clients that don't use a proxy. We therefore decide to augment our ICMP measurement with 0trace, which we discuss in the next section.

3.4.2 0trace RTT

We use Zalewski's 0trace technique [8] for the purpose of determining the round-trip time between the server (where the 0trace measurement is initiated) and a connecting client. Again, we take advantage of the fact that our setting has the client establish a WebSocket connection to the server, meaning that we have an already-established TCP connection that 0trace can use for its TTL and RTT measurement.

We developed a Go package that implements 0trace. This package uses Linux's raw socket API to send manually-crafted TCP segments with incrementing IP time-to-live (TTL) values. We carefully craft 0trace packets to match the TCP five-tuple of the WebSocket connection,[1] so that firewalls between client and server will not interfere. Our 0trace implementation keeps incrementing the TTL until either (i) the responding IP address is identical to the client's IP address; or (ii) until TTL 32 is reached. For each TTL, we send three redundant probes to account for potential packet loss [23]. If we don't receive a response to any of our three probe packets within five seconds, we mark the given TTL as unresponsive. The RTT of our 0trace measurement (Δ_{0T}) is the round-trip time of the probe packet *that made it the farthest* to the client, i.e., the probe packet with the largest TTL.

We specifically use 0trace because it can get past (CG)NAT and stateful firewalls by operating on existing TCP connections. Furthermore, even if 0trace measurements do terminate at the NAT-box due to network level rules, we argue that the distance between a NAT and the user's machine is less than 1,000 km as most are typically deployed by their ISP [24].

[1]The five-tuple consists of IP source and destination addresses; TCP source and destination ports; and the IP protocol.

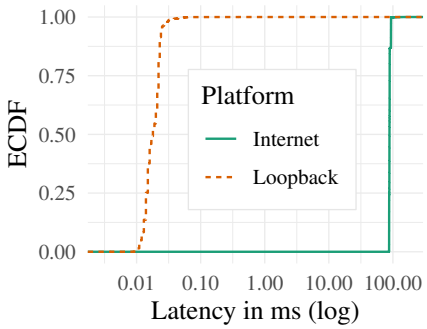


Figure 4: Latency distribution of TCP handshake measurements over two networks.

Being equipped with four techniques that measure the RTT across three OSI layers, we now devise a decision tree that helps us collapse all our measurements into a single verdict: does the client use a proxy?

3.5 Implementation and Deployment

CalcuLatency combines our application, transport, and network-layer measurements into a single service. We implemented CalcuLatency in both Go and JavaScript (WebSocket pings). The Go service consists of (i) a Web server with an endpoint that speaks WebSocket, (ii) a Go package that records TCP handshakes to extract round trip times, and (iii) a package that runs a 0trace measurement to the client.

While developing our 0trace Go package, we take advantage of the ICMP error requirements stated in RFC 1812 and RFC 792, which state that the returned ICMP error message will contain (parts of) the original datagram's data [25, 26]. We find that the ICMP error message reliably contains the IP header of the original datagram in the ICMP datagram. Hence, we design our server to keep track of the IP ID in for each TTL-limited probe it sends out, and it uses the IP ID obtained from the IP header of the original datagram from the received ICMP error packet to correspond the hop IP to the particular TTL value. Using the packet sent time and packet received time, the server calculates the RTT for each hop that responds with the ICMP error. It repeats the process until (if) it reaches the VPN IP, or until the maximum TTL value. All our code is available online under a free software license but we omit the URL to preserve our anonymity.

In real-world systems, CalcuLatency can be deployed as part of a CAPTCHA. The CAPTCHA can be non-interactive and consist of HTML and JavaScript, which initiates a WebSocket connection to the CalcuLatency server. In the first phase, CalcuLatency sends hundred WebSocket pings to the client to estimate the application-layer RTT. After CalcuLatency determines the application-layer latency, we calculate the RTT of the TCP handshake, send ICMP pings, and use 0trace to determine the network-layer RTT, as discussed

above. Clients that CalcuLatency deems to be using a proxy can be flagged for manual inspection.

4 Building Block Evaluation

Before evaluating CalcuLatency in its entirety, we study its building blocks in isolation.

4.1 Reliability of WebSocket Pings

Unlike our network-layer RTT methods, which are typically handled by the kernel's network stack, our application-layer method—WebSocket—traverses not only the kernel's network stack but also the client's browser, and is therefore subject to more jitter. To characterize and approximate this jitter, we built an HTTPS-based Web service that sends 10,000 sequential WebSocket echo requests to the client and measured the round-trip time $\Delta_{WS_1}, \ldots, \Delta_{WS_{10,000}}$ for each request. We made these measurements over the loopback interface to eliminate networking delays and isolate processing delays.

Figure 11 in the Appendix illustrates the results for two consumer laptops (ThinkPad X1, MacBook Pro) running two browsers. The median RTT for all distributions is less than 1.4 ms, and 99% of RTT measurements completed in less than 2.4 ms. All distributions exhibit a long tail, presumably due to transient load spikes. *CalcuLatency must account for this by running multiple measurements, spaced out over time.*

4.2 Reliability of TCP Handshake RTT

We conduct an experiment to understand the reliability of the TCP handshake in measuring RTT. We used Go to build an HTTP server that served a static index page. In parallel to serving this index page, the server uses libpcap to record the TCP handshake; in particular the time delta between the server responding to the client's SYN with its SYN/ACK segment and the subsequent receipt of the client's ACK. We built a shell script that establishes 86,400 TCP connections: one connection per second for 24 hours. We ran this experiment in two network settings: in the "Loopback" setting, client and server run on the same machine, communicating over the loopback interface. By excluding the Internet, this setting highlights computational latency. In the "Internet" setting, client and server run on separate machines, communicating over the Internet. The client ran on a VPS in Ohio, U.S. while the server ran on a VPS in Paris, France.

Figure 4 illustrates the results of this experiment. In the "Internet" setting, we see a minimum, median, and maximum latency of 87, 88, and 171 ms, respectively. 99% of handshakes exhibit a latency of less than 94 ms—only 7 ms more than the minimum. The "Loopback" setting, we observe a median latency of 18 µs. 99% of handshakes exhibit a latency of less than 34 µs. As expected, we observe a long tail in both and we account for outliers by running repeated measurements.

The TCP RTT is readily available and straightforward to calculate, but we only get a single measurement per TCP connection. One could work around this limitation by having the client establish multiple TCP connections to the server.

4.3 Reliability of 0trace Pings: Considering the Variance of Latency Across the Internet

To evaluate the reliability of 0trace, we conduct a large scale measurement with endpoints having a variety of configurations to understand potential effects on the results. For this experiment, we take advantage of the RIPE Atlas network.

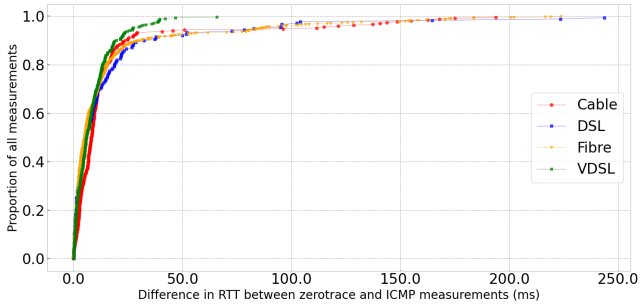
First, we set up an HTTPS server that runs our 0trace code whenever a client connects to the server. Next, we instruct the RIPE Atlas probe to run an **ICMP** and a **TCP traceroute** using both the IPv4 and v6 addresses towards the server, and an **"SSL" measurement** to fetch our server's TLS certificate, which trigger our 0trace measurement toward the client's IPv4 address. For each RIPE Atlas probe, we measure three different RTT values (0trace and two traceroute RTTs). We repeat each of these measurements three times to account for any transient errors. We then determine the absolute difference between the 0trace RTT and the other traceroute RTTs, which serve as ground truth in each case. Appendix B contains a breakdown of the available RIPE probes and our selection.

Our initial experiments to explore whether running latency measurements at different times of the day influence the measured RTT difference (presented in Appendix C and Figure 13) revealed no significant differences even when the RTTs were measured every 3 hours. *To maximize the utility of our measurements considering concurrency limits and resource constraints, we trade-off the small benefit in reliability of running multiple experiments per day, for the diversity of a larger number of RIPE probes over a long period of time.*

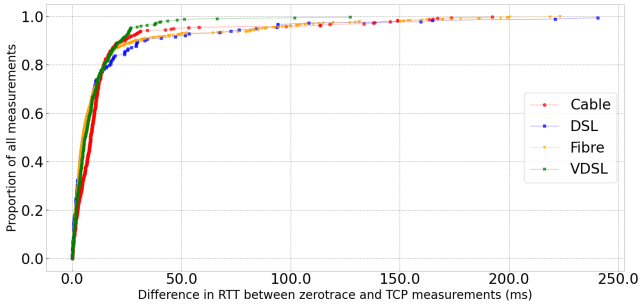
For our measurements, we selected probes that are "connected" (not abandoned or disconnected), with a public IPv4 and IPv6 address, valid country code, with all the desirable tags. Note that while we request all eligible probes to participate in our measurements, not all of them do [27]. **We had 2,350 RIPE Atlas probe IP-probe ID pairs** that fully completed the SSL, ICMP and TCP traceroute measurements, i.e. they either reached our server directly or their last hop IP address was in the same AS as our server. The distribution of these probes over continents, ASNs, and different access technology is described in the Appendix B.

Differences in local and last-mile access technology We evaluate our hypothesis of whether the probe's last-mile access technology (derived from the probe's tags) has large effects on the latency seen from the server side.

We present our results for the RTT difference between the 0trace result and the probes' TCP and ICMP traceroutes separately. We find that VDSL has three outlier RTT differences



(a) Difference between 0trace-determined RTT and RIPE Atlas-determined ICMP RTT in ms.



(b) Difference between 0trace-determined RTT and RIPE Atlas-determined TCP RTT.

Figure 5: **Comparing the RTT differences between probes with different local last-mile access technology—** In 80% of *all measurements* the RTT difference is less than 18ms. However RTT differences in TCP are lower and more stable. Removing three outliers from VDSL, we see that the differences between the access technology become more pronounced for the remaining 20% of the values, with cable performing worse, and Fibre performing marginally better with no outliers.

measuring 388ms, 1970ms, and 2640ms and upon investigation we note that these were due to two German RIPE probes in AS8881 and AS3320 which failed to reach far in the traceroute. To improve readability of our figure, we have removed these three cases and note them separately in Figure 14 in Appendix F. From Figure 5, we see that 95% of all ICMP measurements have an RTT difference lower than 111ms in Cable, 94.93ms in DSL, 86ms in Fibre[2], and 31.3ms in VDSL. Similarly, 95% of all the TCP measurements have an RTT difference lower than 46.3ms in Cable, 93.8ms in DSL, 87.9ms in Fibre, and 30.7ms in VDSL (including outliers).

In summary, we see that the RTT differences in TCP are lower and more stable, that is, 0trace measurements are closest to the TCP ground truth RTTs. We note that in 80% of *all measurements* the RTT difference is less than 18ms. We see that the differences between the access technology become more pronounced for the remaining 20% of the measurements, with cable performing worse, and Fibre performing slightly better than others considering it has no outliers.

Our experiments show that given that 0trace is able to match ground truth for >80% cases, and in 95% of cases has a maximum difference of 111ms in RTT when compared with ground truth. Given the results of our experiment, we can reason about the generalizability of our measurements. We can be confident that when measuring long distance proxies, these differences of 111ms will not hinder CalcuLatency's ability to detect the proxies, and in the remaining 5% of cases, it may inflate the endpoint RTT, which may lead to false negatives that CalcuLatency can tolerate.

5 Evaluating CalcuLatency in Practice

We conduct a thorough evaluation of CalcuLatency using a two-pronged approach. First, we conduct an evaluation with a controlled testbed where we control the webserver, the client, and test our system with a variety of different VPN products and proxy servers, that run multiple different protocols. We test the system with mobile connections, various different user locations, without any proxies connected, and then by connecting to various VPN and tunnelling protocols. Next, we extend on this experiment by conducting a large evaluation with volunteer Internet users. This mimics a real-world deployment test for CalcuLatency because we draw on a geographically diverse population that's representative of the average Internet user. For this experiment, we set up our measurement server at a university network and advertised our crowd-source measurement page via social media channels.

5.1 Control Testbed Evaluation

We conduct a controlled evaluation of our CalcuLatency service by creating a testbed and collecting various measurements ourselves. In our experiment, we controlled the webserver that conducted the measurements. We had teammembers run tests from their devices from fifteen different home and mobile networks using four different popular browsers—Chrome, Mozilla Firefox, Safari, and Brave.

Our network locations include three cities in the U.S. and one city in Canada, UAE, and India each from multiple networks. We conduct both direct measurements from these networks and also run measurements while connecting to various different VPN and proxy servers around the world. We instrumented an automation script using Selenium to trigger measurements from multiple different browsers on a computer at the same time. In both the direct connection case and when we turned on a VPN, we use this automation script to conduct multiple experiments from different browsers towards the same server. We re-ran a direct measurement each time we started experiments, from each browser, and repeated

[2]We are preserving the spelling used in the RIPE Probe Tag

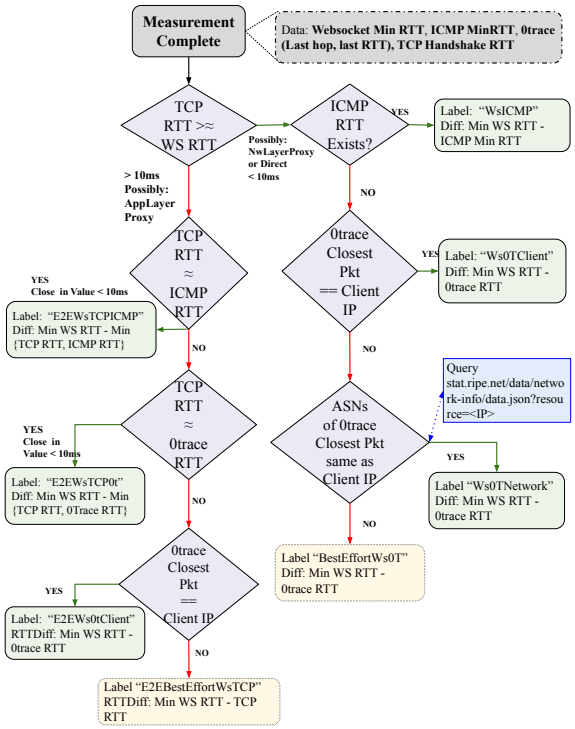


Figure 6: **Decision Tree—**We create a decision tree that uses the round-trip times measured using our four different techniques. Based on their results, we use a combination of these values to decide whether a particular experiment can be labeled as a possible proxy connection. The two yellow nodes refer to cases that lack some of the measurement results and are hence, labeled "best effort" calculations.

it between different experiments. The VPNs and proxies used to conduct these experiments include ten popular, commercial VPN providers including Astrill VPN, CyberGhost VPN, ExpressVPN, IPVanish, IVPN, Mozilla, Mullvad, NordVPN, Private Internet Access, Surfshark. These VPN providers offer multiple different protocols such as Wireguard, OpenVPN, and other proprietary protocols such as OpenWeb and Lightway. We also instrumented our own SOCKS5 proxy servers and tested them as well. We do not measure or reason about security and privacy aspects of the tested commercial VPN providers; our aim is to simply connect to various servers they offer and evaluate our CalcuLatency measurement service.

5.1.1 Data Characterization

In total, we conducted 891 unique experiments with 354 unique client IPs. Our measurements came from 337 unique VPN or proxy server IPs belonging to 82 different autonomous systems (ASes), and 17 unique direct, client IPs belonging to 12 different ASes. Overall, we collected data from four different countries.

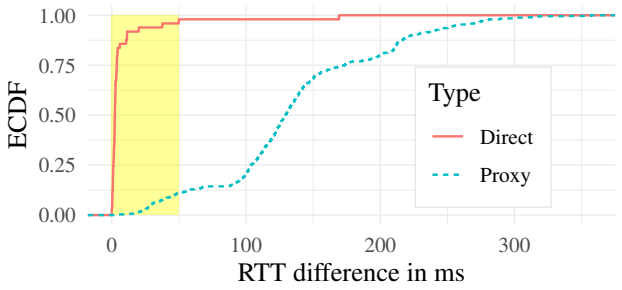


Figure 7: **ECDF of RTT difference for the 891 control, testbed evaluation measurements**—From our measurements, we find that over 98% of direct measurements and less than 11% of VPN measurements have an RTT difference below 50 ms (shaded in yellow).

5.1.2 Results

We calculate the RTT difference for each experiment using a decision tree that we devised (Figure 6).

First, we check if the TCP handshake RTT (Δ_{TL}) $\gtrapprox$ websocket RTT (Δ_{AL}), i.e. if they are within $\pm$ 10 ms. If so then the experiment is a network layer proxy or a direct measurement. In this case, the Δ_{TL} cannot be used for calculating the RTT difference. Hence, we use the ICMP RTT, if it exists, and if not, we will use 0trace RTT as Δ_{NL}. In order to differentiate and identify which value was used as Δ_{NL} and to record the conditions that are met according to the decision tree, we assign labels for each end node of the tree. For instance, in this case, if ICMP RTT does not exist (i.e. ICMP was not supported by the client IP), we will use 0trace RTT and assign different labels based on whether 0trace reached the client IP (*0TClient), 0trace did not reach the client but reached the same ASN as the client (*0TNetwork), and label it a "best effort" if none of these conditions is met. We only use the 0trace RTT from the last successful hop for the calculation.

However, if $\Delta_{TL} \lessapprox \Delta_{AL}$, then the experiment could be an application layer proxy. In this case, we check if $\Delta_{TL} \approx$ to either of the Δ_{NL} (ICMP RTT, 0trace RTT). If not, we follow the same procedure as above and use ICMP RTT if it exists, and if not, fallback to 0trace if it reaches the client IP or its ASN. Finally, if none of those conditions are met, we label it a "best effort" and use TCP RTT to calculate the RTT difference.

In our control testbed evaluation, **we find that in 98.0% of the direct measurements (48 of 49) the calculated RTT difference is less than 50 ms**, as shown in Figure 7. Upon investigating, the anomalous direct measurement belonged to an Indian mobile network provider, where the ICMP failed and 0trace was unable to reach even the same network, hence it was a "best effort" calculation. We will revisit the "best effort" cases in §5.3.

Next, we investigate the VPN measurements, we find that the RTT difference is above 50 ms in 89.1% of all VPN mea-

surements (750 of 842), shown in Figure 7. Among the rest of the 10.9% of VPN experiments, we find that in 60.9% of them (56 of 92), the VPN server is close to the user geographically, i.e. the straight line distance between user and VPN server is less than 650 miles. This is equal to the straight line distance between Washington DC and Boston, MA (635 miles) or the straight line distance between Mountain View, California and San Diego, California (685 miles). This is expected, as specified in our assumptions, our technique primarily seeks to identify longer-distance remote VPN users.

We investigated the 14 unique VPN server IPs belonging to 36 of 92 measurements (39.14%) where the user to VPN distance was above 650 miles. We found that six of those 14 IPs belonged to AS9009 (corresponding to 17 of 36 measurements), and for all these IPs, the VPN provider claims their location to be in a Latin American country (Mexico, Costa Rica, The Bahamas, Venezuela, Chile, Argentina). However, when we investigated their ICMP RTTs as measured from our server in the U.S. Midwest region, we find that all their RTTs ranged between 28 ms–52 ms. We used the speed of the Internet approximation provided by Katz-Bassett et al. [28] which is $\frac{4}{9}$ c, where c is the speed of light traveling in a vacuum, and find that for all of these experiments, **the locations of the VPN server as reported by the VPN provider is an impossibility**. Their true geolocation appears to be much closer than that what is reported.

For the remaining 19 experiments with eight unique VPN server IPs, we were not able to disprove the advertised location of the server with the RTT measurements. However, five VPN servers belonged to the same VPN provider which may have a policy to inflate ICMP values. The other two IPs did not have a reliable ICMP value for us to confirm or disprove their advertised location.

Overall, we find that 50 ms is a viable threshold for the RTT difference to distinguish a direct measurement from one coming through a remote VPN or proxy. We expand on this evaluation by conducting a real-world evaluation to increase the diversity of our "direct" connections and user locations.

5.2 Real-world Crowdsourced Evaluation

Next, we conduct an evaluation of our system in practice, using real-world measurements. We developed the CalcuLatency system, created a web server, and deployed it on a subdomain of our university. To obtain ground truth about the measurements, we create a form that users fill out to give us information regarding their setup. We ask users to optionally provide us with an email with which we can reach them, whether they are connecting to us though a mobile or a desktop, who their internet service provider is, and their current location (to reason about the measured latencies and physical distance). We then ask users if they are connecting to us via a VPN, or directly. If they are using a VPN, we request them to enter their VPN server location if known, and any details about their VPN provider.

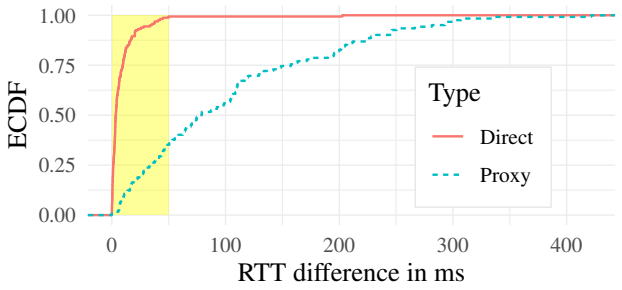


Figure 8: **ECDF of RTT difference for the 283 public crowdsourced evaluation measurements**—From our collected measurements, we find that over 98% of direct measurements and 36.1% of VPN measurements have an RTT difference below 50 ms (shaded in yellow).

We recruited users by publicizing a call for participation on Twitter using the authors' twitter handles and collected data for this experiment over a period of 15 days. We did not target or advertise towards specific Twitter users and chose not use platforms like Prolific or MTurk as they have not been evaluated to be generalizable for network measurement (but rather for qualitative studies).

5.2.1 Data Characterization

We collected 283 unique experiments from 252 unique client IPs, with (self-reported) 161 direct measurements and 122 VPN measurements. Our 161 direct measurements came from 145 unique client IPs from 93 different autonomous systems (ASes). Our 122 VPN measurements came from 109 unique VPN IPs belonging to 51 different ASes. The most number of measurements for the direct measurements came from ASN 7922, followed by ASNs 701, 7018, 21928 and 3320 belonging to Comcast, Verizon, AT&T, T-Mobile, and Deutsche Telekom. We provide a more detailed breakdown of these IPs by ASN in the Appendix E. Overall, we collected data from over 37 different countries from all (six) continents.

5.2.2 Results

Based on the same decision tree explained in §5.1 and Figure 6, we calculate the RTT difference and label each experiment with the appropriate label from the tree. **We find that 98.8% of all direct measurements have an RTT difference below 50 ms** (159 of 161) as shown in Figure 8, following the same trend as our controlled testbed evaluation, despite a large increase in the variety and diversity of our collected direct measurements. Upon investigating the remaining 1.2% (2 of 161) measurements, we find that one measurement was conducted through Safari and indicates that the IP is an iCloud

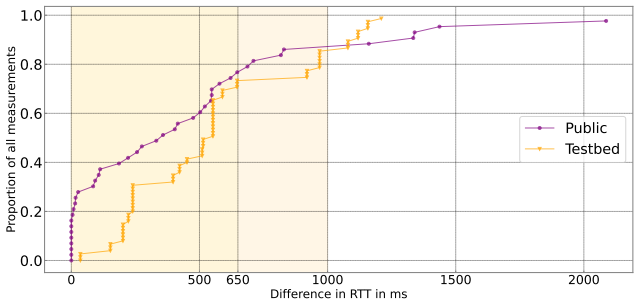


Figure 9: **CDF of the distance between user and VPN when RTT difference < 50ms**—Majority of VPN experiments with a low RTT difference (below 50ms) are located close to the user. 66.2% of these cases, the proxy is only upto 650 miles away from the user (in yellow), and 89% of these cases, the proxy is < 1000 miles from the user (in orange).

Private Relay IP, we conclude that the user may have overlooked the fact that private relay was turned on, and hence this is a "proxy" measurement. The other experiment had an abnormally large 0trace RTT value, labeled a "best effort" calculation (We explore this in §5.3).

Next, we investigate the VPN measurements, the RTT difference in 63.9% of measurements (78 of 122) are above the threshold of 50 ms, also shown in Figure 8. Of the 36.1% VPN measurements whose RTT difference is below 50 ms, we find that in 77.3% of the cases (34 of 44), the user and the VPN server are below 650 miles apart, which we consider as the minimum threshold for it be a "remote proxy", as shown in Figure 9. We use the locations provided by the user in our form and also compare the VPN IP-geolocation. Another 2.2% (1 of 44) was an experiment mislabeled as a VPN measurement, although the client IP indicates it is in the same location as the user. Of the remaining 20.5% of measurements (9 of 44), we did not see any proof to disprove the apparent location of the VPN.

5.3 Results Combining the two Evaluations

Overall, if we consider 50 ms to be the RTT difference threshold for a connection to be labelled as a remote proxy, our method has a low *false positive rate of 0.95%* (2/210 direct measurements). On the other hand, we find that 14.1% (136 of 964) VPN measurements had an RTT difference below this threshold. However, only 2.9% (28/964) of VPN measurements are located over 650 miles away from the user, and are legitimate (i.e. excluding VPNs with fake locations, and mislabeled experiments). Thus, we consider 2.9% our method's *false negative rate*.

What percentage of VPN servers respond to ICMP pings? In our evaluations, we collected a total of 434 unique VPN IPs, and we find that over 94.2% of these IPs (409/434) responded to ICMP pings. A majority of the VPN server IPs that did not respond to ICMP pings belong to Astrill VPN (14/25), which may have a policy for some/all of its servers to not respond to ICMP pings. Of the rest, four VPN servers belonged to personal OpenVPN VPN servers, three other servers others belonged to SOCKS5 proxies, and the remaining four were miscellaneous VPN services including iCloud Private Relay.

Does removing all best-effort calculations improve our method? Yes! Combining both sets of evaluation, we see that exactly 4.0% of all measurements (47/1174) was labeled "best effort calculations", which means that the experiment did not contain successful ICMP measurements, and that TCP RTT measurement (Δ_{NL}) is too close in value to Δ_{AL}, and 0trace did not reach the client or even the same network as the client IP. Since these anomalous measurements only comprise 4% of all measurements, we do an investigation into what our analysis would look like if we removed these measurements. When a service provider deploys CalcuLatency, they can achieve this by simply labeling all "best effort" experiments as requiring a re-run and have the client conduct another round of measurements.

From Figure 10, we see that a 100% of all direct measurements have an RTT difference less than 50 ms, and 86.2% of VPN measurements (808 of 937) have an RTT difference greater than 50 ms. Of the 13.8% (129 of 937) VPN measurements that have an RTT difference below 50 ms, we find that 66.7% of measurements are less than 650 miles away from them user and another 13.2% advertise fake VPN locations as we found in §5.1. The remaining 26 VPN experiments whose RTT difference is less than 50 ms but distance from the user is more than 650mi are false negatives, and the average distance between the VPN server and user in these cases is 1089 miles.

In summary, in a more conservative analysis setting where we only consider the experiments where we obtain all measurement data points (96% of all collected measurements), *we do not find any false positives and false negative rate is 2.77% (26/937).* In production systems, the experiments that we exclude in this analysis, i.e. those that are labelled "best-effort", can be marked by service providers for deeper inspection and for re-running of the experiment to remove transient issues.

6 Ethics

Before running a public, crowd-sourced evaluation of our system detailed in § 5.2, we contacted our institution's IRB and were informed that our project is exempt from regulation. Our crowdsourcing evaluation web service presents a input form and did not trigger any measurements until the user reads, inputs data about the measurement, and consents to the measurements. The index page also outlines the measurements conducted, data collected, and that any latency data collected may possibly be published to help future research, after anonymizing the last octet of each visitor's IP address. From the web service, we measure the user's latency using

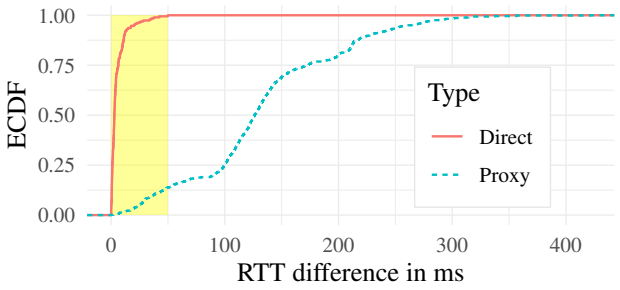


Figure 10: **ECDF of RTT difference of 1127 reliable measurements**—This analysis contains measurements that did not have a "best effort" label from our decision tree. 100% of all direct measurements and 86.2% of all VPN measurements have an RTT difference below 50 ms (in yellow).

four different methods and any data collected is only accessible to the authors of this paper.

We acknowledge that there are potential misuses of our CalcuLatency system that server-side operators and service providers could use to penalize VPN users. However, services currently employ black-box techniques to do similar blocking that maybe even more discriminatory. Our solution aims to identify users that connect to far-off, remote VPNs specifically to abuse systems, and will serve as one of several anti-fraud identifiers. Previous work has found that a majority of VPN users use it for privacy and security purposes [29]. Such legitimate VPN users can still connect to servers closer to themselves to enjoy the privacy benefits of the VPN and still evade our remote proxy detection.

7 Limitations

CalcuLatency cannot reveal proxy users in all cases. We can only detect proxy users if the proxy is sufficiently far enough from the user, from our measurements this distance appears to be at least 650 miles or more. Otherwise if the user and proxy are closer, the RTT difference may be too small to detect the proxy. This may be a prohibitive limitation for some service providers, and a non-issue for others because users may deliberately choose proxy servers that are geographically far away from them, to disguise their physical location. For example, users defrauding rewards networks want access to higher-reward locations, or users of streaming services often use VPNs to appear to be in another country, to get access to geo-restricted content. Each service operator can decide based on their business needs if they are conservative (treat ambiguous users as direct users) or be more aggressive (treat ambiguous users as proxy users). The service provider can also choose to subject ambiguous users' connections to further tests to confirm observations.

Our technique becomes less accurate in the presence of highly restrictive firewalls that discard all ICMP traffic. For perfect accuracy, we need to receive ICMP error packets from the client itself. The farther away we are from the client, topologically, the less accurate our RTT measurements become. While such restrictive firewalls do exist, our results suggest that they are the exception rather than the rule. We find from our evaluation that 94.2% of all VPN or proxy server IPs respond to ICMP pings, and over 56% of direct connections respond to ICMP pings, which is much higher than reported before. Additionally even if direct ICMP pings are disallowed, 0trace can function as long as the firewall does not discard ICMP error responses.

If a server-client pair uses the Multipath TCP (MTCP) protocol, it could reduce the effectiveness of CalcuLatency, due to the fact that this protocol uses multiple network routes concurrently within the same TCP connection, which could cause skewed results when comparing the 0trace RTT measurement to the application-layer RTT measurement since they could have traveled different paths. However, Shreedhar et al. determine that MTCP usage is extremely low compared to TCP (0.5%), and the vast majority of MTCP traffic is from a single company (Apple), which means that CalcuLatency would still be viable in the vast majority of use-cases [30].

Finally, we also present an analysis of how latency measurements can vary when it comes to dual-stack endpoints, where IPv6 connections to the server side can influence a bloat in the measured latency differences. We present an analysis using RIPE Atlas probes in the Appendix D. However, CalcuLatency only supports IPv4 based 0trace measurements, and we leave the optimization of IPv6 based latency calculation for detecting proxy-enabled abuse for future work.

8 Related Work

We provide an overview of related work on latency measurement techniques and its applications, Internet liveness tests, and proxy detection methods.

Latency Measurements: In a 2021 blog post, Tschacher writes that using in-browser measurements and on the server side by measuring RTT of the incoming TCP/IP handshake, website owners can infer that a visitor is using a proxy using the latency difference [31]. However, the blog post conducts a limited evaluation and relies only on one incoming TCP/IP connections and the handshake to identify RTT which may be affected by packet loss and transient factors.

Weinberg et al. [10] used ping-time measurements to hosts in known locations to estimate the locations of 2,269 proxy servers. They also found that over 90% of VPN servers and their first hop routers tested ignore ICMP pings and do not send time exceeded packets. Hopper et al. [32] estimate that an Internet host in 2007 can be uniquely identified by knowing its RTT to five randomly chosen hosts. Pelsser et al. [33] studied whether ICMP ping provides a good estimation of

delay and find that from an application perspective, ICMP ping actually gives a poor estimate of the delay and jitter as it can vary between flows.

Jiang et al. propose two techniques that a passive in-path monitor can use to measure the TCP round-trip time including the handshake [34]. Unlike this work, CalcuLatency has access to bidirectional flows and can analyze all three segments of the TCP handshake. In 2020, Livadariu et al. found that using RTT measurements from probing an IP from multiple vantage points (including from within its own AS) is a more viable strategy than using geolocation databases [35].

Cloud Providers: Dang and Mohan [36] investigated latency specifically towards cloud service providers in 2021, with a focus on developing regions. They found that geographical distance has a high impact on latency towards cloud systems. Kashaf et al. [37] determine that a majority (95%) of websites that are hosted in or serve traffic primarily to Africa rely on third-party cloud infrastructure. These two conclusions support the efficacy of CalcuLatency in lower traffic regions. Dang and Mohan conclude that the last-mile technology used is the greatest factor for latency when reaching a cloud provider, but the results from our RIPE Atlas measurements show that the last-mile technology does not have a large impact on the measured RTT.

Internet Liveness: In their CCR'18 paper, Bano et al. perform Internet-wide scans to study the population of addresses that respond to probes [20]. They found ICMP probes are the most effective to discover alive hosts but TCP probes can further add to the population of alive hosts. A combination of ICMP, TCP, and (to a lesser extent) UDP results in the most complete picture of liveness. Bano et al.'s results inform how CalcuLatency attempts to determine an address's network-layer or transport-layer latency.

Proxy Detection: Much work has focused on detecting proxies; mostly on detecting Web proxies, VPNs, and Tor. Weaver et al.'s PAM'14 paper studied the prevalence of in-path Web proxies by sending controlled application-layer measurements between clients and a server, controlled by the researchers using Netalyzr [38]. In an S&P'19 paper, Mi et al. studied the emerging ecosystem of end user systems that serve as proxies to others—often without the knowledge of the owners of the proxies [39].

Hoogstraaten [40] explored several server-side VPN detection methods, such as using existing IP information databases (WHOIS, rDNS), fingerprinting TCP options like advertised MSS, and even timing based measurements. But they only propose limited latency based measurements to identify internet-layer proxies, and conducted a short proof of concept. The closest related work is by Webb et al. [15] who also proposed detecting proxies and VPNs based on traffic timing and latency. They measure the RTT for each connecting IP address and flag anomalies in the distribution of these RTTs as proxies. However, their method must be trained for each IP address, and hence is not practical.

9 Discussion and Conclusion

In this paper, we present and evaluate our system CalcuLatency that incorporates several network RTT measurement techniques and leverage the application-layer and network-layer differences in roundtrip-times when a user connects to a service using a proxy (which are absent when the user connects to the service directly). We implement and evaluate each building block of our system individually: WebSocket RTT measurement on the application-layer, TCP handshake RTT recorded on the transport-layer, and ICMP ping and 0trace hop-enumeration RTT on the network-layer. We integrate all these techniques into one system called CalcuLatency and conduct a two-pronged evaluation: a control testbed environment where we test multiple different VPN products, proxy protocols, and server locations with over 337 unique VPN or proxy server IPs tested. To expand the diversity of our evaluation, we rally users and collect a public, crowdsourced, real-world evaluation of our system, which gained us 283 measurements from 37 different countries in all six continents.

Our evaluations reveal that a round-trip time difference of 50 milliseconds between the application and network-layer latencies is a viable, empirical threshold to consider a particular client as a remote VPN or proxy connection. We open-source our code in order to help encourage future research [41, 42].

CalcuLatency provides a preliminary, labelling technique for service-providers, and must not serve as the only signal for detecting "malicious proxy traffic". Not all VPN users are attackers and not all VPN or proxy traffic is abusive. Hence, our method only serves as one of the signals and service providers must leverage business-specific logic and make decisions on flagging connections as possible abuse traffic. For example, reward networks can flag certain connections as malicious and use such signals over time to detect and curb abusive consumers.

We acknowledge that service providers could potentially misuse this method to penalize all proxy use. However, we emphasize that our method's inherent design of detecting long-distance proxies and VPNs still protect legitimate proxy use and its users. Users could evade detection by using proxies close to their location, which gives them better performance and the requisite privacy and security features of the VPN.

Ensuring the adoption, success, and sustainability of privacy-focused business models rely heavily on the availability of computationally cost-effective and easily deployable techniques. Service providers must strike a delicate balance between implementing robust abuse-prevention mechanisms and maintaining a rigorous commitment to privacy. With CalcuLatency, we provide an open-source technique that can be readily deployed as a software service that can support efforts that prioritize user interests and their privacy.

10 Acknowledgment

The authors are grateful to Armin Huremagic, Riya Agarwal, Farzad Siraj for their help with the evaluations. We thank the reviewers for their feedback, and RIPE Atlas for generously granting credits to conduct an evaluation of our techniques. This material is based upon work supported by the National Science Foundation under grant numbers CNS-2141512, and CNS-2237552.

References

[1] Todd Spangler. *Digital Advertising Slowed in 2022 but Was Still Up 10.8%*. URL: https://variety.com/2023/digital/news/digital-advertising-revenue-2022-growth-pwc-research-iab-1235601801/ (visited on 05/03/2023).

[2] PwC. *Outlook 2022: The US Digital Advertising Ecosystem*. URL: https://www.pwc.com/us/en/industries/tmt/library/assets/pwc-iab-2022-outlook.pdf (visited on 10/01/2021).

[3] Amnesty International. *Surveillance giants: How the business model of Google and Facebook threatens human rights*. https://www.amnesty.org/download/Documents/POL3014042019ENGLISH.PDF. Amnesty International, 2019.

[4] *Presearch*. https://presearch.io/pre.

[5] *Prolific*. URL: https://www.prolific.com/participants.

[6] *Brave*. https://brave.com/brave-ads/.

[7] *Qmee*. https://www.qmee.com/.

[8] Michal Zalewski. *0trace – traceroute on established connections*. Jan. 2007. URL: https://lwn.net/Articles/217023/.

[9] *The WebSocket API (WebSockets)*. https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API. May 31, 2023.

[10] Zachary Weinberg, Shinyoung Cho, Nicolas Christin, Vyas Sekar, and Phillipa Gill. "How to Catch when Proxies Lie: Verifying the Physical Locations of Network Proxies with Active Geolocation". In: *IMC*. ACM, 2018. URL: http://www.contrib.andrew.cmu.edu/~nicolasc/publications/Weinberg-IMC18.pdf.

[11] Reethika Ramesh, Leonid Evdokimov, Diwen Xue, and Roya Ensafi. "VPNalyzer: Systematic Investigation of the VPN Ecosystem". In: *NDSS*. The Internet Society, 2022. URL: https://dx.doi.org/10.14722/ndss.2022.24285.

[12] Manaf Gharaibeh, Anant Shah, Bradley Huffaker, Han Zhang, Roya Ensafi, and Christos Papadopoulos. "A look at router geolocation in public and commercial databases". In: *Proceedings of the 2017 Internet Measurement Conference*. 2017.

[13] Ingmar Poese, Steve Uhlig, Mohamed Ali Kaafar, Benoit Donnet, and Bamba Gueye. "IP geolocation databases: Unreliable?" In: *ACM SIGCOMM Computer Communication Review* (2011).

[14] *Can I use WebSocket?* URL: https://caniuse.com/?search=websocket (visited on 06/06/2023).

[15] Allen T. Webb and A. L. Narasima Reddy. "Finding Proxy Users at the Service Using Anomaly Detection". In: *CNS*. IEEE, 2016. URL: https://ieeexplore.ieee.org/document/7860473.

[16] Hao Ding and Michael Rabinovich. "TCP Stretch Acknowledgements and Timestamps: Findings and Implications for Passive RTT Measurement". In: *ACM CCR* 45.3 (2015). URL: https://www.sigcomm.org/sites/default/files/ccr/papers/2015/July/0000000-0000002.pdf.

[17] Toke Høiland-Jørgensen, Bengt Ahlgren, Per Hurtig, and Anna Brunstrom. "Measuring Latency Variation in the Internet". In: *CoNEXT*. ACM, 2016. URL: https://dl.acm.org/doi/pdf/10.1145/2999572.2999603.

[18] Gregor Maier, Anja Feldmann, Vern Paxson, and Mark Allman. "On Dominant Characteristics of Residential Broadband Internet Traffic". In: *IMC*. ACM, 2009. URL: https://www.icir.org/vern/papers/imc102-maier.pdf.

[19] Bryan Veal, Kang Li, and David Lowenthal. "New Methods for Passive Estimation of TCP Round-Trip Times". In: *PAM*. Springer, 2005. URL: https://dkl.cs.arizona.edu/publications/papers/pam05.pdf.

[20] Shehar Bano et al. "Scanning the Internet for Liveness". In: *ACM CCR* 48.2 (2018). URL: https://ccronline.sigcomm.org/wp-content/uploads/2018/05/sigcomm-ccr-final175.pdf.

[21] Diwen Xue, Reethika Ramesh, Arham Jain, Michalis Kallitsis, J. Alex Halderman, Jedidiah R. Crandall, and Roya Ensafi. "OpenVPN is Open to VPN Fingerprinting". In: *Security*. USENIX, 2022. URL: https://www.usenix.org/system/files/sec22-xue-diwen.pdf.

[22] Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. "ZMap: Fast Internet-Wide Scanning and its Security Applications". In: *Security*. USENIX, 2013. URL: https://zmap.io/paper.pdf.

[23] Ram Sundara Raman, Mona Wang, Jakub Dalek, Jonathan Mayer, and Roya Ensafi. "Network Measurement Methods for Locating and Examining Censorship Devices". In: *ACM International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*. 2022.

[24] Philipp Richter et al. “A Multi-perspective Analysis of Carrier-Grade NAT Deployment”. In: *IMC*. ACM, 2016. URL: https://dl.acm.org/doi/pdf/10.1145/2987443.2987474.

[25] IETF. *RFC 1812: Requirements for IP Version 4 Routers*. https://datatracker.ietf.org/doc/html/rfc1812. 1995.

[26] IETF. *RFC 792: Internet Control Message Protocol*. https://datatracker.ietf.org/doc/html/rfc792. 1981.

[27] RIPE Atlas Docs. *Starting your own Measurements (User-defined Measurements)*. https://atlas.ripe.net/docs/getting-started/user-defined-measurements.html#the-user-defined-measurements.

[28] Ethan Katz-Bassett, John P John, Arvind Krishnamurthy, David Wetherall, Thomas Anderson, and Yatin Chawathe. “Towards IP Geolocation Using Delay and Topology Measurements”. In: *IMC*. ACM, 2006. URL: https://conferences.sigcomm.org/imc/2006/papers/p7-bassett.pdf.

[29] Reethika Ramesh, Anjali Vyas, and Roya Ensafi. “"All of them claim to be the best": Multi-perspective study of VPN users and VPN providers”. In: *32nd USENIX Security Symposium (USENIX Security 23)*. Anaheim, CA: USENIX Association, 2023.

[30] Tanya Shreedhar, Danesh Zeynali, Oliver Gasser, Nitinder Mohan, and Jörg Ott. *A Longitudinal View at the Adoption of Multipath TCP*. 2022. arXiv: 2205.12138 [cs.NI].

[31] Nikolai Tschacher. *Detecting Proxies and VPN’s [sic] with Latency Measurements*. June 2021. URL: https://web.archive.org/web/20210614154432/https://incolumitas.com/2021/06/07/detecting-proxies-and-vpn-with-latencies/ (visited on 02/15/2023).

[32] Nicholas Hopper, Eugene Y. Vasserman, and Eric Chan-Tin. “How Much Anonymity does Network Latency Leak?” In: *ACM Transactions on Information and System Security* 13.2 (2010). URL: https://www-users.cse.umn.edu/~hoppernj/tissec-latency-leak.pdf.

[33] Cristel Pelsser, Luca Cittadini, Stefano Vissicchio, and Randy Bush. “From Paris to Tokyo: On the Suitability of ping to Measure Latency”. In: *IMC*. ACM, 2013. URL: https://inl.info.ucl.ac.be/system/files/IMC2013-short-tokyo-ping.pdf.

[34] Hao Jiang and Constantinos Dovrolis. “Passive Estimation of TCP Round-Trip Times”. In: *ACM CCR* 32.3 (2002). URL: http://ccr.sigcomm.org/archive/2002/jul02/ccr-2002-3-jiang.pdf.

[35] Ioana Livadariu, Thomas Dreibholz, Anas Saeed Al-Selwi, Haakon Bryhni, Olav Lysne, Steinar Bjørnstad, and Ahmed Elmokashfi. “On the Accuracy of Country-Level IP Geolocation”. In: *Proceedings of the Applied Networking Research Workshop*. ANRW ’20. Virtual Event, Spain: Association for Computing Machinery, 2020, pp. 67–73. ISBN: 9781450380393. DOI: 10.1145/3404868.3406664. URL: https://doi.org/10.1145/3404868.3406664.

[36] The Khang Dang, Nitinder Mohan, Lorenzo Corneo, Aleksandr Zavodovski, Jörg Ott, and Jussi Kangasharju. “Cloudy with a chance of short RTTs: analyzing cloud connectivity in the internet”. In: *Proceedings of the 21st ACM Internet Measurement Conference*. IMC ’21. Virtual Event: Association for Computing Machinery, 2021, pp. 62–79. ISBN: 9781450391290. DOI: 10.1145/3487552.3487854. URL: https://doi.org/10.1145/3487552.3487854.

[37] Aqsa Kashaf, Jiachen Dou, Margarita Belova, Maria Apostolaki, Yuvraj Agarwal, and Vyas Sekar. “A First Look at Third-Party Service Dependencies of Web Services in Africa”. In: *Passive and Active Measurement*. Ed. by Anna Brunstrom, Marcel Flores, and Marco Fiore. Cham: Springer Nature Switzerland, 2023, pp. 595–622. ISBN: 978-3-031-28486-1.

[38] Nicholas Weaver, Christian Kreibich, Martin Dam, and Vern Paxson. “Here Be Web Proxies”. In: *PAM*. Springer, 2014. URL: https://www.icir.org/vern/papers/netalyzr-proxies.pam14.pdf.

[39] Xianghang Mi et al. “Resident Evil: Understanding Residential IP Proxy as a Dark Service”. In: *Security & Privacy*. IEEE, 2019. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8835239.

[40] Hans Hoogstraaten. “Evaluating server-side internet proxy detection methods”. MA thesis. Leiden University, 2018. URL: https://studenttheses.universiteitleiden.nl/access/item%3A2701711/view.

[41] *Zerotrace*. URL: https://github.com/censoredplanet/zerotrace-code.

[42] *CalcuLatency Evaluation Code*. URL: https://github.com/censoredplanet/calculatency-code.

[43] *RIPE Atlas Probes Archive, Accessed 01-11-2024*. https://ftp.ripe.net/ripe/atlas/probes/archive/2024/01/20240111.json.bz2.

Continent	Top 1% ASes		Top 10% ASes		Top 50% ASes	
	Elig.	**Com.**	Elig.	**Com.**	Elig.	**Com.**
Europe	29.8	**31.3**	65.4	**66.2**	87.2	**87.3**
N. America	36.5	**24.6**	67.3	**61.4**	88.8	**87.8**
Oceania	28.9	**30.1**	53.0	**39.7**	87.2	**82.2**
S. America	7.2	**15.6**	32.5	**28.1**	66.2	**65.6**
Asia	10.9	**9.7**	41.5	**40.0**	77.2	**74.2**
Africa	15.9	**43.8**	39.7	**43.8**	71.4	**75.0**

Table 1: Distribution of probes over the top 1%, 10% and 50% stratified by the continent of the probes. "Elig." refers to the percentage of eligible probes that exist in RIPE Atlas and "Com." refers to the percentage of probes that actually participated in and completed our measurements successfully. This table shows that the distribution of eligible probes closely follow the distribution of the probes that actually participated in our measurements, showing that our selection strategy did not skew our results.

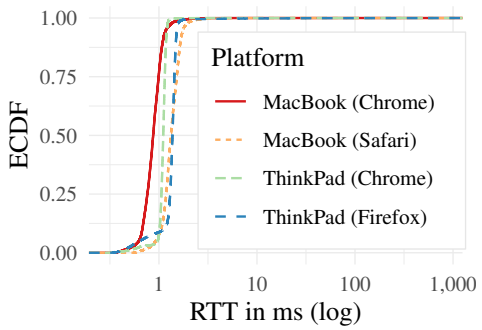


Figure 11: RTT distribution of WebSocket measurements on four platforms.

A Reliability of WebSocket Pings

Figure 11 illustrates the results from our WebSocket ping measurement with 10,000 WebSocket echo requests from two consumer laptops (ThinkPad X1, MacBook Pro) with two browsers: ThinkPad is a X1 Carbon 8th generation equipped with a i5-10210U CPU, and ran Chrome 111 and Firefox 111. The 2023 MacBook Pro is equipped with an M2 Pro CPU, and ran Safari 16.3 and Chrome 111. We find that the median RTT is less than 1.4ms for all measurements and 99% of the RTT measurements completed in less than 2.4ms. Hence, we account for this by running multiple measurements, spaced out over time in CalcuLatency.

B RIPE Atlas Probes Characterization

In our analysis outlined in 4.3, we downloaded the RIPE Atlas probes list latest on January 11, 2024 [43]. The total number of probes by unique probe ID was 38,554. But many probes have the same public IPv4 address, and the number of unique IPv4 addresses is 27,004.

We had to filter probes by eligibility based on multiple criteria. We found that 1,497 (3.9% of 38,554) probes had no public IPv4 OR IPv6 addresses. We found 588 (1.5%) probes had a bad country code where the country code either did not exist or the length of the country code was not two (specified by ISO-3166). We found that there were 1,107 (2.9%) probes that had undesirable tags attached to them, namely "system-geoloc-disputed", "core", "cloud", "vps", "ixp", "ipv6-only", or probe's tags start with "data-center" or "datacenter".

Importantly, we found that that 24,454 (63.4%) probes were either disconnected and/or abandoned, and were hence unusable. We also had to filter out 126 (0.3%) probes that do not have an IPv4 address specifically because 0trace relies on the IPv4 stack on the endpoint contacting us. We identified a set of 97 valid "tags" that determine usable probes. We eliminated 5,382 (14%) of probes that had **none** of the 97 desirable tags.

Of the 38,554 probes in the RIPE Atlas probe file, the total number of ineligible probes as defined by the criteria above was 33,154 (85.99% of 38,554). **Therefore, the number of eligible probes were 5,400 (14%).** Note that while we request all these eligible probes to participate in our measurements, not all of them do [27]. RIPE Atlas documentation mentions that it is possible for probes to not participate if the requested probes were too busy to take on new jobs, or had changed statuses, i.e. become disconnected or offline.

As mentioned in Section 4.3, we have 2,350 RIPE Atlas probe IP-probe ID pairs that fully completed the SSL, ICMP and TCP traceroute measurements. Overall, there are 2,304 unique RIPE Atlas probe IPs belonging to 2,193 unique probe IDs, because some probe IPs have multiple probe IDs associated with them and some probes have multiple IPs. The probes we have in our measurement covered all continents: Europe (1539), North America (378), Australia/Oceania (73), South America (32), Asia (155), and Africa (16).

The distribution of these probes over the top 1%, 10% and 50% of ASes per continent is shown in Table 1. We see that the distribution of eligible probes closely follow the distribution of the probes that participated in our measurements, thereby showing that our selection strategy did not skew our results.

The distribution of eligible and participated probes across different access technology shown in Table 2.

C Measurements done at different times of day and days of the week

We tested the hypothesis that running latency measurements at different times of the day in the local time of the probe

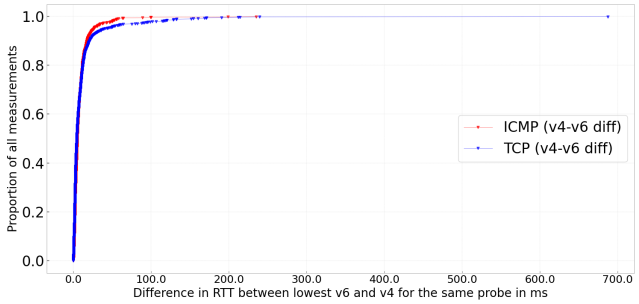


Figure 12: **RIPE Atlas IPv4 and v6 RTT Differences**—From our measurements, we find that in over 85.4% of the probes the difference between the RTTs of the terminal hops done from the traceroutes of IPv4 and IPv6 addresses (ICMP and TCP separately) is less than 50ms, which is exactly our margin threshold. In all cases above 50ms, the IPv6 traceroute RTT was markedly higher.

Tags (Associated RIPE Tags)	Eligible	**Participated**
Fibre (Fibre, FTTH)	2124	**906**
Cable (Cable)	1071	**365**
VDSL (Only VDSL or VDSL2)	475	**268**
DSL (DSL and not VDSL/VDSL2)	356	**160**

Table 2: Distribution of probes over access technology in the set of eligible probes and probes that actually participated in our measurement. This table shows that the distribution of tags follows the participation rate of ≈46%

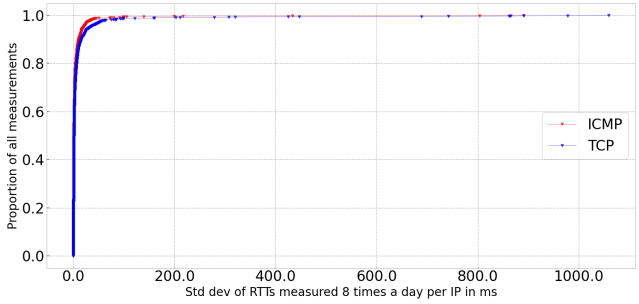


Figure 13: **CDF of Standard Deviation of RTTs per RIPE Probe IP** measured eight times a day—We see that in both ICMP and TCP traceroutes the standard deviations in over 90% of all probes is less than 9.2ms and 10.9ms respectively.

could lead to variance in expected RTT due to different traffic patterns at different times. So, we restricted our measurements to run at eight different times during the day, with respect to the local time of the probe, extrapolated from the latitude and longitude of the probe. We also ran these measurements on weekdays and weekends over two weeks to observe any differences. We had 1,345 probes that completed the TCP traceroute measurements, and 1,196 that completed the ICMP traceroutes over multiple days.

Our measurements show that in both ICMP and TCP traceroutes, the standard deviations of the RTTs throughout the day is low per IP address. Figure 13 shows that in 90% of probes, the standard deviation is lower than 9.2ms and 10.9ms in ICMP and TCP respectively. Moreover, in over 95% of probes, the standard deviation is lower than 15.7ms and 21.1ms in ICMP and TCP respectively. The outlying 5% measurements are not belonging to any specific group of probes and are equally spread out among the tags, locations, and times.

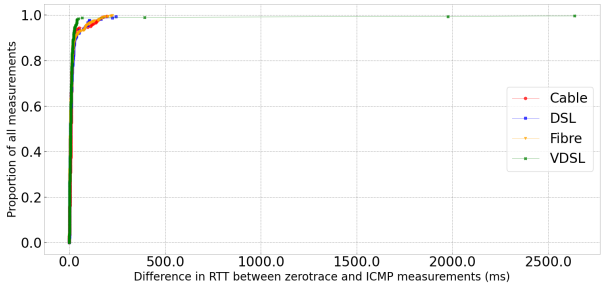
Since RIPE Atlas has a daily quota and a concurrency limit for measurements, our aim is to maximize the usefulness of our measurements. If we run measurements at different times of the day and repeat each measurement thrice to account for errors, we can only measure a maximum of 595 probes a day (Maximum credits per day: 1,000,000, divided by credits need per measurement round per probe (420) times 4 repeats). But since our experiments revealed that there is no significant difference in latency for 95% of the cases when measured at different times of the day, *we make the trade-off of missing out the 5% of probes with some variance in measurements taken at different times of the day, and rather choose to expand our measurements to include a larger variety and diversity of probes from different geolocations and last-mile technology.* In the rest of this section, we present results from our wider, diverse measurement run that was aimed at maximizing the number of probes, and conducted over a three day period (Sunday–Tuesday).

D RIPE Atlas Probes v4 vs v6 Differences

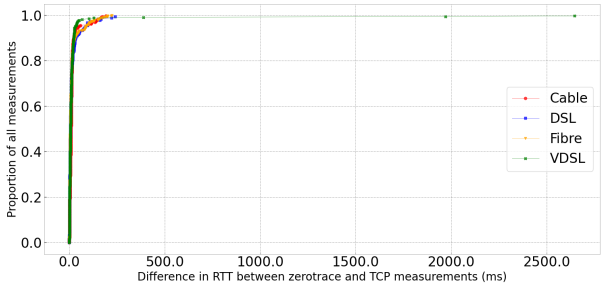
Since RIPE Atlas allows us to run Internet measurements from their probes which have IPv4 and IPv6 dual stack probes, we wanted to identify if our CalcuLatency and its evaluation have any significant effects if the tested endpoints are dual-stack. However, we note that our 0trace system is currently only able to run and measure towards an IPv4 address, since we rely on the IPID field and ICMP(v4) error responses in our system design. If the endpoint makes a TCP handshake via their IPv6 address, 0trace will not be able to run towards that host.

Nevertheless, we conducted measurements from the IPv4 and IPv6 addresses of the same probe. We find that the RTT difference between the terminal hops in the ICMP traceroute and TCP traceroute. We find that in ≈ 90% of the probes, the RTT difference between the IPv4 and IPv6 traceroutes in a given protocol (ICMP/TCP) is below 50ms, as shown in Figure 12. There is a long tail with the RTT differences, and upon manual investigation we notice that in each case, the IPv6 traceroute ended early, and had bloated RTT values and hence the difference turned out to be large.

We note that our CalcuLatency at the moment only supports IPv4 based 0trace measurements and so, we present this



(a) Difference between 0trace-determined RTT and RIPE Atlas-determined ICMP RTT in ms.



(b) Difference between 0trace-determined RTT and RIPE Atlas-determined TCP RTT.

Figure 14: **Comparing the RTT differences between probes with different local last-mile access technology**

analysis in the appendix, and leave the optimization of IPv6 based latency calculation for detecting proxy-enabled abuse for future work.

E Public Real-World Crowdsourced Evaluation ASN distribution

As mentioned in Section 5.2, we had 161 direct measurements that came from 145 unique client IPs belonging to 93 different autonomous systems (ASes), and 122 VPN measurements that came from 109 unique VPN IPs belonging to 51 different ASes. Overall, we collected data from over 37 different countries from all (six) continents.

We provide a more detailed breakdown of the IPs by autonomous system number. Our direct measurements had at least two measurements from the following AS numbers: 7922, 701, 7018, 21928, 3320, 36375, 33915, 3209, 24309, 1257, 22773, 35807, 577, 61832, 27, 20115, 5089, 3215, 6167. The rest of the 74 ASes had one measurement from them. Our VPN measurements had at least two measurements from the following AS numbers: 9009, 212238, 60068, 136787, 39351, 16509, 136557, 132825, 137409, 63949, 20473, 13335, 60781, 197706. The rest of the 34 ASes had one measurement from them.

F Differences in local and last-mile access technology

Figure 14 contains all the data from our evaluation of our hypothesis of whether different last-mile access technologies have large effects on the latency seen from the server side. We divide and present our results for the RIPE Atlas probe's TCP and ICMP traceroutes separately. In Section 4.3, we presented results without outliers for better visibility. Here, we present all our results.

We see that there are only three outliers, all of which were tagged with VDSL as the access technology. As noted in Section 4.3, these outliers had an RTT difference of 388ms, 1970ms, and 2640ms and upon investigation we note that these were due to two German RIPE probes in AS8881 and AS3320 which failed to reach far in the 0trace traceroute. These cases would be characterized as "best effort" in the real world operation of the CalcuLatency system and would be marked for retrying and not used for determination.