An ISP-Scale Deployment of TapDance

Sergey Frolov[1], Fred Douglas[3], Will Scott[5], Allison McDonald[5], Benjamin VanderSloot[5], Rod Hynes[6], Adam Kruger[6], Michalis Kallitsis[4], David G. Robinson[7], Steve Schultze[2], Nikita Borisov[3], J. Alex Halderman[5], and Eric Wustrow[1]

[1]University of Colorado Boulder [2]Georgetown University Law Center [3]University of Illinois Urbana-Champaign
[4]Merit Network [5]University of Michigan [6]Psiphon [7]Upturn

Abstract

We report initial results from the world's first ISP-scale field trial of a refraction networking system. Refraction networking is a next-generation censorship circumvention approach that locates proxy functionality in the middle of the network, at participating ISPs or other network operators. We built a high-performance implementation of the TapDance refraction networking scheme and deployed it on four ISP uplinks with an aggregate bandwidth of 100 Gbps. Over one week of operation, our deployment served more than 50,000 real users. The experience demonstrates that TapDance can be practically realized at ISP scale with good performance and at a reasonable cost, potentially paving the way for long-term, large-scale deployments of TapDance or other refraction networking schemes in the future.

1 Introduction

Censorship circumvention tools typically operate by connecting users to a proxy server located outside the censoring country [3, 12, 15, 18]. Although existing tools use a variety of techniques to conceal the locations of their proxies [5, 9, 13, 17, 19], governments are deploying increasingly sophisticated and effective means to discover and block the proxies [7, 8, 20].

Refraction networking [16][1] is a next-generation circumvention approach with the potential to escape from this cat-and-mouse game. Rather than running proxies at specific edge-hosts and attempting to hide them from censors, refraction works via Internet service providers (ISPs) or other network operators, who provide censorship circumvention functionality for any connection that *passes through* their networks. To accomplish this, clients make HTTPS connections to sites that they can reach, where such connections traverse a participating network. The participating network operator recognizes a steganographic signal from the client and appends the user's requested data to the encrypted connection response. From the perspective of the censor, these connections are indistinguishable from normal TLS connections to sites the censor has not blocked. To block the refraction connections, the censor would need to block all connections that traverse a participating network. The more ISPs participate in such a system, the greater the extent of collateral damage that would-be censors would suffer by blocking the refracted connections.

A variety of refraction networking systems have been proposed in recent years [2, 6, 10, 11, 21, 22], representing different trade-offs among practicality, stealthiness, and performance. The basic idea is to watch all of the traffic passing through a router, selecting flows which are steganographically tagged as participating in the protocol, and then modifying that traffic by extracting and making the encapsulated request on behalf of the client. While each of these schemes has been prototyped in the lab, implementing refraction within a real ISP poses significant additional challenges. An ISP-scale deployment must be able to:

- Identify client connections on high-speed backbone links operating at 10–40 Gbps or more. This is at the limits of commodity network hardware.
- Be built within reasonable cost constraints, in terms both of required hardware and of necessary rack space at crowded Internet exchange points.
- Operate reliably without disrupting the ISP's network or the reachable sites clients connect to.
- Have a mechanism for identifying reachable sites for which connections pass through the ISP, and for disseminating this information to clients.
- Coordinate traffic across multiple Internet uplinks or even multiple ISPs.

To demonstrate that these challenges can be solved, we constructed a large trial deployment of the TapDance refraction scheme [21] and operated a trial deployment in partnership with two mid-sized network operators: a

[1]Previous works used the term *decoy routing*, which confusingly shares the name of a specific refraction scheme. We use refraction networking as an umbrella term to refer to all schemes.

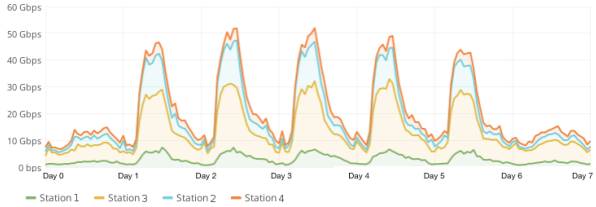

Figure 1: **Station Traffic**—Our four stations processed mirrored traffic from 10- and 40-Gbps ISP upstream links in order to detect connections from TapDance clients. We show traffic processed over one week, which peaked above 55 Gbps.

regional ISP and a large university. Our goal was to understand: (*i*) the scale of traffic a refraction system built within reasonable constraints today can realistically process, (*ii*) the experience for users of refraction in contrast to traditional proxy-based circumvention, and (*iii*) the impact on ISPs of operating refraction infrastructure.

This paper presents initial results from that deployment. We discuss the design and engineering considerations that we dealt with in its construction, and we present the first data supporting the real-world practicality of refraction at ISP scale. Our client and station code is publicly available at https://refraction.network.

2 Deployment Overview

Over the years, several refraction schemes have been proposed. The initial generation includes Cirripede, Curveball, and Telex [10, 11, 22]. Each of these schemes employed an inline blocking element located at an ISP that could observe network flows, look for tagged connections, and selectively block flows between the censored client and the reachable server. This style of system is difficult to deploy, as it introduces a high-risk inline blocking device into an ISP's network.

We chose to implement a second generation refraction networking scheme, called TapDance [21], which was designed to be easier for ISPs to deploy than earlier proposals. TapDance operates by co-locating a "station" at each of the ISP's Internet uplinks. Unlike with previous schemes, the TapDance station does not need to alter or drop any traffic on the link; rather, it merely needs to be able to passively inspect a copy of the traffic and to inject new packets. This can be accomplished using either an optical splitter or a router port configured to mirror the link. TapDance clients send incomplete HTTPS requests to reachable sites, which are chosen so that the client's route to the site passes by the TapDance station. Clients tag the ciphertext of these connections in a way that is observable to the TapDance station, but not to a censor. Since the HTTPS request is incomplete, the reachable site will not respond. Meanwhile, the TapDance station will impersonate the server and covertly communicate with the client.

We partnered with two network operators: Merit Network, a medium-sized regional ISP and University of Colorado Boulder, a large public university. We worked with each to deploy TapDance stations in a configuration that would have visibility into most of the traffic entering and exiting their respective autonomous systems. In all, we deployed four stations, with three at Merit and one at the University of Colorado.

In order to achieve a large user base in a short time, we partnered with Psiphon [15], a widely used proxy-based circumvention tool. We implemented a TapDance client as a modular transport, and it was distributed to a fraction of Psiphon's user base during the trial, as a software update to the Android version of the Psiphon application. From users' perspective, our client was invisible; it simply provided another method of reaching circumvention service. This arrangement also helped ensure user privacy, since data was encrypted end-to-end from the clients to servers operated by Psiphon.

Our successful at-scale deployment of TapDance for censored users provides strong evidence that the technique operates well within the bandwidth, latency, and middlebox constraints that are typical of heavily censored network environments. The scale and duration of the deployment were small enough that we do not believe we came to the attention of censors, so we do not claim to have obtained evidence about TapDance's resistance to adversarial countermeasures. However, the trial did exercise our system's operational behavior and ISP logistics.

The trial took place during the spring of 2017. We have decided not to disclose the precise time window of the trial, in order to mitigate risks to end-users and the people supporting them.

3 Scaling TapDance

TapDance was designed to be easy for ISPs to deploy, and it proved to be neither technically difficult nor expensive for either network operator to provide the required traffic visibility. This was accomplished by configuring gateway routers to mirror traffic on a port that was attached to our station hardware. For packet injection, we simply used the stations' default network interfaces, which were not subject to any filtering that would prevent spoofing.

Bandwidth and traffic volume varied by station location, with two of the stations (both at Merit) operating on 40 Gbps links and the other two on 10 Gbps links. Figure 1 shows the traffic processed by each of the four stations during one week of the trial.

Space constraints at the peering locations limited each TapDance station to a single commodity 1U server. This limited the amount of CPU, RAM, and network hardware available to each station and required us to engineer for

efficiency. Station 3, which handled the most traffic, was provisioned with an 8-core Intel Xeon E5-2667v3 CPU (3.20 GHz), 64 GB of RAM, and a quad-port Intel X710 10GbE SFP+ network adapter. The hardware specifications of other stations varied slightly.

3.1 Handling 40Gbps Links

We reimplemented the TapDance station from scratch, primarily in Rust. Rust is an emerging low-level systems language that provides compile-time guarantees about memory safety, which lowers the risk of remote compromise. To efficiently process packets at 40 Gbps line rates, our implementation is built on the PF_RING library and kernel driver [14], operating in zero-copy mode. By splitting incoming traffic onto multiple cores we were able to handle full line rate traffic with only occasional dropped packets, which, due to the design of TapDance, do not interfere with the normal operation of an ISP. Our experience demonstrates that, even in large installations, a software-based implementation of TapDance can be practical, avoiding the need for costly specialized hardware.

The station uses C in three main areas. First, it uses a C implementation of the Elligator [1] tagging scheme for better performance during initial detection of TapDance flows. Next, it uses OpenSSL to handle TLS work, including manually assembling TLS connections from secrets extracted from TapDance tags. Finally, it uses `forge_socket`, a Linux kernel module that allows a userspace application to create a network socket with arbitrary TCP state, in order to utilize Linux's TCP implementation while pretending to be the server.

Handling multiple TapDance users is highly parallelizable, so the station is designed to use multiple cores by running multiple processes with absolutely no communication among them. However, a single TapDance session typically spans multiple TLS flows. This means that, once a user has begun a TapDance session on a particular station process, that same process must see all future TLS flows in that session. To achieve this, we configure PF_RING to spread flows based on the IP pair, rather than the full TCP 4-tuple, so that the client's changing source port will not send the traffic to a different process.

Our four stations ran on a total of 34 cores (excluding a dedicated PF_RING core per station), with the most loaded station using 14 cores. These 34 cores were able to comfortably handle a peak of close to 14,000 new TLS connections per second, with each connection being checked for a TapDance-tagged request.

3.2 Client Integration

We implemented a from-scratch TapDance client in Go, a language that can be integrated in mobile applications and that, like Rust, guards against memory corruption vulnerabilities. Our client was integrated into Psiphon's existing circumvention application as a library presenting a new reliable transport interface — that is, the same interface that Go programs see when using TCP or TLS.

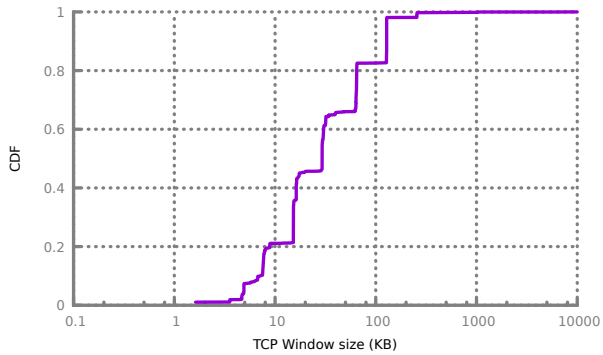

Figure 2: **TCP Window Size of Candidate Hosts** — We measured the TCP window size for each candidate reachable host, in order to find ones suitable for TapDance.

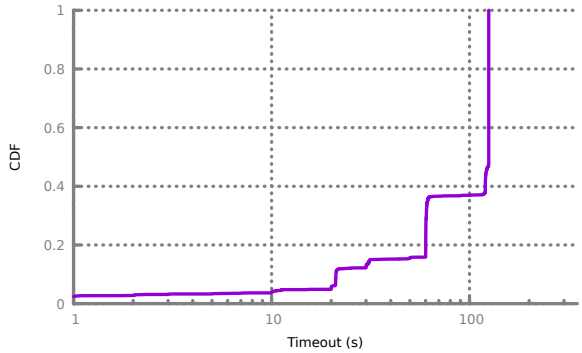

Figure 3: **Timeouts of Candidate Hosts** — We measured how long (up to a limit of 120 seconds) candidate reachable hosts would leave open an incomplete HTTP request.

Through the trial, we logged only a minimal amount of information from users due to privacy concerns. For example, we did not log the source IP address of clients nor the destination page they requested. Instead, we logged a connection identifier generated by the client and used for a session duration, as well as the aggregate number of bytes uploaded and downloaded in the session.

3.3 Selecting Reachable Sites

Unlike previous lab-scale prototypes, our implementation required a mechanism by which clients could discover reachable, uncensored sites with our stations on-path. We achieved this by compiling a list of potential sites and shipping it with the client. We updated the list daily and pushed updates to clients the first time they connected each day.

To construct a list of candidate sites, we started by using Censys [4] to retrieve the list of all hosts serving HTTPS with browser-trusted certificates from within the AS or a

customer AS of our participating network operators. This initial set was typically around 5500 sites, although there was a small amount of churn.

Next, we retrieved `robots.txt` from each site. Sites could decline to be used by our clients by adding a specific entry to this file. To advertise this, we included a URL in the client's `User-Agent` string that led to a site explaining how TapDance works and how to opt out. No websites chose to be excluded during our month-long trial.

A server's suitability also depends on its TCP window size and HTTP timeout duration. In TapDance, a client can only send up to one TCP window of data before the server will respond with stale ACK packets. Similarly, a TapDance connection can only be used until the point where the real server would respond with an HTTP timeout or otherwise attempt to close the connection.

Maximizing these parameters enables connections to last longer and transfer more data. We measured them for each site and included them in the list sent to clients. CDFs of these measurements are shown in Figures 2 and 3. Requiring a minimum 15 KB TCP window eliminated 24% of candidate sites, and a 30 second HTTP timeout eliminated 11%. Together, these minimums reduced the number of viable sites by 34%. Additionally, for sites to be suitable, they need to support the TLS ciphersuites that our client implements (the `AES_128_GCM` family). This requirement eliminated an average of 32% of candidates.

Finally, we made a test connection to each candidate using an automated version of our client that was positioned so that a station was guaranteed to observe all traffic. If this was successful, we added the host to the list of potential reachable sites sent to clients.

These tests ensure that a client's connection to a server will be usable for TapDance as long as the client's connection passes by a station. However, routing is in general difficult to predict, and so this last condition is not guaranteed to hold. We located our stations so that most traffic from international hosts in the operators' networks would be observed. Since some connections would not be visible to the stations, the clients were programmed to select sites from their lists at random, and to retry with a different site if one did not work.

During the trial, we released only half of the viable hosts to clients, in order to better test the load on individual hosts and to retain backup hosts should the censors begin to block the hosts by IP address. A daily median of 450 hosts were available, with an average overlap of 366 from the previous day.

3.4 Handling Multiple Stations

Since we had multiple stations within the same ISP, we needed to account for the possibility that a connection would switch from being observed by one station to another mid-flow, due to routing instability. What we found was that individual client-to-reachable site routes were stable and generally only passed by a single station, but that it was common for two different clients to get to the same reachable site using routes that passed by two different stations.

In our design, we decided to minimize communication between stations in order to simplify the implementation and reduce latency. However, this means that for a client to have an uninterrupted session with the proxy, it must communicate with only a single station. During the session, multiple connections must be made to reachable sites, with the client-to-proxy traffic being multiplexed over them. Therefore, to ensure that these flows all use the same station, we had clients pick a single reachable host for the lifetime of a given session.

3.5 Managing Load on Reachable Sites

In our tests with our own Nginx and Apache servers, we noticed that under default configurations, these web servers limited the number of simultaneous connections they would allow. For example, after 150 connections, a default Apache webserver will stop responding to new connections until one closes. Since a client will continue using a chosen reachable site for a long period, we were concerned that "hot spots" could develop where particular hosts received disproportionate load.

We addressed this potential issue by attempting to limit the number of concurrent users to any individual site. Because users may reach a site through multiple stations, this requires coordination between stations to track how many active clients are using a particular site.

We added support in our stations to allow a central collector to inform a station that a particular site was overloaded and it should turn away additional clients attempting to use that site. We initially set the per-site limit to 30 concurrent connections to prevent inadvertent overload, and the site load remained well below our expected threshold for the duration of the trial.

In addition to this change, we also gave clients that failed to connect to a site an increasing timeout before trying the next site. This exponential back off ensured that, if a station or other failure occurred, the load clients pushed on sites or other network infrastructure would decrease rapidly.

4 Trial Results

We present an initial analysis of the results from our trial deployment in the spring of 2017.

4.1 At-Scale Operation

A major goal of this deployment was validating the hardware and operational requirements for running TapDance in a production setting. We were able to run our system while processing 40 Gbps of ISP traffic from commodity 1U servers. The observed traffic over our measurement

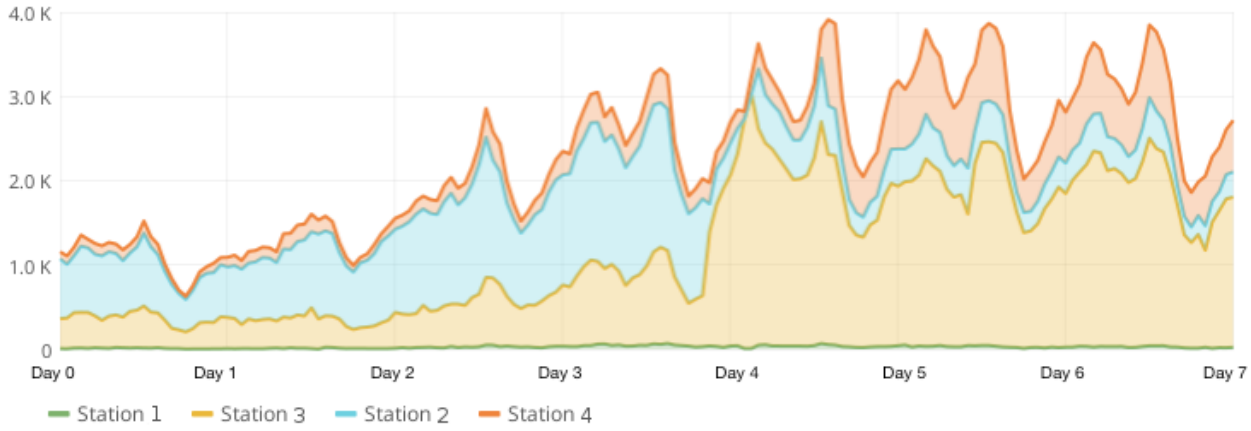

Figure 4: **Concurrent Sessions**—This plot shows the number of active TapDance user sessions, which peaked at 4,000.

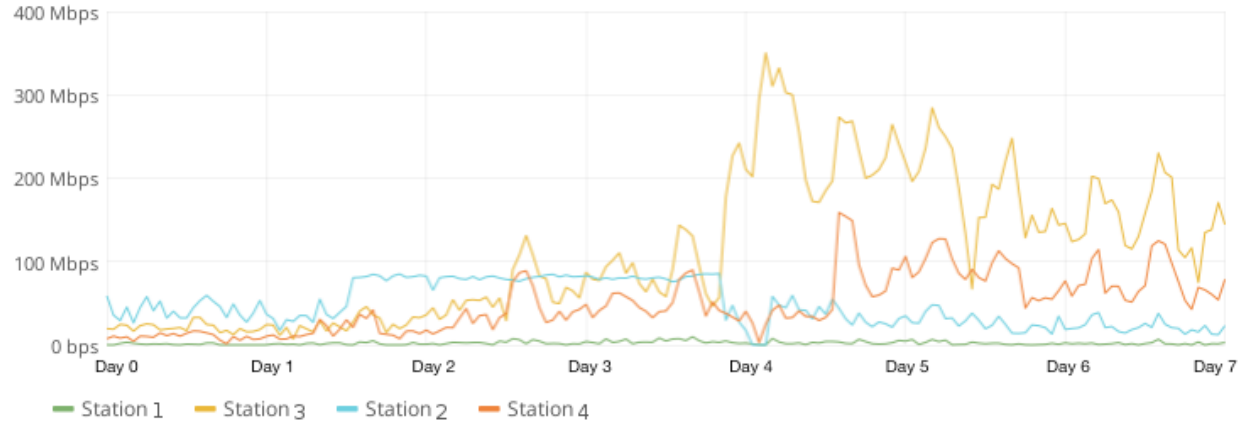

Figure 5: **User Traffic (Downstream)**—This plot shows how much user traffic our deployment served to clients. In response to the saturation of the 100 Mbps link on Station 2, we took steps on Day 3 to migrate user load to Station 3.

week is shown in Figure 1, showing a cumulative processing peak of 55 Gbps across stations, and a single station processing more than 20 Gbps. During our trial, the CPU load on our stations remained below 25%, although this figure alone is not representative of achievable performance, which is heavily dependent on scheduling between the processors and network card.

4.2 User Traffic

Over the trial, we served over 50,000 unique users, according to Psiphon statistics. For privacy reasons, we did not track identifying information of users during our trial. Instead, Psiphon clients reported the last successful connection time, whenever they reconnected. This allowed Psiphon to query for connections with a last connect time before the current day to get a unique daily user count, as shown in Figure 7. Figure 4 shows the number of concurrently active users over the trial week. At peak, TapDance served over 4,000 users simultaneously, with peaks on a single station over 3,000 concurrent users. Interestingly, we see two peaks per day, which is not explained by user timezone distributions.

On day four of the trial, we discovered a bottleneck that was limiting use on the most popular station (Station 2). At this station, the injection interface was misconfigured to be over a 100 Mbps management interface. This link was saturated by the amount of traffic being injected by our station, making downloads noticeably slower for users of this station. The station was at a remote facility where it was impractical to reconfigure the network, so we worked with our Merit to shift traffic headed to a large subset of reachable sites so that it passed a different station (Station 3) with a 1 Gbps injection interface. This solution improved download performance considerably, reflected in the observed usage in Figure 5.

Figure 6 shows a lower-bound estimate on the bandwidth available to users. The average throughput of 5 KB/s shown in this CDF does not reflect users' actual

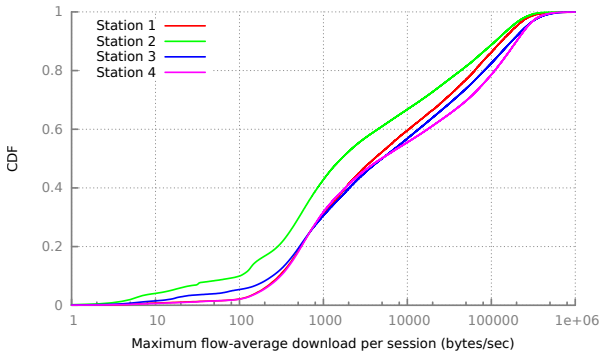

Figure 6: **Session Throughput** — This CDF shows the average downstream throughput achieved by the fastest sub-flow during each client session. Note that this is a lower-bound for actual throughput, as it is possible that sub-flows were not actively used for their entire duration.

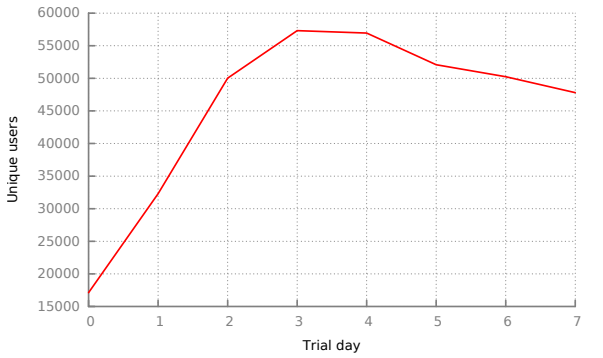

Figure 7: **Unique daily users** — The number of unique connected clients over each day of our measurement week. At peak, we handled over 57,000 unique users.

experience, as many sessions included in the CDF did not see real usage. The structure of our deployment was such that a users' entire device's internet traffic would be tunneled through a single, long-running TapDance session. These sessions must be occasionally restarted, due to network instability or other problems (see Figure 8). Therefore, many of the observed sessions happened entirely while the user was not using their device at all.

Figure 8 shows the distribution of client session durations. The median individual session length to a station was on the order of a few minutes. We reviewed logs to determine that about 20% of sessions ended in a timeout (where the client was not heard from for over 30 seconds).

4.3 Impact on Reachable Sites

During the trial, we measured the impact of multiple clients using the same site to reach TapDance. As described in Section 3.3, we are attentive to the number of clients simultaneously using a given site, as large numbers of clients could lead to resource exhaustion.

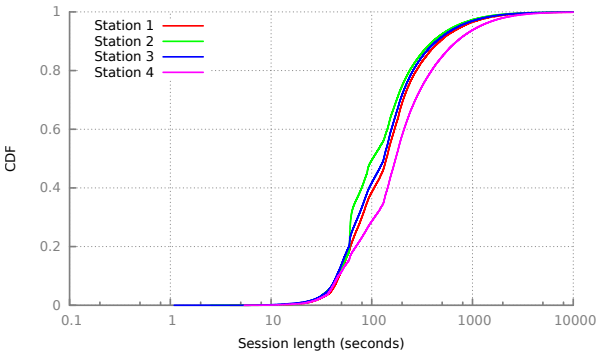

Figure 8: **Session Length** — This CDF shows client session durations over our trial, i.e., the time each client stayed continuously connected to a station without interruption. When sessions were interrupted (due to network failures or timeouts), the client automatically reconnected.

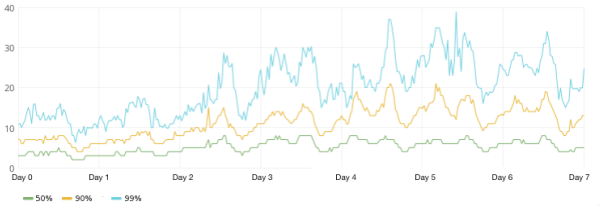

Figure 9: **Clients per Site** — This plot shows how many clients were connected to the average, 90th-, and 99th-percentile reachable site during our trial deployment.

Figure 9 shows the load on the median, 90th, and 99th percentile sites over the trial. The median load remained generally evenly spread, with the typical site seeing around 5 clients connected simultaneously, with only 10% of sites ever having more than 20 simultaneous users.

Mid-week, we shifted more users toward Station 4 by increasing how often clients picked sites that were likely to be served by that station. We simultaneously increased the maximum simultaneous connections allowed to each site from 30 to 45. This is reflected in the 99th percentile site load, which peaked at 37 concurrent users to a single site.

5 Acknowledgements

The authors thank the anonymous reviewers and our shepherd Roger Dingledine for their thoughtful comments and guidance. We also wish to thank our partners, especially Merit Network, Psiphon, and the University of Colorado IT Security and Network Operations. Finally, we thank Christine Jones and Daniel Ellard of Raytheon BBN Technologies for their help in assessing and testing the system.

6 Conclusion

Refraction networking offers a new path forward in the censorship arms race, potentially shifting the balance in favor of Internet freedom. At the same time, real-world deployment poses significant challenges. In this paper, we presented initial results from the first real-world refraction networking deployment. Our experience demonstrates that refraction can operate at ISP-scale and support tens of thousands of real users. Through this trial, we have identified and overcome several challenges related to scaling, operation, and maintenance of refraction networking. The lessons of our experience are likely to be useful for future research and deployment efforts. We hope that this first success paves the way for wider refraction networking deployments in the near future.

References

[1] BERNSTEIN, D. J., HAMBURG, M., KRASNOVA, A., AND LANGE, T. Elligator: Elliptic-curve points indistinguishable from uniform random strings. In *20th ACM Conference on Computer and Communications Security* (Nov. 2013), CCS, pp. 967–980.

[2] BOCOVICH, C., AND GOLDBERG, I. Slitheen: Perfectly imitated decoy routing through traffic replacement. In *23rd ACM Conference on Computer and Communications Security* (Oct. 2016), CCS, pp. 1702–1714.

[3] DINGLEDINE, R., MATHEWSON, N., AND SYVERSON, P. Tor: The second-generation onion router. In *13th USENIX Security Symposium* (Aug. 2004), pp. 303–320.

[4] DURUMERIC, Z., ADRIAN, D., MIRIAN, A., BAILEY, M., AND HALDERMAN, J. A. A search engine backed by Internet-wide scanning. In *Proceedings of the 22nd ACM Conference on Computer and Communications Security* (Oct. 2015).

[5] DYER, K. P., COULL, S. E., RISTENPART, T., AND SHRIMPTON, T. Protocol misidentification made easy with format-transforming encryption. In *20th ACM Conference on Computer and Communications Security* (Nov. 2013), CCS, pp. 61–72.

[6] ELLARD, D., JACKSON, A., JONES, C., MANFREDI, V., STRAYER, W. T., THAPA, B., AND WELIE, M. V. Rebound: Decoy routing on asymmetric routes via error messages. In *40th IEEE Conference on Local Computer Networks* (Oct. 2015), LCN, pp. 91–99.

[7] ENSAFI, R., FIFIELD, D., WINTER, P., FEAMSTER, N., WEAVER, N., AND PAXSON, V. Examining how the Great Firewall discovers hidden circumvention servers. In *15th ACM Internet Measurement Conference* (Oct. 2015), IMC, pp. 445–458.

[8] ENSAFI, R., WINTER, P., ABDULLAH, M., AND CRANDALL, J. Analyzing the Great Firewall of China over space and time. In *Proceedings on Privacy Enhancing Technologies* (2015), PETS, pp. 61–76.

[9] FIFIELD, D., LAN, C., HYNES, R., WEGMANN, P., AND PAXSON, V. Blocking-resistant communication through domain fronting. In *Proceedings on Privacy Enhancing Technologies* (2015), PETS, pp. 1–19.

[10] HOUMANSADR, A., NGUYEN, G. T. K., CAESAR, M., AND BORISOV, N. Cirripede: Circumvention infrastructure using router redirection with plausible deniability. In *18th ACM Conference on Computer and Communications Security* (Oct. 2011), CCS, pp. 187–200.

[11] KARLIN, J., ELLARD, D., JACKSON, A. W., JONES, C. E., LAUER, G., MANKINS, D. P., AND STRAYER, W. T. Decoy routing: Toward unblockable Internet communication. In *1st USENIX Workshop on Free and Open Communications on the Internet* (Aug. 2011), FOCI.

[12] Lantern. https://getlantern.org/.

[13] MOHAJERI, M. H., LI, B., DERAKHSHANI, M., AND GOLDBERG, I. Skypemorph: Protocol obfuscation for Tor bridges. In *19th ACM Conference on Computer and Communications Security* (Oct. 2012), CCS, pp. 97–108.

[14] NTOP. PF_RING. http://www.ntop.org/products/pf_ring.

[15] Psiphon. https://psiphon.ca.

[16] Refraction Networking: Internet freedom in the network's core. https://refraction.network/.

[17] THE TOR PROJECT. obfs4 (the obfourscator) specification. https://gitweb.torproject.org/pluggable-transports/obfs4.git/tree/doc/obfs4-spec.txt.

[18] uProxy. https://www.uproxy.org/.

[19] WEINBERG, Z., WANG, J., YEGNESWARAN, V., BRIESEMEISTER, L., CHEUNG, S., WANG, F., AND BONEH, D. Stegotorus: A camouflage proxy for the Tor anonymity system. In *19th ACM Conference on Computer and Communications Security* (Oct. 2012), CCS, pp. 109–120.

[20] WILDE, T. Knock knock knockin' on bridges' doors. Tor Blog, Jan. 7, 2012. https://blog.torproject.org/blog/knock-knock-knockin-bridges-doors.

[21] WUSTROW, E., SWANSON, C. M., AND HALDERMAN, J. A. Tapdance: End-to-middle anticensorship without flow blocking. In *23rd USENIX Security Symposium* (Aug. 2014), pp. 159–174.

[22] WUSTROW, E., WOLCHOK, S., GOLDBERG, I., AND HALDERMAN, J. A. Telex: Anticensorship in the network infrastructure. In *20th USENIX Security Symposium* (Aug. 2011).