Assessing Support for DNS-over-TCP in the Wild

Jiarun Mao[1], Michael Rabinovich[1], and Kyle Schomp[2]

[1] Case Western Reserve University, Cleveland OH 44106, USA
[2] Akamai Technologies, Cambridge MA 02142, USA
{jxm959, michael.rabinovich}@case.edu
kschomp@akamai.com

Abstract. While the DNS protocol encompasses both UDP and TCP as its underlying transport, UDP is commonly used in practice. At the same time, increasingly large DNS responses and concerns over amplification denial of service attacks have heightened interest in conducting DNS interactions over TCP. This paper surveys the support for DNS-over-TCP in the deployed DNS infrastructure from several angles. First, we assess resolvers responsible for over 66.2% of the external DNS queries that arrive at a major content delivery network (CDN). We find that 2.7% to 4.8% of the resolvers, contributing around 1.1% to 4.4% of all queries arriving at the CDN from the resolvers we study, do not properly fallback to TCP when instructed by authoritative DNS servers. Should a content provider decide to employ TCP-fallback as the means of switching to DNS-over-TCP, it faces the corresponding loss of its customers. Second, we assess authoritative DNS servers (ADNS) for over 10M domains and many CDNs and find some ADNS, serving some popular websites and a number of CDNs, that do not support DNS-over-TCP. These ADNS would deny service to (RFC-compliant) resolvers that choose to switch to TCP-only interactions. Third, we study the TCP connection reuse behavior of DNS actors and describe a race condition in TCP connection reuse by DNS actors that may become a significant issue should DNS-over-TCP and other TCP-based DNS protocols, such as DNS-over-TLS, become widely used.

1 Introduction

The DNS protocol, along with the massive infrastructure that runs it, is one of the key components of the Internet, providing mapping services from names to various data records, most notably from human-readable hostnames to IP addresses [29]. Over its decades of development, numerous use cases for DNS have emerged, such as distributing tokens used as proof of website ownership [4], providing access to cryptographic signatures for verifying both the integrity of the DNS records themselves [10] and the follow-up application traffic [13], and facilitating mechanisms for enhancing email security [11, 18]. Diverse usage scenarios have led to a wide range of DNS message sizes, including some very large messages.

DNS can use either UDP or TCP as transport, but UDP is overwhelmingly used in practice: only 0.02% of DNS queries in our log of DNS queries at a major content delivery network (CDN) arrive over TCP. Using UDP is attractive because it is light-weight, and most DNS messages fit within a datagram and even within the 512-byte limit stipulated for DNS-over-UDP, per RFC 883 [28]. Moreover, the optional EDNS0 payload size allows larger UDP messages [37].

However, there are still messages that may be too large even for EDNS0 payload size. Further, large UDP responses make DNS an easy vector for amplification denial of service attacks [26, 32], since an attacker can (*i*) elicit a large response from a legitimate server using a comparatively small query and (*ii*) spoof the source IP address of the victim, causing the server to send its (large) response to the victim. Thus, there is an increased interest in expanding the use of TCP for DNS interactions. Support for TCP allows much larger DNS messages (up to 65535 bytes) and effectively limits amplification attacks as large messages are not transferred until after the TCP handshake verifies the authenticity of the client's IP address; note that packets exchanged during the handshake are all equal size, precluding amplification in packet size – if not number of packets due to server-side retries [23] – during the handshake itself.

According to the protocol, a DNS interaction can occur over TCP at the discretion of either the client or the server. The client can simply send its query over TCP. The server can force a ”TCP fallback” by responding to a UDP query with a partial UDP response and the truncated flag (TC) to indicate the truncation. The client should then retry the query over TCP.

The heightened interest in DNS-over-TCP brings an important question: if one party chooses to use TCP for DNS interactions, is there a risk that the other party may not support it properly, given that the current practice is to use UDP virtually exclusively? In fact, a past study of DNSSEC deployment provided an indication that some resolvers may not be TCP-capable [24], adding impetus to a more comprehensive look into this issue.

In this paper, we investigate the support for DNS-over-TCP in the deployed DNS infrastructure from several angles. First, we assess the support for TCP-fallback by recursive resolvers, using various measurement techniques to explore different classes of resolvers, that in the aggregate are responsible for a large portion of Internet DNS activity. Second, we assess DNS-over-TCP support by authoritative DNS servers (ADNS) serving many domains – including popular ones – and a large number of content delivery networks. Third, we study the behavior of these DNS actors with regard to an important aspect of DNS-over-TCP behavior, namely, reuse of TCP connections for multiple queries. Our key findings are:

- We show that the egress resolvers follow complex patterns in interacting with authoritative servers that force TCP-fallback, with only half of resolutions exhibiting a “canonical” pattern of one UDP query followed by one TCP query. Consequently, we design and validate an algorithm for characterizing egress resolvers' TCP-fallback capability from complex patterns.

- Among the egress resolvers we study, we find 2.7% to 4.8% to be incapable of TCP-fallback. Moreover, by analyzing the DNS logs of the major CDN, we show that these TCP-fallback incapable resolvers tend to be generally as active as their TCP-fallback capable counterparts, as they account for 1.1% to 4.4% of DNS queries received by the major CDN from the resolvers we study. We believe content providers are unlikely to move to a technology that leads to failure of such a fraction of DNS queries and potentially cuts off a non-negligible amount of their consumers.
- We find that around 3% of popular websites, and 5% of domains at large, with at least some ADNS failing to answer DNS-over-TCP queries. Moreover, a surprisingly large fraction of CDNs, 11 out of 47 CDNs we consider, have at least one authoritative DNS server with no DNS-over-TCP support. Again, we believe, with these results, resolver operators would be hesitant to unilaterally switch to DNS-over-TCP and potentially block their users from a non-negligible portion of Internet content.
- We identify an edge case in the DNS-over-TCP protocol that can cause unnecessary query retries and uncover some DNS-over-TCP implementation bugs in two major CDNs. We propose simple changes to the protocol that would remove this vulnerability.
- We demonstrate that, despite the steady decrease in the number of open resolvers, active scanning can still discover egress resolvers[3] responsible for a substantial portion of the Internet DNS activity. In all, the egress resolvers discovered via our active scanning techniques contributed 66.2% of the DNS queries in the major CDN's DNS logs. Because this CDN handles a large portion of the global Internet traffic, we believe the discovered egress resolvers are responsible for a generally commensurate fraction of the overall DNS activity.

The datasets collected through active measurements and analyzed in this study that are not related to the major CDN are publicly available at [1].

2 Terminology

DNS literature uses the terms "resolver", "recursive resolver (RDNS)", and "local DNS server (LDNS)" to refer to the servers that provide DNS recursive resolution service to end-user devices. We will use the term resolver. "Open resolver" refers to a resolver that accepts DNS queries from the public Internet, as opposed to being restricted to specific clients. The term "authoritative DNS server (ADNS)" refers to servers that maintain DNS records for a section of the DNS namespace (a "zone") and are able to provide authoritative DNS responses to queries for names within the zone.

Some DNS deployments exhibit complex resolution paths (Figure 1) involving multiple resolvers as discussed in [34]. Following [34], we refer to the resolver that receives DNS queries directly from end clients as "ingress resolver", while

[3] The resolvers that directly interact with authoritative DNS servers – see Section 2.

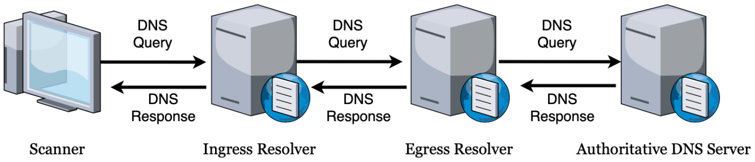

Fig. 1: An example of a complex resolution path which involves ingress and egress resolvers.

an “egress resolver” communicates directly with authoritative DNS servers. The ingress resolver may also act as an egress resolver or may forward queries from the end clients, potentially through several intermediaries, to an egress resolver that obtains the response from the authoritative servers and forwards the response back to the ingress resolver, which finally sends the response back to the client. In the latter case, the ingress resolver is often referred to as a “forwarder”.

A DNS query includes multiple fields. Of particular relevance to this paper are the query string, denoted “QNAME”, which is a name within the DNS namespace and the query type, denoted “QTYPE”, which indicates what type of resource is desired (e.g., “A” for an IPv4 address, “MX” for a mail server, and “TXT” for arbitrary text).

DNS TCP fallback refers to a scenario in which the ADNS sets the TC flag in its response to a UDP query, indicating a truncated response (we also use ”TC response” for brevity), and the querying resolver repeats its query through TCP to retrieve the full response. Modern DNS platforms often employ collaborative resolution, where a resolver may forward the follow-up querying tasks to one of its resolver peers [3,31,34], complicating the analysis of TCP-fallback capability. In our characterization of resolvers’ TCP behavior, we identify TCP-fallback capable resolvers by the initial UDP query, whether the subsequent TCP fallback arrives from the same resolver or from a peer. Because we find negligible number of DNS interactions to be originally conducted over TCP, we equate DNS-over-TCP capability of resolvers with their TCP-fallback capability in this paper.

3 Related Work

Geoff Huston measured DNS TCP-fallback support among resolvers in 2013 [17]. He found that 83% of 80,505 measured resolvers are capable of TCP-fallback. In our study 8 years later, we find much greater support, with 95.2% - 97.3% of 116,851 measured resolvers being TCP-fallback capable. A further explanation for the difference besides time is that Huston did not consider collaborative resolution, where the the original UDP and the followup TCP queries come from different resolvers (see Section 4.3).

TCP fallback was also considered by Moura et al. as part of a broader study on the implications of large DNS responses [30]. By using a passively collected dataset at .nl TLD, this study observed many more recursive resolvers than we could with active measurements (over 3M vs. over 100K in our study) but at the expense of less accurate analysis due to inability to craft special queries with unique names[4]. In particular, this study could only indirectly bound the effect of collaborative resolution, while we are able to associate queries that belong to the same resolution, via names that are used only once, whether or not they come from the same resolver. Moura et al. also assesses the TCP fallback failure to be more prevalent than we observed: after correcting for the specifics of Google's resolution platform, they found roughly 7–10% of truncated responses to lack a TCP query follow-up[5]. In contrast, we estimate that, out of all queries to a major CDN from the resolvers we study, around 1.1–4.4% of queries are from the resolvers incapable of TCP fallback.

In 2016, Shulman and Waidner [35] studied the support for DNS-over-TCP in ADNS and found nearly 20% of the 170K ADNS serving the top 50K domains in the Alexa list could not return a DNS response over TCP. In contrast, we find 423K (95%) out of 445K tested ADNS in our All Domain List (described in Section 5) to be *always* TCP-capable[6]. Clearly, we observe substantially lower levels of incapability. Further, Shulman and Waidner found that the majority of failures (13% of all ADNS) occurred *after* the TCP handshake, while the experimental client was waiting for the DNS response. In our study, we found the opposite: 3.8% of the total tested ADNS failed exclusively during the TCP handshake and only 1.2% ever failed after the TCP handshake.

Vixie and Schryver [38] propose opportunistically setting the TC flag in DNS responses as a mitigation technique for reflection and amplification attacks, rather than dropping queries outright. A legitimate resolver can then retry over TCP, while the attacker gains no amplification from the responses. However, this technique is only effective if the legitimate resolvers support DNS-over-TCP.

Several works [5, 7, 25, 40] study the impact of encrypting DNS (i.e., DNS-over-TLS, or DoT, [15] and DNS-over-HTTPS, or DoH, [12]). In particular, Zhu et al. [40] argue that DNS-over-TLS has acceptable performance costs, in large part due to connection and TLS negotiation reuse. While both DOT and DOH use TCP for transport, these are new protocols that require an explicit adoption by the parties. In contrast, DNS-over-TCP and TCP fallback are part of the existing DNS standard that must be supported by every party. Thus, in principle, an ADNS should be able to switch to TCP unilaterally. Our study examines the extent to which this holds in practice. At the same time, the race condition we uncover (Section 6) applies to both DoT, which explicitly adopts the connection management from DNS-over-TCP (see Section 3.4 in [15]), and DoH as judged from our personal communication with one of the leading CDNs. Thus, in this aspect, our study informs potential support for encrypted DNS.

[4] Other available passive datasets, such as DITL [9] entail a similar limitation.

[5] Indeed, 47% of 15-21% of TC responses initially found without a TCP query follow-up were to Google resolvers, virtually all of which the authors later assessed to be successful fallback to TCP, leaving 7–10% of TC responses still without a TCP followup.

[6] While in Section 5 we categorize TCP support by domain to demonstrate the impact to clients using TCP as their transport medium, here we categorize by ADNS for easy comparison to Shulman and Waidner.

4 TCP Fallback Support by Recursive Resolvers

4.1 Methodology

To investigate DNS-over-TCP support in resolvers, we collect datasets from four sources. Three of the datasets are actively collected; each offers a view into a specific population of resolvers and we collect all three to provide a larger view of resolvers on the Internet. The fourth dataset is aggregated logs from the ADNS servers of a major CDN, which we use to assess the coverage of our study and evaluate the activity level of the resolvers collected in the active datasets.

A limitation of our study is that it focuses on DNS interactions over IPv4 and does not consider IPv6. While we are currently expanding our experiments to cover IPv6, we note that IPv4 still dominates DNS traffic: in the major CDN dataset, queries conducted over IPv4 outnumber IPv6 by 11:1. Further, there is no reason to believe that the choice of the network-layer protocol version would affect the application-level behavior of DNS resolvers[7].

Next, we describe the details of the experiments run to produce each dataset.

Open Resolvers Scan We scan the entire IPv4 public address space (barring reserved addresses and our exclusion list, see Section 7) between February 10 and 11, 2021, with DNS queries for names from our experimental zone, in search of open resolvers. If the scanned IP address returns a DNS response with a NOERROR response code and a resource record, the scanner will send follow-up queries to the discovered ingress resolver to assess support for DNS-over-TCP. Because some ingress resolvers dynamically change IP addresses – a previous study shows that 52.2% of the ingress resolvers change their IP addresses within a week [22] – we launch the DNS-over-TCP capability testing right after the discovery of an ingress resolver, to minimize interference caused by address churn. All queries encode the IP address of the ingress resolver and a nonce so that they cannot be answered from cache.

In probing the discovered ingress resolvers, our goal is to test the TCP-fallback capability of the egress resolvers the open resolvers use by forcing the egress resolver to switch to TCP after a UDP query. The scanner sends two follow-up queries, one A-type and one MX-type (for ease of comparison with our other datasets below which use the same type). Our experimental zone has a single ADNS server that is configured to respond to both queries with the TC flag set and 0 resource records, so that the egress resolvers must re-query over TCP to obtain the records.

Enterprise Resolvers Scan In an effort to assess recursive resolvers used by major enterprises, which are often protected by firewalls that forbid any DNS queries from the outside, we leverage a technique introduced by Klein et al. [19] to induce MX queries from the resolvers used by enterprises to our ADNS. The experiment was conducted on February 15, 2021, and used the Majestic Top

[7] Indeed, a study [30] concerning large DNS responses did include both IPv4 and IPv6 traffic and did not note significant variations in behavior between the two.

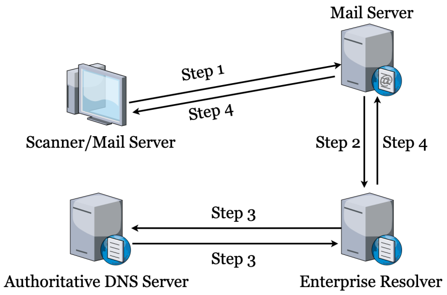

Fig. 2: Inducing a DNS resolution from a private enterprise resolver.

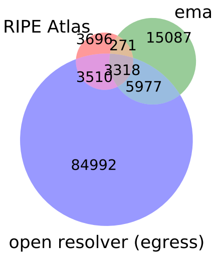

Fig. 3: Egress resolvers discovered in our experiments.

Million “Root Domains List” [27] – which includes many major enterprises – fetched on February 14, 2021.

We measure TCP fallback support of enterprise resolvers using the following process, illustrated in Figure 2:

1. For each mail server used by the enterprise domain (obtained from the MX records of the domain), in the decreasing preference order, we send an email through SMTP and, if failed, through SMTPS, using Python’s *smtplib* library. We stop iterating through the mail servers in this enterprise domain if *smtplib* reports no error. We use random strings as recipients of our emails (i.e., “[random]@majestic.domain”), which are highly unlikely to collide with existing recipients, and we embed the enterprise domain in the sender’s email address (i.e., “research@majestic-domain.our.zone”).
2. When the destination mail server receives the email for the non-existent recipient, the mail server should generate a delivery-status-notification (DSN) and send it back to the original sender, as required by RFC 5321 [21]. To return the DSN, the mail server must lookup via DNS the MX record of the original sender’s domain (i.e., “majestic-domain.our.zone”), although some resolvers send our ADNS an A-type query for the same name, instead. We interpret the latter as a misunderstanding of implicit MX records stated in RFC 2821 [20], which allows mail servers to use the A record of the sender’s domain if no MX records are found. For the purposes of our study, either an MX-type or A-type query from the resolver is sufficient, so we use both.
3. The enterprise resolver interacts with our ADNS to resolve either an MX or A query over UDP. Since our ADNS responds to either query with an empty answer section and the TC flag, the resolver must support TCP-fallback in order to successfully obtain the MX and A records.
4. If the mail server receives a successful response from its resolver, the mail server then sends the DSN back to our scanner (which we set up to double as the MX server for our zone). Regardless, we can assess the TCP-fallback capability of the resolver by observing a TCP query from it (or lack thereof).

Note that for the above technique to work, several prerequisites must be met. First, the mail servers must be willing to receive mail from our SMTP client. Second, the mail server must attempt to send the DSN back to the original sender. Often in our measurement the mail servers indicate an error within the SMTP interaction instead of resolving our MX record and delivering the DSN. Further, some mail servers have catch-all inboxes and provide no feedback about non-existing recipients at all. Another frequent occurrence is that the resolver performs the MX lookup, but the mail server does not send a DSN message to our scanner. In this case, we can still ascertain that the resolver is TCP-fallback capable since the lookup completed.

Another assumption behind this experiment is that enterprise mail servers use the enterprise's general resolution path rather than resolve queries themselves (in which case we would be measuring the TCP-fallback capability of the mail servers rather than the enterprise resolvers). A supporting evidence for this assumption is that there is a large overlap between the resolvers discovered in our enterprise and open resolver scans (see below).

RIPE Atlas Probes Experiment We use the RIPE Atlas platform to gain an insight into another slice of resolvers. RIPE Atlas [33] is a volunteer-based Internet measurement platform managed by RIPE NCC, where each volunteer deploys a measurement probe in their home or institutional network, and experimenters can conduct various measurements from the probes, including DNS resolutions. RIPE Atlas has probes deployed worldwide, but particularly dense deployment in Europe and North America. We used 11,522 probes in our experiments – all available probes that were suitable for our purpose, i.e., those having an IPv4 prefix and not tagged as having problems in resolving DNS queries.

Similar to the technique in the open resolver scan, we embed the probe ID in the DNS query, which makes each probe's query unique and ensures that no resolvers reuse cached responses to answer queries from multiple probes. Further, we direct the probes to use their configured DNS resolvers – whether they happen to be open or closed – to process our queries. Thus, the RIPE Atlas dataset explores a potentially distinct set of resolvers from our open resolver and enterprise email scans. Like the open resolver scan, a RIPE Atlas probe sends a type-A and a type-MX query to its configured resolver, and our ADNS replies to both of these queries with zero resource records and the TC flag.

DNS Logs of a Major CDN Lastly, we collect one week in February 2021 of aggregated logs from the authoritative DNS servers of a major CDN. The dataset includes source IP addresses (i.e., egress resolver) and the counts of queries from each address fielded by the CDN during that week. Due to the high aggregation level, we are not able to identify whether the resolvers in the logs are TCP-fallback capable by observing TCP follow-up queries to TC flag responses. Instead, we use the CDN dataset to assess the coverage of our study and use the relative counts of queries to evaluate the popularity of the resolvers we measure.

Measurement Instrumentation Losses Our measurements are susceptible to packet losses due to unreliable UDP messages, the ADNS potentially not responding to some queries due to overload, and the scanner not exploring discovered ingress resolvers. We can't formally assess the number of UDP datagrams lost in-network before they reach our hosts, but we have no reason to suspect an unusually high packet losses during our experiments. On the ADNS side, we find that 16 out of 13,836,165 total UDP queries were left without response across all three datasets, which is negligible. Among the TCP queries, 13,793 out of 11,692,210 TCP queries were not responded to, a high response rate of 99.9%. Moreover, our classification strategy (see Section 4.3) is based on the TCP queries that arrive; thus, the unresponsive TCP queries do not affect the accuracy of our egress resolver classification.

As time is a factor in our algorithm (see Section 4.3), we investigate how quickly our ADNS responds to UDP queries. The average response time for UDP queries is 38μs, with 3.0ms standard deviation, so the response is usually very quick. Indeed, the ADNS response took greater than 1s for only 302 (0.002%) of UDP queries, still substantially below the 2s threshold used in our algorithm. We conclude that delays in our ADNS do not materially affect our analysis.

Another source of measurement loss is on the scanner during the open resolver scan. Our scanner uses an LRU cache to keep track of the most recent 200K {QNAME, QTYPE} pairs that produced answers from scanned ingress resolvers. This is to ensure that our scanner does not send follow-up queries to superfluous repeated responses to our scanning queries that some ingress resolvers keep sending. The entries in the cache are identified using a 64-bit xxHash value for performance, and the check if an arriving response is already present in the LRU cache may return false positives on collisions, in which case our scanner will not measure the responding ingress resolver. Our scanner explored 3,051,701 out of 3,052,913 (99.96%) responding ingress resolvers. The small number of unexplored ingress resolvers slightly reduces the number of assessed egress resolvers but does not materially affect our results.

4.2 Datasets

In the open resolver scan, we discovered 97,797 egress resolvers, and 3,052,913 open ingress resolvers.[8] It is important to note that, while the open resolver scan only engages with DNS resolution systems through open ingresses, the measured egress resolvers include both open and closed ones. In particular, we found a sizable number of egress resolvers discovered through the open resolver scan to also serve enterprise networks.

In the email scan, we discovered 24,653 egress resolvers, serving enterprises responsible for 192,164 websites from the Majestic 1M list. In the rest of the

[8] We discover substantially more open ingress resolvers than other recent scans ([2,36]). The likely cause of this discrepancy is that those scans do not include ingress resolvers that respond from a different port (not 53) than the port used in probing, a behavior first observed in [34]. Indeed, our results include 2,010,584 (65.9%) ingress resolvers that do respond from port 53, closely matching the number of resolvers reported by the previous scans.

paper, we refer to these resolvers as “serving a domain” or “used by a domain” for brevity, meaning that these resolvers provide resolution services for clients from the enterprises that operate or are responsible for those domains. We stress that in this experiment, we assess resolvers that the enterprises use, regardless of whether they are operated by themselves or an external DNS service provider.

Finally, in the RIPE Atlas measurement, we discovered 10,795 egress resolvers.

While each of the above datasets aims to capture a distinct class of egress resolvers – the open resolver scan finds public resolvers or those used by open forwarders, the email scan finds resolvers used by enterprises, and the RIPE Atlas scan finds closed resolvers in use by home networks or institutions – unsurprisingly, there is some overlap. Figure 3 is a Venn diagram of the egress resolvers in each dataset. While the overlap is significant, each datasets contributes a significant number of net new egress resolvers. Thus, all three datasets contribute to a more comprehensive picture of DNS-over-TCP support in the wild. In the aggregate, we discover and investigate 116,851 egress resolvers across all three datasets. These resolvers contribute 66.2% of the external queries in the CDN logs. Note that the CDN uses DNS internally as part of the platform operation, and DNS queries originating from the CDN do not represent end-user resolutions. Since this CDN delivers large amounts of popular content accessed by most Internet users, this finding shows that, despite drastic reduction in the number of open resolvers, active measurements can still assess a significant portion of the DNS activity on the Internet. For the rest of the paper, we focus on the aggregate set of egress resolvers, unless otherwise stated.

4.3 Resolver Categorization Algorithm

Our open resolver scan finds a large fraction of failed DNS interactions when the ADNS forces TCP fallback: almost a quarter (24.35%) of the overall type-A resolutions involving TCP fallback ultimately fail, i.e., the ingress resolver returns a response with an error flag, or response with no TC flag and no resource records, or no response at all (despite having been discovered through a successful UDP-based DNS probe) or the response carrying a wrong transaction ID. However, it is unclear if these failures are not due to a potential bias in the set of the ingress resolvers we are able to interact with, since they are selected based on a specific trait (being open to external queries) that distinguishes them from all other ingress resolvers. Thus, our assessment of the resolvers' TCP fallback capability focuses on characterizing egress resolvers rather than on whether or not the ingress resolvers ultimately returns the answer to our scanning host. As noted in Section 4.1, we are able to assess the egress resolvers responsible for two-thirds of the overall DNS activity observed by ADNS of the major CDN.

Identifying TCP-fallback capability of an egress resolver in our datasets requires associating the UDP query it sends with the subsequent TCP query. Several factors significantly complicate this seemingly easy task. First, we find that a single query emitted by our scanner often elicits multiple queries at our

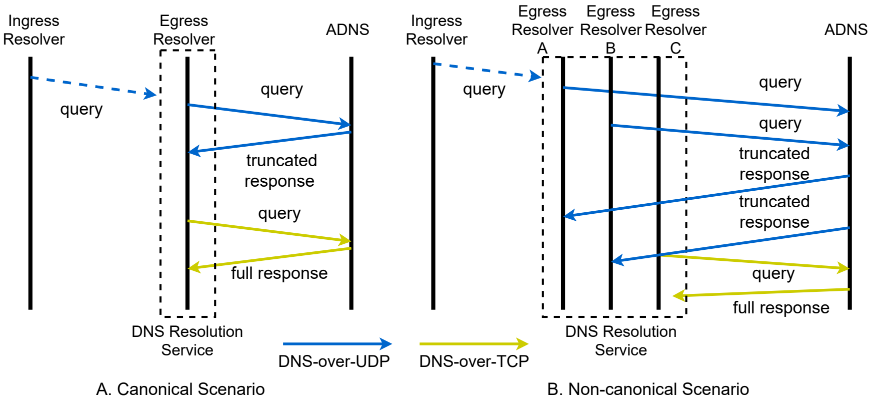

Fig. 4: Examples of a "canonical scenario" where it is trivial to match the UDP and TCP queries and a "non-canonical scenario" where it is hard to tell whether resolver A or B triggers the TCP-fallback.

ADNS (see Section 4.4). This complicates matching a UDP query with the followup TCP query because multiple UDP queries may be candidates for being associated with the same TCP query. Second, these queries may come from different egress resolvers, and in fact TCP queries may come from *different* egress resolvers than the UDP queries they succeed, a behavior indicative of collaborative resolution previously reported (e.g., [3,31,34]).

To illustrate, in the "canonical" TCP fallback scenario shown in Figure 4A, matching the UDP and TCP queries, and identifying the resolver involved as TCP fallback capable, is trivial – there is only one query of each type. However, Figure 4B shows an example of a scenario, where it is unclear, from the ADNS vantage point, which of the two UDP queries triggered TCP-fallback, and which of the two resolvers, A or B is TCP fallback capable.

This section presents our algorithm to determine TCP-fallback capability, as well as the rationale for selecting key parameters of the algorithm.

At the beginning of our data processing pipeline, packet traces captured at our ADNS during the three active measurements are fed to a preprocessing stage. A converter extracts all the DNS messages from the trace, including messages in UDP datagrams as well as reassembling TCP streams to find DNS messages across segments. Next, the few UDP queries that our ADNS did not respond to (see Section 4.1) are discarded. In these cases, the egress resolver did not receive a response, let alone one with the TC flag, so we cannot measure TCP-fallback. Conversely, the TCP queries to which our ADNS did not respond are retained because the TCP query even without a response is sufficient to identify TCP-fallback. If the egress resolver repeats the TCP query after the query to which it received no response, the repetitions are discarded to prevent potential incorrect association with other UDP queries.

Next, the algorithm attempts to associate the UDP and TCP queries and mark matched UDP queries as "TCP-fallback success", unmatched UDP queries as "TCP-fallback failure", and the queries that it could not unambiguously match to a single TCP query as "indeterminate". Finally, the algorithm classifies egress resolvers as TCP-fallback capable or not based on the prevalence of their UDP query markings.

Associating UDP and TCP-Fallback Queries The heart of the algorithm in UDP-to-TCP query matching. It does this by first separating the DNS queries by QNAME and QTYPE, since the UDP and TCP-fallback query must match on these fields. The queries are then clustered such that all the UDP queries in a cluster can plausibly be associated only with the TCP queries in the same cluster, and no cluster can be split without removing some plausible UDP-to-TCP query association. By "plausible association" we mean a UDP query followed by a TCP query within a certain time threshold $t_{threshold}$, a parameter in the algorithm. UDP queries outside any clusters have no plausible associations with TCP queries and are marked as "TCP-fallback failures". Our matching algorithm proceeds to act upon individual clusters.

Appendix A has the full algorithm for constructing query clusters. Briefly, a cluster has the following attributes:

1. For any UDP query within a cluster, there must be at least one TCP query that is within $t_{threshold}$ seconds after it. Otherwise, the cluster can be split after the UDP query in question.
2. For each pair of consecutive TCP queries in a cluster, T_i and T_{i+1}, there is a UDP query U that precedes T_i and also precedes T_{i+1} by less than $t_{threshold}$ seconds. This ensures that U can be plausibly associated with both T_i and T_{i+1} – otherwise, the cluster could be split after T_i.
3. A cluster can include no UDP queries (e.g., if there is a spurious isolated TCP query or a TCP query that is removed from a preceding UDP query by more than $t_{threshold}$), but always includes at least one TCP query.
4. As a corollary of (1), a cluster always ends with a TCP query in chronological order.

Figure 5 shows an example of a series of queries that arrive at ADNS in the aftermath of a single probe to an ingress resolver. UDP query #13 doesn't precede any TCP query, and therefore is labeled as TCP-fallback failed by our algorithm. The rest of the queries are split into four clusters The first cluster consists of queries #1 through #4 because UDP query #1 is within $t_{threshold}$ seconds before TCP query #4. TCP query #5 is in a cluster alone because there are no UDP queries within the time threshold preceding it. UDP query #6 and TCP query #7 make up the third cluster, and the remaining queries #8 through #12 form the last cluster.

Within a cluster, the algorithm scans queries in chronological order and attempts to greedily assign each UDP query to the next unclaimed TCP query within the $t_{threshold}$ window. If all UDP queries are successfully assigned to their own exclusive TCP query, they are marked as TCP-fallback success. Otherwise, all UDP queries that could plausibly match to the same TCP query are

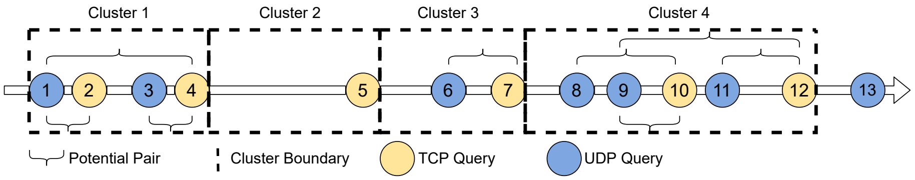

Fig. 5: Splitting queries into clusters based on plausible TCP-fallback association.

marked as indeterminate since we cannot determine conclusively which triggered TCP fallback and which did not.

Returning to the clusters in Figure 5, while both UDP queries #1 and #3 can be associated with TCP query #4, this would leave TCP query #2 unmatched – an unlikely scenario. The algorithm instead greedily matches UDP query #1 with TCP query #2, and UDP query #3 with TCP query #4. Thus, both UDP queries are marked as TCP-fallback success. UDP query #6 is trivially associated with TCP query #7 because there are no other UDP queries that can plausibly associate with TCP query #7. In the fourth cluster, the algorithm will attempt to greedily match UDP query #8 with TCP query #10, then UDP query #9 with TCP query #12, and then is left with UDP query #11 that could not match to a dedicated TCP query. At this point the algorithm will mark all three UDP queries as indeterminate because it cannot tell if UDP queries #8 and #9 should share TCP query #10 (and thus be marked indeterminate) leaving UDP query #11 to be exclusively matched to TCP query #12 (and thus marked TCP-fallback success), or UDP query #8 should be exclusively matched to TCP query #10 (thus marked TCP-fallback success) while UDP queries #9 and #11 should share TCP query #12 (and thus be marked indeterminate).

Selecting Time Threshold The algorithm uses one parameter which must be tuned, $t_{threshold}$ – the time for matching UDP queries to potential TCP-fallback queries. Logically, this time comprises the propagation time of the UDP response with TC flag from the ADNS to the egress resolver, processing time at the egress resolver, and the TCP handshake followed by the initial data segment carrying the TCP query. Clearly, this time can vary substantially among resolvers.

To estimate $t_{threshold}$, we look at scenarios where we can unambiguously associate UDP and TCP queries, i.e., those resolutions that contain exactly one UDP query followed by exactly one TCP query. We call these scenarios “canonical”. Because in the canonical scenarios we can be confident that the TCP query is a result of TCP-fallback from the UDP query, we measure the time between the queries and use the times to inform our choice of $t_{threshold}$.

Figure 6 shows the cumulative distribution function (CDF) of the measured time between UDP and TCP queries in the canonical scenarios in our three datasets. In all three datasets, nearly 100% of TCP-fallback happens within 2 seconds. Thus, we choose 2 seconds for $t_{threshold}$ as a conservative estimate of the maximum time TCP-fallback should take. The CDFs do show that this threshold will miss a small number of cases where TCP-fallback takes longer

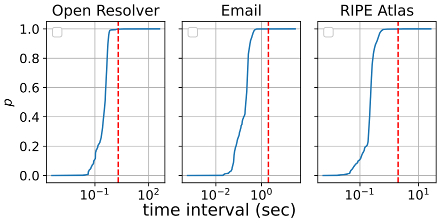

Fig. 6: CDF of the fallback time gaps for canonical scenarios in all three datasets. The red dash line shows $t_{threshold}$.

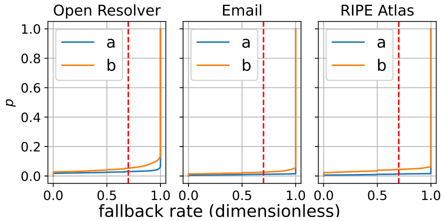

Fig. 7: CDF of the (a) optimistic and (b) pessimistic fallback rates of the resolvers across all three datasets. The red dash line shows $t_{threshold}$.

than 2 seconds. However, a resolver that requires more than 2 seconds to fallback can be treated as TCP-fallback incapable (or at least "impaired") since it will negatively impact the end-user experience.

Categorizing Egress Resolvers The above algorithm labels individual UDP queries as leading to TCP-fallback success, failure, or indeterminate. However, moving from the characterization of individual queries to the overall characterization of the egress resolvers involves several complicating issues.

One issue is that many egress resolvers in our datasets handle multiple queries, and we find that the labeling of those queries sent by the same resolver may not be consistent. There are many possible causes of this inconsistency. First, UDP queries in complex resolution patterns may be marked indeterminate rather than TCP-fallback success or failure due to coinciding with UDP queries from other egress resolvers, with different TCP fallback capability. Second, UDP queries that arrive from the same IP address may in fact come from different resolvers in a server farm behind a common NAT box, again with different TCP-fallback capability. Finally, egress resolvers may selectively perform TCP-fallback due to rate limits, policy, or other factors. Thus, we characterize the overall TCP-fallback capability of an egress resolver by the predominance of their individual UDP query markings.

Another complication is that indeterminate UDP queries introduce uncertainty into the characterization. To address this issue, we bracket its effect by treating indeterminate queries as TCP fallback-success or failure and considering, respectively, the optimistic and pessimistic successful fallback rates of resolvers. We refer to the resolvers characterized under these assumptions as, respectively, optimistically and pessimistically TCP-fallback (in)capable.

Figure 7 shows the CDFs of the successful fallback rate for egress resolvers in each of our three datasets. A majority of resolvers exhibit successful fallback rate of 1.0, removing the first complication in characterizing their TCP fallback capability. In the aggregated dataset, the successful fallback rate is 1.0 for 102,802 to 110,394 (88.0%- 94.5% of all) of the egress resolvers depending on pessimistic or optimistic treatment of indeterminate queries, making these resolvers optimistically or pessimistically TCP-fallback capable. To classify the minority of egress resolvers with the successful fallback rate between 0.0 and 1.0, we opt to take a threshold approach. At around 0.7 on the X-axis (the dotted red line), we note that the lines appear to be fairly flat, meaning that large steps in the threshold would produce small changes in the classification. Thus, we classify the egress resolvers with the successful fallback rate of 0.7 or higher as TCP-fallback capable. In addition to being insensitive to variations in value, this threshold requires the resolver to exhibit fallback success – at least under the optimistic assumptions – in a sizable majority of UDP queries in order to be considered TCP-fallback capable.

In the rest of the paper, we will use a range to represent the number of TCP-fallback capable and incapable resolvers according to the optimistic and pessimistic calculations.

Validating the Algorithm We validate our algorithm by looking at the end-to-end resolution outcomes for A-type queries in our open resolvers scan, with the following three assumptions:

1. If a DNS query is resolved exclusively by TCP-fallback *incapable* egress resolvers, the ingress resolver should reply to our scanner with a failed DNS response or no response at all.
2. If a DNS query is resolved exclusively by TCP-fallback *capable* egress resolvers, the ingress resolver should reply to our scanner with a successful DNS response.
3. Should our matching algorithm fail to match a UDP query with the fallback due to the latter being delayed by longer than $t_{threshold}$, the end-to-end response would also be delayed. As mentioned earlier, characterizing the resolver involved as incapable is not unjustified because of the excessive response time.

Note that we expect these assumptions to only hold as general trends because – as discussed earlier – the same egress resolver (as represented by its IP address) may exhibit inconsistent TCP-fallback capability in different resolutions but in the end is categorized by the algorithm as either TCP-fallback capable or not.

We consider the end-to-end resolution successful if our scanner receives a response with the matching transaction ID and the NOERROR response code, and which either carries the TC flag (71.8% of successful responses) or, if the TC flag is not set, includes some resource records in the answer section (28.2% of successful responses); otherwise the resolution is considered to have failed. A response that carries the TC flag – which may have zero answer records – we consider successful because, after such a response, a real client would retrieve the answer records through TCP. This assumption is conservative because one cannot preclude the possibility that the upstream resolvers blindly copy back the ADNS response with TC flag, and without the ability to perform TCP-fallback. We could not further assess the TCP-fallback capability of the resolution path by sending a TCP query to the ingress resolver, because most ingress resolvers that we found to be open to UDP queries are not open to TCP queries. However, 65.7% of all transactions considered successful – including 37.5% with the TC flag – contain at least one resource record returned by our ADNS over TCP, precluding the possibility of the success misjudgment.

Of all the resolutions that were handled by optimistically TCP-fallback capable egress resolvers only, 88.9% returned a successful response to our scanner; in contrast, for the resolutions handled by optimistically TCP-fallback incapable egress resolvers only, 79.9% returned failed responses. Furthermore, for the 20.1% of queries that were proxied by optimistically TCP-fallback incapable resolvers only but came back to our scanner as successful responses, we found that their end-to-end response time is significantly higher than the queries that were proxied by TCP-fallback capable egress resolvers. Comparing the cumulative distribution functions of the two sets of response times above, shown in Figure 8, 75% of resolutions involving all incapable egress resolvers had end-to-end response time over 1 second, while only 25% of resolutions involving all capable egress resolvers were as slow.

These results show high correspondence between our algorithm's characterization of TCP-fallback capability of egress resolvers and positive end-to-end resolution outcomes when fallback is required. We conclude that our algorithm can successfully characterize resolvers from this perspective.

Cluster pattern	% of clusters
U	0.7%
UU	0.4%
>2 UDP queries	2.0%
T	2.9%
UT	50.4%
UUT	1.7%
UTUT	19.2%
UUTT	13.6%
Other	9.0%

Table 1: Pattern prevalence of query clusters. Many rare patterns are aggregated together in "Other".

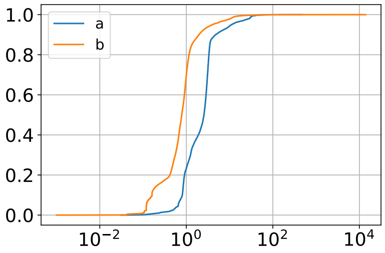

Fig. 8: CDF of the end-to-end response time for successful TCP-fallback responses with (a) optimistically TCP-fallback incapable and (b) optimistically TCP-fallback capable egress resolvers

4.4 DNS Resolution Patterns

Query clusters represent scenarios unfolding in individual DNS resolutions of a domain name. We now examine the patterns of UDP and TCP queries in these clusters. This analysis sheds light on the sequence of DNS transactions comprising a single resolution when the authoritative DNS server forces a TCP fallback. Given that each cluster includes a sequence of UDP and TCP queries for the same QNAME/QTYPE pair, we represent query patterns in clusters as sequences of symbols "U" and "T" to denote, respectively, the UDP and TCP queries in a cluster, in chronological order.

Table 1 breaks down the query clusters by pattern. Note that the table includes patterns composed exclusively of UDP queries even though the algorithm does not consider them as "clusters" for the matching purpose. These patterns include all UDP queries for a given QNAME/QTYPE pair that cannot be plausibly associated with any TCP queries, i.e., are TCP-fallback failures.

First, we note that 2.9% of clusters contain a singleton TCP query that could not be matched with any UDP query according to our algorithm. This does not match our expectation that unsolicited TCP queries are very rare, and therefore we investigate them further. The singletons account for 0.8% of all DNS queries in our dataset and arrive from 1549 different egress resolvers. One possible explanation for the singletons is TCP-fallback that took longer than $t_{threshold}$ (2) seconds. Indeed, our ADNS did receive UDP queries from all but 15 of these resolvers at some point during our scans, and 79% of the singletons are preceded by at least one other cluster that does contain a UDP query. Still, since our algorithm could not associate the singletons with any UDP query, they are excluded from our analysis, potentially leading to UDP queries being marked as TCP-fallback failure. In an effort to determine whether the singletons affect our results, we bound their potential impact by marking each as TCP-fallback success, since the egress resolvers involved are clearly capable of DNS-over-TCP. We find that the results of our analysis in Section 4.5 do not change discernibly since most of the egress resolvers sending the singletons are determined to be TCP-fallback capable anyway from their other interactions: the number of TCP-fallback capable resolvers increases by just 26 to 33 for optimistic and pessimistic cases, respectively, from over 100K TCP-fallback capable resolvers found without the singletons. Thus, we exclude the singleton TCP queries for simplicity.

Surprisingly, the "canonical" expected scenario, where a single UDP query is followed by a single TCP-fallback query, represents barely a majority of interactions, only 50.4%. Others involve multiple redundant queries, with a sizable number, 83,987 (9.0%), of interactions following complex patterns beyond those listed in the table. While DNS scans typically leave a low-rate residual trickle of queries arriving long after the scan, the fact that so many resolvers, including well-known and well-administered ones, routinely send redundant queries, is unlikely due to obscure bugs. These behaviors more likely represent complicated resolution processes and require a separate study. In the meantime, the complexity and diversity of the cluster patterns highlight the need for a commensurately sophisticated matching algorithm, of the kind we use in this paper, to identify the TCP-fallback capabilities of the resolvers involved.

4.5 Results

In this section, we share the results of our analysis. As shown in Table 2, of the 116,851 egress resolvers discovered across all three datasets, between 111,284 to 113,673 (95.2% - 97.3%) are TCP-fallback capable according to the pessimistic and optimistic classification, respectively. This leaves a sizeable number, under either classification approach, of incapable egress resolvers in the wild, suggesting that the Internet is not ready for DNS-over-TCP, as content providers are

Resolver Category	Num.	% resolvers	% queries to CDN
Optimistically capable	113,673	97.3%	98.9%
Pessimistically capable	111,284	95.2%	95.6%
All egress resolvers	116,851	100%	100%

Table 2: TCP-fallback capabilities of egress resolvers.

unlikely to move to a protocol that cuts off a non-negligible amount of their consumers.

We also note that many resolvers support EDNS0 extended payload [6], which allows handling of oversized messages over UDP and thus reduces the need for TCP. Interestingly, we find that TCP-fallback incapable resolvers are *more* likely to support EDNS0 than their TCP-fallback capable counterparts. From our ADNS logs, 65,501 (56.0%) of all the egress resolvers support EDNS0, while 2037 (64.1%) of optimistically classified, and 3633 (65.3%) of pessimistically classified TCP-fallback incapable resolvers support EDNS0. We note, however, that EDNS0 and DNS-over-TCP are not interchangeable because in EDNS0 the server must agree to the payload size the client advertises and very large messages may still exceed it, and because DNS-over-TCP mitigates amplification attacks while EDNS0 exacerbates them. In any case, TCP support is mandatory per RFC 7766 [8], so egress resolvers that support EDNS0 but not TCP are still in violation of the specification.

We consider the significance of the TCP-fallback incapable egress resolvers, that is, how actively they are used in practice. To this end, we turn to the DNS Logs of a Major CDN dataset to assess how many DNS queries the egress resolvers drive. The 2.7% - 4.8% of egress resolvers that are TCP-fallback incapable contribute 1.1% to 4.4% of all queries arriving at the CDN from the resolvers we study. Thus, TCP-incapable egress resolvers are roughly as active as their TCP-fallback capable counterparts.

Enterprise-Centric View Out of the 999,546[9] domains in the Majestic Top Million "Root Domains List" used in our Enterprise Resolvers Scan, we were able to receive DNS queries from resolvers serving 192,164 domains, or 19.2%. The rest either refused our SMTP connection or responded with an error within the SMTP interaction, without ever sending our ADNS a DNS query. In this section, we classify the domains based upon the TCP-fallback capability of the DNS resolvers used by the enterprises that own those domains – which we again stress could be operated or administered by the enterprises themselves or third-party DNS service providers. Note that the relationship of domains to resolvers may be many-to-many: we observe both (*i*) domains served by several resolvers, with or without TCP-fallback capability (see below for the techniques we use to detect these cases), and (*ii*) cases where a resolver serves multiple domains. Moreover, the same resolver may successfully fallback to TCP when conducting a resolution on behalf of one domain, yet fail to fallback for another domain.

[9] We missed 454 domains due to an issue with retrieving the full list from Majestic's website.

One way we detect that a domain uses multiple resolvers is when the domain's ADNS contains multiple MX records, and our scanner tries several of the listed mail servers searching for one that would accept our email. Some mail servers trigger DNS queries even if they don't accept our email, and we observe these queries coming from different egress resolvers. However, as reported in Section 4.3, some DNS interactions involve complex resolution patterns, and we also find that a single attempt at an email to a domain may trigger repeated DNS queries from this domain, sometimes from different egress resolvers. This provides us with another, incidental, way to observe a domain that uses multiple resolvers. Of all the measurable domains, we detected 103,549 (53.9%) domains use multiple recursive resolvers.

We break down the measurable 192,164 domains into the following categories according to the resolvers that serve them, again using our pessimistic and optimistic classification, respectively:

- 190,045 - 191,371 (98.9% - 99.6%) domains use only resolvers classified as TCP-fallback capable in the enterprise dataset, and we observe at least one successful TCP fallback in the email scan of each of these domains. This means that the clients of the enterprises responsible for these domains are unlikely to be negatively impacted by an ADNS-triggered TCP fallback.
- 1028 - 479 (0.5% - 0.3%) domains are served by both TCP-fallback capable and incapable resolvers, and we still observe at least one successful TCP fallback when scanning these domains.
- 937 - 160 (0.5% - 0.1%) domains were not observed to use any TCP-fallback capable resolvers, yet we observe at least one successful TCP fallback when scanning each of these domains. Note that this is possible, since a resolver classified as TCP-fallback incapable may still fallback for some queries (<70%).
- 152 (0.1%) domains do not use any TCP-fallback capable resolvers, and we observed no TCP queries from our email scans of the corresponding domains.
- 2 (0.00%) domains are served by at least one TCP-fallback capable resolver (i.e., the resolver performed fallback for *some* websites), yet no TCP queries were observed when scanning the corresponding domains.

Overall, compared to our resolver-centric results, enterprises are more prepared for unilateral TCP fallback enforced by ADNS, with 98.9% - 99.6% of the domains using exclusively TCP-fallback capable resolvers and only 154 domains served by resolvers that did not send any TCP queries during our scan.

5 DNS-over-TCP Support by Authoritative DNS Servers

We now turn to the other side of DNS interactions and consider DNS-over-TCP support by ADNS. Failure to support DNS-over-TCP is a violation of RFC 7766 [8], so in this section we examine whether ADNS in the wild accept unsolicited TCP connections and respond to the DNS queries sent over those connections. To this end, we consider three sets of domain names: (*i*) 10.6 million domains queried in a week through a resolution service operated by the major CDN ("All Domains"), measured on January 3 and 4, 2022, and (*ii*) popular websites as represented by the Majestic top-1K "root domains" ("Popular Websites"), measured on January 5, 2022 and (*iii*) 47 content delivery networks, measured on January 5, 2022. whose domains we identified in a separate project ("CDNs"). For the All Domains set, we extract second-level domains (e.g., "example.com") from the QNAME in queries the resolution service receives. If the domain includes two two-character labels (e.g., "co.uk") we also extract the third label (e.g., "example.co.uk") to cover country-code domains[10]. Next, and for all three sets, we discover the NS records of the ADNS serving the domain names in the sets, using publicly available recursive resolvers. Finally, we resolve – using public recursive resolvers again – the name in each NS record to A records, resulting in one or more IP addresses of the ADNS serving each domain name. We note that, in the case of large-scale ADNS operators, such as CDNs, very high-volume content providers, or third-party DNS service providers, this technique is likely to obtain only a small subset of the potentially large number of ADNS operated by the ADNS platform, since some providers use anycast to distribute queries among their ADNS or return different NS records to different resolvers. Still, even with this incomplete measurement, we find – as discussed below – a number of ADNS with problematic DNS-over-TCP support.

To assess the TCP support of a domain name, we send a TCP query to each ADNS serving the domain name. An ADNS supports TCP if it both accepts the TCP connection and responds to the DNS query, both within a 2 second timeout. We note that a TCP query may fail either because the ADNS does not support DNS-over-TCP or because the ADNS is offline. While accepting a TCP connection is sufficient to determine that the ADNS is online, a time out at this step may simply indicate the ADNS is offline. In this case, we send 5 UDP queries with a 0.5 second interval between each, to check the ADNS status. If the ADNS successfully responds to any one of them, the ADNS is online and TCP-incapable, otherwise it is offline and we discard the ADNS from our experiment. Note that our experiments are measuring either the behavior of the ADNS or the middleboxes fronting the ADNS, and cannot distinguish between the two. However, that is a distinction without meaning: If an ADNS is only accessible through a middlebox that blocks DNS-over-TCP, then our analysis reflects the experience of an actual DNS client attempting, per RFC, to use TCP to communicate with this ADNS. In all, we measure DNS-over-TCP support for 445,293 ADNS servers in the All Domains, 2835 ADNS in the Popular Websites, and 224 ADNS in the CDNs sets, respectively. In addition, 22,980 ADNS servers were found to be offline in the All Domains set and thus excluded; no ADNS were offline in the Popular Websites or CDN sets.

[10] While this technique is not exhaustive in discovering all domains from the available QNAMEs, it serves to provide a reasonably broad list for our subsequent measurements. In hindsight, using Mozilla's public DNS suffix list at https://publicsuffix.org/ would have been better.

Domain Category	All Domains		Popular Websites		CDNs	
	Num.	%	Num.	%	Num.	%
TCP queries succeed with all ADNS	10,067,248	95.1%	954	96.5%	36	76.6%
TCP queries fail with all ADNS	280,981	2.7%	6	0.6%	10	21.3%
Mixed outcome of TCP queries	242,571	2.3%	29	2.9%	1	2.1%
All domains tested	10,590,800	100%	989	100%	47	100%

Table 3: Domains using ADNS with a given DNS-over-TCP capability.

Table 3 summarizes the results of this experiment[11]. Over 5% of All Domains fail to resolve a TCP query through some of their ADNS. Among popular websites, which one would hope to be administered well, still around 3% exhibit failures to resolve the website over TCP from at least some of their ADNS. Surprisingly, CDNs are even more likely to fail: 10 of the 47 CDNs studied do not support DNS-over-TCP at all, at least by the ADNS probed. See Appendix B for the tested CDN list and individual CDN results. We conclude that egress resolvers choosing to switch to TCP as their transport medium will encounter a non-negligible amount of resolution failures as a result.

6 Race Condition in DNS-over-TCP Connection Reuse

The RFC standards [8, 29] recommend that a client and server keep TCP connections open to amortize the cost of the TCP handshake over multiple DNS resolutions. These standards further recommend letting clients initialize the closing process, but both parties are free to close the connection at will, after either a timeout or the completion of a DNS resolution.

Our ADNS supports TCP connection reuse, using the following policy: (1) A new TCP connection times out if no query arrives on this connection for two seconds, and (2) A TCP connection used by some queries times out after being idle for five seconds. The timeout values are shorter than the defaults of some mainstream DNS server implementations such as BIND9 due to the high traffic in our experiments but still in compliance with the recommended settings of RFC 7766 ("it is RECOMMENDED that the default server application-level idle period be on the order of seconds") [8].

A number of egress resolvers observed across our scans in Section 4.1 are shared among multiple ingress resolvers or different enterprise domains. When we happen to scan these resolvers in close time proximity, we observe them reusing their TCP connections to our ADNS. Of the 114,909 resolvers that used TCP for some of their queries to our ADNS, we observed 3653 to reuse a TCP connection, with the longest connection lasting over 24 hours. Furthermore, these connection-reusing resolvers are very active, accounting for 37.5% of all the queries from the resolvers we discovered in the major CDN's DNS logs. In fact, by deliberately grouping email probes close in time for the enterprises that

[11] The table includes only 989 of the 1K popular websites because eleven of these domains either provide no NS records, the name servers listed fail to resolve to IP addresses, or none of the IP addresses responded to TCP or UDP queries.

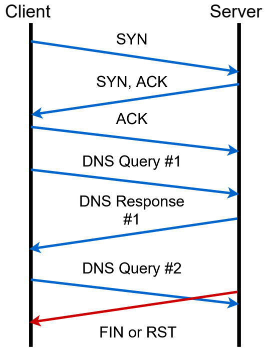

Fig. 9: An example of a race condition, where the client is attempting to reuse an established connection that the server is closing.

share egress resolvers, we were able to induce more connection reuse and observe 13.5% of enterprise resolvers (3135 out of 23197 that established TCP connections to our ADNS) reuse their connections. Clearly, TCP connection reuse is deployed in production on resolvers that are responsible for a significant part of the Internet DNS activity.

As a worrisome observation, we found incidents of a race between a connection reuse attempt from the egress resolver and a connection closing by our ADNS. Significantly, this race condition is not an artifact of a bug in an implementation but is a behavior allowed by the protocol specification. Consider the scenario in Figure 9, where the client would like to reuse the connection, and the server decides to close the connection while the second query from the client is in flight. In this scenario, the second query arrives at the closing server, in which case the server is either unable to receive the query from the socket, or unable to send the response back, depending upon implementation. In either case, the second query will be unanswered, and the client must retry, adding delay.

While this edge case can occur both when the server times out on an idle connection and when the server does not support connection reuse at all and closes the connection immediately after returning a DNS response, the latter is especially problematic as it makes the race condition more likely. Indeed, with connections closed after idling, a busy resolver interacting with a popular website may never experience long enough idle time for the connection to close, and the race described here would not arise. At the same time, the immediate unilateral closing of the connection by ADNS will have a high chance to coincide with the next client query, triggering the race.

Furthermore, one incident of this edge case may impact more than one query, particularly since TCP, as a streaming protocol, can transmit multiple DNS queries back-to-back, even within a single segment. Also, in the extreme case, a query retried after encountering the race condition might be sent over another active TCP connection to the ADNS – in fact, this may not be uncommon for busy public resolvers and ADNSs serving large or multiple zones (such as those

Category	Number	%
All established TCP connections can be reused	662	67.3%
None of the established TCP connections can be reused	258	26.2%
Mixed outcome of connection reuse	63	6.4%
All websites that established some TCP connections	983	100%

Table 4: TCP connection reuse by popular websites.

for CDNs). When the ADNS does not support connection reuse, this would lead to repeated failed resolution attempts.

In order to investigate the prevalence of immediate closing by ADNS, we consider handling of TCP connections by ADNS of (*i*) popular websites and (*ii*) a number of CDNs, since their behavior affects all content providers that subscribe to their service. We focus on ADNS that are likely to be busy in this experiment because, as discussed above, they are more prone to trigger the race.

To assess this behavior, our Popular Websites and CDNs scans described in Section 5 actually include two TCP queries over each connection, with the second query sent one second after receiving the response to the first. In accordance with the recommendation of RFC 7766, we expect connection reuse without explicitly including the EDNS0 edns-tcp-keepalive option (see Section 6.1).

Popular Websites Among the 2835 name servers serving the popular websites, 53 servers do not accept TCP connections or fail to respond to the first DNS query in a TCP connection – violating the DNS protocol. Of the remaining servers, only 1861 (66.9%) were observed to respond to the second query at least once. Table 4 breaks down the websites according to their connection reuse behavior. The take-away point is that a third of ADNS servers close their TCP connections immediately after sending a response, and a third of top-1K websites are affected by this behavior as at least some of their ADNS are in this category.

This result shows that roughly a third of popular websites use name servers that leave them highly exposed to the race condition, as some of their name servers do not support connection reuse. Thus, DNS-over-TCP performance for these websites may suffer.

Content Delivery Networks Turning to the ADNS serving content delivery networks, the results are quite different. Out of 37 CDNs that support DNS-over-TCP by at least some of their ADNS, 33 always support TCP connection reuse, while only two-thirds of the popular websites do. However, two out of four CDN providers that do not support TCP connection reuse are two major CDNs, which we label CDN1 and CDN2. In fact, based on their name servers we probed, these two CDNs behaved abnormally – albeit differently – in response to the resolver's attempt to reuse a connection. Neither explicitly closed the connection between servicing the first query and the second. CDN1 did ACK the second query, but then immediately initiated TCP connection closing. CDN2 ACK'ed the second query as well, but did not send a DNS response before our scanner timed out and initiated closing of the connection[12].

[12] We contacted both CDN1 and CDN2 about our findings. CDN1 acknowledged the bug, and CDN2 did not respond.

Overall, we observe that resolvers often follow RFC recommendations and attempt to reuse a TCP connection for DNS-over-TCP queries. At the same time, popular websites commonly close the connections immediately, and some major CDNs, serving large portions of Web traffic, mishandle connection reuse. Either case is problematic: immediate close can lead to a race condition and possible performance degradation, and mishandling leads to a deterministic failure of queries that reuse existing connections and a potentially significant disruption in the DNS resolution process.

6.1 Deployment of edns-tcp-keepalive

RFC 7828 [39] allows clients and servers to negotiate the idle timeout of the current TCP connection through an EDNS0 edns-tcp-keepalive option. The client signals its desired TCP connection keepalive time to the server. Upon receipt, the server can determine the value it wishes to use – either accepting the client's value, modifying it, or rejecting the persistent connection – and return its decision to the client. Support for this EDNS0 option can mitigate the TCP race condition discussed above, especially if the server maintains the connection slightly longer than as negotiated.

RFC 7828 is already supported by some widely deployed DNS software, including BIND9. We assess the adoption of this mechanism by popular websites. To this end, we rescan the authoritative servers of Majestic top-1K websites and the 47 CDNs, including the edns-tcp-keepalive option in the first query to each ADNS. The second query is still sent 1 second after receiving the response to the first, and we set the keepalive to 2 seconds to be significantly larger than our actual interval.

Unfortunately, we find support for the edns-tcp-keepalive option to be sparse. Only 263 ADNS, serving 140 top-1K websites, negotiate the edns-tcp-keepalive option with our scanner by appending their corresponding options in DNS responses. 244 (92.8%) out of 263 of these ADNS set this value to 30 seconds, which is the default option value in BIND9, as well as the default waiting time before it timeouts an idle TCP connection. Further, none of the CDN we study support edns-tcp-keepalive, at least judged by the ADNS we probed. In addition, we find that 36 ADNS servers in Popular Websites set the edns-tcp-keepalive option in their responses but close the connection before the specified keepalive value, resulting in a failed second query. This is obviously a non-compliant behavior.

We note that RFC 7828 does not address the situation where a resolver explicitly requests a keepalive in its first query, the ADNS refuses, but before the resolver receives the response with the refusal, another query arrives at the resolver. An opportunistic resolver may pipeline the second query through the current connection, and it will be left unanswered.

6.2 Addressing the Connection Reuse/Closing Race

We showed a high potential for the race between TCP connection reuse and closing: many resolvers attempt to reuse a connection when it is available and many ADNSs close their connections right after sending a response. As long as DNS interactions over TCP are rare, the few TCP queries that do occur are unlikely to encounter another connection for reuse, and the above race will be rarer still. Indeed, on a 5-minute packet trace of TCP queries at one of the ADNS servers at the major CDN, resolvers reused only 19 of the 15436 observed TCP connections. Still, we believe this issue needs to be addressed at the protocol level, before it can inflict practical harm, in the case the current activities within the DNS community do lead to a shift towards connection-oriented DNS communication. We stress that this issue extends beyond DNS-over-TCP, as DNS-over-TLS [14] and DNS-over-QUIC [16] explicitly inherit connection reuse policies from DNS-over-TCP. We believe the following simple modifications to these policies would remove a possibility for the race.

- A resolver *must* not reuse a TCP connection unless an explicit edns-tcp-keepalive negotiation has been completed, so that resolver would know for how long the ADNS will maintain the connection.
- Similar to timed wait in TCP, An ADNS *must* retain an active connection for 2MSL (maximum segment lifetimes) beyond the negotiated keepalive duration. At the same time, the resolver *must not* reuse a connection beyond the negotiated keepalive duration.
- As an optional optimization (a further study would be needed to decide if this is worth the complications), a resolver may indicate its support for TCP connection reuse in an EDNS0 option with its initial UDP query. An ADNS that supports persistent connections may then indicate a default keepalive value with its UDP TC response to such a query, allowing the client to immediately learn the possibility of reusing the fallback connection. The client, in the TCP fallback, can choose any keepalive value that does not exceed the indicated default. The ADNS *must* accept this value during the TCP interaction.

7 Ethical Considerations

We realize our scans may be confused for malicious activity by some scanned networks, or be otherwise unwanted. We employ the following measures (representing to the best of our knowledge the best practices developed by the measurement community) to minimize the affect to the Internet.

- We randomly shuffle the targets of our open resolver scan to avoid high probing rates for a given network and triggering alerts for address scanning.
- We encode our contact information in the query strings used in our open resolver measurement. The strings are formatted according to the following pattern: "[keyword]-[target-IP]-email-[email-addr].our.zone".
- All the scanner machines we used in this paper have publicly accessible reverse DNS records, and the organizations of these machines can be looked up in WHOIS database by their IP addresses.

- We embed the same message in the email sent in our enterprise resolver measurements as well.
- We maintain an exclusion list, which includes the IP addresses and the hostnames of the organizations who previously expressed their unwillingness to join experiments. We exclude these IP addresses and hostnames from subsequent measurements.

We received 14 complaints and inquiries (including one notification from a public shared spam reporting service) from our enterprise email scans. We responded to all of them quickly and excluded from our future experiments those who expressed their unwillingness to participate. We did not receive any complaints from our open resolver and RIPE Atlas scan.

8 Conclusion

In this paper, we assess the support of DNS-over-TCP, which has generated significant interest due to its more secure nature. On the one hand, while we find significantly higher support for TCP fallback by recursive resolvers than prior studies, there is still a number of resolvers that are not capable of TCP fallback. In particular, we assess 116,851 egress resolvers, responsible for 66.2% of all queries to a major CDN's ADNS platform, and find that 2.7%-4.8% of them, contributing 1.1% to 4.4% of all queries from the resolvers we measured, were unable to perform a TCP fallback when instructed by an ADNS. Thus, ADNS operators deciding to force DNS-over-TCP usage via TCP fallback face the risk of cutting off a non-negligible amount of their potential users.

On the other hand, we find a number of authoritative name servers, including those serving some popular websites and several content delivery networks, to not accept DNS queries over TCP. Indeed, around 3% of popular websites, and over 5% of domains at large, are served by at least some ADNS failing to answer queries over TCP, while 9 out of 47 CDNs we consider also exhibit this behavior. Thus, a resolver operator choosing to switch to DNS-over-TCP today would essentially make its users unable to reliably access these domains as well as all content delivered by these CDNs.

Further, we uncover a race condition that may occur in DNS-over-TCP and find that 32.4% of authoritative DNS servers serving Majestic top-1K popular websites are vulnerable to this condition. Finally, we observe abnormal behavior by two major CDNs in their DNS-over-TCP support.

We hope our findings will inform DNS operators who consider possible adoption of DNS-over-TCP and help improve DNS-over-TCP support of the platforms that have already adopted it.

Acknowledgement We thank the anonymous reviewers, and especially our shepherd, Alessio Botta, for useful comments and guidance. We are indebted to CWRU IT organization ("UTech") for its continued support, without which this research would not be possible. The work of Jiarun Mao and Michael Rabinovich was supported in part by NSF through grant CNS-1647145.

References

1. https://osf.io/6ysxv/
2. Al-Dalky, R., Rabinovich, M., Schomp, K.: A look at the ECS behavior of DNS resolvers. In: Proceedings of the ACM Internet Measurement Conference. pp. 116–129 (2019)
3. Al-Dalky, R., Schomp, K.: Characterization of collaborative resolution in recursive DNS resolvers. In: International Conference on Passive and Active Network Measurement. pp. 146–157. Springer (2018)
4. Barnes, R., Hoffman-Andrews, J., Kasten, J.: RFC 8555: Automatic certificate management environment (acme) (2019)
5. Böttger, T., Cuadrado, F., Antichi, G., Fernandes, E.L., Tyson, G., Castro, I., Uhlig, S.: An empirical study of the cost of DNS-over-HTTPS. In: Proceedings of the ACM Internet Measurement Conference. pp. 15–21 (2019)
6. Damas, J., Graff, M., Vixie, P.A.: RFC6891: Extension mechanisms for DNS (EDNS0) (2013)
7. Deccio, C., Davis, J.: DNS privacy in practice and preparation. In: Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies (CoNEXT). pp. 138–143 (2019)
8. Dickinson, J., Dickinson, S., Bellis, R., Mankin, A., Wessels, D.: RFC 7766: DNS transport over TCP-implementation requirements (2016)
9. DNS-OARC: https://www.dns-oarc.net/oarc/data/ditl (2018)
10. Eastlake, D.: RFC 2535: Domain name system security extensions (1999)
11. Hansen, T., Crocker, D., Hallam-Baker, P.: RFC 5585: DomainKeys identified mail (DKIM) service overview (2009)
12. Hoffman, P., McManus, P.: RFC 8484: Dns queries over https (doh) (2018)
13. Hoffman, P., Schlyter, J.: RFC6698: The dns-based authentication of named entities (DANE) transport layer security (tls) protocol: Tlsa (2012)
14. Hu, Z., Zhu, L., Heidemann, J., Mankin, A., Wessels, D., Hoffman, P.: RFC 7858: Specification for dns over transport layer security (tls) (2016)
15. Hu, Z., Zhu, L., Heidemann, J., Mankin, A., Wessels, D., Hoffman, P.: RFC7858: Specification for dns over transport layer security (tls) (2016)
16. Huitema, C., Shore, M., Mankin, A., Dickinson, S., Iyengar, J.: Specification of DNS over Dedicated QUIC Connections. Internet-Draft draft-huitema-quic-dnsoquic-07, work in Progress
17. Huston, Geoff: https://labs.ripe.net/Members/gih/a-question-of-dns-protocols (Aug 2013)
18. Kitterman, S.: RFC 7208: Sender policy framework (SPF) for authorizing use of domains in email (2014)
19. Klein, A., Shulman, H., Waidner, M.: Internet-wide study of DNS cache injections. In: IEEE INFOCOM. pp. 1–9. IEEE (2017)
20. Klensin, J.: RFC 2821: Simple mail transfer protocol (2001)
21. Klensin, J.: RFC 5321: Simple mail transfer protocol (2008)
22. Kührer, M., Hupperich, T., Bushart, J., Rossow, C., Holz, T.: Going wild: Large-scale classification of open DNS resolvers. In: Proceedings of the ACM Internet Measurement Conference (2015)
23. Kührer, M., Hupperich, T., Rossow, C., Holz, T.: Exit from hell? reducing the impact of amplification DDoS attacks. In: 23rd {USENIX} Security Symposium ({USENIX} Security 14). pp. 111–125 (2014)

24. Lian, W., Rescorla, E., Shacham, H., Savage, S.: Measuring the practical impact of DNSSEC deployment. In: 22nd USENIX Security Symposium. pp. 573–588 (2013)

25. Lu, C., Liu, B., Li, Z., Hao, S., Duan, H., Zhang, M., Leng, C., Liu, Y., Zhang, Z., Wu, J.: An end-to-end, large-scale measurement of DNS-over-encryption: How far have we come? In: Proceedings of the ACM Internet Measurement Conference. pp. 22–35 (2019)

26. MacFarland, D.C., Shue, C.A., Kalafut, A.J.: Characterizing optimal DNS amplification attacks and effective mitigation. In: International Conference on Passive and Active Network Measurement. pp. 15–27. Springer (2015)

27. Majestic Top Million Root Domains List: `https://majestic.com/reports/majestic-million` (2021)

28. Mockapetris, P.: RFC 883: Domain names - implementation and specification (1983)

29. Mockapetris, P.V.: RFC 1035: Domain names-implementation and specification (1987)

30. Moura, G.C.M., Müller, M., Davids, M., Wullink, M., Hesselman, C.: Fragmentation, truncation, and timeouts: Are large dns messages falling to bits? In: Passive and Active Measurement. pp. 460–477 (2021)

31. Randall, A., Liu, E., Akiwate, G., Padmanabhan, R., Voelker, G.M., Savage, S., Schulman, A.: Trufflehunter: Cache snooping rare domains at large public dns resolvers. In: Proceedings of the ACM Internet Measurement Conference. pp. 50–64 (2020)

32. van Rijswijk-Deij, R., Sperotto, A., Pras, A.: DNSSEC and its potential for DDoS attacks: a comprehensive measurement study. In: Proceedings of the ACM Internet Measurement Conference. pp. 449–460 (2014)

33. RIPE: Atlas. `https://atlas.ripe.net/` (2021)

34. Schomp, K., Callahan, T., Rabinovich, M., Allman, M.: On measuring the client-side DNS infrastructure. In: Proceedings of the ACM Internet Measurement Conference. pp. 77–90. ACM (2013)

35. Shulman, H., Waidner, M.: Is the Internet Ready for DNSSEC: Evaluating Pitfalls in the Naming Infrastructure. In: International Workshop on Traffic Monitoring and Analysis (TMA) (2016)

36. The Shadowserver Foundation: `https://scan.shadowserver.org/dns/` (2020)

37. Vixie, P.: RFC 2671: Extension Mechanisms for DNS (EDNS0) (1999)

38. Vixie, P., Schryver, V.: DNS response rate limiting (DNS RRL). `http://ss.vix.su/\~vixie/isc-tn-2012-1.txt` (2012)

39. Wouters, P., Abley, J., Dickinson, S., Bellis, R.: RFC 7828: The edns-tcp-keepalive EDNS0 option (2016)

40. Zhu, L., Hu, Z., Heidemann, J., Wessels, D., Mankin, A., Somaiya, N.: Connection-oriented DNS to improve privacy and security. In: 2015 IEEE symposium on security and privacy. pp. 171–186. IEEE (2015)

A Matching Algorithm

Algorithm 1 Split an array of DNS queries with a given (QNAME,QTYPE) pair into an array of clusters (as defined in Section 4.3). Queries that cannot be assigned to any cluster are added to a pseudo cluster.

Input: array *queries* ← all UDP and TCP queries with the given QNAME and QTYPE pair in chronological order.

Input: float *max_time* ← maximum delay between a UDP query and a TCP query in TCP-fallback.

Output: array of arrays *clusters* ← each inner array is a cluster of UDP/TCP queries. The last inner array is the pseudo cluster of UDP queries that cannot be assigned to any cluster. The *clusters* appear in the outer array in chronological order, i.e., the first query in a later cluster comes after the last query in an earlier cluster. The queries within a cluster are also in chronological order.

```
procedure SplitIntoClusters($queries$, $max\_time$)
    $clusters$ ← empty array
    $pseudo\_cluster$ ← empty array
    $cluster$ ← empty array
    for each TCP query $q_{tcp}$ in $queries$ in chronological order do
        $qs_{udp}$ ← UDP queries in $queries$ within $max\_time$ before $q_{tcp}$
        if $(qs_{udp} \cap cluster) = \emptyset$ and $cluster \neq \emptyset$ then        ▷ cluster boundary
            append $cluster$ to $clusters$
            $cluster$ ← empty array
        append each query $q$ in $qs_{udp}$ to $cluster$ unless $q$ is already in $cluster$
        append $q_{tcp}$ to $cluster$
    if $cluster$ is not empty then
        append $cluster$ to $clusters$
        remove all queries in $clusters$ from $queries$
    for each $q_{udp}$ in $queries$ do        ▷ All remaining queries in $queries$ are UDP
        add $q_{udp}$ to $pseudo\_cluster$
    append $pseudo\_cluster$ to $clusters$
    return $clusters$
```

Algorithm 2 Label DNS queries according to whether they represent TCP-fallback successes, failures, or indeterminate cases.

Input: map *m* ← QNAME and QTYPE pair (*"qpair"*) as **keys** and arrays of chronologically ordered queries with the corresponding *qpair as **values***.

Input: float *max_time* ← maximum delay between a UDP query and a TCP query in TCP-fallback.

Output: map *labels* ← queries as **keys** and labels of success, failure, or indeterminate as **values**.

```
1:  $labels$ ← empty map
2:  **for** $qpair$, $queries$ in $m$ **do**
3:      $clusters$ ← SplitIntoClusters($queries$, $max\_time$)
4:      **for** each $cluster$ in $clusters$ excluding pseudo cluster **do**
5:          $used\_qSet_{tcp}$ ← empty set
6:          $indeterminate\_qSet_{udp}$ ← empty set
7:          $pending\_qSet_{udp}$ ← empty set
8:          $pending\_count\_qs_{udp}$ ← 0
9:          $owners$ ← empty map (from UDP queries to tentatively matching TCP queries)
10:         **for** each $query$ in $cluster$ **do**
11:             **if** $query$ is a UDP query **then**                ▷ **else** statement in line 36
12:                 $first\_q_{tcp}$ ← the first TCP query in $cluster$ following $query$ and not in $used\_qSet_{tcp}$
13:                 **if** $first\_q_{tcp}$ is null **then**
14:                     add $query$ to $indeterminate\_qSet_{udp}$
15:                     add queries in $pending\_qSet_{udp}$ to $indeterminate\_qSet_{udp}$
16:                     $pending\_count\_qs_{udp}$ ← 0
17:                 **else**
18:                     add $first\_q_{tcp}$ to $used\_qSet_{tcp}$
19:                     $owners[query]$ ← $first\_q_{tcp}$
20:                     **if** $indeterminate\_qSet_{udp}$ is empty **then**
21:                         add $query$ to $pending\_qSet_{udp}$
22:                         $pending\_count\_qs_{udp} + +$
23:                     **else**
24:                         $last\_q_{udp}$ ← last query in $indeterminate\_qSet_{udp}$
25:                         **if** time($first\_q_{tcp}$) − time($last\_q_{udp}$) $>$ $max\_time$ **then**
26:                             **for** each $q$ in $indeterminate\_qSet_{udp}$ **do**
27:                                 $labels[q]$ ← indeterminate
28:                                 remove $owners[q]$ from $cluster$
29:                                 remove $q$ from $cluster$
30:                             clear $indeterminate\_qSet_{udp}$
31:                             $used\_qSet_{tcp}$ ← $\{first\_q_{tcp}\}$
32:                             $pending\_qSet_{udp}$ ← $\{query\}$
33:                             $pending\_count\_qs_{udp}$ ← 1
34:                         **else**
35:                             add $query$ to $indeterminate\_qSet_{udp}$
36:             **else**                                          ▷ **if** statement in line 11
37:                 **if** $pending\_count\_qs_{udp} > 0$ **then**    ▷ Can be 0 if $cluster$ only has a TCP query
38:                     $pending\_count\_qs_{udp} - -$
39:                     **if** $pending\_count\_qs_{udp} = 0$ **then**
40:                         **for** each $q$ in $pending\_qSet_{udp}$ **do**
41:                             $labels[q]$ ← success
42:                         clear $pending\_qSet_{udp}$
43:         **for** each $q$ in $indeterminate\_qSet_{udp}$ **do**
44:             $labels[q]$ ← indeterminate
45:     **for** each $q$ in pseudo $Q_t$ cluster **do**
46:         $labels[q]$ ← failure
47: return $labels$
```

B CDN Targets Tested

Below we list the 47 CDNs we tested in this study, which includes 17 (shown in bold font) out of 25 CDNs listed at CDN Planet (https://www.cdnplanet.com/, accessed on Jan 2, 2022). The parenthetical information lists the domain name employed and whether all, some, or none of the ADNS tested are TCP capable.
advancedhosterscdn(11799613.pix-cdn.org, all),
akamai(www.a1776.g1.akamai.net, all),
amazoncloudfront(www.d2qjncoblxi5md.cloudfront.net, all),
aryaka(hd.itrip.com.top.aads1.net, none), azion(18697b.ha.azioncdn.net, all),
belugacdn(www.cdn.famefocus.com.i.belugacdn.com, none),
bitgravity(www.pc-ap.bitgravity.com, all),
bunnycdn(www.planetedomo.b-cdn.net, some),
cachefly(www.vip1.g5.cachefly.net, none), **cdn77**(www.1650447009.rsc.cdn77.org, all),
cdnetworks(www.kisa.or.kr.cdngc.net, none),
cdnify(karnataka.a.cdnify.io, all), cdnsun(www.239827766.r.cdnsun.net, all),
cdnvideo(bfm.cdnvideo.ru, all),
cedexis(mobile.interflora.fr.fasterize.it.2-01-295f-000e.cdx.cedexis.net, all),
chinacache(hpcc-page.cncssr.chinacache.net, all),
chinanetcenter(www.v4q3iig12pcnka.wscloudcdn.com, none),
cloudflare(www.upra.org.cdn.cloudflare.net, all), cubecdn(mr.sp.cubecdn.net, all),
edgecast(www.cs109.adn.edgecastcdn.net, all),
facebook(scontent.xx.fbcdn.net, all), **fastly**(www.prod.seamless.map.fastlylb.net, all),
google(www.g0e1hw.feedproxy.ghs.google.com, all),
highwinds(www.cds.v2f8x7x9.hwcdn.net, all),
incapsula(www.hs2rptk.x.incapdns.net, all),
internap(www.6a2809e8d5.site.internapcdn.net, none),
keycdn(p-frpa00-v4.kxcdn.com, all),
leasewebcdn(www.5ad9c8cb35308834bf7d93d4e09de97e.lswcdn.net, all),
level3(www.vc.sporttube.com.c.footprint.net, none),
limelight(ualsharp.vo.llnwd.net, none),
maxcdn(www.creative-watch-new-pull.4ncfzftyhcv4rwo.netdna-cdn.com, all),
medianova(img-cimri.mncdn.com, none),
netlify(www.campusmanagement.netlify.com, all),
ngenix(www.cntraveller-st.cdn.ngenix.net, all),
nyiftw(www.nyi.nyiftw.net, all), onapp(316150366.r.worldcdn.net, all),
optimalcdn(www.cdn.optimalcdn.com, all),
quantil(www.oversea.dtwscache.speedcdns.com, none),
reflectednetworks(www.e-static.pornmd.com.sds.rncdn7.com, all),
rocketcdn(www.mediacdn.karnaval.com.streamprovider.net, all),
singularcdn(h2.singularcdn.net.br, all),
stackpath(www.adoramapix-8u9vvrwnlphhiqnu.stackpathdns.com, all),
swiftcdn(secure.aims.jns.swiftserve.com, all),
unicorncdn(xc3uk5s3rf.unicorncdn.net, all), wordpress(www.2.gravatar.com, all),
yottaa(www.a19af6306e7c013695900a3ba3fac80a.yottaa.net, all),
zenedge(104-225-137-39-tls12.zenedge.net, all)