technische universität dortmund

Bachelor Thesis

Analysis of authoritative DNS infrastructure failures

Ege Girit

January 14, 2023

Reviewer:
Prof. Dr. Christian Rossow
BSc. Jonas Bushart

Contents

1 Introduction

1.1 Motivation

Domain Name System (DNS) is a fundamental protocol that was not designed with security in mind. There are many types of attacks aimed at exploiting vulnerabilities in DNS [1]. If name resolution does not work properly, there is no alternate protocol for name resolution, and the connectivity between users and the server might be compromised. This makes DNS a popular target for hackers. Many systems rely on DNS to remain functional. Therefore, errors or attacks on DNS are of particular importance for everything based on it. If an authoritative name server is unreachable, the client usually has to expect longer waiting times or errors. However, there is a lack of knowledge about the specific behaviors of resolvers during cyber attacks and authoritative server failures. This gap in understanding presents an opportunity for further research.

DNS has been the focus of significant DDoS attacks in recent years. DDoS stands for Distributed Denial of Service and is a type of cyber attack in which a large number of computers, often ones that have been hacked, are used to overwhelm a target infrastructure with an excessive amount of traffic. This is done to make the targeted network or website unavailable to users. A DDoS attack can put a lot of strain on the target server by flooding it with a huge amount of traffic, making it difficult or impossible for the server to process and respond to legitimate requests. As a result, some network packets may not be sent or may be lost, causing problems for legitimate users. A DDoS attack targeting a server may also cause collateral damage by disrupting other services that share infrastructure, such as links, servers, and routers, even if they are not the main focus of the attack [2]. Some examples of DDoS attacks include:

- In 2002, the root servers was hit by a DDoS attack. The attackers sent a lot of traffic to the targeted servers in an attempt to make them unavailable. As a result, the DNS infrastructure and internet services slowed down globally [3].
- In 2015, the root servers was attacked again by DDoS attacks. This made it hard for the servers to handle the traffic and caused them to become overloaded and unable to respond [4, 5].
- In 2016, a botnet called “Mirai” infected a large number of internet-connected devices with malware and used them to launch DDoS attacks on the DNS provider Dyn. These attacks made it impossible to access several popular websites such as Airbnb, GitHub, Reddit, Twitter and Netflix [6].

These attacks show how important it is to secure the DNS infrastructure and the potential consequences of vulnerabilities in this critical component of the internet.

1.2 Structure of the Work

In the first section of this paper, we provide a summary of the DNS, which forms the foundation of our focus. This overview covers the basics of DNS and sets the stage for the subsequent discussions. Building on this foundation, we explain why we believe it is important to study resolver behavior during an authoritative server failure. In the second part of the paper, we describe the specific characteristics of resolvers we want to examine, the sources we use for testing, and the metrics we use to determine the success of an attack. We also outline the test environment and analysis methods. In the third part, we describe the experiments and present the results of our experiments using various graphs to illustrate the findings. We then evaluate these results and draw conclusions about what the tests reveal. Finally, we summarize our findings and discuss the extent to which our results may differ from reality and whether further research is needed in this area.

1.3 History of DNS

The need for the DNS arose from the need to address the IP addresses of servers, which are difficult for humans to remember, with easy-to-remember names. When a user enters a server's name into their computer or device, DNS is used to resolve the name to the corresponding IP address, enabling the user to connect to the server.

Before the DNS, there was only a centralized text file (hosts.txt) consisting of static mappings that allowed users to resolve domain names to their associated IP addresses. This text file had to be updated after each change and distributed to all computers to provide a mapping between these names and their corresponding network addresses. With the growing number of hosts and websites, it became impractical to manage such a centralized system. The administrative overhead associated with managing every possible domain name on the internet would be too great, and this central database would not scale well. To address these challenges, DNS designers abandoned the flat naming approach, where the names do not have any internal structure or clear connections to each other, and instead implemented a decentralized model with a hierarchical naming architecture that became the internet standard in 1987. The official documentation of these standards is described in Request for Comments (RFC) 1034 and 1035 [7, 8]. However, there are numerous other RFCs that detail various aspects of the DNS.

1.4 How DNS Works

Today we have a globally distributed collection of databases that form a tree structure. System administration is completely decentralized, and responsibility for delegation is handed over to organizations. There are two main functions of DNS, which are name resolution and name space. A name space refers to the way that domain names are structured and used in terms of what makes a name valid. In this work, we will focus on the core function of DNS, which is the name resolution. Several actors play a role in resolving names to their corresponding IP addresses. The main actors are:

- The client
- Recursive resolver
- Authoritative name servers

The client is interested in the answer of the name resolution, which is the IP address associated with a domain name. Clients are typically built into operating systems, but they can also be provided by third-party software such as browsers. In this paper, we refer to the client as a device that initiates the resolution process by sending a query. To start the name resolution process, a client sends a DNS query to a recursive resolver. The resolver then takes the role of the client and starts the querying process by trying to find the answer to the query on behalf of the client and returning the results to the client.

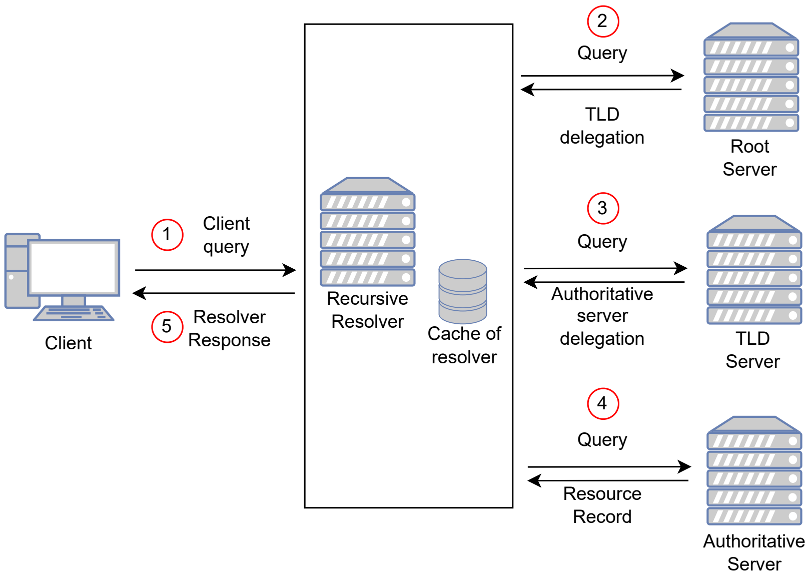

Figure 1.1: An example of a DNS lookup process

The DNS hierarchy consists of a number of different types of servers that work together to resolve domain names to IP addresses. The recursive resolver acts as a middleman between the client and the DNS hierarchy by contacting the DNS hierarchy to get the DNS response. The root and TLD server are two instanciations of authoritative name servers. The first server that the recursive resolver contacts is the root name server. Root servers are located all around the world and are responsible for keeping track of IP addresses and locations of TLD (top-level domain) name servers, which are associated with domain extensions such as .com or .net. The root name server sends back the appropriate TLD server based on the queried host's domain extension to the recursive resolver. The TLD server receives the next request from the recursive resolver and responds with the appropriate authoritative name server. The authoritative name server stores the DNS resource records that map domain names to IP addresses, and it responds to the recursive resolver's final request with the queried hostname's IP address. If the IP address of the queried domain name is unavailable on the authoritative server, the name server will return an error.

A DNS resource record is a type of data that is stored in a DNS database and used to provide information about various aspects of a domain name or IP address. There are several different types of DNS resource records, each with a specific purpose. In this paper we focus on A (IPv4 Address) record, which maps a domain name to an IP address.

To inform whether the DNS request was processed successfully or if an error occurred, the RCODE (Response Code) field in a DNS packet is utilized to indicate the status of a DNS response. A few of the most common RCODE values are 0 (NOERROR), which signals a successful response, and 2 (SERVFAIL), indicating a failure during the resolution process.

Because there are many servers involved in the resolution process, a request from a client can consume a lot of network resources before the resolution succeeds or fails. Therefore, there are optimization mechanisms such as caching, which shorten the resolution time and save resources. The purpose of caching is to temporarily store previously requested data so that future requests for that data can be served faster without contacting all servers in the DNS hierarchy. The recursive resolver stores the cached resource records and these cached entries have a TTL (Time to live) value. This value represents the number of seconds that the resolver should keep the entry in its cache. A lower TTL value results in more frequent resolution lookups and less effective utilization of caches. In the contrary, a higher TTL value causes the resolver to cache the resource record for extended periods, thereby reducing the resolution time for the duration of the TTL. However, in practice, the resolver may alter this TTL value based on its implementation, resulting in the entry being kept for more or less time [9]. Previous research has demonstrated that resolvers generally shorten the TTL time rather than keeping the data beyond the TTL value [10]. The resolver can also manually clear the cache before the TTL value expires for various purposes such as to free up storage space.

When a resolver receives a DNS query, it first checks its cache to see if it has a recent copy of the requested resource record. If the record is found in the cache, the resolver can return the record to the client without needing to send a query to an upstream DNS server. After a cache entry of the resolver expires, it can be refreshed if a new response to the query is received by the resolver. If the resolution process takes too much time, the query times out. A DNS retransmission is the process of sending a DNS query again if no response or an error response is received from the DNS server. If DNS packets are lost during transmission, the receiving system wont receive the necessary information and the sender will need to retransmit the packet. DNS retransmissions are typically triggered by a timeout, which occurs when the DNS server does not respond within a certain time frame. Depending on its implementation, the resolver may retransmit the query multiple times before giving up and returning an error to the user. In cases of high packetloss, the number of retransmissions may increase, resulting in longer response times for the client. For a DNS request, there can be multiple duplicate requests, and therefore multiple responses to these requests. If a resolver has not received an answer to a query, the resolver can repeat this query, resulting in duplicate queries and possible duplicate responses.

DNS packets are usually sent via UDP (User Datagram Protocol), which is a connectionless, unreliable protocol that allows for small packet size but does not provide guarantees for packet delivery or receipt. It is also possible to transmit DNS packets using TCP, which is a more reliable protocol, particularly when the size of a DNS packet exceeds the maximum size for a UDP packet.

1.5 Focus of the Work

In this study, we examine the behavior of resolvers in the event of a partial or complete failure of authoritative DNS servers. Through our experiments, we aim to understand whether the behavior of resolvers follows certain patterns and how the defensive mechanisms of DNS work against DDoS attacks. By analyzing the results of these experiments, we hope to gain a better understanding of the impact of different parameters on communication with the name server and to measure various metrics such as resolution latency, failure rate and stale record rate. This information could be useful in developing strategies to defend against cyber attacks and improve the performance and reliability of DNS.

2 Structure of the Experiments

2.1 Details of the Test Environment

For our experiments we use an authoritative name server at Saarland University, that uses BIND9 (Berkeley Internet Name Domain) version 9.16 as an open-source name server software to be able to interact with DNS. We only use one name server with one IP address without the use of any anycast structure or secondary servers. Our authoritative name server acts as both a server, providing DNS answers to a resolver, and a client, creating the DNS queries. To distinguish between client and server network traffic, we use separate IP addresses for the client and the server. To generate and send DNS requests in our experiments, we use the DNSPython library, Zmap and RIPE Atlas API on the client side. The command-line utility "tcpdump" is used on both our authoritative server and client to capture and display incoming and outgoing network packets transmitted over the network such as DNS queries and responses. To isolate the recorded network traffic for the respective resolver communications and filter out irrelevant packets that do not belong to our experiment, we use Wireshark.

"iptables" is a command-line utility that allows users to modify the behavior of the Linux firewall by configuring the tables and chains of rules maintained by the Linux kernel firewall. Iptables uses a set of tables, which contain chains of rules that match packets and determine what to do with them. With iptables, users can create firewall rules that allow or block traffic based on various parameters such as IP addresses, protocols, and port numbers. To simulate an attack on our authoritative server, iptable rules are used by configuring it to randomly drop incoming UDP and TCP packets on the server at a specified rate. This allows us to study the effects of packetloss, which can occur as part of a DDoS attack, on the performance and behavior of resolvers without causing harm to any real systems. This simulation may not be entirely accurate because it does not take queueing delays into account and is only performed at the final router [10]. However, the impact of this limitation is likely to be minimal due to the dominant influence of packetloss on the overall results [11]. The dropped network packets that were filtered by the iptables rules can still be seen in the network capture logs.

After we have performed the experiments, the filtered capture files are evaluated and the data of interest is extracted and visualized by the use of various plots to gain insights into the resolver behaviors. This helps us to understand how the resolvers respond to different scenarios and identify patterns or trends in their behavior.

Given the low adoption rate of IPv6 in DNS and the server's lack of capability to support IPv6, we used IPv4 for our experiments. This enabled us to focus on a widely used protocol and ensure that our results are relevant to a broad audience. However, an alternative approach could have been to include both IP protocols in our measurements, in order to broaden the scope of our target audience.

2.2 Parameters to be Tested

In our experiments, we want to examine the impacts of a simulated cyber attack's effects against the DNS infrastructure. A high query resolution time means high waiting time or even a query timeout for the client. If the resolution fails, the client will be unable to connect to the requested server. The resolver may compensate for lost network packets during transmission by sending retransmissions to the name server. If an authoritative server fails, the resolver's caching mechanism can resolve client queries for a limited time. Overall, we examine the resolver's performance by measuring the following key metrics:

- DNS resolution latency, which is the time it takes for a client to receive a response from a DNS resolver in response to a DNS query.
- Failure rate of responses to examine the percentage of failed DNS resolutions.
- The amount of DNS retransmissions sent by the resolver, to examine how the packetloss amplifies the DNS traffic by query retransmissions and how the retransmissions help to resolve the query.
- The underlying transport protocol of the DNS packet (either UDP or TCP).
- The rate of stale records to examine whether a resolver is able to answer a client query in case of a name server failure when the cached resource records are expired.

The rate of stale records is only measured in some of our experiments where we examine caching behavior of resolvers in detail.

2.3 Test Sources

An Internet Service Provider's (ISP) DNS resolver is a server that is run by the client's ISP. Public recursive resolvers, which are operated by a third party, are an option for people who don't want to use their ISP's DNS resolver. These resolvers can be used by anyone on the internet. A public recursive DNS resolver is not connected to a specific ISP and can be used by anyone, no matter what ISP they have. One major difference between an ISP's DNS resolver and a public recursive resolver is that public recursive resolvers may offer extra security features, such as protection against phishing.

Given the large number of resolvers with varying configurations, it is not feasible to test all the resolvers worldwide in detail and draw general conclusions about resolver behavior. Therefore, we needed to design our experiments in a way that provides a representative sample of most types of resolvers. This will enable us to draw meaningful insights about resolver behavior while still being feasible to conduct. The types of resolvers we tested in our experiments are as follows:

- Widely used open (public) resolvers
- Resolvers of worldwide uniformly distributed RIPE Atlas probes
- Wild open resolvers found via DNS scans

For our experiments, we first identified widely used public DNS operators. The DNS operators we have chosen are the following, which were obtained from the website publicdns.xyz on December 6, 2022: AdGuard [12], CleanBrowsing [13], Cloudflare [14], Dyn, Google [15], Level3, OpenDNS [16], Yandex [17] and Quad9 [18]. These operators represent a diverse range of DNS services that are commonly used by a large number of users. Many operators have multiple resolver IP addresses that can have different configurations. In our experiments, we examined a total of 9 operators and 22 open resolver IP addresses. We carefully selected the IP addresses to ensure that we had at least one resolver IP address for each operator configuration. This allowed us to test the different configurations of each operator. Our experiments on stale records showed that different configurations of the same operator can behave differently when an authoritative DNS server fails. This highlights the importance of considering the specific configuration of an operator when studying resolver behavior.

RIPE Atlas is a global network of probes that help us measure the internet's connectivity and reachability. The RIPE Atlas probes are small devices that are distributed around the world and connected to the internet through various types of connections. Each probe sends measurements to the RIPE Atlas servers and the collected data is used to provide a detailed view of the internet's performance, including metrics such as latency, packetloss, and throughput. The data collected by RIPE Atlas is used to understand the performance and reliability of the internet. With the use of RIPE Atlas, we are able to examine the performance of the selected probe devices. The selection of the probes was achieved by the usage of Metis Atlas probe selection tool [1].

In order to carry out our experiments with a large number of resolvers, we used Zmap to scan the IPv4 address space and pinpoint all IP addresses that appeared to be functioning as DNS servers, with the use of a denylist for certain network ranges. We were able to identify around 6 million IP addresses which were then narrowed down by applying various filtering criteria for more accurate results.

[1] https://ihr.iijlab.net/ihr/en-us/metis/selection

3 Experiments and Results

3.1 Open Resolver Packetloss Experiment

Since we are unable to perform a real cyber attack on our authoritative name server, we simulate a DDoS attack by using iptable rules to introduce random packetloss for incoming UDP and TCP packets on the server at a specified packetloss rate. For the experiment, we used various rates of packetloss and these packetloss rates are 0% (which represents the network latency without packetloss), 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 85%, 90% and 95%. To ensure that our measurements accurately reflect the full query lookup process without the benefit of the final answer, we used a unique query name for each query sent to the resolver. Without caching, the resolver must contact the DNS hierarchy to obtain the final answer in order to retrieve the IP address associated with a queried name. The use of retransmissions and caching can significantly decrease resolution failures for less than TTL duration of the resource record in case of 90% packetloss [10]. However, in this experiment, the only defense mechanisms against query resolution failures are through the use of DNS retransmission and the resolver's ability to switch to TCP, which allows us to better isolate and examine the effects of these mechanisms.

We conducted the packetloss experiment on September 5, 2022, and repeated the same experiment on December 20, 2022, to verify our results. For each packetloss rate, we sent 49 queries to each resolver. This allows us to collect a sufficient amount of data to draw meaningful conclusions about the behavior of the resolvers at different packetloss rates. In this experiment, we introduced a one second delay after each query sending. After sending 49 queries to a resolver for a packetloss rate, we added a cooldown phase of 10 minutes, where we kept the network capture going to be able to see any delayed DNS packets. After the cooldown phase, we continued the experiment with the next packetloss rate, until all the packetloss rates are processed. For a DNS query, we calculate the latency of the DNS resolution in our experiments by calculating the time difference between the first DNS query and the first DNS response to it in case of duplicate responses.

3.1.1 Open Resolver Packetloss Experiment Results

To visualize the observed resolution latencies for the client, we use a violin plot, which allows us to see the range as well as the density of the latencies. The thicker part of the violin plot represents the denser regions of the data, while the thinner part represents the sparser regions of the data.

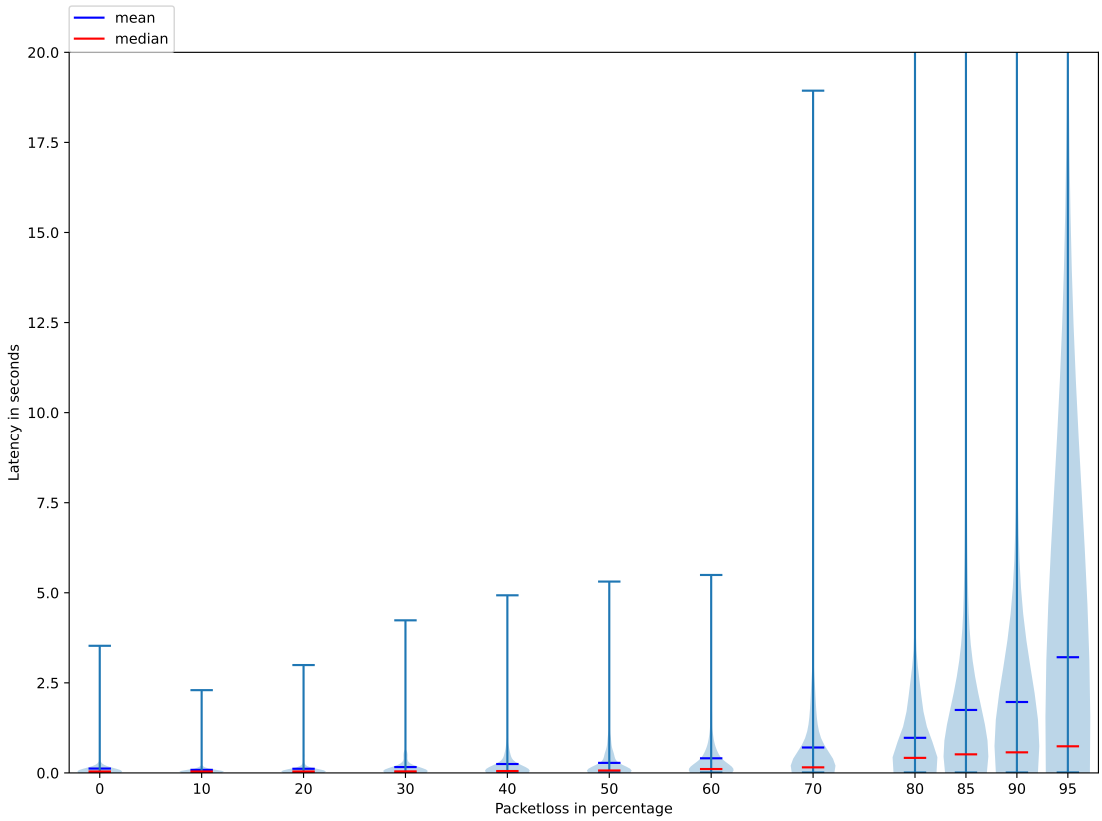

Figure 3.1: Latency plot of RCODE OK responses (Client side)

Without the packetloss simulation, the median latency for successfully resolved queries (indicated by an RCODE of OK) on the client side was approximately 0.03 seconds, whereas the mean latency was 0.1 seconds. The median latency increases by a factor of 1.7 for packetloss rates between 0% and 50%, while the mean latency increases by a factor of 2.2 in figure 3.1. If we compare the latencies at a packetloss rate of 0% to a packetloss rate of 95%, we see that the median latency increases by a factor of 18, while the mean latency increases by a factor of 26. Since the median is more resistant to outliers and the mean is heavily influenced by extreme values, it is generally more reliable to use the median value when performing the latency analysis. In our experiments, we observe many individual outliers, particularly from the Yandex resolver, which significantly influenced the mean values due to their extreme values. There is a significant difference in the amount of latencies observed at various packetloss rates in figure 3.1. For example, we see that there are 1072 latencies observed at a packetloss rate of 0%, and 577 latencies at 85%, while there are only 95 latencies at a packetloss rate of 95% due to the increasing failure rates.

The response latencies for queries that failed to resolve (indicated by an RCODE of SERVFAIL) are represented in a box plot in figure 3.2, which shows how long a resolver attempted to resolve a query before returning an error to the user. The results indicate that as the packetloss rate increases above 40%, some of the resolvers take longer to try to resolve the query. Although some operators such as Google, Cloudflare, and Cleanbrowsing had relatively low SERVFAIL response latencies compared to other operators, the latencies of other operators increased by more than 10 times when the packetloss rate was 95%. The reason for this increase in SERVFAIL latency will be explained later when we present the findings about retransmission amounts.

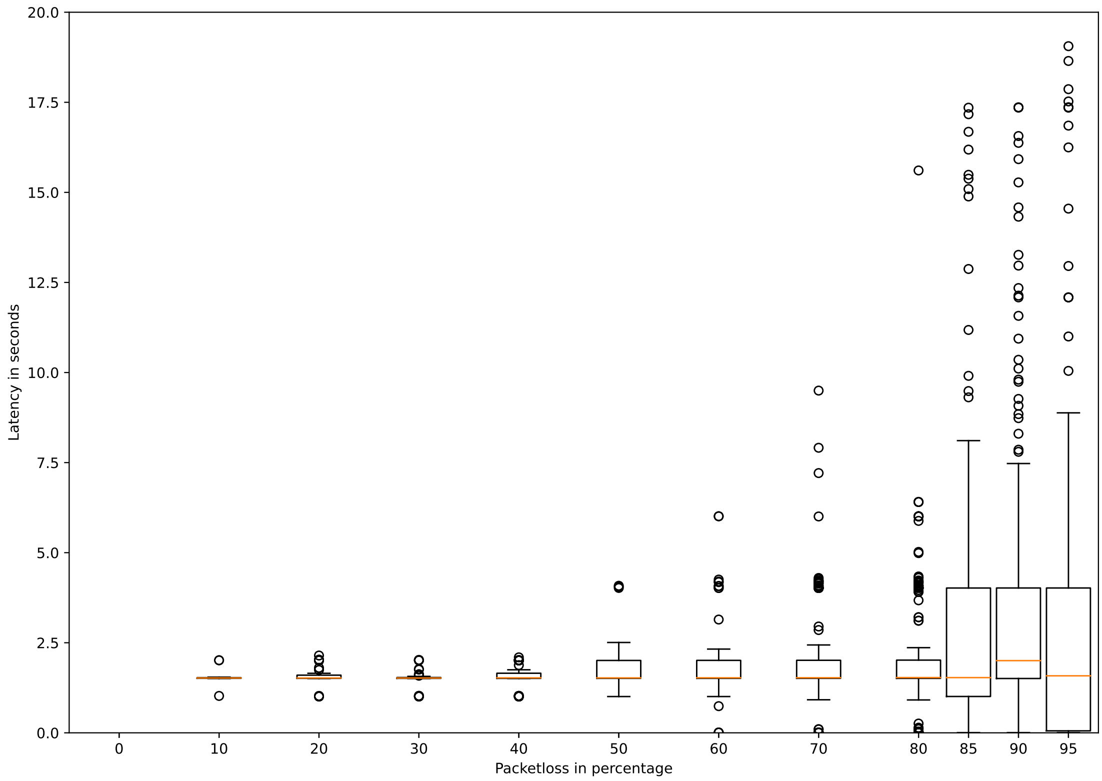

Figure 3.2: Latency plot of RCODE SERVFAIL responses (Client side)

At a packetloss rate of 50%, the failure rate of the queries is 14% and it increases to 90% at a packetloss rate of 95%, which can be seen in figure 3.3.

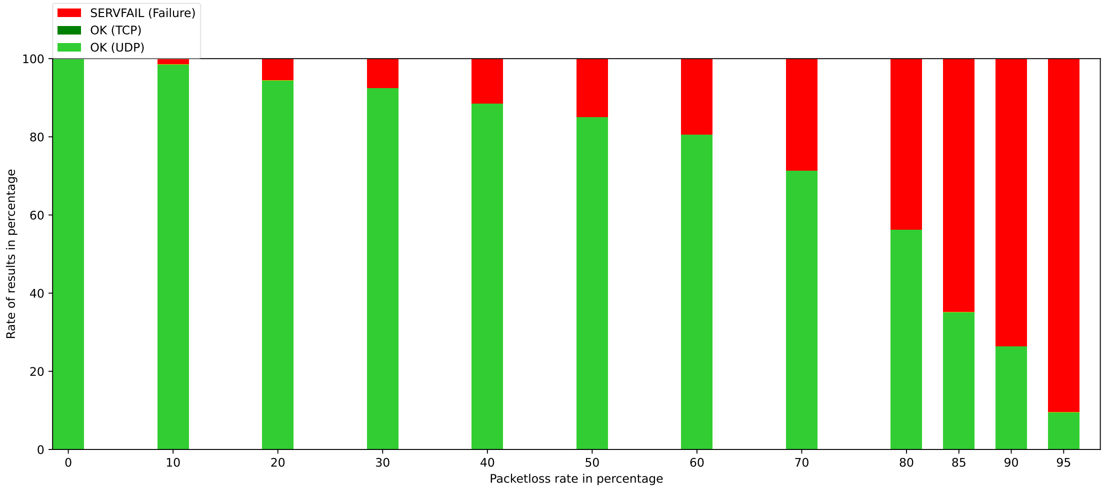

Figure 3.3: Success/Failure rates of all resolvers (Client side)

The number of missing queries on the name server side, which are queries that are not forwarded from the resolver to the name server, increased significantly at a packetloss rate of 70% and higher (3.4). We are able to detect the missing queries by comparing the queries sent from the client with the queries received by the authoritative server. If a non-cached query sent by the client is not received by the authoritative server, it is considered a missing query. Google, Cloudflare, one Cleanbrowsing resolver and two Adguard resolvers were able to redirect all client queries to the name server, while the other operator resolvers failed to send some of the client queries to the server, especially when the packetloss was above 70%. Most missing queries were observed for the Level3, Dyn, and OpenDNS operators.

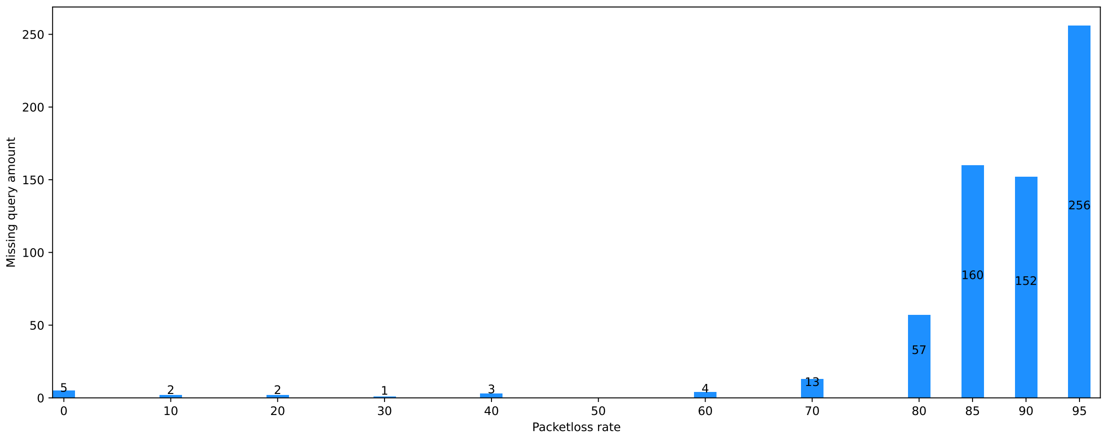

Figure 3.4: Missing query amount for each packetloss rate (Client side)

A DNS resolver that fails to redirect a client query to the authoritative server could be caused by issues such as network connectivity problems, security measures, or resource limitations on the resolver side.

For a single query sent by the client we counted the number of retransmissions received by the authoritative server for that query at each packetloss rate, and used a violin plot to visualize the retransmission amounts. While the mean and median number of retransmissions for a single query increase with a higher packetloss rate, the maximum value tends to decrease when the packetloss rate is above 60%. Prior research has demonstrated that DNS retransmissions during DDoS attacks can cause legitimate traffic to increase by up to 8 times for servers [10]. When we examine the violin plot for query retransmissions in figure 3.5, we observe that a single query can result in up to 18 retransmissions, which is the case for 80% packetloss rate. From a packetloss rate of 80% to 95%, the maximum amount of retransmissions for a single query begins to decrease. It is possible that some resolvers may recognize that there is an issue with the authoritative name server and give up on resolving the query earlier, resulting in a response of SERVFAIL. In our experiment, Quad9 sent the most retransmissions (18) to resolve a single query, while Google, Cloudflare and Cleanbrowsing only sent a maximum of 1 retransmission for each query.

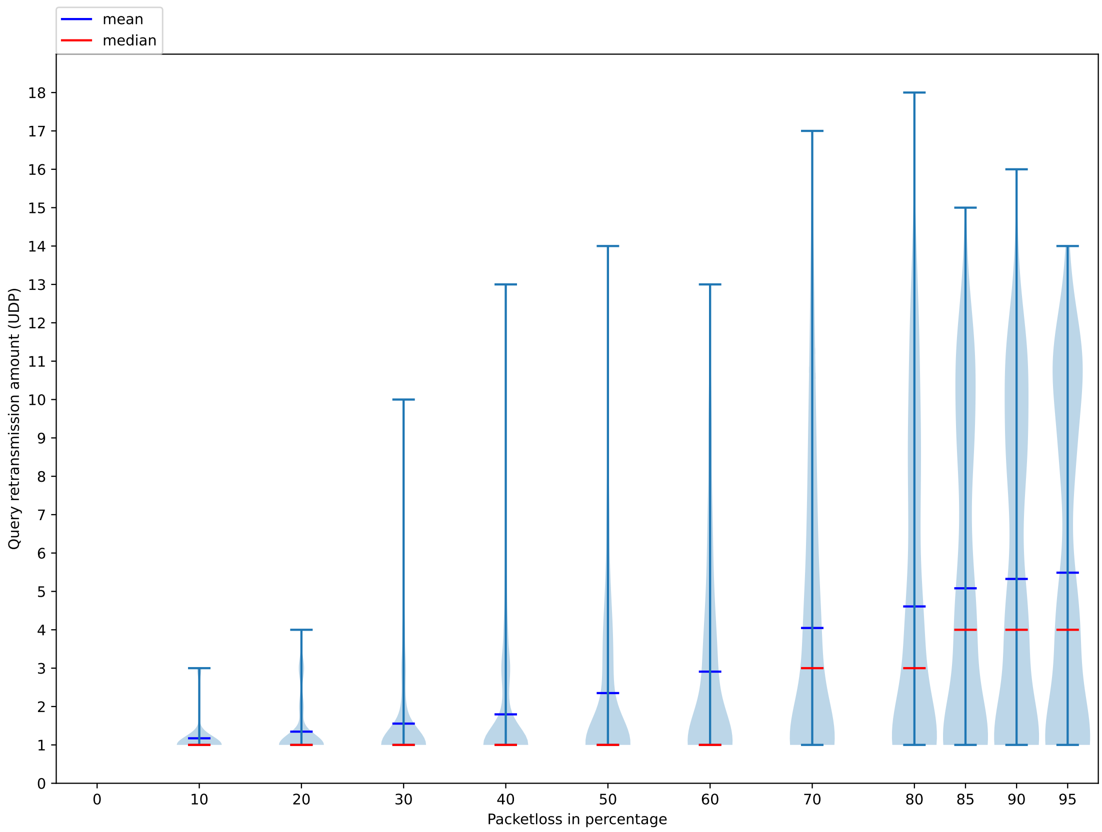

Figure 3.5: UDP query retransmissions observed at authoritative server side

While most of the resolvers sent multiple retransmissions for a single query in case of packetloss, we observed an interesting behaviour exhibited by Cloudflare, Cleanbrowsing and Google. For each packetloss rate, these resolvers sent at most one retransmission for each failed query. While Cloudflare sends a single retransmission for each failed query resolution, Google and Cleanbrowsing do not attempt to resolve every failed query, which lowers their success rates for queries. Because these three operators only perform one retransmission, they give up on the resolution process earlier than other operators, which send multiple retransmissions. This causes their SERVFAIL response latencies to be much lower compared to other operators.

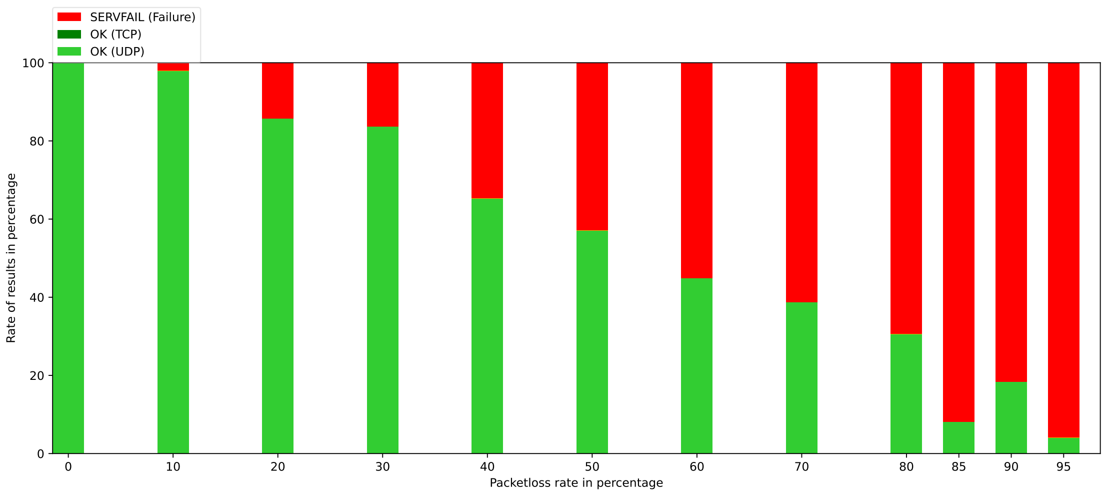

Figure 3.6: Success/Failure rates of Google (Client side)

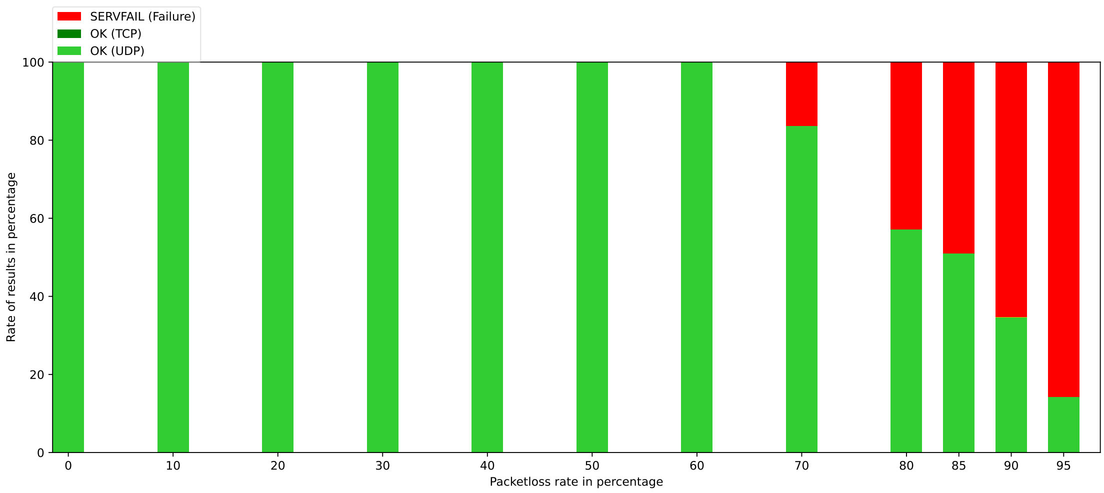

Figure 3.7: Success/Failure rates of Adguard (Client side)

When we compare the failure rates of Google's resolver in figure 3.6, which sends very few retransmissions, to Adguard (Figure 3.7), which sends the most retransmissions in this experiment overall, we can see how the number of retransmissions can increase the success rates. While the Google resolver had a failure rate of over 50% at a packetloss rate of 60%, Adguard was able to resolve all queries with a 100% success rate due to the retransmissions.

Sending a large number of retransmissions can place additional burden on an already overloaded name server, but it also increases the likelihood of a successful query resolution. On the other hand, when a resolver sends very few or no retransmissions in case of failure, this will have limited benefits for query resolution depending on the rate of dropped packets. That's why we believe it is important to find a balance between the success rate and the amplified network traffic, in order to increase the success rate of client queries without adding too much to the network load. Resolvers do not have certain knowledge of whether the authoritative server is under attack, so they can only infer whether there is a problem with the name server by monitoring the success rates and latencies of queries and responses from the name server.

Upon examining the number of unanswered queries on the client side (3.8), which are queries that have no corresponding response from the resolver and result in a timeout on the client side, we noticed that up to a packetloss rate of 60%, there were only a few unanswered queries for each packetloss rate. However, starting at a packetloss rate of 60%, the number of unanswered queries began to increase significantly for the Quad9, and Dyn operators, while the other operator resolvers were responsive to the client most of the time.

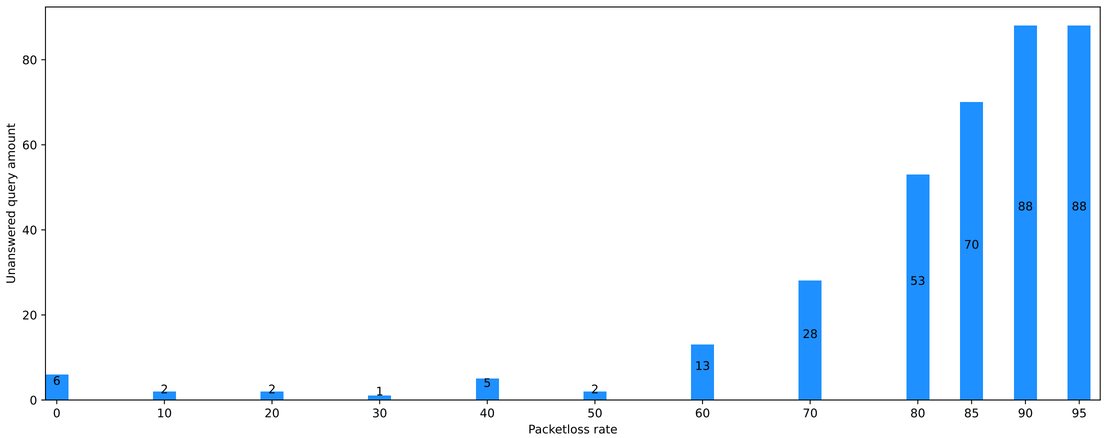

Figure 3.8: Unanswered query amount for each packetloss rate (Client side)

When a client sends a query to a resolver, the client is unaware if the resolver uses retransmissions to resolve the query. Because of these retransmissions sent by the resolver to the name server, the amount of unanswered queries on the authoritative server side is much higher than on the client side. A query resolution that results in an unanswered query counts as a failed resolution, increasing the failure rate.

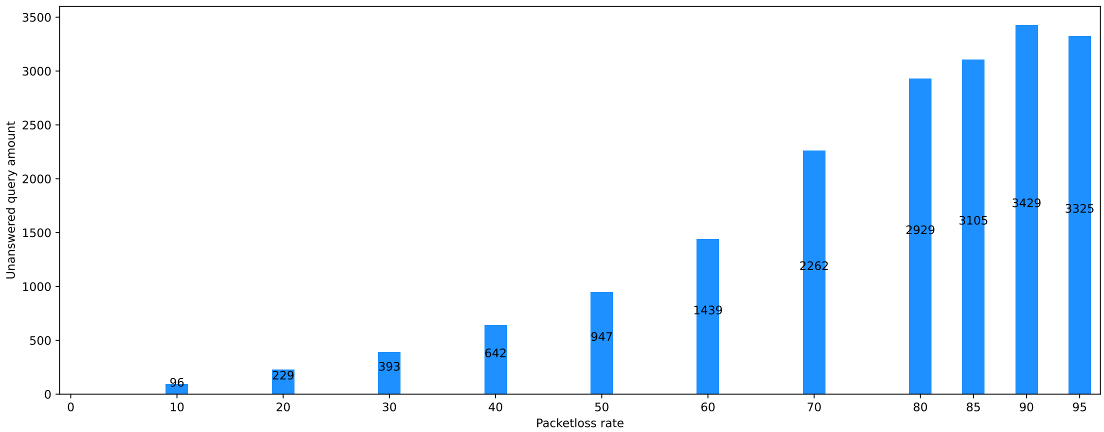

Figure 3.9: Unanswered query amount for each packetloss rate (Authoritative server side)

A DNS resolver that does not respond to a client's query, despite receiving the answer from the authoritative server, can be caused by issues such as high traffic, security filtering, software or hardware failure, or resource exhaustion.

None of the experimented resolvers switched to using TCP on their own and used UDP as the underlying transport protocol for the entire experiment.

3.2 RIPE Atlas Packetloss Experiment

We conducted our packetloss experiment with the RIPE Atlas measurement system using a total of 830 probes that were evenly distributed around the world. The RIPE Atlas probe query timeout value is set to 30 seconds to prevent the report from indicating "no answer" when latency is less than 30 seconds. This ensures that the report reflects the latency of queries, even when they are processed relatively slowly. We conducted the experiment over two days, due to the daily credit limit. On the first day, we tested packetloss rates of 40%, 60%, 70%, 80%, 90%, and 95%. On the second day, we tested packetloss rates of 0%, 10%, 20%, 30%, 50%, and 85%. For each packetloss rate and each selected probe, we sent 20 queries. Similar to the packetloss experiment with open resolvers, we used unique query names for each probe query by prepending the probe ID and the time stamp of the query to each query name to prevent caching. This helps make sure the results show the performance of a full query lookup, rather than being influenced by cached responses.

3.2.1 RIPE Atlas Packetloss Experiment Results

Errors or issues with the configuration of the RIPE Atlas probe device, or policies implemented by the respective resolver, may be responsible for the occurrence of REFUSED answers. These types of issues can cause DNS queries to be rejected or blocked, resulting in the REFUSED response. We excluded DNS responses that returned REFUSED as RCODES from the experiment, which made up only 0.05% of the responses.

In contrast to our experiment with widely used open resolvers, we observed that approximately 10 probe resolvers switched to using TCP to communicate with our server when they were unable to get a response after a certain number of UDP retransmissions. In this experiment, DNS packets that are sent over TCP make up less than 1% of the captured network traffic, which indicates that switching to TCP is not a common strategy among the observed resolvers.

Some resolvers used random capitalization for the query names they sent. This can help protect against Kaminsky attacks because it makes it harder for attackers to guess the query names. It is a simple way to add more randomness to the messages when other parts of the DNS packet cannot be changed. The usage of random capitalization was not observed in the open resolver experiment.

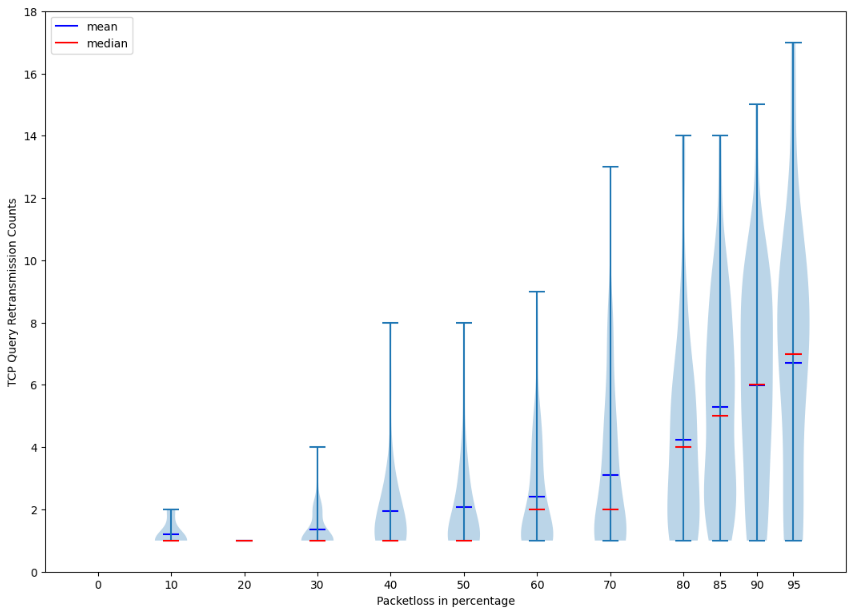

Figure 3.10: Retransmission amounts for a single query (TCP)

Another difference between the probe resolvers and the open resolvers we examined is the number of retransmissions in figure 3.11. In our open resolver packetloss experiment, we only received up to a maximum of 18 retransmissions. At a packetloss rate of 95%, the median value of retransmissions for the RIPE Atlas probe resolvers was 4, while the

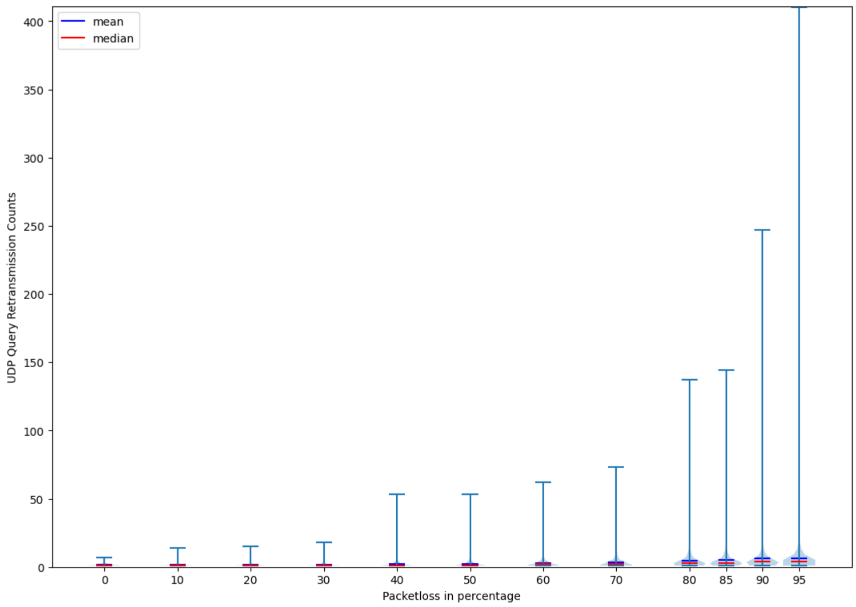

Figure 3.11: Retransmission amounts for a single query (UDP)

mean was 6. However, one RIPE Atlas probe resolver caused up to 395 retransmissions to resolve a single query when the packetloss rate was 95%, using more than 10 different source IP addresses, which significantly adds to network load. Upon closer examination of the capture logs to understand the high number of retransmissions, we found that even though the query was answered by the name server multiple times, the resolver continued to aggressively query our server for 30 seconds, causing 395 retransmissions over multiple IP addresses. It might be possible that the IP addresses of the resolver were unable to properly synchronize with each other, resulting in the resolver continuing to send queries despite having received an answer. At a packetloss rate of 90%, the maximum number of retransmissions was around 250. Only 37 of the 830 probes sent more than 18 retransmissions, which was the maximum number of observed retransmissions in the open resolver experiment.

In figure 3.12, the failure rate of client queries at a packetloss rate of 95% is slightly above 80%, similar to the rates observed in the open resolver experiment although the overall retransmission amounts were not as high as in the public resolver experiment.

When the packetloss simulation was not applied, the median latency for queries that were successfully resolved on the client side was around 0.08 seconds, and the average latency was 0.16 seconds. On average, client response latency increased about 6 times from a packetloss rate of 0% to 50%, and increased 18 times from 0% to 95%. Median latency increased about 8 times from 0% to 50% packetloss rate, and increased 20 times from 0% to 95% packetloss rate. It is recommended to use a maximum query timeout value of 10 seconds when sending a DNS query. With a packetloss rate of 0%, there were almost no responses with a latency greater than 10 seconds. When the packetloss rate was 50%, only 0.01% of the responses were above 10 seconds, and this rate increased to 0.12% when the packetloss rate was 95%

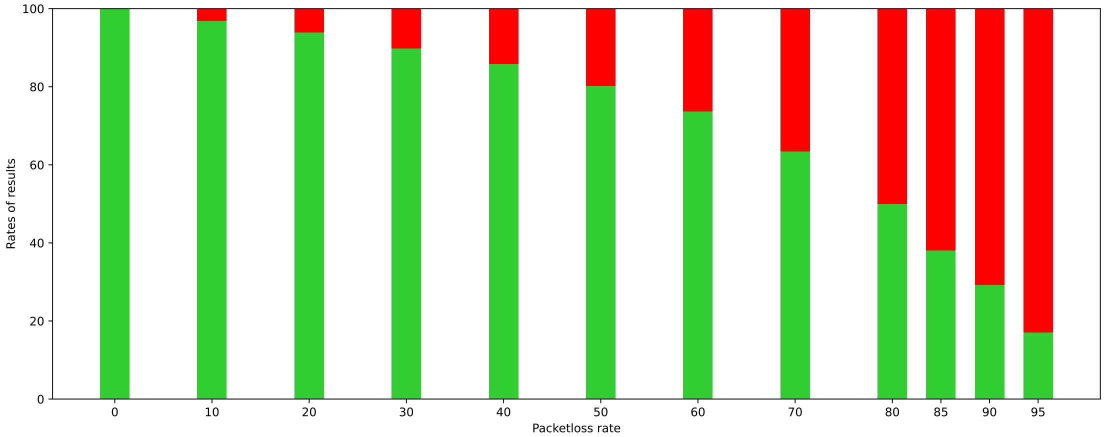

Figure 3.12: Success/Failure rates of RIPE Atlas probes

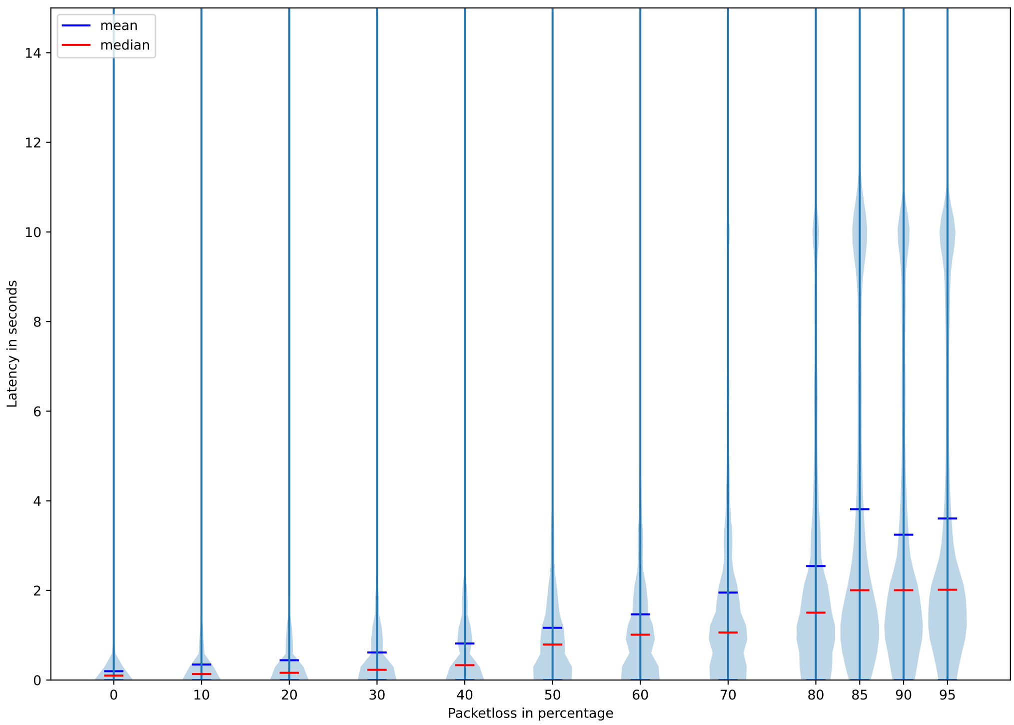

Figure 3.13: Latencies of RIPE Atlas probes (Client side)

3.3 Wild Open Resolver Experiment

We expanded our packetloss experiment to include a larger number of IP addresses by scanning the IPv4 address space and identifying all IP addresses that appeared to be functioning as DNS servers. The scan excluded private IP space and all network which have opted out of being scanned. We then conducted the same packetloss experiment using these resolvers, but modified the procedure so that only one query is sent to each identified resolver IP due to the large amount of IP addresses. This approach allowed us to test a larger number of IP addresses, around 1.9 million in total, but with fewer queries per resolver. It's worth noting that the number and IP address of the identified resolvers may change frequently, so it's possible that some of the resolvers may stop responding during the course of the experiment.

3.3.1 Wild Open Resolver Experiment Results

Since public recursive resolvers do not need any authorization to resolve domain names, these resolvers can be exploited as a way to launch various types of attacks. Recent studies in 2014 have discovered approximately 26 million recursive resolvers. Out of the identified IP addresses, only about 1 million remained stable and continued to provide DNS resolution within a week [19]. Another study in 2018 identified about 3 million recursive resolvers [20] that do not require authorization to resolve domain names, where some of them did not conform the DNS standards described in RFC1034 [7] and RFC1035 [8] and some open resolvers redirect users to malicious destinations reported as malware, phishing, etc. In this experiment, we mainly focused on how latency is affected when packetloss is introduced due to time constraints and the large amount of data we collected.

We identified approximately 6 million IP addresses that seemed to be recursive resolvers. After filtering out those that did not send valid DNS responses with an RCODE value of OK, the number of IP addresses decreased to 1.9 million. We then measured the query resolution latencies of each response for each packetloss rate. Upon analysing the observed latencies, we identified numerous outliers at every packetloss rate.

At a packetloss rate of 0%, the median latency of responses on the client side begins at approximately 0.4 seconds, and the average latency is 0.6 seconds. The median value of the latencies increased by only 1.2 from a packetloss rate of 0% to 50%, and by 3.8 from a packetloss rate of 0% to 95%, while the mean increased by a factor of 2 from a packetloss rate of 0% to 50%, and by 5.5 from a packetloss rate of 0% to 95%. By comparing these results with previous packetloss experiments, we can see that the overall increase in latency in the wild open resolver experiment was much lower, even though we observed many outlier queries that were responded to after up to a maximum of one hour.

In addition to response latencies, we found around 71000 resolver IP addresses that did not send the correct IP address to the client. These incorrect responses may cause clients to be redirected to malicious websites, as concluded by [19, 20].

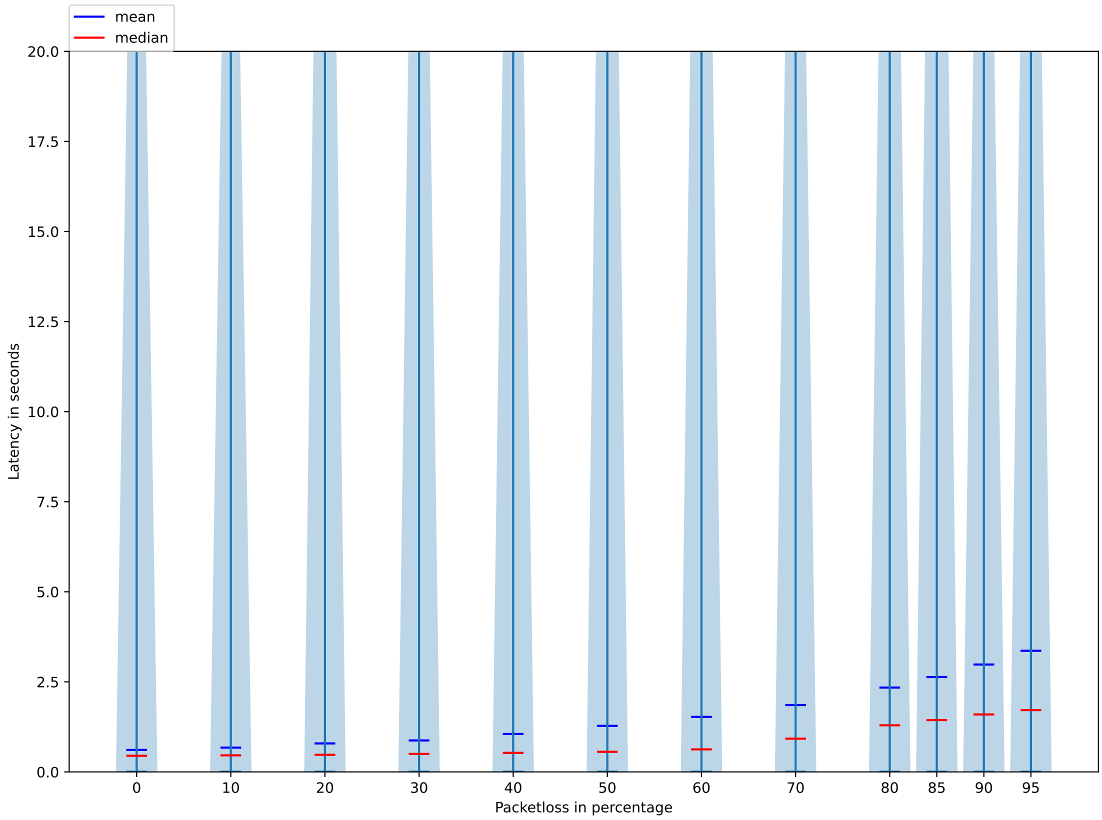

Figure 3.14: Latencies of IP addresses found via DNS Scan (Client side)

3.4 Stale Record Support Experiment

If the recursive resolver detects that the authoritative name server is unreachable, then the resolver may respond the client with expired resource records (stale records) in its cache as proposed in RFC 8767 [21]. We wanted to measure how widespread this strategy is among resolvers. A DNS record stored in a resolver's cache is considered stale when the record's TTL value has expired. Many resolvers clear the expired entries from their caches automatically to free up storage space. The deletion of stale records depends on the implementation of the cache data structure and the configuration of the resolver. The calculation of whether a record is stale in BIND9 only happens dynamically during the lookup. BIND9 does not know which lookups expire when and it uses a hash table to store and manage stale records.

Besides its advantages for static IP addresses or IP addresses that infrequently change, stale DNS records can also cause issues with the resolution of domain names for IP addresses that change often, leading to errors or incorrect information being returned to users. If a DNS record is associated with an IP address that is no longer in use or has been reassigned to a different domain, the resolver may still return the old IP address when a user attempts to access the domain. This can cause users to be directed to a wrong website or to receive an error message when trying to access the domain. Recursive resolvers can be configured to delete cache entries that have been stored for a long time after a certain amount of time has passed. For example, BIND9 removes these entries after one week by default [22].

The goal of our first stale record experiment is to determine if a resolver is able to keep answering the client queries with stale records in case of total unavailability of the authoritative server, even though the resolver's cache entries are expired. This experiment also allows us to see if stale records can help in case the server has an outage.

Public recursive resolvers often use anycast to route traffic to multiple endpoint devices using a single destination address. This helps to distribute network traffic across multiple devices or locations, improving resilience and performance. With anycast, DNS queries are routed to the closest or most available DNS server, which can improve the speed and reliability of DNS resolution for users worldwide. However, the use of anycast can result in isolated caches across multiple servers. Each resolver IP address can have a different number of caches and the resolvers might be deployed behind a load balancer architecture to distribute the network load. The implementation of the load balancer influences the decision, which server should answer the incoming DNS query. With this architecture, each individual server might maintain a cache that is isolated from the other servers. To increase the cache hit rates, some architectures might use a shared cache pool that can be used by multiple servers [23]. Due to the varying number of caches among different recursive resolvers, we have to treat each resolver differently and be almost certain that we have first filled all the caches of a resolver with a resource record to be able to create stale record entries at the resolver side.

We divide our first stale record experiment in two phases, namely the prefetching phase and the stale phase. During the prefetching phase, we send the same DNS query multiple times to the resolvers in order to increase the likelihood that all of the resolver's caches will be hit with this resource record with a chance greater than 95%, assuming the load balancers use a uniformly distributed implementation. We want to hit all the caches of the resolver with this resource record because if a cache is not hit, and we send the same DNS query to the resolver in the stale phase and hit an unfilled cache, the resolver's response will most likely be a resolution error because of the packetloss simulation on the authoritative server side. If we miss a resolver's cache in the prefetching phase, a stale record for that missed cache can't exist, therefore we cannot definitively test if the resolver supports stale records or not. One of the challenges of this phase is the unknown cache amount of the resolvers. Every resolver can have different cache amounts and these cache amounts could be implemented in a way, so that it varies during the day based on the network load of the resolver. To estimate the number of caches for each operator's resolvers, we conducted separate experiments at different times of the day and used the maximum found cache amount from these experiments as the result.

To estimate the number of caches of a resolver, we created a resource record with a TTL value of 600 on our authoritative server and sent this query to the resolver every 2 seconds for a total duration of 600 seconds and observed the received responses' TTL value. When the resolver caches the first response with the TTL value of 600 from our name server and our second query sent to the resolver hits the same cache again, the received response's TTL value should be 598 because 2 seconds have passed since the last query. If we observe a TTL value of 600 for a response after the first query, this may suggest that we hit a new cache. After sending the same query for 600 seconds, we count the number of observed responses with a TTL value of 600, which is likely to be the number of caches of the resolver. However, this method of estimating the number of caches does not work on resolvers that alter the TTL value (which was the case with Yandex resolvers in our experiments) and the resolvers might also refresh the cache during the experiment, which can result in inaccurate results. A resolver may also use a different number of caches depending on its network traffic, which is why we conducted this estimation experiment multiple times throughout the day.

Our results show that Cloudflare and OpenDNS have the highest amount of caches, with 18 each. Some operators such as CleanBrowsing had varying numbers of caches throughout the day. After determining the number of caches for all resolvers, we started our stale record experiment using the identified cache amounts for each resolver. These cache amounts helped us determine how many queries were needed to fill the resolver caches with a 95% probability in the prefetching phase. For example, to fill all of the caches of a resolver with about 18 possible caches with a 95% probability, we would need to send 53 queries assuming a uniform distribution. In order to fill all 18 caches of a resolver with a probability greater than 99.9%, we would need to send more than 120 queries. This would not be practical to do in a short time frame and would not be benign, as it would put a higher load on the resolver.

To determine if a resolver is actually sending an outdated record rather than fetching a new record from our server, we altered the IP addresses of the DNS responses on our server after the prefetching phase, so that we can examine the received response on the client side using this IP address, if the answer of the resolver was stale or not. After populating the caches of all resolver IP addresses during the prefetching phase, we simulated the selected packetloss rates on our authoritative name server. With the simulated packetloss rate, we began the stale phase by sending the same DNS query to the resolvers again. We then examined the DNS responses and extracted the results of this experiment by analyzing the IP addresses of the responses on the client side. In each stale record experiment, we used a randomly generated string with a length of 3 that is included in the query names sent during the experiment. This ensures that subsequent

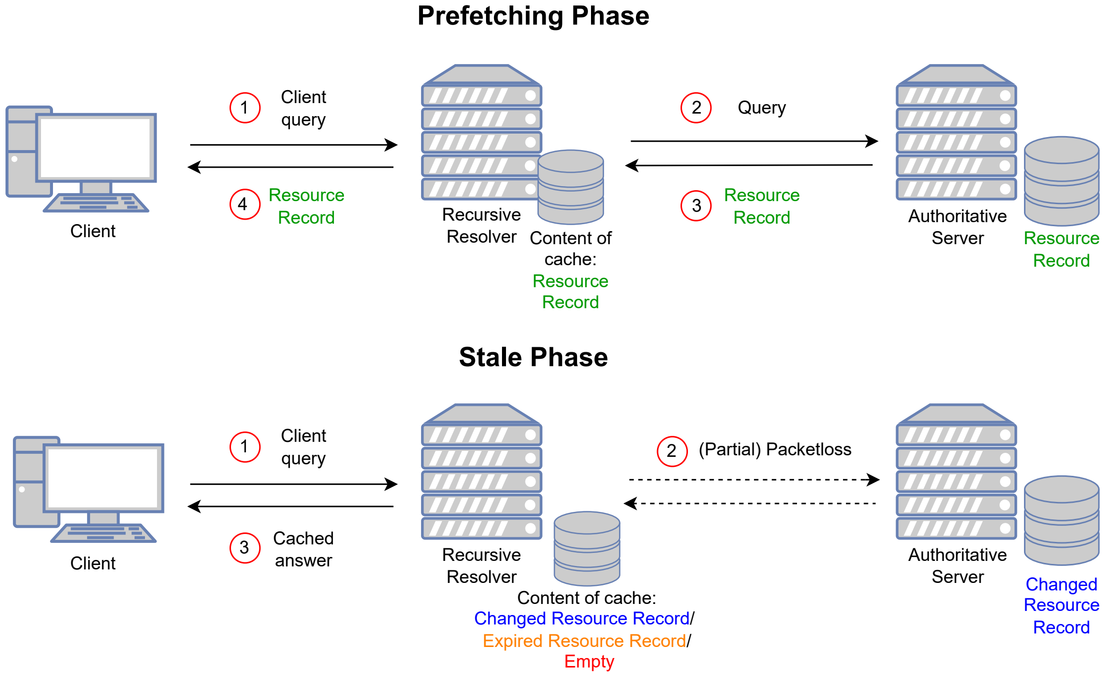

Figure 3.15: Experiment phases (Root name server and the TLD name server are omitted)

experiments do not interfere with each other.

3.4.1 Stale Record Support Experiment Results

The first stale record experiment separated the selected resolvers we were investigating into two groups: those that frequently send stale records and those that do not send stale records. We then compared the results between these two groups to see how the stale records affected DNS resolution.

During our experiments, we only received stale records from 6 of the 22 IP addresses we examined. We noticed that a DNS operator might utilize stale records for some of their IP addresses, while other IP addresses of the operator may not use stale records at all. For example, we selected 3 resolver IP addresses belonging to the operator OpenDNS and found that 2 of them sent stale records, while the other did not. One resolver IP address of OpenDNS, which does not support stale records, does not offer any security filtering. In contrast, resolvers of OpenDNS with stale record support provide security filtering options for the users. Our strategy of selecting one IP address per configuration from each operator proved useful, as different configurations may behave differently in the event of a failure.

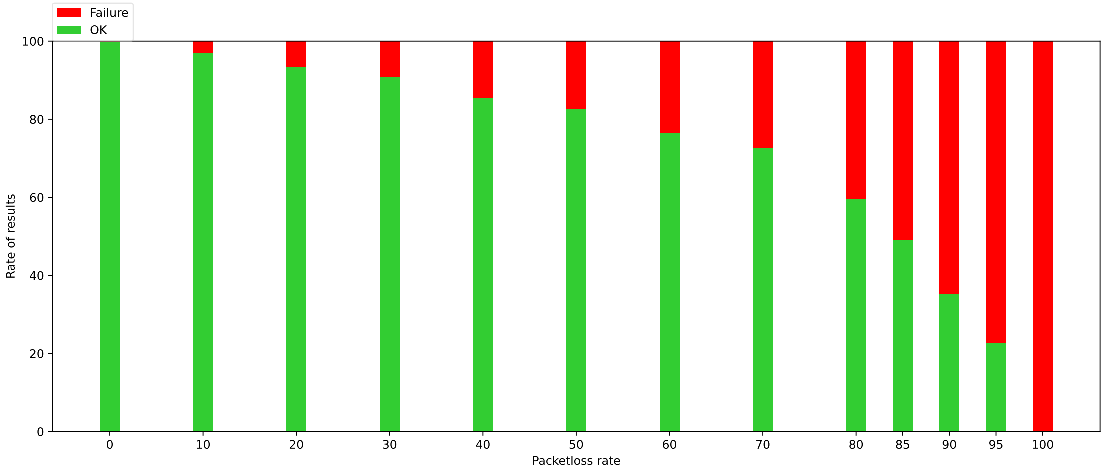

Figure 3.16: Success/Failure rates of resolvers that do not support stale records

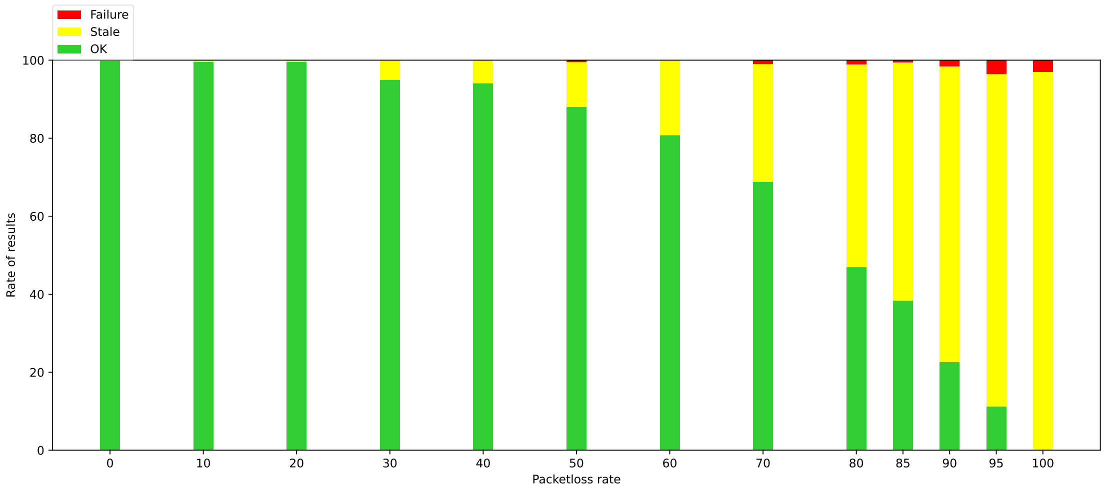

Figure 3.17: Success/Stale/Failure rates of resolvers that support stale records

When we compare the failure rates of these two groups, we clearly see that the query resolution process significantly benefits from the stale records. With stale record supported resolvers, the failure rate was always below 3% in all packetloss rates including complete name server unavailability.

While stale records can be useful for IP Addresses that do not change often, a stale DNS record is no longer accurate or relevant when a domain name is transferred to a new owner, when a website's IP address changes, or when a resource is no longer available. As suggested by [24], we observed that the recursive resolvers did not serve stale records when the authoritative server was able to respond to a query.

3.5 Stale Record TTL Experiment

After the first experiment with stale records, we wanted to determine how long a resolver will send stale records, if it is able to send them at all. By answering this question, we can estimate how long a resolver can provide DNS responses as if there is no failure on the name server. In this second stale record experiment, we also want to examine the impact of the TTL value on the rate of stale records. Since it is not feasible to experiment with all possible TTL configurations, we chose a few TTL values to test, which are 60, 300, 900, and 3600. After the prefetching phase, we simulated 100% packetloss on our name server to determine if the resolvers would continue to answer client queries in the event of total unavailability of the name server. During the stale phase of the experiment, we sent a query every TTL seconds for a total of 6 hours, anticipating that the resolver might refresh the stale record for the next TTL seconds. We selected this query frequency in the stale phase in order to make the potentially refreshed resource record stale again after waiting for the TTL seconds.

3.5.1 Stale Record TTL Experiment Results

The results of the experiment indicate that the TTL value of the records may influence the duration of stale records for some configurations. We expected that a higher TTL value might cause the resolver to continue serving the stale record for a longer period of time. However, we observed that resolvers in our experiment tended to send more stale records for a longer duration with lower TTL values. When the highest TTL value of 3600 was used, we never received a single stale record for 4 of the IP addresses, while OpenDNS rarely sent stale records. The Dyn and OpenDNS resolvers were able to send stale records throughout the entire duration of the experiment when the TTL value was 60, whereas Cloudflare began returning SERVFAIL responses after 85 minutes and stopped serving stale records after 160 minutes in figure 3.18.

The stale records we observed had either a TTL value of 30 or 0. The RFC 8767 recommends that the returned stale records should have a TTL value greater than 0, with a recommended value of 30 seconds due to the historical problems with the TTL value of 0 [21]. Operator Dyn consistently sent stale records with a TTL value of 30, Cloudflare usually sent stale records with a TTL value of 30 and very rarely sent records with a TTL value of 0, and OpenDNS always sent stale records with a TTL value of 0. A TTL value of 0 means that the DNS record should not be cached and must be retrieved from the authoritative server every time it is needed. However, if a record should have already expired but we receive a TTL value of 30, this could also signal that the resolver refreshes this record by itself by adding additional time for this entry to be cached. It is possible that the resolvers check if a stale record is being requested frequently in order to determine whether to retain the stale record in the cache because they are not able to retain that record indefinitely. Stale records that are frequently requested may be retained in the resolver cache for a longer period of time.

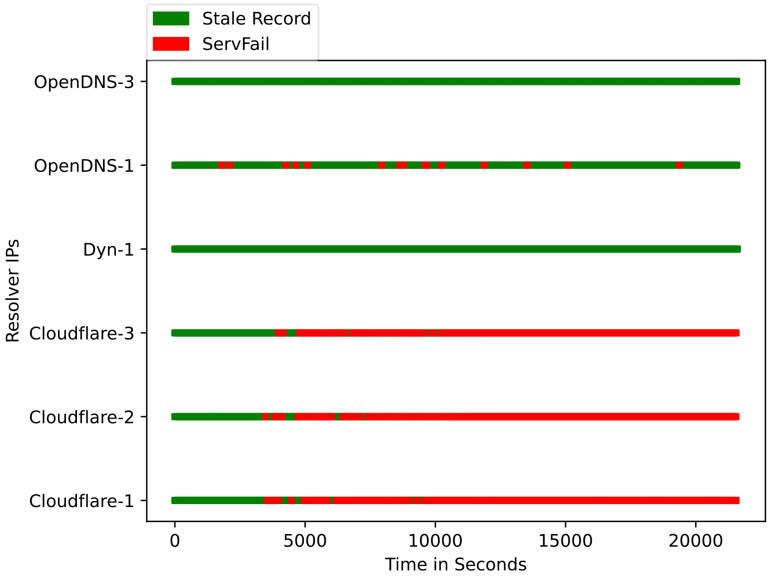

Figure 3.18: Stale record duration experiment (TTL 60, probing interval 60)

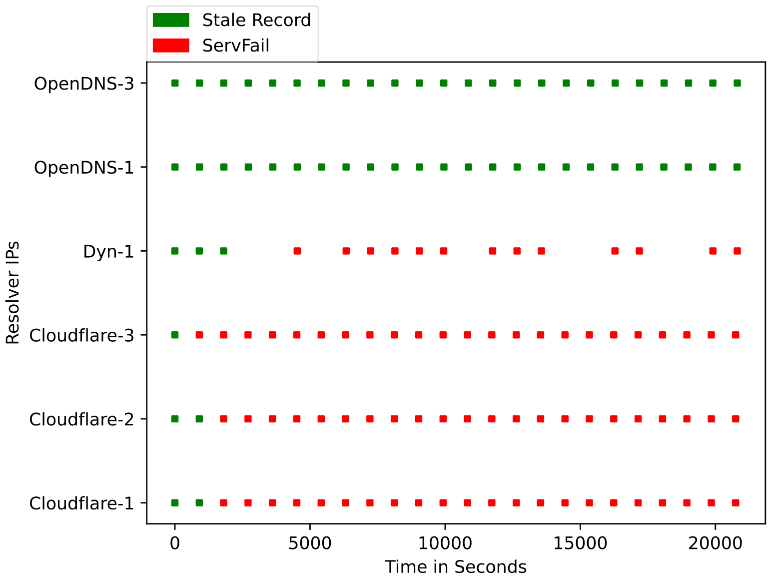

Figure 3.19: Stale record duration experiment (TTL 900, probing interval 900)

The latencies of stale records are very similar to latencies of SERVFAIL responses, which are queries that failed to resolve. The reason for the similarity is that the resolver sends a stale record after attempting to get a response from the authoritative name server and failing to establish contact.

3.6 Stale Record Query Frequency Experiment

To further narrow down the parameters that affects the rate of stale record, we conducted another experiment with a constant query sending frequency and varying TTL values to be able to understand if the querying frequency has an influence on the served stale records. We used the same TTL values in the previous experiment and sent a query to each resolver every 2 minutes in the stale phase instead of waiting for the TTL interval after each query.

3.6.1 Stale Record Query Frequency Experiment Results

We anticipated that when a query is frequently requested, the resolver might take this into consideration by keeping track of the requested stale records and serving stale records more frequently for the requested query. While the results suggest that frequency of queries may influence stale record rates, they do not clearly indicate whether this effect leads to higher or lower rates in general. Comparing the plots of this experiment with the plots of the stale record TTL experiment shows that Cloudflare resolvers only saw a slight improvement in stale records when the query frequency increased. When the query frequency was set to 2 minutes and the TTL value were raised above 300, OpenDNS resolvers began to return much more SERVFAIL responses than stale records. When we compare figure 3.19 to figure 3.21, the Dyn resolver, which had the most consistent rate of stale records overall, began to show an increase in stale record rates as the query frequency increased when the TTL value was set to 900, while the OpenDNS resolvers tended to return more SERVFAIL responses than stale records. As in the previous TTL experiment, the highest TTL value of 3600 caused all resolvers except OpenDNS to consistently return SERVFAIL responses in figure 3.22.

Before serving stale records, Dyn and OpenDNS attempted to obtain a response from the name server by sending a maximum of 9 retransmissions, while Cloudflare only sent 1 retransmission, as shown by the previous packetloss experiment. Dyn typically sends 8 retransmissions, while OpenDNS usually sends 4, occasionally sending 9. OpenDNS resolvers sometimes do not contact our name server to obtain the resource record unlike the other operators, which leads to lower latency values at times. However, when OpenDNS does send retransmissions, its latency is generally around 4 seconds, with a maximum of 6 seconds. Dyn consistently has a latency of 1.8 seconds and Cloudflare always has a latency of 2 seconds, regardless of whether the response is OK or SERVFAIL.

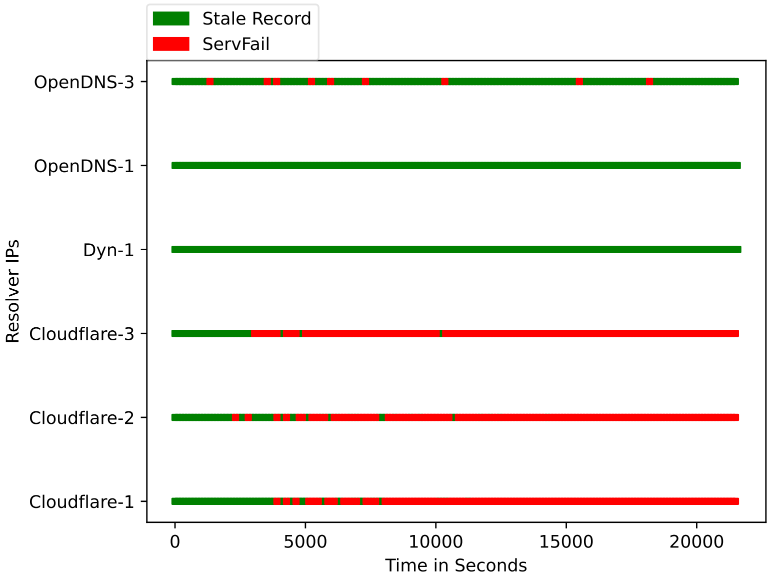

Figure 3.20: Stale record query frequency experiment (TTL 60, probing interval 2 minutes)

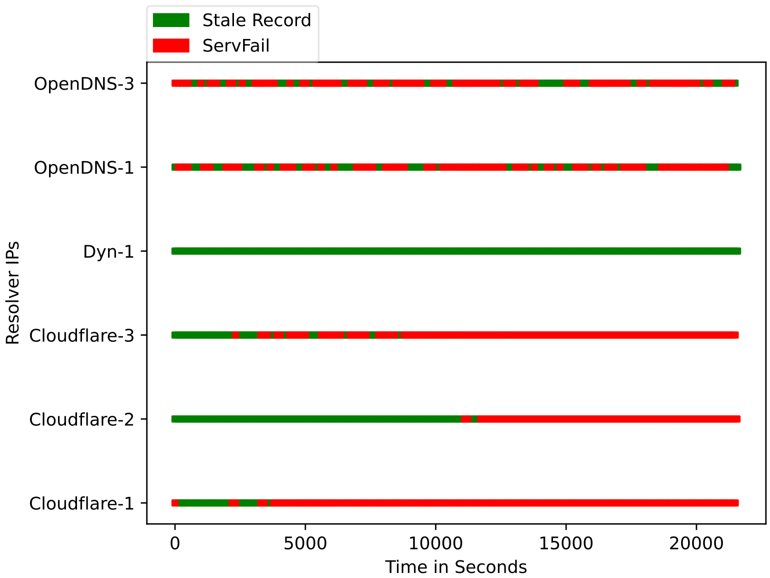

Figure 3.21: Stale record query frequency experiment (TTL 900, probing interval 2 minutes)

We believe that keeping track of the frequency of stale record requests may help reduce failure rates in the event that no response is received from the authoritative server.

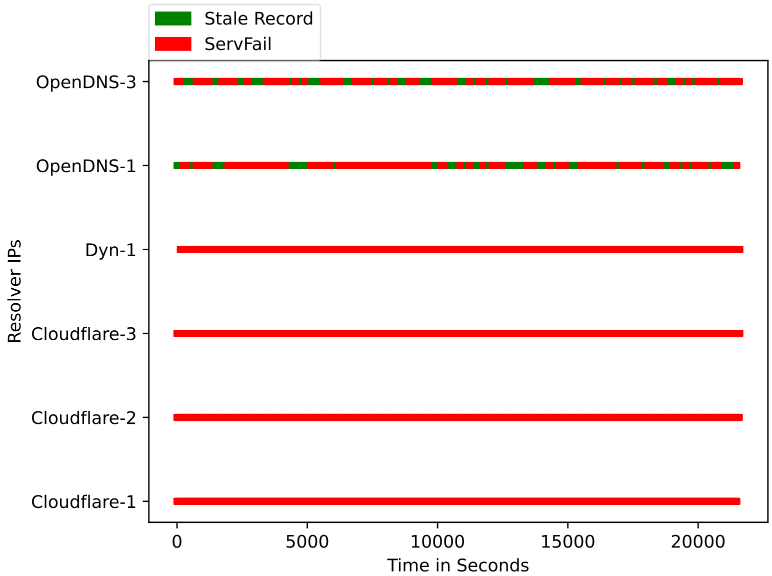

Figure 3.22: Stale record query frequency experiment (TTL 3600, probing interval 2 minutes)

These stale records might refreshed or removed after the authoritative server starts to respond.

3.7 RIPE Atlas Stale Record Experiment

In order to determine the prevalence of stale records among the RIPE Atlas probe resolvers, we conducted the stale record experiment with a TTL value of 60 using 739 RIPE Atlas probes worldwide. It would be impractical and time-consuming to individually estimate the cache capacity for each RIPE Atlas probe resolver. Therefore, we made the assumption that each probe device's resolver had a maximum cache capacity of 18, which was the maximum observed during our previous experiment. We then carried out the prefetching phase accordingly for each probe. The duration of the stale phase of the experiment was set to 3 hours and the probing interval in the stale phase was set to 2 minutes.

3.7.1 RIPE Atlas Stale Record Experiment Results

The experiment results show that none of the 739 RIPE Atlas probe resolvers returned a stale record in the stale phase of the experiment where the authoritative server had 100% packetloss simulation. This means that these resolvers would not be able to respond to client queries in the event of a name server failure if the cache entries of the resolver expire.

3.8 DNS Truncation Experiment

IP spoofing is a type of cyber attack in which an attacker alters the source IP address of a packet to make it appear as if it came from a different device. This is typically done to hide the identity of the attacker, or to launch attacks on networks or individual devices. DNS Cookies provide a lightweight mitigation mechanism against attacks such as reflection attacks and DDoS attacks from spoofed sources [25]. DNS Cookies work by adding an additional layer of protection to the DNS protocol, allowing DNS servers and resolvers to authenticate each other and verify the integrity of DNS responses. In order to benefit from DNS Cookies and decrease their vulnerability to the threats, both the client and server must support it. As shown by [26] the adoption of cookies is limited and higher adoption rates are needed to benefit from them. Another way to protect against IP-Spoofing attacks is by using TCP as the underlying transport protocol. With TCP, two devices can establish a reliable connection which uses sequence numbers to ensure the correct delivery of data. To initiate the connection and synchronize the sequence numbers of the sender and receiver, a 3-way handshake is used between the communication partners. Due to the random generation of the initial sequence number, it becomes difficult for an attacker to intercept the TCP session by guessing the sequence number on the client, which makes TCP much more secure than UDP. Therefore, manipulating the source IP address of a packet via TCP on an existing connection is more challenging than manipulating the source IP address a UDP packet. UDP has no sequence numbers, therefore it is possible to insert manipulated data that be treated as if they were coming from the real host. DNS queries are usually sent via the UDP protocol and as a result, it is possible to send a forged DNS query with a spoofed IP address to a resolver.

In the truncation experiment, we wanted to see the effects of a name server switching to a policy where it only accepts DNS packets transmitted via TCP, in order to filter out spoofed network traffic. For this purpose, we modified our iptables rule to only simulate packetloss on UDP packets and not TCP. We used a response policy zone in BIND9 on our authoritative server to be able to answer a DNS query sent via UDP with a DNS response packet, where the truncation bit of the packet is set and there is no answer section in the response. TCP Packetloss may cause issues with 3-way handshake when the packetloss rate is very high, therefore we only simulated packetloss on UDP packets, since spoofed packets are very unlikely to be sent over TCP.

The truncation bit is a flag in the DNS header that indicates whether the response to a DNS query has been truncated due to size constraints. It is used to allow DNS clients and servers to handle responses that are too large to fit in a single DNS message. If a DNS response is larger than the maximum size of a DNS message, the server can set the truncation bit in the response to indicate that the response has been truncated and that the client should send a new query using a different mechanism, such as TCP, to retrieve the full response.

A response policy zone (RPZ) in BIND9 is a feature that allows the DNS server to modify its responses to DNS queries based on a set of rules. The RPZ is a special type of zone that contains the rules for modifying DNS responses. These rules can be used to block access to certain domains, redirect traffic to different domains, or provide alternate responses for specific queries. Once the RPZ zone is configured, the DNS server will apply the rules in the RPZ zone to all DNS queries that it receives. If a query matches one of the rules in the RPZ zone, the DNS server will modify its response accordingly. Using an RPZ can potentially affect the latency of DNS queries, as the DNS server must perform additional processing when handling a query for a domain that has an associated RPZ policy. This additional processing may slightly increase the time it takes for the DNS server to resolve the query and return a response to the client.

3.8.1 DNS Truncation Experiment Results

Our previous packetloss experiment with the resolvers of DNS operators showed that none of the resolvers we selected switched their communication channel from UDP to TCP by themselves. All public resolvers in this experiment were able to switch to TCP and send queries and responses via TCP due to the implemented authoritative server policy. The success rate of query resolutions of the truncation experiment show close similarity to the packetloss experiment.

The main difference between the open resolver packetloss experiment and the truncation experiment is that when a resolver was able to switch to TCP, it always received a response from the authoritative server, due to the reliability of TCP. After switching to TCP, the maximum amount of observed TCP retransmissions was 4. If a packet is not delivered during transmission using UDP, the sender will not be aware of the failure and will not attempt to send the packet again. In contrast, TCP is designed to handle packetloss and will detect when a packet has not been received by the intended recipient. It will then retransmit the lost packet to ensure that all the data is successfully delivered. When a resolver is able to use TCP, it will take on the responsibility of retrying the transmission of the query until it is acknowledged by the name server. Using TCP is similar to sending as many retransmissions as needed to make sure the response is received from the server, but this also has the downside of increasing network traffic by resending TCP packets until the receiver acknowledges the received packets. If the packetloss is temporary and the connection is able to recover, then TCP will retransmit the lost packets and the communication will continue. However, if the packetloss is persistent and the connection is unable to recover, then the TCP connection will be terminated after a certain amount of time.

Since the TCP communication only happens between the resolver and the authoritative name server, the client does not know whether the resolver gets the resource record from the name server via TCP.

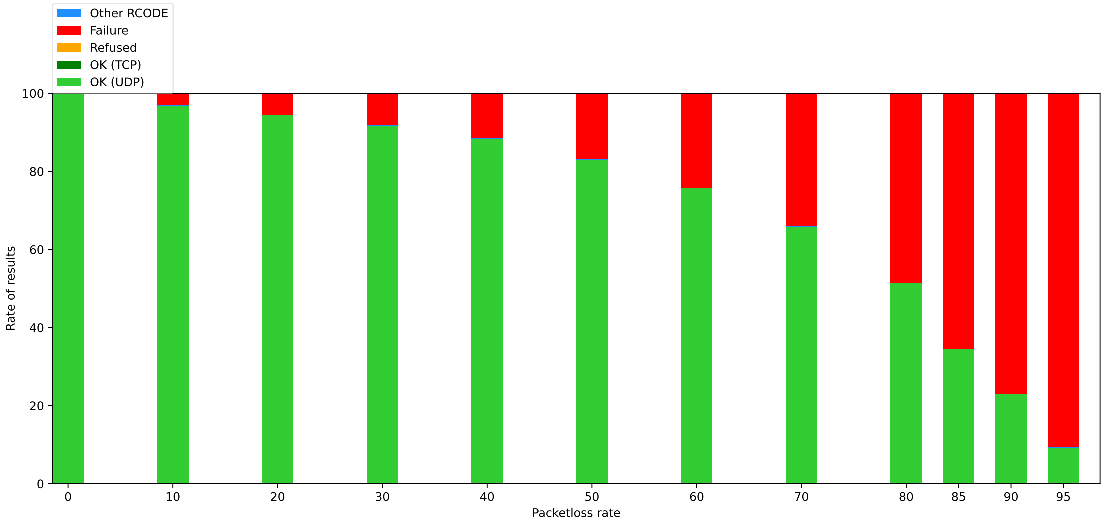

Figure 3.23: Success/Failure rates of truncation experiment

In figure 3.25, when we observe the resolution latencies from the client's perspective, we can see a general increase in resolution time. While the resolvers were able to respond to the name server's TCP request swiftly and received an answer, some operator resolvers took longer to return the retrieved answers to the client compared to the packetloss experiment latencies in figure 3.26. The increase in resolution time was mostly visible for the operator Quad9, resulting in almost double the latency for the resolver.

This suggests that while switching to TCP may be advantageous for the authoritative server itself, it can also lead to an increase in resolution time for certain resolvers.

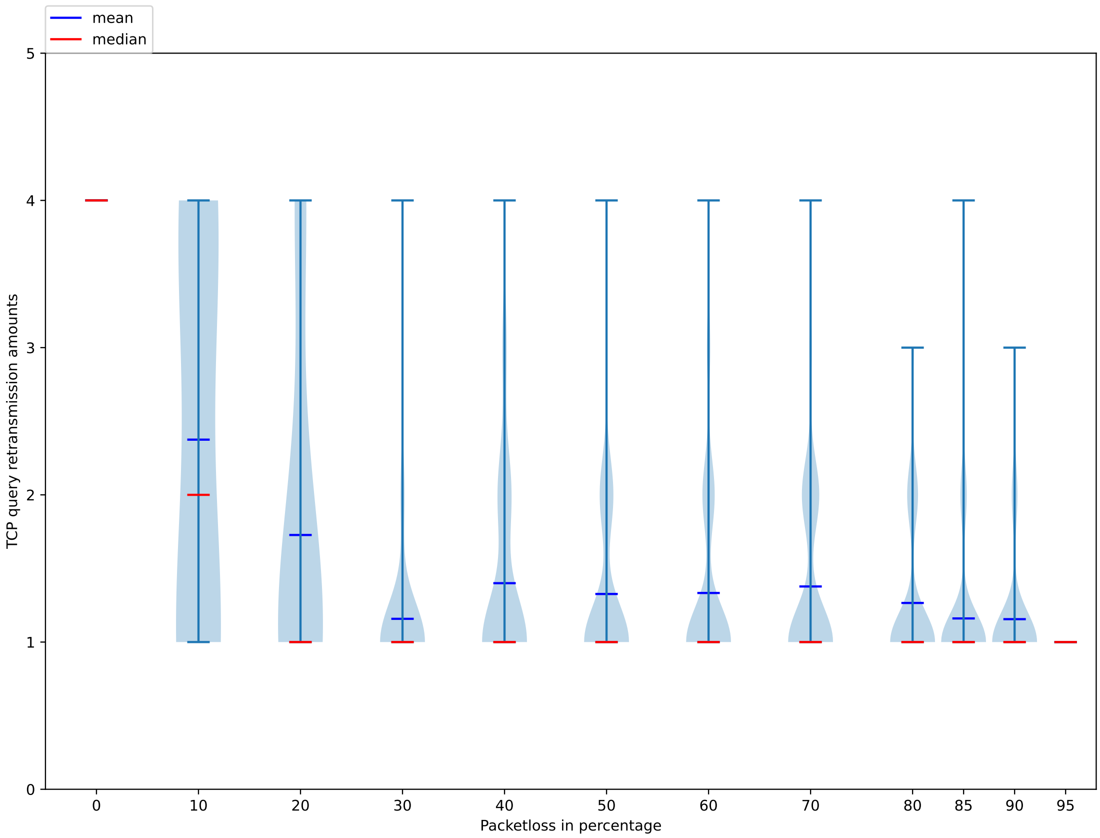

Figure 3.24: Retransmission amounts for a single query (TCP)

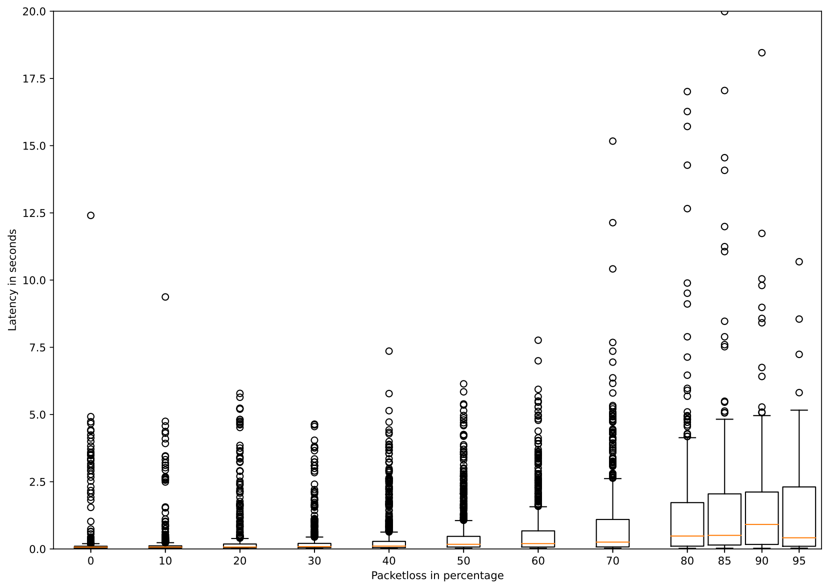

Figure 3.25: Latencies of truncation experiment (Client side)

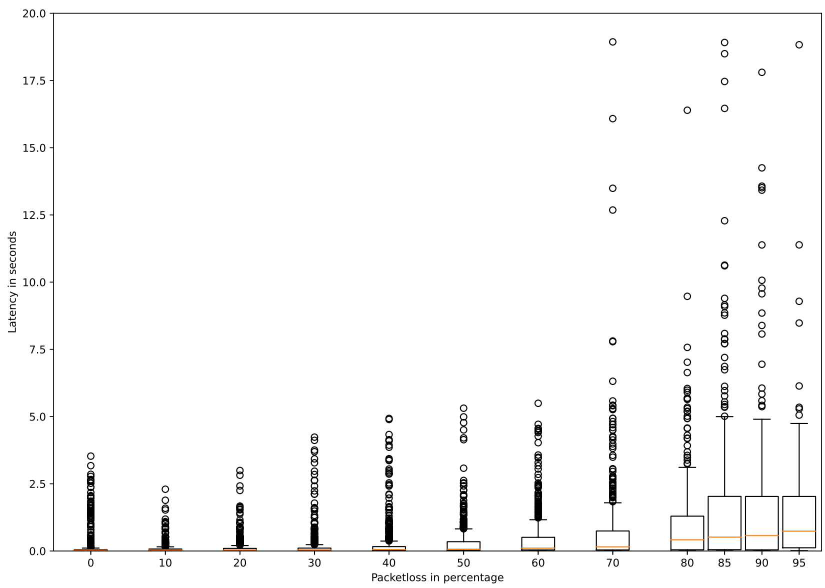

Figure 3.26: Latencies of packetloss experiment (Client side)

4 Summary of the Experiments

To summarize the main findings and implications of our research: If a resolver's caches are full and it supports stale records, these stale records can significantly reduce the failure rate when the authoritative server is unreachable. The serving stale strategy is beneficial for IP addresses that do not change frequently, but it does not help IP addresses that change often. The strategy of serving stale records is not widely supported among the resolvers we tested.

Our experiments did not provide clear evidence on how the rate of stale records is affected by parameters such as TTL value and query sending frequency, as various resolver configurations behaved differently from each other. In some cases, higher TTL values resulted in a higher frequency of stale records being served, while in others, increasing the TTL value led to fewer stale records being sent. Notably, when using a TTL value of 3600, nearly all of the resolvers consistently returned SERVFAIL responses in the stale record experiments.

DNS retransmissions can help resolve queries, but they also amplify network traffic and increase resolution latency in the event of packetloss. Resolvers that made the most use of DNS retransmissions were able to successfully resolve all client queries at a packetloss rate of up to 60%. Both the number and frequency of retransmissions affect the increase in response latency. Our experiment's results showed that a packetloss rate of 95% can cause an increase in latency of more than 18 times, and in rare cases, a single query can amplify network traffic by around 400 times.

Using TCP as the underlying protocol prevents IP spoofing. Switching the transport protocol from UDP to TCP can be useful when a name server detects a high volume of DNS traffic, as it can help filter out spoofed IP addresses. Switching to TCP also ensures that the resolver receives a response from the name server. However, the clients may experience increased latency as a drawback.

5 Discussion

In our experiments, we used only one authoritative name server, but it is common to use multiple authoritative name servers in practice. Doing so can improve the reliability and performance of the DNS system. If one server goes down, the other servers can still respond to DNS queries. Multiple servers can also help to improve performance by allowing more queries to be handled concurrently, which reduces response time. According to RFC 1034 [7], every zone must be available on at least two authoritative nameservers.

As described in section 2.1, the packetloss simulation on the authoritative server by the usage of iptables is not entirely accurate [10]. In addition to the network load, a real DDoS attack can also cause delays in queueing and impact the CPU of the victim server. If the attack generates sufficient traffic, it can use up all of the server's available CPU resources.

The test sources we used in our experiments were mainly PCs and not other devices such as mobile devices. The Metis selection tool was used to choose one RIPE Atlas probe per Autonomous System Number. We did not have full control over the RIPE Atlas probe devices as they can differ from one probe to another and we cannot classify the selected probes as a typical Internet client. The experiments in this paper can be extended by adding more operators and more resolver IPs. While our study includes a variety of recursive resolvers, we do not claim that our sample of recursive resolvers represents a generalization of all resolvers.

If an open resolver is targeted by a cyber attack, it can redirect the attack to any server on the internet because anyone can use open resolvers without authorization. This illustrates the importance of configuring open resolvers in a way that they do not amplify network traffic and cause harm to the victim. In this paper, we focused on the perspective of the client and the authoritative server and inferred the behavior of the resolver from our findings. Since we do not own a resolver, we are unable to measure the internal workings of a resolver in detail. Further research could focus on a case where the resolver is under cyber attack, rather than the authoritative name server. Additionally, these experiments could also be expanded by taking DNSSEC into account, but due to time constraints, we excluded DNSSEC in our experiments.

While the aforementioned limitations may affect the results, we believe that our findings are still relevant for environments that do not have these limitations, and these limitations do not cause the results to deviate greatly from reality.

6 Related Works

Giovane et al. [10] has demonstrated the importance of caches and retransmissions in the event of a DDoS attack and suggests that using longer TTL values may help defend against DDoS attacks. However, our findings indicate a negative impact on stale record rates when using very high TTL values.

To protect against IP spoofing, DNS cookies can be a lightweight solution, Jacob et al. [26] demonstrated that the adoption rate of the cookies not high enough and the enforcing cookies may cause configuration issues. In our experiments we examined an alternative solution to DNS cookies which is the strategy of switching to TCP. However, we found that this strategy was not as prevalent among the resolvers we examined, and it may even be less popular than using DNS cookies. DNS cookies have the advantage of utilizing UDP packets, which generally have a smaller size than TCP packets.

As our experiments only simulate packetloss on the authoritative name server, it is not possible for us to evaluate the impact of DDoS attacks on infrastructure beyond the name server. An investigation of the inner workings of a DDoS scrubber and analysis of various DDoS attacks revealed that such attacks may cause collateral damage to other services and affect many more victims than the targeted addresses [2]. The study also discovered that DDoS attacks leave distinctive marks on DNS traffic, which can be used to detect them early on.

Measures taken to defend against DDoS attacks may also affect legitimate traffic if the attack traffic is indistinguishable from regular traffic. A case study demonstrated that even with traffic scrubbing, DDoS attacks can disrupt resolution for hundreds of thousands of domains [27]. The authors of the paper combined two existing data sets to study recent DDoS attacks against authoritative DNS infrastructure and found that a DDoS attack can cause DNS resolution time to increase 100 times, or make it completely unreachable, which highlights the effectiveness of using anycast in nameserver infrastructure to make it resilient against DDoS attacks.

Recent studies identified millions of open resolver IP addresses by scanning the IPv4 address space and found that some of these resolvers do not follow DNS standards and even exhibit malicious behavior, such as redirecting users to malicious websites [19, 28]. We scanned the IPv4 space to identify open resolvers and analysed the impact of packetloss on resolution latency using the identified resolvers. Our results confirm that there are many open resolvers that do not provide accurate responses to clients.

7 Ethical Considerations

All the experiments we introduced in this paper are performed in a benign manner so that our experiments does not cause unusual amount of network traffic on the resolver side. It is generally not appropriate to send a large number of queries to a DNS resolver in a short period of time, as this can put a strain on the resolver and potentially impact its performance for other users. There is no specific threshold for how many queries is considered excessive, as it can vary depending on the capabilities of the DNS resolver and the overall level of activity on the network. The highest frequency that the queries are sent to a resolver was the prefetching phases of the stale records experiments. In the prefetching phase, we have sent two queries every second to a resolver with a high cache count for about 30 seconds to be able to hit all the caches of the resolver in a short time.

During the IPv4 address space scan in the wild open resolver experiment, we excluded private IP addresses from the scan. The host that conducted the scan has a reverse DNS record associated with its IP address, making it possible to contact us and be added to the denylist if an IP address wishes to not be included in future scans.

The experiments use zones created specifically for testing, which have no effect on other users and run on a system that is only used for the experiments.

8 Conclusion

DNS packets are typically transmitted via UDP, and DNS retransmissions play a crutial role in addressing the unreliability of UDP and compensating packetloss. We believe it is important for resolvers to find a balance between the amplified network traffic caused by retransmissions and the success rate of name resolutions for the client.

Serving stale records is preferable to serving nothing, as resolvers attempt to reach the authoritative server before serving a stale record. Therefore, we believe that utilizing a strategy of serving stale records can be beneficial in the event of an authoritative server failure. As mentioned in RFC 8767 [24], we also suggest that monitoring the frequency of stale record requests (particularly records with higher TTL values) may help reduce failure rates when no response is received from the authoritative server. These stale records may be refreshed or removed once the authoritative server starts responding again.

There are general countermeasures against DDoS attacks that prevent packetloss, such as scrubbing services, which work by identifying and filtering out malicious traffic, while allowing legitimate traffic to pass through, and CDNs, which are servers that are spread out across different locations and are designed to deliver content more efficiently to users. However, CDNs decrease the effectiveness of caching by typically configuring lower TTL values to support DNS-based load balancing [27]. DDoS attacks usually affect network bandwidth, but they can also overload the CPU of the victim server, causing it to be unable to handle its usual tasks, such as processing incoming packets in time. To address this issue, one can increase both the bandwidth and CPU power of the server. CDNs and scrubbing services achieve this by scaling horizontally with multiple distributed machines, where each server handles a portion of the total traffic and applies filters as needed.

When a name server implements a policy where it only accepts DNS packets sent over TCP, it can filter out spoofed DNS packets to mitigate the effects of an attack. However, this may also increase overhead and latency, as TCP requires a 3-way handshake and more bytes to be sent over the network.

Bibliography

[1] Suranjith Ariyapperuma and Chris J. Mitchell. Security vulnerabilities in dns and dnssec. 2007.

[2] Giovane C. M. Moura, Cristian Hesselman, Gerald Schaapman, Nick Boerman, and Octavia de Weerdt. Into the ddos maelstrom: a longitudinal study of a scrubbing service. 2020.

[3] Paul Vixie, Gerry Sneeringer, and Mark Schleifer. Events of 21-oct-2002. *http://c.root-servers.org/october21.txt*, 2002.

[4] Giovane C. M. Moura, Ricardo de O. Schmidt, John Heidemann, Wouter B. de Vries, Moritz Müller, Lan Wei, , and Christian Hesselman. Anycast vs. ddos: Evaluating the november 2015 root dns event.

[5] Root Server Operators. Events of 2015-11-30. *https://root-servers.org/media/news/events-of-20151130.txt*, 2015.

[6] Ojo Ademola. Dyn ddos cyberattack: A position paper. 2021.

[7] P. Mockapetris. Rfc 1034. *https://www.rfc-editor.org/rfc/rfc1034*, 1987.

[8] P. Mockapetris. Rfc 1035. *https://www.rfc-editor.org/rfc/rfc1035*, 1987.

[9] Unbound documentation. *https://unbound.docs.nlnetlabs.nl/en/latest/*, 2021.

[10] Giovane C. M. Moura, John Heidemann, Moritz Müller, Ricardo de O. Schmidt, and Marco Davids. When the dike breaks: Dissecting dns defenses during ddos. 2018.

[11] Giovane C. M. Moura, Ricardo de O. Schmidt, John Heidemann, Wouter B. de Vries, Moritz Müller, Lan Wei, and Christian Hesselman. Anycast vs. ddos: Evaluating the november 2015 root dns event.

[12] Adguard. *https://adguard-dns.io/en/public-dns.html*.

[13] Cleanbrowsing. *https://cleanbrowsing.org/*.

[14] Cloudflare. *https://1.1.1.1/*.

[15] Public dns. *https://developers.google.com/speed/public-dns/*.

[16] Opendns. *https://www.opendns.com*.

[17] Yandex. *https://dns.yandex.com/.*

[18] Quad9. *https: //quad9.net.*

[19] Marc Kührer, Thomas Hupperich, Jonas Bushart, Christian Rossow, and Thorsten Holz. Going wild: Large-scale classification of open dns resolvers. 2015.

[20] Jeman Park, Aminollah Khormali, Manar Mohaisen, and Aziz Mohaisen. Behavioral analysis of open dns resolvers.

[21] David C Lawrence, Warren "Ace" Kumari, and Puneet Sood. Serving stale data to improve dns resiliency.

[22] ISC BIND. Bind 9 configuration reference. *https://ftp.isc.org/isc/bind9/9.16.7/doc/arm/.*

[23] Google. Public dns. *https://developers.google.com/speed/public-dns/docs/performance.*

[24] D. Lawrence and W. Kumari. Serving stale data to improve dns resiliency-02.

[25] Internet Engineering Task Force. Domain name system (dns) cookies, rfc 7873. *https://www.rfc-editor.org/rfc/rfc7873*, 2016.

[26] Jacob Davis and Casey Deccio. A peek into the dns cookie jar, an analysis of dns cookie use. 2021.

[27] Raffaele Sommese, KC Claffy, Roland van Rijswijk-Deij, Arnab Chattopadhyay, Alberto Dainotti, Anna Sperotto, and Mattijs Jonker. Investigating the impact of ddos attacks on dns infrastructure. 2022.

[28] Jeman Park, Aminollah Khormali, Manar Mohaisen, and Aziz Mohaisen. Where are you taking me? behavioral analysis of open dns resolvers. 2019.

Eidesstattliche Versicherung

(Affidavit)

Girit, Ege

Name, Vorname
(surname, first name)

197921

Matrikelnummer
(student ID number)

■ Bachelorarbeit
(Bachelor's thesis)

☐ Masterarbeit
(Master's thesis)

Titel
(Title)

Analysis of authoritative DNS infrastructure failures

Ich versichere hiermit an Eides statt, dass ich die vorliegende Abschlussarbeit mit dem oben genannten Titel selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

I declare in lieu of oath that I have completed the present thesis with the above-mentioned title independently and without any unauthorized assistance. I have not used any other sources or aids than the ones listed and have documented quotations and paraphrases as such. The thesis in its current or similar version has not been submitted to an auditing institution before.

Dortmund, 15.01.2023

Ort, Datum
(place, date)

Unterschrift
(signature)

Belehrung:
Wer vorsätzlich gegen eine die Täuschung über Prüfungsleistungen betreffende Regelung einer Hochschulprüfungsordnung verstößt, handelt ordnungswidrig. Die Ordnungswidrigkeit kann mit einer Geldbuße von bis zu 50.000,00 € geahndet werden. Zuständige Verwaltungsbehörde für die Verfolgung und Ahndung von Ordnungswidrigkeiten ist der Kanzler/die Kanzlerin der Technischen Universität Dortmund. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Täuschungsversuches kann der Prüfling zudem exmatrikuliert werden. (§ 63 Abs. 5 Hochschulgesetz - HG -).

Die Abgabe einer falschen Versicherung an Eides statt wird mit Freiheitsstrafe bis zu 3 Jahren oder mit Geldstrafe bestraft.

Die Technische Universität Dortmund wird ggf. elektronische Vergleichswerkzeuge (wie z.B. die Software „turnitin") zur Überprüfung von Ordnungswidrigkeiten in Prüfungsverfahren nutzen.

Die oben stehende Belehrung habe ich zur Kenntnis genommen:

Official notification:
Any person who intentionally breaches any regulation of university examination regulations relating to deception in examination performance is acting improperly. This offense can be punished with a fine of up to EUR 50,000.00. The competent administrative authority for the pursuit and prosecution of offenses of this type is the Chancellor of TU Dortmund University. In the case of multiple or other serious attempts at deception, the examinee can also be unenrolled, Section 63 (5) North Rhine-Westphalia Higher Education Act (*Hochschulgesetz, HG*).

The submission of a false affidavit will be punished with a prison sentence of up to three years or a fine.

As may be necessary, TU Dortmund University will make use of electronic plagiarism-prevention tools (e.g. the "turnitin" service) in order to monitor violations during the examination procedures.

I have taken note of the above official notification:*

Dortmund, 15.01.2023

Ort, Datum
(place, date)

Unterschrift
(signature)

***Please be aware that solely the German version of the affidavit ("Eidesstattliche Versicherung") for the Bachelor's/ Master's thesis is the official and legally binding version.**