Defragmenting DNS

Determining the optimal maximum UDP response size for DNS

Sunday 5th July, 2020

Axel Koolhaas
axel.koolhaas@os3.nl
Security and Network Engineering
University of Amsterdam

Tjeerd Slokkker
tjeerd.slokker@os3.nl
Security and Network Engineering
University of Amsterdam

***Abstract*—The Domain Name System (DNS) is the core technology of the Internet translating hostnames to Internet Protocol (IP) addresses. DNS uses the connectionless User Datagram Protocol (UDP) by default, which causes problems with Path MTU Discovery. This is because DNS servers are stateless, and do not remember queries they have already answered. The Path MTU (PMTU) should be used as maximum size to stop fragmentation from happening. Extension Mechanisms for DNS (EDNS(0)) expands DNS with the UDP Message Size field, which communicates the response size capability of the resolver. This allows resolvers to specify the EDNS(0) they support.**

In this research, we aim to provide data for a considered optimal maximum EDNS(0) UDP message size, by measuring the PMTU to which resolvers and stub resolvers on the Internet are subject. We did this by creating an environment to serve different sized DNS responses and querying this environment across the Internet. This is done with the cooperation and supervision of NLnet Labs and aligns with the goals DNS Flag Day 2020. Our ambition is to suggest defaults for the maximum EDNS(0) message size for DNS.

Based on our results, we recommend an EDNS(0) message size of 1372 for IPv4 and 1332 for IPv6 stub resolvers in internal networks. We recommend to use an EDNS(0) message size of 1232 for stub resolvers that communicate to resolvers using IPv6. IPv4 stub resolvers can still use an EDNS(0) message size of 1372. With regards to resolvers, we recommend an EDNS(0) message size of 1232 for IPv4 and 1332 for IPv6.

***Index Terms*—DNS, MTU, Fragmentation, PMTU, EDNS(0)**

I. INTRODUCTION

The Internet is constructed as an interconnected network of networks where messages, which in this paper we refer to as packets, are exchanged. Not every network supports the same packet size, defined as the Maximum Transmission Unit (MTU). To cope with this, Internet Protocol (IP) packets can be divided into smaller pieces, called fragments. Thus, fragmentation can occur if the number of bytes of an IP packet exceed the MTU of a network. Fragmentation can cause problems for the Domain Name System (DNS).

The problems fragmentation causes for DNS are numerous. They vary from connectivity problems to security vulnerabilities. This is described in further detail in Section III. The biggest concern is connectivity. Since DNS is at the core of the Internet, no connectivity means no connection for most end users, unless they remember IP addresses. Therefore, network operators try to reduce fragmentation occurrence for DNS packets.

DNS Flag Day [1] is an initiative by several corporations and organizations with the endeavor to improve the current state of DNS employed worldwide. This is done by examining interoperability and performance issues with the DNS brought forth on industry mailing lists and conferences. To solve these issues, standards and solutions are enforced during the Flag Day by collaboratively modifying the way DNS resolvers behave. The upcoming DNS Flag Day, planned to be held in October 2020, aims to tackle the problems with IP fragmentation of DNS packets.

This research aims to substantiate the choice for a recommended packet size and provide an optimal MTU value for DNS packets, suggesting a default for network operators to handle. This aligns with the goals of DNS Flag Day [1], improving the functioning of the Internet.

A. Structure

The remainder of this paper is structured as follows: In Section III, we provide some background information and examine the highlights of previous work done by others that relates to ours. In Section IV, we define our experiments and environment along with the structure of gathering the results and analysing them. We present the results of our experiments in Section V. In Section VI, we discuss our findings. Using our findings, we will draw conclusions and present those in Section VII. The last Section VIII, contains suggestions for future work.

II. RESEARCH QUESTION

For this research, we have formulated the following research question:

What is the optimal EDNS message size to avoid IP fragmentation?

Which results into the following sub-questions:

- *Is there a difference between IPv4 and IPv6 regarding MTU sizes?*
- *Which EDNS message size is best in terms of support for DNS stub resolvers?*

- *Which EDNS message size is best in terms of support for DNS resolvers?*

III. Background

IP fragmentation introduces fragility to Internet communications. Fragmentation occurs if the number of bytes that a single IP packet can convey in an Internet path is exceeded. This limit is called the Path MTU (PMTU). The resulting fragility is troublesome, because it is computationally expensive, holds state, prone to errors, and susceptible to attacks [2].

Path MTU Discovery (PMTUD) discovers the PMTU between two nodes. This is used as an alternative to in transit fragmentation. PMTUD does this conservatively. However, this results into a PMTU that is usually less than the actual PMTU [2]. Also, PMTUD relies on Internet Control Message Protocol (ICMP) fragmentation needed (type 3, code 4) for IPv4 and ICMPv6 Too Big (PTB) messages for IPv6. These message can be unsupported or even blocked by nodes on the path. Worse, the ICMP messages can be forged [3]. IPv6 also discourages the use of fragmentation, thus it can only be applied by source nodes, and not by routers along a packet's delivery path [4].

PMTUD and the resulting fragmentation also causes problems with DNS servers, which are known to be stateless. This is because if such an ICMP message is received, the DNS server does not remember the original question anymore. Therefore, it cannot resend the fragmentated answer.

The original DNS standard by Mockapetris *et al.* [5], specifies the use of port 53 with both Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) for transport. It recommends UDP for standard queries and demands TCP for zone transfers. Using UDP as initial protocol is due to the lower overhead and better performance. The DNS packet also has the TrunCation (TC) bit, which indicates that a response has been truncated. This is due to it exceeding the length permitted on the transmission channel. The length permitted in original DNS UDP messages is 512 octets. If a response has the TC bit set, the client will retransmit the DNS query over TCP.

Extension Mechanisms for DNS (EDNS(0)) expands and enhances DNS and is the default for modern DNS servers. EDNS(0) adds information to DNS messages in the form of a pseudo-Resource Record (RR) of the type OPT, due to lack of space in the DNS message header [6]. One noteworthy improvement is the increase of the maximum UDP packet size from 512 octets to a larger size, with 4096 octets as starting point suggestion. It this with a new field named the UDP Message Size, which communicates the response size capability of the resolver.

DNS works according to a client-server model, where the domain name space is subdivided into a tree structure. Authoritative name servers are leaves within the tree, and provide the final answer to DNS queries. The client program that queries name servers is called a resolver, Mockapetris *et al.* [5]. Resolvers can be further divided into two subclasses, namely stub resolvers and resolvers. In this paper, we refer to a stub resolver as a computer program querying the authoritative name server itself without an in-between party, whereas a resolver is queried by a stub resolver to do the lookup for them.

A. Related work

Weaver, Kreibich, Nechaev, *et al.* [7] showed with a dataset gathered using Netalyzr that IP fragmentation causes problems for DNS, especially with DNSSEC and other size-sensitive DNS features. When using fragmented UDP datagrams, 8% of the sessions could not send fragmented UDP and, more importantly for DNS, 9% could not receive fragmented UDP datagrams. Van Den Broek, Rijswijk-Deij, Sperotto, *et al.* [8] expanded on this, showing that as much as 10.5% of all resolvers suffer from fragmentation-related connectivity issues. They based their observations on networks traces recorded from an authoritative name server of SURFnet. They also verified their results using traces from an authoritative name server at the University of Pennsylvania.

Different techniques have been used to measure and analyze the PMTU regarding DNS. The article by Toorop [9] presents a technique to experiment with different EDNS message sizes. Different sub-domains are used, where each produces different responses. For example, querying `1280.gorilla.nlnetlabs.nl TXT` produces an answer of 1280 bytes. Accordingly, querying `1600.gorilla.nlnetlabs.nl TXT` produces a fragment of 1496 bytes and one of 160 bytes. This fragmentation is due to the maximum MTU. This method requires custom name servers, which decrease the reproducibility of resulting research. Thus, we prefer to use default components. Furthermore, this exact method is not suited for our research, shown in Section IV, since different queries are used for different sizes, whereas we want to be able to receive different response sizes based on the same query.

DNS-OARC [10] published a method that can be used to determine the maximum reply size between a DNS server and a resolver. They use a different method compared to Toorop [9] to test different reply sizes. They do this with a custom DNS server and chained CNAME responses. The custom DNS server sends multiple replies, where each reply decreases in size. This makes the resolver follow the CNAME of the largest reply it receives. This way, a single query can be used to test for different reply sizes. We came up with a DNAME implementation based on this method, shown in Section IV.

Fujiwara and Vixie [11] wrote a draft on fragmentation avoidance in DNS. In this draft, a suggestion is made on a possible maximum DNS/UDP payload size. "To allow for possible IP options and faraway tunnel overhead, a useful default for maximum DNS/UDP payload size would be 1400." However, there is no scientific support for this suggestion. In our research, we used this suggestion and tested it for both IPv4 and IPv6. They also mention some reasons to stay away from IP-fragmentation, e.g., "Fragmentation Considered Poisonous" by Herzberg and Shulman [12], "IP fragmentation attack on DNS" by Hlavacek [13], "Domain Validation++

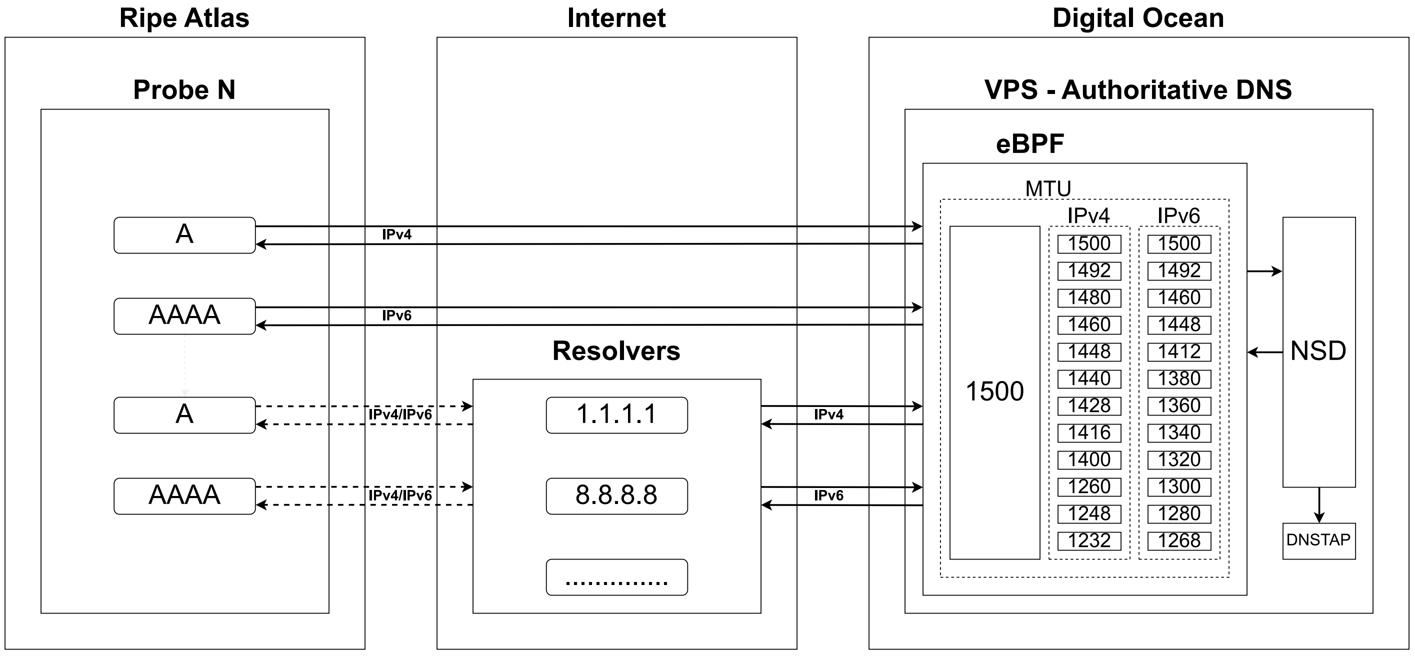

Fig. 1. Test Environment. Each arrow indicates a data stream between components. The solid arrows between Probes and VPS, and resolvers and VPS are the queries that are being researched. The dotted arrows from probes to resolvers are out of our measuring influence.

For MitM-Resilient PKI" by Brandt, Dai, Klein, *et al.* [14], and lastly "Security Implications of Predictable Fragment Identification Values" by Gont [15]. Besides the attack vectors they also mention that in RFC 8085 [16], it is specified that an application SHOULD NOT send UDP datagrams which result in IP-packets that exceed the MTU along the path to the destination. Bonica [2] summarized that IP fragmentation introduces fragility to Internet communication.

IV. Methodology

A. Test Environment

To determine the optimal EDNS message size, we conducted theoretical research and practical experiments. The theoretical research consists of consulting existing literature regarding this subject, previous research, and results from NLnet Labs. For the experiments, we used an online environment with a Name Server Daemon (NSD), an open source, authoritative name server developed by NLNet Labs [17].

Figure 1 illustrates our testing environment. It consists of three main components. The first component is the RIPE (Réseaux IP Européens) Atlas platform. RIPE Atlas is an Internet data collection system based on a global network of devices, called probes and anchors, that can actively measure Internet connectivity [18]. The second component are paths through the Internet, which includes third party DNS resolvers, e.g., Cloudflare or Google or network resolvers. The last component is a Virtual Private Server (VPS) or Virtual Machine (VM) providing DNS, hosted by Digital Ocean.

This VPS also consists of multiple components, one of them being NSD. The NSD instance is configured with two zone files, one being only available through IPv4 and one only available through IPv6. This is achieved by delegating the respective zone file through an NS record that only an address of corresponding protocol can reach. Each zone file has DNAME records to allow for different sized responses. We want to keep this as generic as possible, answering with A and AAAA records. Since DNS was primarily developed for looking up IP addresses, all resolvers should support these address records. All DNS queries to NSD are logged with dnstap. "Dnstap is a flexible, structured binary log format for DNS software. It uses Protocol Buffers to encode events that occur inside DNS software in an implementation-neutral format [19]."

We use separate queries to distinguish between IPv4 and IPv6 along with the distinction between stub resolvers and resolvers. The reason behind the differentiation between the stub and resolver is because assumed discrepancies, e.g., the PMTU between stubs and resolvers. IPv4 and IPv6 are tested through delegation. The distinction between a stub or resolver is made using parameters of RIPE Atlas measurements. Therefore, to account for all cases, we defined four unique queries to measure the PMTU between the probes and the authoritative DNS server. Each RIPE Atlas probe will send all four queries. These queries are:

- `$r-$t-$p-rslv.1500-plus0.pmtu4.rootcanary.net A`
- `$r-$t-$p-rslv.1500-plus0.pmtu6.rootcanary.net AAAA`
- `$r-$t-$p-stub.1500-plus0.pmtu4.rootcanary.net A`
- `$r-$t-$p-stub.1500-plus0.pmtu6.rootcanary.net AAAA`

The variables `$r-$t-$p`, denoted by a `$`, are used to add a random number, a timestamp, and the probe identifier respectively to the query. This makes it possible to correlate the results from RIPE Atlas with the data gathered by dnstap.

Multiple MTU sizes need to be tested, and therefore it is necessary that NSD answers to different sized queries with responses that match in size. We facilitate this using Delegation Name (DNAME) records. For each MTU that we will analyze, we create a DNAME record with the MTU size included in the name. For example, to test an MTU of 1500 bytes, 1500-plus0 needs to be queried or likewise, for an MTU of 1280 bytes, 1280-plus0 will be queried. Thus, the padding is facilitated by DNAME RDATA records that introduce extra characters in the query. These extra characters are always the letter "x". The fact that this padding is possible with DNAMEs, is because the RFC [20] does not allow DNAME RDATA to be compressed.

However, the variable of `$p`, the probe ID, differs in length between 1 and 7 characters. To combat this we created a padding option in the query. This is where the plus0 in the query is used for, where the addition is represented using hexadecimals. So for example, a query with a probe ID length of 7 characters needs to query 1500-plus0 to pad the answer to 1500 bytes. For a probe ID length of 1 character, a query with 1500-plusc should be used to pad the answer to 1500 bytes. This is due to the short probe ID occupying less space in the query opposed to a large probe ID. So 1500-plusc pads the query with 12 bytes to the same length as a query with a probe ID length of 7 characters and a padding of 0 bytes. The zones files with all the padding can be found in the appendices A and B.

We received ongoing Atlas measurements from RIPE that utilize all probes available at all times. With regular user credentials it would be challenging to constantly spawn new measurements based on previous results. These RIPE Atlas queries are static and it is not possible to make different queries based on MTU size. Thus, in collaboration with NLnet Labs, we used extended Berkeley Packet Filter (eBPF) [21] to rewrite the query of incoming packets to the MTU sizes we want to measure. These are subsequently processed by NSD, where afterwards the query inside the response packet is again rewritten to the incoming query value.

We try to measure a different MTU every hour, beginning with the Atlas measurement activating all online probes and making them perform the constant queries. DNS queries arrive with the question $r-$t-$p-stub.1500-plus0.pmtu4.rootcanary.net A at the associated interface of the VM. eBPF will rewrite the question section from 1500 to the MTU we want to measure at that given moment, e.g., 1400. It will also rewrite the addends of the plus part in the query depending on the length of the probe ID. The rewritten packet will then continue to NSD, which will respond with an answer of 1400 bytes in this example. The answer will then return through eBPF, which rewrites the question section back to the original 1500-plus0. Thus, eBPF does not change the outgoing answer section. However, to make sure IP fragmentation does not occur, eBPF will add the Don't Fragment (DF) flag to all outgoing IPv4 packets. The eBPF filter needs to be rebuild and reloaded to test a different MTU size for IPv4 and IPv6. This setup can potentially be extended to set an MTU per unique probe identifier.

B. Tests

IPv4 and IPv6 have different header sizes and different requirements regarding the MTU. Hence it is important to make a distinction in MTU measurements based on them. It is also important to test the resolver and stub independently, since we assume that resolvers are most of the time located in higher performing networks.

For UDP, fragmentation should be avoided for resiliency and authenticity reasons. That entails that an EDNS message size lower than most PMTUs needs to be chosen. However, it should be noted that the lower the message size, the earlier TCP is required to answer queries, impacting performance. To reduce performance impact caused by fallback over TCP (2 extra round trips) to a minimum, an EDNS message size needs to be chosen which is as close to the highest supported PMTU as possible.

Based on information of Fujiwara and Vixie [11], DNS Flag Day [1], and Arends, Austein, Larson, *et al.* [22], we determined which MTUs need to be tested. The maximum MTU is 1500 bytes, as this is the default on most of the Internet. The lowest is 1232 bytes for IPv4 and 1268 bytes for IPv6, as suggested by Arends, Austein, Larson, *et al.* [22]. These are MTU sizes, but we need to take into account that the EDNS message size is the MTU minus the IP and UDP headers. This varies depending on IPv4 and IPv6, where for IPv4 we subtract 28 bytes, which aligns with the IPv4 header and UDP header. In case of IPv6, we subtract 48 bytes, which is the size of the IPv6 header in addition with the UDP header. For both cases, we assume that no option fields will be enabled. If the EDNS message size of a query is smaller than the EDNS message size we want to test at that moment, then we will discard that result. This is due to the resolver not supporting the EDNS message size we want to measure. Also, all TCP queries will be discarded as we only look at failed UDP queries. We mark a query as failed, if the query contains a EDNS message size higher or equal than the measured EDNS message size or if the probe did not receive an answer.

We added extra MTU sizes to cover more within this research and get an more elaborate overview. These extra MTU sizes are based on the size of optional (extension) headers and possible encapsulation in tunnels. However, not all probes are online for the entire day. To get as much results from all probes as possible, we measured all chosen MTUs two times per day with an interval of 12 hours. This results in 12 measurements from 00:00 to 11:59 and from 12:00 to 23:59.

C. Gathering Results

We need to gather and correlate the data of the sending side, i.e., the Atlas probe doing the query, and the receiving side, i.e., the NSD DNS server. We use the `dnstap` command-line tool to create a reliable byte-stream socket which NSD uses to send serialized event messages. These messages, with stateful DNS information, are then deserialized into JavaScript Object Notation (JSON). We then use the piped logging program `rotatelogs` from Apache to save the JSON formatted DNS logs for every hour.

To acquire the sending data, we utilized RIPE Atlas Cousteau, a Python wrapper around the RIPE Atlas API [23]. We fetch the data from public measurements of which the corresponding identifiers are shown in Table I. We further extended this with other Python scripts which collectively load the JSON formatted NSD logs and the Atlas data.

TABLE I
PUBLIC RIPE ATLAS MEASUREMENT IDENTIFIERS

	Stub	**Resolver**
IPv4	25741785	25741787
IPv6	25741786	25741788

For stub resolver around 10.000 queries were performed. This aligns with what we expect, since there are around 10.000 probes and each probe queries the authoritative nameserver only once. For the probe resolvers, around 20.000 queries were performed. This is because most probes have two or more resolvers configured and all resolvers will be used to send queries.

To perform data analysis and manipulation, we used pandas [24]. We store all data as DataFrame objects for fast and efficient data manipulation. The data is correlated depending on the type of resolver and done for both IPv4 and IPv6 separately.

For the stub resolvers, first the probe IDs from the failed stub Atlas queries are retrieved and the total number of queries are stored. Then we use the IDs to filter the dnstap logs and thus only load the logs from failed probes. Next, we drop duplicates since we only care about unique queries. This yields the total number of queries, the number of failed queries, and allows us to calculate the percentage of failed queries.

For DNS resolvers, the process needs to be done differently. First the dnstap logs are loaded and split on their transport protocol. We remove all results where the padding `x`'s occur in the query. This is due to caching where the resolver already tries to resolve the query. The occurrence of these padding `x`'s means we do not have 100% cache-hit free results. Next, we merge the separate DataFrames on their unique variable section, i.e., `$r-$t-$p`. This correlates the initial UDP EDNS message size with the subsequent TCP queries.

Then, we fetch the total number of queries from RIPE Atlas and use this together with the failed results to calculate the percentage of failed resolver queries.

Finally, all the hourly results are printed in JSON format. This result includes separate MTU values for IPv4 and IPv6, since we make a distinction as mentioned in the previous Subsection IV-B

D. Analysing Results

The processed data is plotted using Seaborn, a Python data visualization library based on matplotlib. We take the mean of all the MTU values and include the standard deviation.

We also saw discrepancies in the results which led us to further analyze our results by looking at the ASN of IP addresses of failing queries, seen in Section V. This was done with the aid of the pyasn library [25].

All our code is publicly available under the BSD license at Github [26].

V. RESULTS

We look at which EDNS message size is best in terms of support for DNS stub resolvers and DNS resolvers. As we have four different categories of queries, each will be presented on its own. First, we will look into the results of the DNS stub resolvers. Secondly, we will look into the results of the DNS resolvers. Secondly, the results are displayed with an MTU size on the x-axis, not the corresponding EDNS message size. The graphs are plotted from data collected in 2 consecutive days.

A. Stub resolvers

For the IPv4 stub resolver, the results are presented in Figure 2. It stands out that the MTU size 1500 has a higher mean failed percentage than the other MTUs. The MTU of 1500 results in a mean of 18.92 percentage of failed queries. An MTU of 1492 drops 12.54 percentage point to a mean percentage of 6.38. Going down to an MTU of 1480 it drops 1.49 percentage point. Then going to a MTU of 1460 it drops 1.97 percentage point, to a mean percentage of 2.92. The remainder of the MTUs keep going down in failed percentage up to the MTU of 1400. After this the results of the remaining MTUs are around 0.9 percent of failed queries. So for the two highest tested MTU 1500, we observe a significant higher failure rate, whereas the other tested MTUs differ around 5.5 percent point of failed queries.

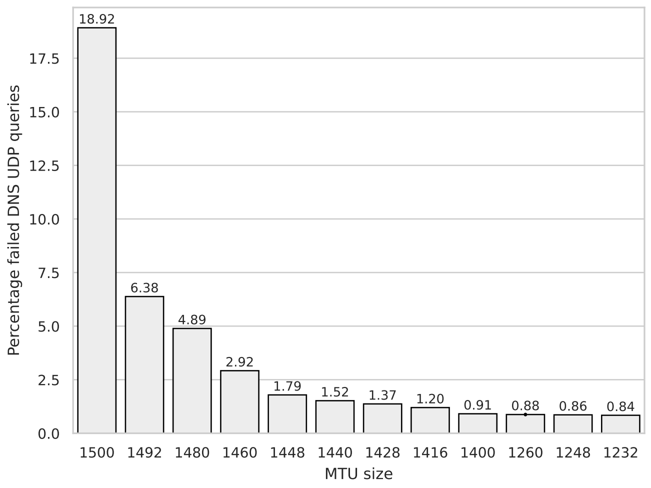

Fig. 2. Percentage of failed DNS UDP queries per MTU size for IPv4 stub resolvers

The results of the IPv6 stub resolver experiments are presented in Figure 3. An MTU of 1500 has a mean of 26.16

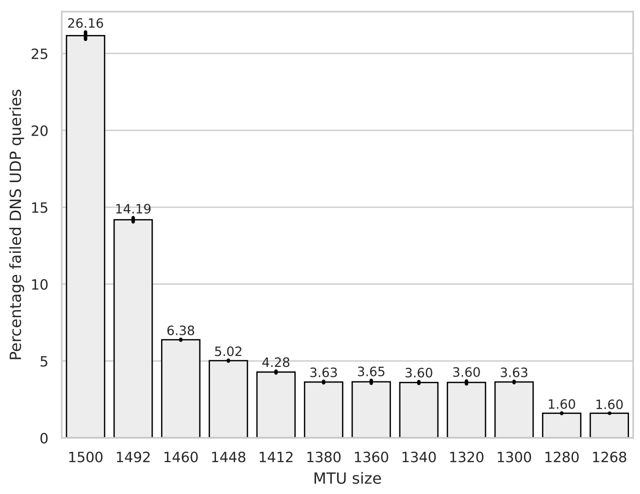

Fig. 3. Percentage of failed DNS UDP queries per MTU size for IPv6 stub resolvers

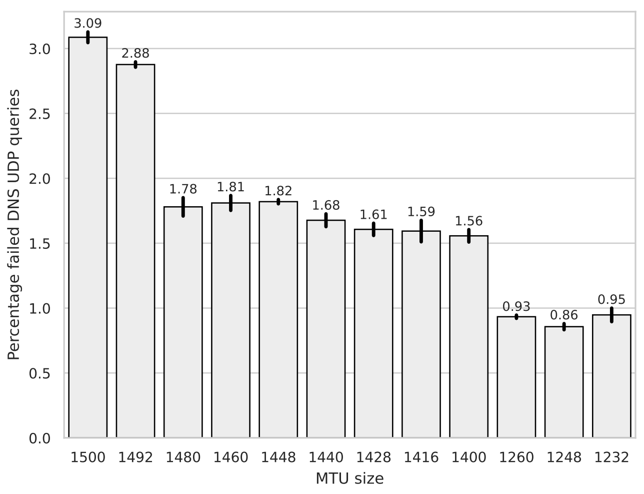

Fig. 4. Percentage of failed DNS UDP queries per MTU size for IPv4 resolvers

percent of failed queries. Going to an MTU of 1492 the failed queries dropped to a mean of 14.19 percent. An MTU of 1460 compared to 1500 dropped 19.78 percentage point, to a mean of 6.38 percent failed queries. 1380 up to including 1300 all are comparable equal in the mean percentage of failed queries. The next noticeable difference is going down to an MTU of 1280.

B. Resolvers

There are three groups of MTUs that stand out. The first group is the MTUs 1500 and 1492. Those have a mean percentage of failed queries around 3 percent, which is considerably more than most of the remaining MTUs. The second group is the MTUs 1480 up to and including 1400. Those have a mean percentage of failed queries around 1.65 percent. The last group is the MTUs 1260, 1248, and 1232. These only have a mean percentage failed queries of around 0.9 percent.

For IPv6 resolvers we do not see these three groups. We do see the MTUs 1500 and 1492 having a higher mean percentage than the remaining MTUs. Those have a mean percentage of failed queries around 1.35 percent. The remaining MTUs are decreasing in a linearly like fashion. So starting at the MTU 1460 with a mean percentage of 0.81, going down to MTU 1268 with a mean percentage of 0.43.

VI. Discussion

The background information including the related work helped us determine how the optimal EDNS message size needs to be determined. It showed that a lower EDNS message size does not equal a better performance. Also, a higher message size does not imply improvement. So an ENDS message size needs to be chosen which is as close to the highest supported path MTU as possible.

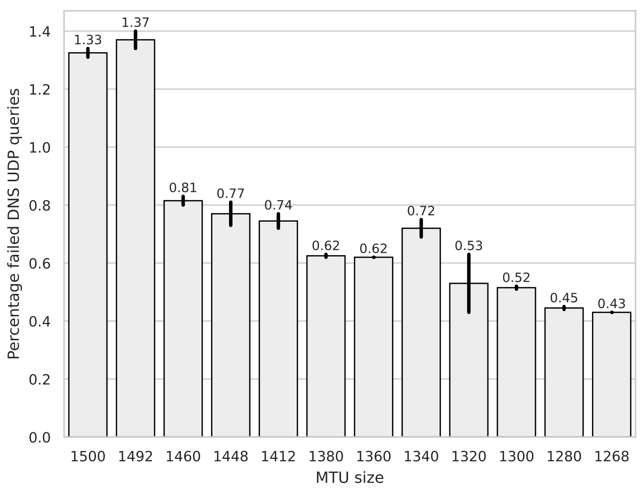

Fig. 5. Percentage of failed DNS UDP queries per MTU size for IPv6 resolvers

In our method we described four different queries. Two queries are for the stub resolvers and two are for the resolvers. As the specifications of IPv4 and IPv6 allow for different minimum MTUs, we made an individual query for both IPv4 and IPv6 for stub resolvers and resolvers. The results show that there is a significant difference between IPv4 and IPv6 and between the stub resolvers and resolvers. This makes that we will provide suggestions for those four different situations.

What is visible in the graphs is that the path MTUs of 1500 and 1492 have a noticeable higher percentage of failed queries. This is possibly due to tunneling and encapsulation, e.g., VLANs, GRE tunnels, and VPNs.

The standard deviations show that there is little variation in the measured results. Even though the difference between the mean of each MTU is small for the stub and resolvers, it still is significant. The population of stub and resolvers that is within this 0.01% is considerable. For the stub resolver results, 0.01% is around 100 queries and for the resolvers, it is around 200 queries.

For the IPv4 stub resolver test, we initially set the UDP checksum to 0. This should be supported by the RIPE Atlas probes [4]. However, the results we found with this implementation showed at least 13 percent failures for all MTUs. This indicates that the probes do not handle this checksum configuration well. After changing the configuration from setting the checksum to zero to calculating each checksum individually, the results were not failing as often, indicating a more accurate measurement.

The results of the stub resolvers present that, for IPv4, an MTU of 1400 does not differ much from the remaining MTUs, taking the standard deviation into account. So an EDNS message size of 1372 could be a valid option for stub resolvers on IPv4. The fact that the middle group of MTUs in the IPv6 stub resolver results are around the same percentage of failed queries, could be because of failed path MTU discoveries. In case of a real world scenario, these would fallback to 1280 as MTU. In our experiment we tested the path of a stub resolver over the internet to our own authoritative DNS. However, in most scenarios the stub resolver would talk to a resolver within its own network. This makes it difficult to suggest one EDNS message size for this case. To take tunneling and encapsulation into account we suggest to use a MTU of 1380 –EDNS message size of 1332– for internal stub resolvers as this is the highest MTU that has around the same mean percentage of failed queries as the other MTUs. Because of the failing path MTU discoveries, we suggest a MTU of 1280 –EDNS message size of 1232– for stub resolvers that communicate to a resolver. This as it is a trade-off between a higher MTU and a lower mean percentage of failed queries.

Based on the results of the resolvers for IPv4, we suggest that an MTU of 1260 –EDNS message size of 1232– could be chosen. Based on the results for the IPv6 resolvers, we would advice to choose an MTU of 1380 –EDNS message size of 1332–. The fact that for IPv4 there is a middle group of MTUs that are around the same percentage of failed queries, made us dive deeper in the results. We observed there is no single AS that is the cause of the higher percentage failing queries.

As the MTU of 1500 is the highest tested MTU, it is important that the Digital Ocean environment supported MTUs of this size. This was not known in advance, but the experiment showed that there are positive results on a MTU of 1500. This means that our experiment environment supported our maximum MTU, so this was not a bottleneck in our experiments.

A DNS query comes in through a specific path. This path however, can differ for the answer. So because the path selection is dynamic, we cannot conclude anything on the MTU of the incoming packets. Only for the packets we send out. However, to determine the failed resolver queries, we had to look at the incoming packets. In the case that a resolver sends an UDP query that does not arrives at NSD, we are unable to measure that this took place. After a failed attempt, the resolver could come(back) over TCP. In that case we are only able to measure DNS resolvers that had failed UDP queries, when a TCP question arrived.

In our results, we can see that even on the lowest MTU tested there are probes that fail. This could be because the network in between does not support that MTU. However, it could also be the case that there are probes that are always failing. This does have an impact on the height of the percentage of failed queries, but this would not be a big issue, as this would be the case for all tested MTUs. So overall the percentage of failed queries could be lower. We do not suspect that the amount of consistent failing probes is of a big significance.

Lastly, our research tries to depict an accurate representation of the Internet. RIPE atlas probes are mostly located in Europe and the USA. This means that there is a bias in results which can result in a distorted picture of the actual situation. The cause of the distribution of probes could be because RIPE Atlas is presumed to have a tech-bias too. This means that the probes are most likely to be handed out to the tech-savvy crowd that attends RIPE and similar conferences.

VII. Conclusion

In this research, we analyzed a sample of the Internet to determine the optimal maximum UDP response size for DNS. We constructed a reproducible environment with standard components except for the RIPE Atlas probes as population. Based on our results, we recommend an EDNS(0) message size of 1372 for IPv4 and 1332 for IPv6 stub resolvers in internal networks. For stub resolvers that communicate to resolvers, we recommend to use 1232 as EDNS(0) message size for IPv6, IPv4 can still use the 1372. For resolvers, we recommend an EDNS(0) message size of 1232 for IPv4. IPv4 resolvers did have a centered group that had around the same percentage of failed queries. We suspected this to be specific ASs failing, but after analysing the results, no AS could be singled out as the cause of the higher percentage of failed queries. Finally, we suggest an EDNS(0) message size of 1332 for IPv6 resolvers.

VIII. Future work

A. MTUs

In future research more than 12 MTUs could be chosen to analyze. It is possible to research all MTUs available in a specific range. To go even further, looking at specific MTUs for individual probes could give a deeper insight in the supported MTUs on specific networks. This could also help with getting rid of the time bias in the measurements.

Further follow-up research would be analyzing the MTU sizes per AS. This could provide information on supported MTU sizes by different ASs instead of different probes, which could help in looking at ASs that are not supporting the general supported MTU(s). However, it could be the case that there are ASs that have a higher degree of probes in their network.

This could result in drawing the wrong conclusions. So future research could be analyzing the distribution of measurement probes across ASs.

B. Failing Probes

As mentioned in the discussion, there are a number of probes that could always be failing. To get a more accurate result from RIPE Atlas measurements, research could be done to determine which probes are always failing. These probes could then be researched with a higher degree of depth. That way, a better conclusion could be made to say whether the MTU size is the problem or that there is something wrong with the probe. It should be noted that this research should only focus on the experiments we conducted, as the probes support multiple measurement types, e.g., HTTPS, ICMP. So if a probe fails for DNS, it does not mean the probe is broken, but it could be that only the DNS functionality is not working properly or that the probe resides behind a firewall that is filtering out its DNS traffic.

C. Continuation

The Internet is constantly changing and it is important to adapt to these changes. This is also what we try to achieve, as we do not only do suggestions on choosing the optimal EDNS message sizes, but we also illustrated a reproducible environment that is in the public domain. The suggestions we do now are likely to change in the future. Therefore, it is important to actively pursue the continuation of this research, adjusting the EDNS message sizes when due. This keeps DNS performing as optimal as possible.

IX. Acknowledgements

We want to thank everyone who helped us in making this research possible. We are especially grateful for NLnet Labs, who facilitated this research. Likewise, we thank RIPE for providing Atlas measurements. We would also like to thank Willem Toroop and Roland van Rijswijk-Deij particularly, for all their supervision and feedback.

References

[1] DNS Flag Day. (2020). Dns flag day, [Online]. Available: https://dnsflagday.net/.

[2] R. e. a. Bonica, "Ip fragmentation considered fragile," 2020.

[3] F. Gont, "Icmp attacks against tcp," RFC 5927, July, Tech. Rep., 2010.

[4] S. Deering, R. Hinden, *et al.*, *Internet protocol, version 6 (ipv6) specification*, 1998.

[5] P. Mockapetris *et al.*, "Domain names-implementation and specification," 1987.

[6] J. Damas, M. Graff, and P. Vixie, "Extension mechanisms for dns (edns (0))," *IETF RFC6891, April*, 2013.

[7] N. Weaver, C. Kreibich, B. Nechaev, and V. Paxson, "Implications of netalyzr's dns measurements," in *Proceedings of the First Workshop on Securing and Trusting Internet Names (SATIN), Teddington, United Kingdom*, Citeseer, 2011.

[8] G. Van Den Broek, R. van Rijswijk-Deij, A. Sperotto, and A. Pras, "Dnssec meets real world: Dealing with unreachability caused by fragmentation," *IEEE communications magazine*, vol. 52, no. 4, pp. 154–160, 2014.

[9] W. Toorop. (Jun. 2013). Using pmtud for a higher dns responsiveness, [Online]. Available: https://medium.com/nlnetlabs/using-pmtud-for-a-higher-dns-responsiveness-60e129917665.

[10] DNS-OARC. (). Reply size test, [Online]. Available: https://www.dns-oarc.net/oarc/services/replysizetest.

[11] K. Fujiwara and P. Vixie. (Jun. 2020). Fragmentation avoidance in dns, [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-dnsop-avoid-fragmentation/.

[12] A. Herzberg and H. Shulman, "Fragmentation considered poisonous, or: One-domain-to-rule-them-all. org," in *2013 IEEE Conference on Communications and Network Security (CNS)*, IEEE, 2013, pp. 224–232.

[13] T. Hlavacek, *Ip fragmentation attack on dns*, 2013. [Online]. Available: https://ripe67.ripe.net/presentations/240-ipfragattack.pdf.

[14] M. Brandt, T. Dai, A. Klein, H. Shulman, and M. Waidner, "Domain validation++ for mitm-resilient pki," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 2060–2076.

[15] F. Gont, "Security implications of predictable fragment identification values," RFC 7739, DOI 10.17487/RFC7739, February 2016,¡ http://www. rfc-editor. org ..., Tech. Rep., 2012.

[16] L. Eggert and G. Fairhurst, "G. shepherd," udp usage guidelines," BCP 145, RFC 8085, DOI 10.17487/RFC8085, March 2017,¡ https://www. rfc ..., Tech. Rep.

[17] NLnetLabs. (Jun. 2020). Nsd, [Online]. Available: https://nlnetlabs.nl/projects/nsd/about/.

[18] R. N. Staff, "Ripe atlas: A global internet measurement network," *Internet Protocol Journal*, vol. 18, no. 3, 2015.

[19] R. Edmonds and P. Vixie. (Jun. 2020). Dnstap, [Online]. Available: https://dnstap.info/.

[20] S. Rose and W. Wijngaards, "Dname redirection in the dns," RFC 6672, June, Tech. Rep., 2012.

[21] S. McCanne and V. Jacobson, "The bsd packet filter: A new architecture for user-level packet capture.," in *USENIX winter*, vol. 46, 1993.

[22] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, "Protocol modifications for the dns security extensions," RFC 4035, March, Tech. Rep., 2005.

[23] RIPE. (Jul. 2020). Ripe atlas cousteau, [Online]. Available: https://ripe-atlas-cousteau.readthedocs.io/en/latest/index.html.

[24] W. McKinney and P. Team, "Pandas: Powerful python data analysis toolkit," *Pandas—Powerful Python Data Analysis Toolkit*, p. 1625, 2015.

[25] H. Asghari. (Jul. 2020). Pyasn, [Online]. Available: https://github.com/hadiasghari/pyasn.
[26] A. Koolhaas. (Jul. 2020). Defragdns, [Online]. Available: https://github.com/shoaloak/defragDNS.

Appendix

Note that for the zone Listings, the consecutive sequences have been simplified with three dots to reduce size.

A. Zone file IPv4

Listing 1. IPv4 DNS Zone File

```
$ORIGIN pmtu4.rootcanary.net.
$TTL 1

@  25200 SOA   ns.pmtu4.rootcanary.net. sysadm.rootcanary.org. (
         2020062400 ; serial
         10800      ; refresh (3 hours)
         3600       ; retry (1 hour)
         604800     ; expire (1 week)
         300        ; minimum (5 minutes)
         )
   25200 NS ns
ns 25200 A  64.227.79.193



; MSG SIZE: 1472
1500-plus0  DNAME xxx.1500.pmtu4.rootcanary.net.
1500-plus2  DNAME xxxx.1500.pmtu4.rootcanary.net.
...
1500-plusa  DNAME xxxxxxxx.1500.pmtu4.rootcanary.net.
1500-plusc  DNAME xxxxxxxxx.1500.pmtu4.rootcanary.net.



; MSG SIZE: 1464
1492-plus0  DNAME xxxxxxx.1492.pmtu4.rootcanary.net.
1492-plus2  DNAME xxxxxxxx.1492.pmtu4.rootcanary.net.
...
1492-plusa  DNAME xxxxxxxxxxxx.1492.pmtu4.rootcanary.net.
1492-plusc  DNAME xxxxxxxxxxxxx.1492.pmtu4.rootcanary.net.



; MSG SIZE: 1452
1480-plus0  DNAME x.1480.pmtu4.rootcanary.net.
1480-plus2  DNAME xx.1480.pmtu4.rootcanary.net.
...
1480-plusa  DNAME xxxxxx.1480.pmtu4.rootcanary.net.
1480-plusc  DNAME xxxxxxx.1480.pmtu4.rootcanary.net.



1460-plus0  DNAME xxxxxxx.1460.pmtu4.rootcanary.net.
1460-plus2  DNAME xxxxxxxx.1460.pmtu4.rootcanary.net.
...
1460-plusa  DNAME xxxxxxxxxxxx.1460.pmtu4.rootcanary.net.
1460-plusc  DNAME xxxxxxxxxxxxx.1460.pmtu4.rootcanary.net.



1448-plus0  DNAME x.1448.pmtu4.rootcanary.net.
1448-plus2  DNAME xx.1448.pmtu4.rootcanary.net.
...
1448-plusa  DNAME xxxxxx.1448.pmtu4.rootcanary.net.
1448-plusc  DNAME xxxxxxx.1448.pmtu4.rootcanary.net.



1440-plus0  DNAME xxxxx.1440.pmtu4.rootcanary.net.
1440-plus2  DNAME xxxxxx.1440.pmtu4.rootcanary.net.
...
1440-plusa  DNAME xxxxxxxxxx.1440.pmtu4.rootcanary.net.
1440-plusc  DNAME xxxxxxxxxxx.1440.pmtu4.rootcanary.net.
```

```
; MSG SIZE: 1404
1428-plus0  DNAME xxxxxxx.1428.pmtu4.rootcanary.net.
1428-plus2  DNAME xxxxxxxx.1428.pmtu4.rootcanary.net.
...
1428-plusa  DNAME xxxxxxxxxxxx.1428.pmtu4.rootcanary.net.
1428-plusc  DNAME xxxxxxxxxxxxx.1428.pmtu4.rootcanary.net.



1416-plus0  DNAME x.1416.pmtu4.rootcanary.net.
1416-plus2  DNAME xx.1416.pmtu4.rootcanary.net.
...
1416-plusa  DNAME xxxxxx.1416.pmtu4.rootcanary.net.
1416-plusc  DNAME xxxxxxx.1416.pmtu4.rootcanary.net.



; MSG SIZE: 1372
1400-plus0  DNAME x.1400.pmtu4.rootcanary.net.
1400-plus2  DNAME xx.1400.pmtu4.rootcanary.net.
...
1400-plusa  DNAME xxxxxx.1400.pmtu4.rootcanary.net.
1400-plusc  DNAME xxxxxxx.1400.pmtu4.rootcanary.net.



; MSG SIZE: 1252
1280-plus0  DNAME xxxxx.1280.pmtu4.rootcanary.net.
1280-plus2  DNAME xxxxxx.1280.pmtu4.rootcanary.net.
...
1280-plusa  DNAME xxxxxxxxxx.1280.pmtu4.rootcanary.net.
1280-plusc  DNAME xxxxxxxxxxx.1280.pmtu4.rootcanary.net.



1260-plus0  DNAME xxx.1260.pmtu4.rootcanary.net.
1260-plus2  DNAME xxxx.1260.pmtu4.rootcanary.net.
...
1260-plusa  DNAME xxxxxxxx.1260.pmtu4.rootcanary.net.
1260-plusc  DNAME xxxxxxxxx.1260.pmtu4.rootcanary.net.



1248-plus0  DNAME xxxxx.1248.pmtu4.rootcanary.net.
1248-plus2  DNAME xxxxxx.1248.pmtu4.rootcanary.net.
...
1248-plusa  DNAME xxxxxxxxxx.1248.pmtu4.rootcanary.net.
1248-plusc  DNAME xxxxxxxxxxx.1248.pmtu4.rootcanary.net.



; MSG SIZE: 1204
1232-plus0  DNAME xxxxx.1232.pmtu4.rootcanary.net.
1232-plus2  DNAME xxxxxx.1232.pmtu4.rootcanary.net.
...
1232-plusa  DNAME xxxxxxxxxx.1232.pmtu4.rootcanary.net.
1232-plusc  DNAME xxxxxxxxxxx.1232.pmtu4.rootcanary.net.



; MSG SIZE: 1192
1220-plus0  DNAME xxxxxxx.1220.pmtu4.rootcanary.net.
1220-plus2  DNAME xxxxxxxx.1220.pmtu4.rootcanary.net.
...
1220-plusa  DNAME xxxxxxxxxxxx.1220.pmtu4.rootcanary.net.
1220-plusc  DNAME xxxxxxxxxxxxx.1220.pmtu4.rootcanary.net.


;
; Wildcards hack
;

*.1500          A       10.0.150.0
                A       10.1.150.0
                ...
                A       10.77.150.0
                A       10.78.150.0


*.1492          A       10.0.149.2
                A       10.1.149.2
                ...
                A       10.76.149.2
                A       10.77.149.2



*.1480          A       10.0.148.0
                A       10.1.148.0
                ...
                A       10.76.148.0
                A       10.77.148.0


*.1460          A       10.0.146.0
                A       10.1.146.0
                ...
                A       10.74.146.0
                A       10.75.146.0



*.1448          A       10.0.144.8
                A       10.1.144.8
                ...
                A       10.74.144.8
                A       10.75.144.8


*.1440          A       10.0.144.0
                A       10.1.144.0
                ...
                A       10.73.144.0
                A       10.74.144.0


*.1428          A       10.0.142.8
                A       10.1.142.8
                ...
                A       10.72.142.8
                A       10.73.142.8



*.1416          A       10.0.141.6
                A       10.1.141.6
                ...
                A       10.72.141.6
                A       10.73.141.6


*.1400          A       10.0.140.0
                A       10.1.140.0
                ...
                A       10.71.140.0
                A       10.72.140.0


*.1280          A       10.0.128.0
                A       10.1.128.0
                ...
                A       10.63.128.0
                A       10.64.128.0

*.1260          A       10.0.126.0
                A       10.1.126.0
                ...
                A       10.62.126.0
                A       10.63.126.0


*.1248          A       10.0.124.8
                A       10.1.124.8
                ...
                A       10.61.124.8
                A       10.62.124.8


*.1232          A       10.0.123.2
                A       10.1.123.2
                ...
                A       10.60.123.2
                A       10.61.123.2


*.1220          A       10.0.122.0
                A       10.1.122.0
                ...
                A       10.59.122.0
                A       10.60.122.0
```

B. Zone file IPv6

Listing 2. IPv6 DNS Zone File

```
$ORIGIN pmtu6.rootcanary.net.
$TTL 1

@  25200 SOA   ns.pmtu6.rootcanary.net. sysadm.rootcanary.org. (
         2020062400 ; serial
         10800      ; refresh (3 hours)
         3600       ; retry (1 hour)
         604800     ; expire (1 week)
         300        ; minimum (5 minutes)
         )
   25200 NS ns
ns 25200 AAAA  2a03:b0c0:2:f0::1b5:5001

;
; DNAME hack to respond with different sizes depending on l = len(probe_id)
;


1500-plus0  DNAME xxxxxxxxx.1500.pmtu6.rootcanary.net.
1500-plus2  DNAME xxxxxxxxxx.1500.pmtu6.rootcanary.net.
...
1500-plusa  DNAME xxxxxxxxxxxxxx.1500.pmtu6.rootcanary.net.
1500-plusc  DNAME xxxxxxxxxxxxxxx.1500.pmtu6.rootcanary.net.


1492-plus0  DNAME xxxxx.1492.pmtu6.rootcanary.net.
1492-plus2  DNAME xxxxxx.1492.pmtu6.rootcanary.net.
...
1492-plusa  DNAME xxxxxxxxxx.1492.pmtu6.rootcanary.net.
1492-plusc  DNAME xxxxxxxxxxx.1492.pmtu6.rootcanary.net.


1460-plus0  DNAME xxx.1460.pmtu6.rootcanary.net.
1460-plus2  DNAME xxxx.1460.pmtu6.rootcanary.net.
...
1460-plusa  DNAME xxxxxxxx.1460.pmtu6.rootcanary.net.
1460-plusc  DNAME xxxxxxxxx.1460.pmtu6.rootcanary.net.


1448-plus0  DNAME xxxxxxxxxxx.1448.pmtu6.rootcanary.net.
1448-plus2  DNAME xxxxxxxxxxxx.1448.pmtu6.rootcanary.net.
...
1448-plusa  DNAME xxxxxxxxxxxxxxxx.1448.pmtu6.rootcanary.net.
1448-plusc  DNAME xxxxxxxxxxxxxxxxx.1448.pmtu6.rootcanary.net.


1412-plus0  DNAME xxxxxxx.1412.pmtu6.rootcanary.net.
1412-plus2  DNAME xxxxxxxx.1412.pmtu6.rootcanary.net.
...
1412-plusa  DNAME xxxxxxxxxxxx.1412.pmtu6.rootcanary.net.
1412-plusc  DNAME xxxxxxxxxxxxx.1412.pmtu6.rootcanary.net.


1400-plus0  DNAME x.1400.pmtu6.rootcanary.net.
1400-plus2  DNAME xx.1400.pmtu6.rootcanary.net.
...
1400-plusa  DNAME xxxxxx.1400.pmtu6.rootcanary.net.
1400-plusc  DNAME xxxxxxx.1400.pmtu6.rootcanary.net.


1380-plus0  DNAME xxxxx.1380.pmtu6.rootcanary.net.
1380-plus2  DNAME xxxxxx.1380.pmtu6.rootcanary.net.
...
1380-plusa  DNAME xxxxxxxxxx.1380.pmtu6.rootcanary.net.
1380-plusc  DNAME xxxxxxxxxxx.1380.pmtu6.rootcanary.net.


1360-plus0  DNAME xxxxxxxxx.1360.pmtu6.rootcanary.net.
1360-plus2  DNAME xxxxxxxxxx.1360.pmtu6.rootcanary.net.
...
```

```
1360-plusa  DNAME xxxxxxxxxxxxxx.1360.pmtu6.rootcanary.net.
1360-plusc  DNAME xxxxxxxxxxxxxxx.1360.pmtu6.rootcanary.net.


1340-plus0  DNAME xxxxxxxxxxxxx.1340.pmtu6.rootcanary.net.
1340-plus2  DNAME xxxxxxxxxxxxxx.1340.pmtu6.rootcanary.net.
...
1340-plusa  DNAME xxxxxxxxxxxxxxxxxx.1340.pmtu6.rootcanary.net.
1340-plusc  DNAME xxxxxxxxxxxxxxxxxxx.1340.pmtu6.rootcanary.net.

1320-plus0  DNAME xxx.1320.pmtu6.rootcanary.net.
1320-plus2  DNAME xxxx.1320.pmtu6.rootcanary.net.
...
1320-plusa  DNAME xxxxxxxx.1320.pmtu6.rootcanary.net.
1320-plusc  DNAME xxxxxxxxx.1320.pmtu6.rootcanary.net.


1300-plus0  DNAME xxxxxxx.1300.pmtu6.rootcanary.net.
1300-plus2  DNAME xxxxxxxx.1300.pmtu6.rootcanary.net.
...
1300-plusa  DNAME xxxxxxxxxxxx.1300.pmtu6.rootcanary.net.
1300-plusc  DNAME xxxxxxxxxxxxx.1300.pmtu6.rootcanary.net.


1280-plus0  DNAME xxxxxxxxxxx.1280.pmtu6.rootcanary.net.
1280-plus2  DNAME xxxxxxxxxxxx.1280.pmtu6.rootcanary.net.
...
1280-plusa  DNAME xxxxxxxxxxxxxxxx.1280.pmtu6.rootcanary.net.
1280-plusc  DNAME xxxxxxxxxxxxxxxxx.1280.pmtu6.rootcanary.net.


1268-plus0  DNAME xxxxx.1268.pmtu6.rootcanary.net.
1268-plus2  DNAME xxxxxx.1268.pmtu6.rootcanary.net.
...
1268-plusa  DNAME xxxxxxxxxx.1268.pmtu6.rootcanary.net.
1268-plusc  DNAME xxxxxxxxxxx.1268.pmtu6.rootcanary.net.


1232-plus0  DNAME x.1232.pmtu6.rootcanary.net.
1232-plus2  DNAME xx.1232.pmtu6.rootcanary.net.
...
1232-plusa  DNAME xxxxxx.1232.pmtu6.rootcanary.net.
1232-plusc  DNAME xxxxxxx.1232.pmtu6.rootcanary.net.


1220-plus0  DNAME xxxxxxxxx.1220.pmtu6.rootcanary.net.
1220-plus2  DNAME xxxxxxxxxx.1220.pmtu6.rootcanary.net.
...
1220-plusa  DNAME xxxxxxxxxxxxxx.1220.pmtu6.rootcanary.net.
1220-plusc  DNAME xxxxxxxxxxxxxxx.1220.pmtu6.rootcanary.net.

;
; Wildcards hack
;

*.1500          AAAA    2001:db8::1:1500
                AAAA    2001:db8::2:1500
                ...
                AAAA    2001:db8::43:1500
                AAAA    2001:db8::44:1500


*.1492          AAAA    2001:db8::1:1492
                AAAA    2001:db8::2:1492
                ...
                AAAA    2001:db8::43:1492
                AAAA    2001:db8::44:1492


*.1460          AAAA    2001:db8::1:1460
                AAAA    2001:db8::2:1460
                ...
                AAAA    2001:db8::42:1460
                AAAA    2001:db8::43:1460
```

```
*.1448          AAAA    2001:db8::1:1448
                AAAA    2001:db8::2:1448
                ...
                AAAA    2001:db8::41:1448
                AAAA    2001:db8::42:1448


*.1412          AAAA    2001:db8::1:1412
                AAAA    2001:db8::2:1412
                ...
                AAAA    2001:db8::40:1412
                AAAA    2001:db8::41:1412


*.1400          AAAA    2001:db8::1:1400
                AAAA    2001:db8::2:1400
                ...
                AAAA    2001:db8::40:1400
                AAAA    2001:db8::41:1400


*.1380          AAAA    2001:db8::1:1380
                AAAA    2001:db8::2:1380
                ...
                AAAA    2001:db8::39:1380
                AAAA    2001:db8::40:1380


*.1360          AAAA    2001:db8::1:1360
                AAAA    2001:db8::2:1360
                ...
                AAAA    2001:db8::38:1360
                AAAA    2001:db8::39:1360


*.1340          AAAA    2001:db8::1:1340
                AAAA    2001:db8::2:1340
                ...
                AAAA    2001:db8::37:1340
                AAAA    2001:db8::38:1340


*.1320          AAAA    2001:db8::1:1320
                AAAA    2001:db8::2:1320
                ...
                AAAA    2001:db8::37:1320
                AAAA    2001:db8::38:1320


*.1300          AAAA    2001:db8::1:1300
                AAAA    2001:db8::2:1300
                ...
                AAAA    2001:db8::36:1300
                AAAA    2001:db8::37:1300


*.1280          AAAA    2001:db8::1:1280
                AAAA    2001:db8::2:1280
                ...
                AAAA    2001:db8::35:1280
                AAAA    2001:db8::36:1280


*.1268          AAAA    2001:db8::1:1268
                AAAA    2001:db8::2:1268
                ...
                AAAA    2001:db8::35:1268
                AAAA    2001:db8::36:1268


*.1232          AAAA    2001:db8::1:1232
                AAAA    2001:db8::2:1232
```

```
                        ...
                        AAAA    2001:db8::34:1232
                        AAAA    2001:db8::35:1232


*.1220                  AAAA    2001:db8::1:1220
                        AAAA    2001:db8::2:1220
                        ...
                        AAAA    2001:db8::33:1220
                        AAAA    2001:db8::34:1220
```