Updates: RFC 1034

Negative Caching of DNS Queries (DNS NCACHE)

Status of This Memo

Abstract

[RFC1034] provided a description of how to cache negative responses. It however had a fundamental flaw in that it did not allow a name server to hand out those cached responses to other resolvers, thereby greatly reducing the effect of the caching. This document addresses issues raise in the light of experience and replaces [RFC1034 Section 4.3.4].

Negative caching was an optional part of the DNS specification and deals with the caching of the non-existence of an RRset [RFC2181] or domain name.

Negative caching is useful as it reduces the response time for negative answers. It also reduces the number of messages that have to be sent between resolvers and name servers hence overall

network traffic. A large proportion of DNS traffic on the Internet could be eliminated if all resolvers implemented negative caching. With this in mind negative caching should no longer be seen as an optional part of a DNS resolver.

1 - Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

Negative caching is the storage of knowledge that something does not exist. For example we can store the knowledge that a record has a particular value. We can also do the reverse, that is, to store the knowledge that a record does not exist. It is the storage of knowledge that something does not exist, cannot or does not give an answer that we call negative caching.

"QNAME" refers to the name in the query section of an answer, or where this resolves to a CNAME, or CNAME chain, the data field of the last CNAME. The last CNAME in this sense is that which contains a value which does not resolve to another CNAME. Implementations should note that including CNAME records in responses in order, so that the first has the label from the query section, and then each in sequence has the label from the data section of the previous (where more than one CNAME is needed) allows the sequence to be processed in one pass, and considerably eases the task of the receiver. Other relevant records (such as SIG RRs) can be interspersed amongst the CNAMEs.

"NXDOMAIN" is as alternate expression for the "Name Error" RCODE as described in [RFC1035 Section 4.1.1] and the two terms are used interchangeably in this document.

"NODATA" is a pseudo RCODE which indicates that the name is valid, for the given class, but are no records of the given type. A NODATA response has to be inferred from the answer.

"FORWARDER" is a nameserver that is used by another nameserver to resolve queries that it cannot answer from its cache in preference to those that would be queried as the result of looking at the records in the DNS. FORWARDERS are often used to reduce the number of queries sent across wide area networks or to provide DNS services through firewalls.

An understanding of [RFC1034], [RFC1035] and [RFC2065] is expected when reading this document.

2 - Negative Responses

The most common negative responses indicate that a particular RRset does not exist in the DNS. The first sections of this document deal with these case. Other negative responses can indicate failures of a nameserver, those are dealt with in section 8 (Other Negative Responses).

A negative response is indicated by one of the following conditions:

2.1 - Name Error

Name errors (NXDOMAIN) are indicated by the presence of "Name Error" in the RCODE field. In this case the domain referred to by the QNAME does not exist. Note: the answer section may have SIG and CNAME RRs and authority section may have SOA, NXT and SIG RRsets.

It is possible to distinguish between a referral and a NXDOMAIN response by the presense of NXDOMAIN in the RCODE regardless of the presence of NS or SOA records in the authority section.

```
NXDOMAIN RESPONSE: TYPE 1.

Header:
RDCODE=NXDOMAIN
Query:
AN.EXAMPLE. A
Answer:
AN.EXAMPLE. CNAME TRIPPLE.XX.
Authority:
XX. SOA NS1.XX. HOSTMASTER.NS1.XX. ....
XX. NS NS1.XX.
XX. NS NS2.XX.
Additional:
NS1.XX. A 127.0.0.2
NS2.XX. A 127.0.0.3
```

```
NXDOMAIN RESPONSE: TYPE 2.

Header:
RDCODE=NXDOMAIN
Query:
AN.EXAMPLE. A
Answer:
AN.EXAMPLE. CNAME TRIPPLE.XX.
Authority:
XX. SOA NS1.XX. HOSTMASTER.NS1.XX. ....
Additional:
<empty>


NXDOMAIN RESPONSE: TYPE 3.

Header:
RDCODE=NXDOMAIN
Query:
AN.EXAMPLE. A
Answer:
AN.EXAMPLE. CNAME TRIPPLE.XX.
Authority:
<empty>
Additional:
<empty>


REFERRAL RESPONSE.

Header:
RDCODE=NOERROR
Query:
AN.EXAMPLE. A
Answer:
AN.EXAMPLE. CNAME TRIPPLE.XX.
Authority:
XX. NS NS1.XX.
XX. NS NS2.XX.
Additional:
NS1.XX. A 127.0.0.2
NS2.XX. A 127.0.0.3
```

Note, in the three examples of NXDOMAIN responses, it is known that the name "AN.EXAMPLE." exists, and has as its value a CNAME record. The NXDOMAIN refers to "TRIPPLE.XX", which is then known not to exist. On the other hand, in the referral example, it is shown that "AN.EXAMPLE" exists, and has a CNAME RR as its value, but nothing is known one way or the other about the existence of "TRIPPLE.XX", other than that "NS1.XX" or "NS2.XX" can be consulted as the next step in obtaining information about it.

Where no CNAME records appear, the NXDOMAIN response refers to the name in the label of the RR in the question section.

2.1.1 Special Handling of Name Error

There are a large number of resolvers currently in existence that fail to correctly detect and process all forms of NXDOMAIN response. Some resolvers treat a TYPE 1 NXDOMAIN response as a referral. To alleviate this problem it is recommended that servers that are authoritative for the NXDOMAIN response only send TYPE 2 NXDOMAIN responses, that is the authority section contains a SOA record and no NS records. If a non-authoritative server sends a type 1 NXDOMAIN response to one of these old resolvers, the result will be an unnecessary query to an authoritative server. This is undesirable, but not fatal. If a server is listed as a FORWARDER for another resolver it may be necessary to disable sending TYPE 1 NXDOMAIN response for non-authoritative NXDOMAIN responses.

Some resolvers incorrectly continue processing if the authoritative answer flag is not set. This is a problem when your nameserver is listed as a FORWARDER for these resolvers. It is sometimes necessary to force the authority flag on for NXDOMAIN responses when you have such a resolver.

2.2 - No Data

NODATA responses have to be algorithmically determined from the response's contents as there is no RCODE value to indicate NODATA. In some cases to determine with certainty that NODATA is the correct response it can be necessary to send another query.

NODATA is indicated by an answer with a RCODE of NOERROR and no relevant answers in the answer section and either an SOA record in the authority section or no NS records in the authority section. The authority section may contain NXT and SIG RRsets in addition to NS and SOA records. CNAMEs and SIG records may exist in the answer section.

It is possible to distinguish between a referral and a NODATA response by the presence of a SOA record in the authority section or the absence of NS records in the authority section.

```
NODATA RESPONSE: TYPE 1.

Header:
RDCODE=NOERROR
Query:
ANOTHER.EXAMPLE. A
Answer:
<empty>
Authority:
EXAMPLE. SOA NS1.XX. HOSTMASTER.NS1.XX. ....
EXAMPLE. NS NS1.XX.
EXAMPLE. NS NS2.XX.
Additional:
NS1.XX. A 127.0.0.2
NS2.XX. A 127.0.0.3


NO DATA RESPONSE: TYPE 2.

Header:
RDCODE=NOERROR
Query:
ANOTHER.EXAMPLE. A
Answer:
<empty>
Authority:
EXAMPLE. SOA NS1.XX. HOSTMASTER.NS1.XX. ....
Additional:
<empty>
```

```
NO DATA RESPONSE: TYPE 3.

Header:
RDCODE=NOERROR
Query:
ANOTHER.EXAMPLE. A
Answer:
<empty>
Authority:
<empty>
Additional:
<empty>


REFERRAL RESPONSE.

Header:
RDCODE=NOERROR
Query:
ANOTHER.EXAMPLE. A
Answer:
<empty>
Authority:
EXAMPLE. NS NS1.XX.
EXAMPLE. NS NS2.XX.
Additional:
NS1.XX. A 127.0.0.2
NS2.XX. A 127.0.0.3
```

These examples, unlike the NXDOMAIN examples above, have no CNAME records, however they could, in just the same way that the NXDOMAIN examples did, in which case it would be the value of the last CNAME (the QNAME) for which NODATA would be concluded.

2.2.1 - Special Handling of No Data

There are a large number of resolvers currently in existence that fail to correctly detect and process all forms of NODATA response. Some resolvers treat a TYPE 1 NODATA response as a referral. To alleviate this problem it is recommended that servers that are authoritative for the NODATA response only send TYPE 2 NODATA responses, that is the authority section contains a SOA record and no NS records. Sending a TYPE 1 NODATA response from a non-authoritative server to one of these resolvers will only result in an unnecessary query. If a server is listed as a FORWARDER for another resolver it may also be necessary to disable the sending of TYPE 1 NODATA response for non-authoritative NODATA responses.

Some name servers fail to set the RCODE to NXDOMAIN in the presence of CNAMEs in the answer section. If a definitive NXDOMAIN / NODATA answer is required the resolver must query again with QNAME.

3 - Negative Answers from Authoritative Servers

Authoritative name servers MUST add the SOA record from the zone in which a name was not located, or in which the name was found but no data of the requested type, to the authority section of the answer containing a negative response soa that the response can be cached. The TTL of this record is set from the MINIMUM field of the SOA record, not the TTL of the SOA itself, and indicates how long a resolver may cache this negative answer.

If the containing zone is signed [RFC2065] the SOA and appropriate NXT and SIG records MUST be added.

4 - SOA Minimum Field

The SOA minimum field has been overloaded in the past to have three different meanings, the minimum TTL value of all RRs in a zone, the default TTL of RRs which did not contain a TTL value and the TTL of negative responses.

The first of these, the minimum TTL value of all RRs in a zone, has never in practice been used and should now be ignored.

The second, the default TTL of RRs which did not contain a TTL value, is not preserved across zone transfers where each record has a TTL. In fact it is impossible to determine whether the TTL for a record was explicitly set or derived from the default after a zone transfer. Servers MUST provide a mechanism to set the default TTL independent of the MINIMUM field. How this is done in implementation dependent.

The [RFC 1035 Section 5] "MASTER FILES" format format is extended to include the following directive which specifies the default TTL of RRs, which appear after the directive, in the zone file which do not have a TTL of their own.

```
        $TTL <TTL> [comment]
```

This only leaves the third case, TTL of negative responses, as the final and only use for the SOA minimum field.

5 - Caching Negative Answers

Like normal answers negative answers have a time to live (TTL). As there is no record in the answer section to which this TTL can be applied, the TTL must be carried by another method. This is done by using the SOA record from the containing zone and putting it in the authority section with an initial TTL set from the SOA minimum field. This TTL decrements in a similar manner to a normal cached answer and upon reaching zero (0) indicates the negative answer MUST be discarded.

Early advertisement of a service before all the secondaries have a copy of the relevant zone can lead to prolonged denials of service. With this in mind a resolver SHOULD set an upper bound on the TTL of the negative answer it is willing to cache. If it does this the TTL MUST be set to the minimum of the resolver upper bound and the received TTL. A resolver upper bound of one (1) hour is often appropriate.

A negative answer that resulted from a name error (NXDOMAIN) should be cached such that it can be retrieved and returned in response to another query for the same <QNAME, QCLASS> that resulted in the cached negative response.

A negative answer that resulted from a no data error (NODATA) should be cached such that it can be retrieved and returned in response to another query for the same <QNAME, QTYPE, QCLASS> that resulted in the cached negative response.

The NXT record, if it exists in the authority section, MUST be stored such that it can be be located and returned with SOA record in the authority section as should any SIG records in the authority section. For NXDOMAIN answers there is no "necessary" obvious relationship between the NXT records and the QNAME. The NXT record MUST have the same owner name as the query name for NODATA responses.

Negative responses without SOA records SHOULD NOT be cached as there is no way to prevent the negative responses looping forever between a pair of servers even with a short TTL.

6 - Negative answers from the cache

When a server, in answering a query, encounters a cached negative response it MUST add the cached SOA record to the authority section of the response with the TTL decremented by the amount of time it was stored in the cache. This allows the NXDOMAIN / NODATA response to time out correctly.

If a NXT record was cached along with SOA record it MUST be added to the authority section. If a SIG record was cached along with a NXT record it SHOULD be added to the authority section.

7 - Changes from RFC 1034

Negative caching in resolvers is no-longer optional, if a resolver caches anything it MUST also cache negative answers.

Non-authoritative negative answers MAY be cached.

The SOA record from the authority section MUST be cached. Name error indications MUST be cached against the tuple <query name, QCLASS>. No data indications MUST be cached against <query name, QTYPE, QCLASS> tuple.

A cached SOA record MUST be added to the response. This was explicitly not allowed because previously the distinction between a normal cached SOA record, and the SOA cached as a result of a negative response was not made, and simply extracting a normal cached SOA and adding that to a cached negative response causes problems.

Added $TTL directive to master file format.

8 - Other Negative Responses

Caching of other negative responses is not covered by any existing RFC. There is no way to indicate a desired TTL in these responses. Care needs to be taken to ensure that there are not forwarding loops.

8.1 Server Failure (OPTIONAL)

Server failures fall into two major classes. The first is where a server can determine that it has been misconfigured for a zone. This may be where it has been listed as a server, but not configured to be a server for the zone, or where it has been configured to be a server for the zone, but cannot obtain the zone data for some reason, either because the zone file does not exist or contains errors, or because another server from which the zone should have been available either did not respond or was unable or unwilling to supply the zone.

The second class is where the server needs to obtain an answer from elsewhere, but is unable to do so, due to network failures, other servers that don't reply, or return server failure errors, or similar.

In either case a resolver MAY cache a server failure response. If it does so it MUST NOT cache it for longer that five (5) minutes, and it MUST be cached against the specific query tuple <query name, type, class, server IP address>.

8.2 Dead / Unreachable Server (OPTIONAL)

Dead / Unreachable servers are servers that fail to respond in any way to a query or where the transport layer has provided an indication that the server does not exist or is unreachable. A server is deemed to be dead or unreachable if it has not responded to an outstanding query within 120 seconds.

Examples of transport layer indications are:

ICMP error messages indicating host, net or port unreachable.
TCP resets
IP stack error messages providing similar indications to those above.

A server MAY cache a dead server indication. If it does so it MUST NOT be deemed dead for longer than five (5) minutes. The indication MUST be stored against query tuple <query name, type, class, server IP address> unless there was a transport layer indication that the server does not exist, in which case it applies to all queries to that specific IP address.

9 History of Negative Caching

The following is a potted history of negative caching in the DNS and forms no part of the technical specification of negative caching.

It is interesting to note that the same concepts were re-invented in both the CHIVES and BIND servers.

The history of the early CHIVES work (Section 8.1) was supplied by Rob Austein <sra@epilogue.com> and is essentially untouch from what he sent me [MPA].

Sometime around the spring of 1985, I mentioned to Paul Mockapetris that our experience with his JEEVES DNS resolver had pointed out the need for some kind of negative caching scheme. Paul suggested that we simply cache authoritative errors, using the SOA MINIMUM value for the zone that would have contained the target RRs. I'm pretty sure that this conversation took place before RFC-973 was written, but it was never clear to me whether this idea was something that Paul came up with on the spot in response to my question or something he'd already been planning to put into the document that became RFC-973. In any case, neither of us was entirely sure that the SOA MINIMUM value was really the right metric to use, but it was available and was under the control of the administrator of the target zone, both of which seemed to us at the time to be important feature.

Late in 1987, I released the initial beta-test version of CHIVES, the DNS resolver I'd written to replace Paul's JEEVES resolver. CHIVES included a search path mechanism that was used pretty heavily at several sites (including my own), so CHIVES also included a negative caching mechanism based on SOA MINIMUM values. The basic strategy was to cache authoritative error codes keyed by the exact query parameters (QNAME, QCLASS, and QTYPE), with a cache TTL equal to the SOA MINIMUM value. CHIVES did not attempt to track down SOA RRs if they weren't supplied in the authoritative response, so it never managed to completely eliminate the gratuitous DNS error message traffic, but it did help considerably. Keep in mind that this was happening at about the same time as the near-collapse of the ARPANET due to congestion caused by exponential growth and the the "old" (pre-VJ) TCP retransmission algorithm, so negative caching resulted in drastically better DNS response time for our users, mailer daemons, etcetera.

As far as I know, CHIVES was the first resolver to implement negative caching. CHIVES was developed during the twilight years of TOPS-20, so it never ran on very many machines, but the few machines that it did run on were the ones that were too critical to shut down quickly no matter how much it cost to keep them running. So what few users we did have tended to drive CHIVES pretty hard. Several interesting bits of DNS technology resulted from that, but the one that's relevant here is the MAXTTL configuration parameter.

Experience with JEEVES had already shown that RRs often showed up with ridiculously long TTLs (99999999 was particularly popular for many years, due to bugs in the code and documentation of several early versions of BIND), and that robust software that blindly believed such TTLs could create so many strange failures that it was often necessary to reboot the resolver frequently just to clear this garbage out of the

cache. So CHIVES had a configuration parameter "MAXTTL", which specified the maximum "reasonable" TTL in a received RR. RRs with TTLs greater than MAXTTL would either have their TTLs reduced to MAXTTL or would be discarded entirely, depending on the setting of another configuration parameter.

When we started getting field experience with CHIVES's negative caching code, it became clear that the SOA MINIMUM value was often large enough to cause the same kinds of problems for negative caching as the huge TTLs in RRs had for normal caching (again, this was in part due to a bug in several early versions of BIND, where a secondary server would authoritatively deny all knowledge of its zones if it couldn't contact the primaries on reboot). So we started running the negative cache TTLs through the MAXTTL check too, and continued to experiment.

The configuration that seemed to work best on WSMR-SIMTEL20.ARMY.MIL (last of the major Internet TOPS-20 machines to be shut down, thus the last major user of CHIVES, thus the place where we had the longest experimental baseline) was to set MAXTTL to about three days. Most of the traffic initiated by SIMTEL20 in its last years was mail-related, and the mail queue timeout was set to one week, so this gave a "stuck" message several tries at complete DNS resolution, without bogging down the system with a lot of useless queries. Since (for reasons that now escape me) we only had the single MAXTTL parameter rather than separate ones for positive and negative caching, it's not clear how much effect this setting of MAXTTL had on the negative caching code.

CHIVES also included a second, somewhat controversial mechanism which took the place of negative caching in some cases. The CHIVES resolver daemon could be configured to load DNS master files, giving it the ability to act as what today would be called a "stealth secondary". That is, when configured in this way, the resolver had direct access to authoritative information for heavily-used zones. The search path mechanisms in CHIVES reflected this: there were actually two separate search paths, one of which only searched local authoritative zone data, and one which could generate normal iterative queries. This cut down on the need for negative caching in cases where usage was predictably heavy (e.g., the resolver on XX.LCS.MIT.EDU always loaded the zone files for both LCS.MIT.EDU and AI.MIT.EDU and put both of these suffixes into the "local" search path, since between them the hosts in these two zones accounted for the bulk of the DNS traffic). Not all sites running CHIVES chose to use this feature; C.CS.CMU.EDU, for example, chose to use the "remote" search path for everything because there were too many different sub-zones at CMU for zone shadowing to be practical for them, so they relied pretty heavily on negative caching even for local

traffic.

Overall, I still think the basic design we used for negative caching was pretty reasonable: the zone administrator specified how long to cache negative answers, and the resolver configuration chose the actual cache time from the range between zero and the period specified by the zone administrator. There are a lot of details I'd do differently now (like using a new SOA field instead of overloading the MINIMUM field), but after more than a decade, I'd be more worried if we couldn't think of at least a few improvements.

9.2 BIND

While not the first attempt to get negative caching into BIND, in July 1993, BIND 4.9.2 ALPHA, Anant Kumar of ISI supplied code that implemented, validation and negative caching (NCACHE). This code had a 10 minute TTL for negative caching and only cached the indication that there was a negative response, NXDOMAIN or NOERROR_NODATA. This is the origin of the NODATA pseudo response code mentioned below.

NCACHE made default XXXX. xxxx 199?.

Mark Andrews of CSIRO added code (RETURNSOA) that stored the SOA record such that it could be retrieved by a similar query. UUnet complained that they were getting old answers after loading a new zone, and the option was turned off, BIND 4.9.3-alpha5, XXXX 199?. In reality this indicated that the named needed to purge the space the zone would occupy. Functionality to do this was added in BIND 4.9.3 BETA11 patch2, XXXX 199?.

RETURNSOA was re-enabled by default, BIND 4.9.5-T1A, XXXX 199?.

10 Security Considerations

We believed that this document does not introduce any significant additional security threats.

With negative caching it might be possible to propagate a denial of service attack by spreading a NXDOMAIN message with a very high TTL. Without negative caching that would be much harder. A similar effect could be achieved previously by spreading a bad A record, so that the server could not be reached - which is almost the same but not quite. It has the same effect as far as what the end user is able to do, but with a different psychological effect. With the bad A, I feel "damn the network is broken again" and try again tomorrow. With the "NXDOMAIN" I feel "Oh, they've turned off the server and it doesn't exist any more" and probably never bother trying this server again.

For such an attack to be successful you would need to get the NXDOMAIN indiction injected into a parent server (or a busy caching resolver). This can only be done by the use of a CNAME which results in the parent server querying an attackers server.
 Resolvers that are wish to prevent such attacks can query again the final QNAME ignoring any NS data in the query responses it has received for this query.

Implementing TTL sanity checking will reduce the effectiveness of such an attack, because a successful attack would require re-injection of the bogus data at more frequent intervals.

DNS Security [RFC2065] provides a mechanism to verify whether a negative response is valid or not, through the use of NXT and SIG records. This document supports the use of that mechanism by promoting the transmission of the relevant security records even in a non security aware server.

References

[RFC1034]
P. Mockapetris, "DOMAIN NAMES - CONCEPTS AND FACILITIES," STD 13, RFC 1034, November 1987.

[RFC1035]P. Mockapetris, "DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION," STD 13, RFC 1035, November 1987.

[RCF2065]
D. Eastlake, 3rd, C. Kaufman, "Domain Name System Security Extensions," RFC 2065, January 1997

[RFC2119]
S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, March 1997

[RFC2181]
R. Elz, R. Bush, "Clarifications to the DNS Specification," RFC 2181, July 1997.

Author's Address

Mark Andrews
CSIRO - Mathematical and Information Sciences
Locked Bag 17
North Ryde NSW 2113
AUSTRALIA
+61 2 9325 3148
<Mark.Andrews@cmis.csiro.au>