Network Working Group
Request for Comments: 1536
Category: Informational

A. Kumar
J. Postel
C. Neuman
ISI
P. Danzig
S. Miller
USC

October 1993

Common DNS Implementation Errors and Suggested Fixes

Status of this Memo

This memo provides information for the Internet community. It does not specify an Internet standard. Distribution of this memo is unlimited.

Abstract

This memo describes common errors seen in DNS implementations and suggests some fixes. Where applicable, violations of recommendations from STD 13, RFC 1034 and STD 13, RFC 1035 are mentioned. The memo also describes, where relevant, the algorithms followed in BIND (versions 4.8.3 and 4.9 which the authors referred to) to serve as an example.

Introduction

The last few years have seen, virtually, an explosion of DNS traffic on the NSFnet backbone. Various DNS implementations and various versions of these implementations interact with each other, producing huge amounts of unnecessary traffic. Attempts are being made by researchers all over the internet, to document the nature of these interactions, the symptomatic traffic patterns and to devise remedies for the sick pieces of software.

This draft is an attempt to document fixes for known DNS problems so people know what problems to watch out for and how to repair broken software.

1. Fast Retransmissions

DNS implements the classic request-response scheme of client-server interaction. UDP is, therefore, the chosen protocol for communication though TCP is used for zone transfers. The onus of requerying in case no response is seen in a "reasonable" period of time, lies with the client. Although RFC 1034 and 1035 do not recommend any

retransmission policy, RFC 1035 does recommend that the resolvers should cycle through a list of servers. Both name servers and stub resolvers should, therefore, implement some kind of a retransmission policy based on round trip time estimates of the name servers. The client should back-off exponentially, probably to a maximum timeout value.

However, clients might not implement either of the two. They might not wait a sufficient amount of time before retransmitting or they might not back-off their inter-query times sufficiently.

Thus, what the server would see will be a series of queries from the same querying entity, spaced very close together. Of course, a correctly implemented server discards all duplicate queries but the queries contribute to wide-area traffic, nevertheless.

We classify a retransmission of a query as a pure Fast retry timeout problem when a series of query packets meet the following conditions.

a. Query packets are seen within a time less than a "reasonable waiting period" of each other.

b. No response to the original query was seen i.e., we see two or more queries, back to back.

c. The query packets share the same query identifier.

d. The server eventually responds to the query.

A GOOD IMPLEMENTATION:

BIND (we looked at versions 4.8.3 and 4.9) implements a good retransmission algorithm which solves or limits all of these problems. The Berkeley stub-resolver queries servers at an interval that starts at the greater of 4 seconds and 5 seconds divided by the number of servers the resolver queries. The resolver cycles through servers and at the end of a cycle, backs off the time out exponentially.

The Berkeley full-service resolver (built in with the program "named") starts with a time-out equal to the greater of 4 seconds and two times the round-trip time estimate of the server. The time-out is backed off with each cycle, exponentially, to a ceiling value of 45 seconds.

FIXES:

a. Estimate round-trip times or set a reasonably high initial time-out.

b. Back-off timeout periods exponentially.

c. Yet another fundamental though difficult fix is to send the client an acknowledgement of a query, with a round-trip time estimate.

Since UDP is used, no response is expected by the client until the query is complete. Thus, it is less likely to have information about previous packets on which to estimate its back-off time. Unless, you maintain state across queries, so subsequent queries to the same server use information from previous queries. Unfortunately, such estimates are likely to be inaccurate for chained requests since the variance is likely to be high.

The fix chosen in the ARDP library used by Prospero is that the server will send an initial acknowledgement to the client in those cases where the server expects the query to take a long time (as might be the case for chained queries). This initial acknowledgement can include an expected time to wait before retrying.

This fix is more difficult since it requires that the client software also be trained to expect the acknowledgement packet. This, in an internet of millions of hosts is at best a hard problem.

2. Recursion Bugs

When a server receives a client request, it first looks up its zone data and the cache to check if the query can be answered. If the answer is unavailable in either place, the server seeks names of servers that are more likely to have the information, in its cache or zone data. It then does one of two things. If the client desires the server to recurse and the server architecture allows recursion, the server chains this request to these known servers closest to the queried name. If the client doesn't seek recursion or if the server cannot handle recursion, it returns the list of name servers to the client assuming the client knows what to do with these records.

The client queries this new list of name servers to get either the answer, or names of another set of name servers to query. This process repeats until the client is satisfied. Servers might also go through this chaining process if the server returns a CNAME record for the queried name. Some servers reprocess this name to try and get the desired record type.

However, in certain cases, this chain of events may not be good. For example, a broken or malicious name server might list itself as one of the name servers to query again. The unsuspecting client resends the same query to the same server.

In another situation, more difficult to detect, a set of servers might form a loop wherein A refers to B and B refers to A. This loop might involve more than two servers.

Yet another error is where the client does not know how to process the list of name servers returned, and requeries the same server since that is one (of the few) servers it knows.

We, therefore, classify recursion bugs into three distinct categories:

a. Ignored referral: Client did not know how to handle NS records in the AUTHORITY section.

b. Too many referrals: Client called on a server too many times, beyond a "reasonable" number, with same query. This is different from a Fast retransmission problem and a Server Failure detection problem in that a response is seen for every query. Also, the identifiers are always different. It implies client is in a loop and should have detected that and broken it. (RFC 1035 mentions that client should not recurse beyond a certain depth.)

c. Malicious Server: a server refers to itself in the authority section. If a server does not have an answer now, it is very unlikely it will be any better the next time you query it, specially when it claims to be authoritative over a domain.

RFC 1034 warns against such situations, on page 35.

"Bound the amount of work (packets sent, parallel processes started) so that a request can't get into an infinite loop or start off a chain reaction of requests or queries with other implementations EVEN IF SOMEONE HAS INCORRECTLY CONFIGURED SOME DATA."

A GOOD IMPLEMENTATION:

BIND fixes at least one of these problems. It places an upper limit on the number of recursive queries it will make, to answer a question. It chases a maximum of 20 referral links and 8 canonical name translations.

FIXES:

a. Set an upper limit on the number of referral links and CNAME links you are willing to chase.

   Note that this is not guaranteed to break only recursion loops. It could, in a rare case, prune off a very long search path, prematurely. We know, however, with high probability, that if the number of links cross a certain metric (two times the depth of the DNS tree), it is a recursion problem.

b. Watch out for self-referring servers. Avoid them whenever possible.

c. Make sure you never pass off an authority NS record with your own name on it!

d. Fix clients to accept iterative answers from servers not built to provide recursion. Such clients should either be happy with the non-authoritative answer or be willing to chase the referral links themselves.

3. Zero Answer Bugs:

   Name servers sometimes return an authoritative NOERROR with no ANSWER, AUTHORITY or ADDITIONAL records. This happens when the queried name is valid but it does not have a record of the desired type. Of course, the server has authority over the domain.

   However, once again, some implementations of resolvers do not interpret this kind of a response reasonably. They always expect an answer record when they see an authoritative NOERROR. These entities continue to resend their queries, possibly endlessly.

A GOOD IMPLEMENTATION

   BIND resolver code does not query a server more than 3 times. If it is unable to get an answer from 4 servers, querying them three times each, it returns error.

   Of course, it treats a zero-answer response the way it should be treated; with respect!

FIXES:

a. Set an upper limit on the number of retransmissions for a given query, at the very least.

b. Fix resolvers to interpret such a response as an authoritative statement of non-existence of the record type for the given name.

4. Inability to detect server failure:

Servers in the internet are not very reliable (they go down every once in a while) and resolvers are expected to adapt to the changed scenario by not querying the server for a while. Thus, when a server does not respond to a query, resolvers should try another server. Also, non-stub resolvers should update their round trip time estimate for the server to a large value so that server is not tried again before other, faster servers.

Stub resolvers, however, cycle through a fixed set of servers and if, unfortunately, a server is down while others do not respond for other reasons (high load, recursive resolution of query is taking more time than the resolver's time-out, ...), the resolver queries the dead server again! In fact, some resolvers might not set an upper limit on the number of query retransmissions they will send and continue to query dead servers indefinitely.

Name servers running system or chained queries might also suffer from the same problem. They store names of servers they should query for a given domain. They cycle through these names and in case none of them answers, hit each one more than one. It is, once again, important that there be an upper limit on the number of retransmissions, to prevent network overload.

This behavior is clearly in violation of the dictum in RFC 1035 (page 46)

> "If a resolver gets a server error or other bizarre response from a name server, it should remove it from SLIST, and may wish to schedule an immediate transmission to the next candidate server address."

Removal from SLIST implies that the server is not queried again for some time.

Correctly implemented full-service resolvers should, as pointed out before, update round trip time values for servers that do not respond and query them only after other, good servers. Full-service resolvers might, however, not follow any of these common sense directives. They query dead servers, and they query them endlessly.

A GOOD IMPLEMENTATION:

BIND places an upper limit on the number of times it queries a server. Both the stub-resolver and the full-service resolver code do this. Also, since the full-service resolver estimates round-trip times and sorts name server addresses by these estimates, it does not query a dead server again, until and unless all the other servers in the list are dead too! Further, BIND implements exponential back-off too.

FIXES:

a. Set an upper limit on number of retransmissions.

b. Measure round-trip time from servers (some estimate is better than none). Treat no response as a "very large" round-trip time.

c. Maintain a weighted rtt estimate and decay the "large" value slowly, with time, so that the server is eventually tested again, but not after an indefinitely long period.

d. Follow an exponential back-off scheme so that even if you do not restrict the number of queries, you do not overload the net excessively.

5. Cache Leaks:

Every resource record returned by a server is cached for TTL seconds, where the TTL value is returned with the RR. Full-service (or stub) resolvers cache the RR and answer any queries based on this cached information, in the future, until the TTL expires. After that, one more query to the wide-area network gets the RR in cache again.

Full-service resolvers might not implement this caching mechanism well. They might impose a limit on the cache size or might not interpret the TTL value correctly. In either case, queries repeated within a TTL period of a RR constitute a cache leak.

A GOOD/BAD IMPLEMENTATION:

BIND has no restriction on the cache size and the size is governed by the limits on the virtual address space of the machine it is running on. BIND caches RRs for the duration of the TTL returned with each record.

It does, however, not follow the RFCs with respect to interpretation of a 0 TTL value. If a record has a TTL value of 0 seconds, BIND uses

the minimum TTL value, for that zone, from the SOA record and caches it for that duration. This, though it saves some traffic on the wide-area network, is not correct behavior.

FIXES:

a. Look over your caching mechanism to ensure TTLs are interpreted correctly.

b. Do not restrict cache sizes (come on, memory is cheap!). Expired entries are reclaimed periodically, anyway. Of course, the cache size is bound to have some physical limit. But, when possible, this limit should be large (run your name server on a machine with a large amount of physical memory).

c. Possibly, a mechanism is needed to flush the cache, when it is known or even suspected that the information has changed.

6. Name Error Bugs:

This bug is very similar to the Zero Answer bug. A server returns an authoritative NXDOMAIN when the queried name is known to be bad, by the server authoritative for the domain, in the absence of negative caching. This authoritative NXDOMAIN response is usually accompanied by the SOA record for the domain, in the authority section.

Resolvers should recognize that the name they queried for was a bad name and should stop querying further.

Some resolvers might, however, not interpret this correctly and continue to query servers, expecting an answer record.

Some applications, in fact, prompt NXDOMAIN answers! When given a perfectly good name to resolve, they append the local domain to it e.g., an application in the domain "foo.bar.com", when trying to resolve the name "usc.edu" first tries "usc.edu.foo.bar.com", then "usc.edu.bar.com" and finally the good name "usc.edu". This causes at least two queries that return NXDOMAIN, for every good query. The problem is aggravated since the negative answers from the previous queries are not cached. When the same name is sought again, the process repeats.

Some DNS resolver implementations suffer from this problem, too. They append successive sub-parts of the local domain using an implicit searchlist mechanism, when certain conditions are satisfied and try the original name, only when this first set of iterations fails. This behavior recently caused pandemonium in the Internet when the domain "edu.com" was registered and a wildcard "CNAME" record placed at the

top level. All machines from "com" domains trying to connect to hosts in the "edu" domain ended up with connections to the local machine in the "edu.com" domain!

GOOD/BAD IMPLEMENTATIONS:

Some local versions of BIND already implement negative caching. They typically cache negative answers with a very small TTL, sufficient to answer a burst of queries spaced close together, as is typically seen.

The next official public release of BIND (4.9.2) will have negative caching as an ifdef'd feature.

The BIND resolver appends local domain to the given name, when one of two conditions is met:

i. The name has no periods and the flag RES_DEFNAME is set.
ii. There is no trailing period and the flag RES_DNSRCH is set.

The flags RES_DEFNAME and RES_DNSRCH are default resolver options, in BIND, but can be changed at compile time.

Only if the name, so generated, returns an NXDOMAIN is the original name tried as a Fully Qualified Domain Name. And only if it contains at least one period.

FIXES:

a. Fix the resolver code.

b. Negative Caching. Negative caching servers will restrict the traffic seen on the wide-area network, even if not curb it altogether.

c. Applications and resolvers should not append the local domain to names they seek to resolve, as far as possible. Names interspersed with periods should be treated as Fully Qualified Domain Names.

   In other words, Use searchlists only when explicitly specified. No implicit searchlists should be used. A name that contains any dots should first be tried as a FQDN and if that fails, with the local domain name (or searchlist if specified) appended. A name containing no dots can be appended with the searchlist right away, but once again, no implicit searchlists should be used.

Associated with the name error bug is another problem where a server might return an authoritative NXDOMAIN, although the name is valid. A secondary server, on start-up, reads the zone information from the primary, through a zone transfer. While it is in the process of loading the zones, it does not have information about them, although it is authoritative for them. Thus, any query for a name in that domain is answered with an NXDOMAIN response code. This problem might not be disastrous were it not for negative caching servers that cache this answer and so propagate incorrect information over the internet.

BAD IMPLEMENTATION:

BIND apparently suffers from this problem.

Also, a new name added to the primary database will take a while to propagate to the secondaries. Until that time, they will return NXDOMAIN answers for a good name. Negative caching servers store this answer, too and aggravate this problem further. This is probably a more general DNS problem but is apparently more harmful in this situation.

FIX:

a. Servers should start answering only after loading all the zone data. A failed server is better than a server handing out incorrect information.

b. Negative cache records for a very small time, sufficient only to ward off a burst of requests for the same bad name. This could be related to the round-trip time of the server from which the negative answer was received. Alternatively, a statistical measure of the amount of time for which queries for such names are received could be used. Minimum TTL value from the SOA record is not advisable since they tend to be pretty large.

c. A "PUSH" (or, at least, a "NOTIFY") mechanism should be allowed and implemented, to allow the primary server to inform secondaries that the database has been modified since it last transferred zone data. To alleviate the problem of "too many zone transfers" that this might cause, Incremental Zone Transfers should also be part of DNS. Also, the primary should not NOTIFY/PUSH with every update but bunch a good number together.

7. Format Errors:

Some resolvers issue query packets that do not necessarily conform to standards as laid out in the relevant RFCs. This unnecessarily increases net traffic and wastes server time.

FIXES:

a. Fix resolvers.

b. Each resolver verify format of packets before sending them out, using a mechanism outside of the resolver. This is, obviously, needed only if step 1 cannot be followed.

References

[1] Mockapetris, P., "Domain Names Concepts and Facilities", STD 13, RFC 1034, USC/Information Sciences Institute, November 1987.

[2] Mockapetris, P., "Domain Names Implementation and Specification", STD 13, RFC 1035, USC/Information Sciences Institute, November 1987.

[3] Partridge, C., "Mail Routing and the Domain System", STD 14, RFC 974, CSNET CIC BBN, January 1986.

[4] Gavron, E., "A Security Problem and Proposed Correction With Widely Deployed DNS Software", RFC 1535, ACES Research Inc., October 1993.

[5] Beertema, P., "Common DNS Data File Configuration Errors", RFC 1537, CWI, October 1993.

Security Considerations

Security issues are not discussed in this memo.

Authors' Addresses

Anant Kumar
USC Information Sciences Institute
4676 Admiralty Way
Marina Del Rey CA 90292-6695

Phone:(310) 822-1511
FAX: (310) 823-6741
EMail: anant@isi.edu

Jon Postel
USC Information Sciences Institute
4676 Admiralty Way
Marina Del Rey CA 90292-6695

Phone:(310) 822-1511
FAX: (310) 823-6714
EMail: postel@isi.edu

Cliff Neuman
USC Information Sciences Institute
4676 Admiralty Way
Marina Del Rey CA 90292-6695

Phone:(310) 822-1511
FAX: (310) 823-6714
EMail: bcn@isi.edu

Peter Danzig
Computer Science Department
University of Southern California
University Park

EMail: danzig@caldera.usc.edu

Steve Miller
Computer Science Department
University of Southern California
University Park
Los Angeles CA 90089

EMail: smiller@caldera.usc.edu