May 2026
Geoff Huston

DNS Delegation

The DNS Working Group meets at every RIPE meeting. I found this DNS presentation from RIPE 92, held in May 2026, to be particularly interesting, and I'd like to explore this topic in a little more depth here.

A presentation, by ISC's Ondřej Surý, at RIPE 92 was on the topic of DNS provisioning, looking at how to strike a balance between resilience and efficiency in the provisioning of nameservers of DNS zones and the performance of DNS resolvers.

An interesting way to look at this topic is to start with a single DNS recursive resolver that is in a *cold start* state, where there is nothing held in the resolver's local cache. It's an easy experiment to replicate, so let's start with my web server domain name, `www.potaroo.net`, and use a local Bind recursive resolver, running version 9.20.23. It takes this recursive resolver 4 queries to perform this resolution function. The resolver has loaded the IP addresses of the root servers from its configuration data, so the first query is to refresh this root server data by querying one of these root server IP addresses with a query for the nameservers of the root zone;

```
Q: NS? .
A: NS i.root-servers.net., NS d.root-servers.net. ...
      Additional: m.root-servers.net. A 202.12.27.33, ...
```

The resolver then asks one of these servers for the nameservers for the `.net` zone:

```
Q: NS? net.
A: NS f.gtld-servers.net., NS m.gtld-servers.net., …
   Additional: f.gtld-servers.net. A 192.55.83.30, …
```

The third query is to one of these servers and it's after the nameservers for `potaroo.net`: ...

```
Q: NS? potaroo.net.
A: NS ns1.potaroo.net., NS ns2.potaroo.net
   Additional: ns1.potaroo.net. A 203.133.248.2, …
```

The final query is for the terminal name, `www.potaroo.net`

```
Q: A? www.potaroo.net.
A: A 203.133.248.108
```

This is as few queries as one can make to resolve a name that has 3 labels and each label is a zone cut-point. In this case the resolver is performing query name minimisation, but the result would be the same if it were asking for the A record for `www.potaroo.net` in each case, as it would receive the same information in the referral responses from the root and `.net` servers.

In contrast, there are domain names that present a far more detailed resolution pattern. Ondřej's example was `teams.microsoft.com`, which took his Bind resolver running 9.16 some 247 queries! My *cold-start* Bind resolver running 9.20 took just 94 queries, and Ondřej noted that 9.21 has managed to cut this down to 29 queries. Earlier versions of the Bind resolver took up to 247 queries to resolve this name!

In some ways this is a contrived question, as it's unusual for users to encounter a *cold-start* recursive resolver. In a more conventional environment, many of these queries are answerable by the resolver's warm cache and many queries are avoided as a result. But in any case, it's useful to ask why `teams.microsoft.com` is such a challenging name to resolve.

CNAMES

CNAMES, so popular with (Content Distribution Networks (CDNs) due to their ability to isolate a single label out of the control of one DNS zone and insert it into another zone, are one of the challenges to name resolution. A CNAME has the effect of restarting the resolver's name resolution process with the target name. In the case of `teams.microsoft.com` there is a sequence of 4 CNAME records, causing the resolver to perform a total of 5 name resolutions in sequence:

```
teams.microsoft.com.       CNAME  teams.office.com.
teams.office.com.          CNAME  tmc-g2.tm-4.office.com.
tmc-g2.tm-4.office.com.    CNAME  teams-office-com.s-0005.dual-s-msedge.net.
teams-office-com.s-0005.dual-s-msedge.net. CNAME     s-0005.dual-s-msedge.net.
```

In the first CNAME, `teams.microsoft.com.` to `teams.office.com`, the resolver has already cached the nameservers for `.com`, so the resolution of the second name can commence with a query to an already cached `.com` nameserver for the name `office.com`. The second CNAME, `teams.office.com.` to `tmc-g2.tm-4.office.com.`, can use the local cache for the `office.com` nameservers. The next CNAME directs the resolver to restart from the root zone to query for the `.net` servers.

The longer the CNAME chains, combined with a provisioning model where the elements in the chain use different top-level domains, the greater the *cold-start* query load.

Glue

The next factor is the resolution of the names of the name servers. A delegation in the DNS lists the names of the DNS servers that are authoritative for the delegated zone. It is not possible to delegate a domain to an IP address, so it is left to the resolution process to resolve these nameserver names into IP addresses so that they may be queried.

In some circumstances the responding server will also provide the IP addresses of the listed nameservers in the form of *glue records* in the Additional Section of a DNS response. If the name of the nameserver lies within the delegated zone, then it's essential to include these glue records for the nameservers, as otherwise the resolver has no ability to proceed with the resolution operation. Such nameserver names are quaintly termed *in-balliwick* for the delegated zone. Here's an example of *in-balliwick* nameservers:

```
$ dig +short NS potaroo.net
ns2.potaroo.net.
ns1.potaroo.net.
```

It is also permissible to include glue records for *sibling* nameserver names, although it's not strictly necessary to perform such an inclusion, nor is the resolver required to trust these *sibling* glue records in any case. Here's an example of *sibling* records for nameservers:

```
$ dig +short ns ripe.net
manus.authdns.ripe.net.
ns3.lacnic.net.
ns4.apnic.net.
ns3.afrinic.net.
rirns.arin.net.
```

Balliwicks* and *Siblings

I've been intrigued by the use of the term *balliwick* in the context of the DNS and the names of name servers. The word "bailiwick" originated in 15th-century Middle English as a combination of two words: "bailiff" (a word for a law officer that originated as a French term that came to the English language with so many other French words relating to the law and justice with the Noman invasion of 1066) and "wic" (an older Anglo-Saxon term for a village or dwelling place).

By the mid-1800s, American English speakers began using the word *balliwick* figuratively to describe a person's natural sphere of interest, authority, or area of expertise. It essentially means "someone's specific domain." Why the term was revived in America appears to be related to a shift in the mid-1800's in American colloquial writing and journalism that adopted slightly archaic, pedantic, or colourful English terms.

In the DNS the term *in-balliwick* is is used to describe the collection of names that are a "descendant" of a name. For example, the names `test1.example.com` and `test2.example.com` are both in the *balliwick* of `example.com`

Sibling is a term used in the DNS to describe the relationship between two names that have a common immediate parent domain. For example, the name `a.example.com` is a *sibling* name of `b.example.com`.

See RFC 9499 and RFC 9471 for a detailed description of these terms with respect to delegation and glue records in DNS responses.

Obviously, where it can, or must, be trusted, the provision of glue records makes *cold start* resolution more efficient, as otherwise the resolver has to set aside its primary resolution task and start a new resolution process to resolve the name of the name server.

The trust in these glue records for *in-balliwick* nameserver names is essentially a matter of forced trust, as the resolver cannot proceed with the resolution function without using this information. Trust in the glue records for *sibling* nameserver names is potentially misplaced, as the resolver can independently resolve these names in any case (unless the name dependency chain generates some form of loop). The Bind recursive resolver changed its behaviour of the treatment of *sibling* glue records as of Bind 9.20.14, because of a vulnerability relating to cache poisoning with *sibling* glue (CVE-2025-40778) and *sibling* glue is now ignored in more recent releases of this resolver code.

Now let's return to the resolution of the name `teams.microsoft.com`, and the first CNAME target, `teams.office.com`:

```
$ dig +short NS office.com
ns1-05.azure-dns.com.
ns2-05.azure-dns.net.
ns3-05.azure-dns.org.
ns4-05.azure-dns.info.
```

One of these names is a *sibling* record in .com, while the other three nameserver names are going to require resolution from the root zone, as they are out of *balliwick*. For a *cold-start* resolver state this is a burden that appears to add little in terms of overall resilience of the name `office.com`.

All or One?

More generally, there are two different nameserver resolution approaches for recursive resolvers.

One approach is to resolve everything, loading the resolver's cache with all the IP addresses of all the listed nameservers (both A and AAAA records). This can be a liability for resolvers when presented with a name with long CNAME chains, each of which has an extended list of name server names that are all *out-of-balliwick*.

The other approach is to resolve just one nameserver name and potentially just one IP address (in just one IP protocol) and fall back to alternative nameservers and IP addresses if the initial resolution query fails. The question here is whether it's preferable to load the cache with a more complete set of nameserver information for a domain, allowing the resolver to quickly turn to querying other nameservers in the event of a failure to respond, or whether its preferable to reduce the minimum resolution time by taking a direct and minimal path through the delegation hierarchy. This is a higher risk path in terms of resiliency, as failure may cause extended resolution times as the resolver may then be required to resolve other nameserver names, which in the extreme case may result in the resolver encountering a task timeout and returning an unnecessary SERVFAIL response.

The benefit of this minimal query approach is both faster cold cache performance and reduced DNS query loads. It's a case of "assume the best and withhold activating the backup plan until it's necessary."

Parent-Centric? Child-Centric?

Duplicated information in the DNS always represents a difficult decision for resolvers. The names of the nameservers in the delegated (child) zone represent the authoritative source of this information, but in performing name resolution, a *cold start* recursive resolver inevitably follows the nameserver information as provided by the nameserver for the parent zone.

However, there is a choice at this point. The resolver could query the child zone for the nameservers of this zone, and then use these servers thereafter, as well as using these servers to query for names in this child zone. The cost is an additional delay to perform this query of the child zone and then a potential additional delay in order to resolve these names to their IP addresses if they are out of *balliwick*.

What exactly is the risk in not performing this additional query to establish the child domain's listing of a zone's nameservers? On first use, the resolver is essentially forced to use the parent zone's nameservers, and if this data has been compromised then the damage has been done. Even if the resolver then attempts to perform a child-centric query, the issue is that the resolver may have already been misdirected and the query to the child zone servers may be a misdirected query. The pragmatic observation is that there is little to be gained by performing this additional query to the child zone to establish the zone's nameservers as the additional time and effort to do so provides little in the way of additional protection for the integrity of DNS resolution. If you want to have some level of assurance as to the authenticity of a DNS resolution outcome, then DNSSEC represents a more robust approach, albeit with an additional cost.

BIND version 9.21.20 switched to parent-centric delegation model, and in so doing further reduced the query load for *cold-start* name resolution.

DNSSEC

What happens when we add DNSSEC validation to the query? In the case of Bind version 9.18.47 (which I'm using as an authoritative nameserver) the query to a parent zone for the NS records of a delegated zone not only generates a response that lists the nameservers held by the parent, but it also contains the DS record, and its corresponding RRSIG record.

In the case we've looked at, for resolution of the name `www.potaroo.net`, an additional 2 queries are used for DNSSEC validation, namely for the DNSKEY resource records for `potaroo.net` and `net`.

In the case of a CNAME name, DNSSEC validation is required for each CNAME in the name sequence, adding DNSSEC queries according to the label count in each CNAME.

The Bind recursive resolver implements DNSSEC validation as a second step to name resolution, as it queries for the DNSKEY records of each of the delegated domains once the name has been resolved (the DS records have already been retrieved with the prior NS queries). This particular resolver implementation does not perform any kind of seperate DNSSEC validation on delegation records (such as the DNSSEC-signed NS records in the child zone), as it appears to operate on the principle that it's the validity of the resolution response that ultimately matters, not the chain of individual delegations that were used to get to the name.

Resolver Behaviour

There is an obvious tension between resilience and speed in the design of a resolver's query strategy.

One approach is to load a resolver's cache with all the resolved IP addresses of all listed nameservers for delegations. This way the resolver is likely to haver immediately available alternative nameservers available to query should the initial query fail for any reason. Such a comprehensive approach might also see the resolver explicitly query the child zone to load the authoritative list of nameservers for each delegation and only use the parent zone list for the initial discovery of the delegation point.

A suitably cautious resolver would only use glue records for *in-balliwick* nameserver names and discard these glue records once the authoritative list has been retrieved from the child zone and resolved.

However, such a comprehensive collection of delegation data takes both queries and time, and the pragmatic considerations of speed and efficiency of resolution call for a more minimal approach. In this case the resolver will use the parent's list of nameservers for a delegated zone and will prefer the use of an *in-balliwick* nameserver that already has provided its IP address as a glue record in the referral response. This behaviour optimises the performance of the resolver when the DNS infrastructure is responsive, and additional performance penalties may be incurred in the event of DNS response failures.

NameServer Suggestions

What are some useful guidelines to improve the efficiency of name lookups? Here's a short list:

- Prefer in-domain nameserver names where possible – here's a good example:

```
$ dig +short NS google.com
ns3.google.com.
ns1.google.com.
ns4.google.com.
ns2.google.com.
```

- Avoid spreading nameserver names across many top-level domains – here's an example of what to avoid!

```
$ dig +short NS office.com
ns1-05.azure-dns.com.
ns4-05.azure-dns.info.
ns2-05.azure-dns.net.
ns3-05.azure-dns.org.
```

- Try to keep CNAME chains short:

```
$ dig +short A www.apnic.net
CNAME www.apnic.net.cdn.cloudflare.net.
104.18.235.68
104.18.236.68
```

- Limit the number of nameservers – more nameservers does not necessarily engender greater levels of resilience, in general.

If you are after resilience through diversity, it makes more sense to achieve that through the use of multiple IP addresses that are drawn from different route advertisements. Query performance can also be improved by the use of *anycast*, directing the resolver client to the (routing-determined) *closest* authoritative nameserver.

It's an open issue whether it's better to use multiple IP addresses per nameserver name or not. For example:

> The name `google.com` has four nameservers, each of which has a single IPv4 and a single IPv6 address:
>
> ```
> $ dig NS google.com
>
> ; ANSWER SECTION:
> google.com.         172800 IN     NS     ns1.google.com.
> google.com.         172800 IN     NS     ns2.google.com.
> google.com.         172800 IN     NS     ns3.google.com.
> google.com.         172800 IN     NS     ns4.google.com.
>
> ;; ADDITIONAL SECTION:
> ns1.google.com.            172800 IN     AAAA   2001:4860:4802:32::a
> ns2.google.com.            172800 IN     AAAA   2001:4860:4802:34::a
> ns3.google.com.            172800 IN     AAAA   2001:4860:4802:36::a
> ns4.google.com.            172800 IN     AAAA   2001:4860:4802:38::a
> ns1.google.com.            172800 IN     A      216.239.32.10
> ns2.google.com.            172800 IN     A      216.239.34.10
> ns3.google.com.            172800 IN     A      216.239.36.10
> ns4.google.com.            172800 IN     A      216.239.38.10
> ```
>
> The name `cloudflare.net` has five nameservers, each of which has three IPv4 and three IPv6 addresses:
>
> ```
> $ dig NS cloudflare.net
>
> ;; ANSWER SECTION:
> cloudflare.net.            85767  IN     NS     ns4.cloudflare.net.
> cloudflare.net.            85767  IN     NS     ns2.cloudflare.net.
> cloudflare.net.            85767  IN     NS     ns3.cloudflare.net.
> cloudflare.net.            85767  IN     NS     ns1.cloudflare.net.
> cloudflare.net.            85767  IN     NS     ns5.cloudflare.net.
>
> ;; ADDITIONAL SECTION:
> ns1.cloudflare.net. 172166 IN     AAAA   2606:4700:57:1:4e4b:27d7:a29f:3cfa
> ns1.cloudflare.net. 172166 IN     AAAA   2803:f800:52:1:1df9:cfa1:ac40:28fa
> ns1.cloudflare.net. 172166 IN     AAAA   2a06:98c1:56:1:2693:f8af:6ca2:c6fa
> ns2.cloudflare.net. 172166 IN     AAAA   2606:4700:57:1:e6ea:b99c:a29f:3cfb
> ns2.cloudflare.net. 172166 IN     AAAA   2803:f800:52:1:4da:414e:ac40:28fb
> ns2.cloudflare.net. 172166 IN     AAAA   2a06:98c1:56:1:ca7d:4c29:6ca2:c6fb
> ns3.cloudflare.net. 172166 IN     AAAA   2606:4700:57:1:cba:4d93:a29f:3cfc
> ns3.cloudflare.net. 172166 IN     AAAA   2803:f800:52:1:b216:ccbe:ac40:28fc
> ns3.cloudflare.net. 172166 IN     AAAA   2a06:98c1:56:1:2cfa:9492:6ca2:c6fc
> ns4.cloudflare.net. 172166 IN     AAAA   2606:4700:57:1:fd14:b411:a29f:3cfd
> ns4.cloudflare.net. 172166 IN     AAAA   2803:f800:52:1:e69e:46f0:ac40:28fd
> ns4.cloudflare.net. 172166 IN     AAAA   2a06:98c1:56:1:45a3:de4a:6ca2:c6fd
> ns5.cloudflare.net. 172166 IN     AAAA   2606:4700:57:1:cb7b:a51d:a29f:3cfe
> ns5.cloudflare.net. 172166 IN     AAAA   2803:f800:52:1:d48b:2cd:ac40:28fe
> ns5.cloudflare.net. 172166 IN     AAAA   2a06:98c1:56:1:3b6c:6602:6ca2:c6fe
> ns1.cloudflare.net. 172166 IN     A      108.162.198
> ns1.cloudflare.net. 172166 IN     A      162.159.60.250
> ns1.cloudflare.net. 172166 IN     A      172.64.40.250
> ns2.cloudflare.net. 172166 IN     A      108.162.198.251
> ns2.cloudflare.net. 172166 IN     A      162.159.60.251
> ns2.cloudflare.net. 172166 IN     A      172.64.40.251
> ns3.cloudflare.net. 172166 IN     A      108.162.198.252
> ```

```
ns3.cloudflare.net. 172166 IN     A      162.159.60.252
ns3.cloudflare.net. 172166 IN     A      172.64.40.252
ns4.cloudflare.net. 172166 IN     A      108.162.198.253
ns4.cloudflare.net. 172166 IN     A      162.159.60.253
ns4.cloudflare.net. 172166 IN     A      172.64.40.253
ns5.cloudflare.net. 172166 IN     A      108.162.198.254
ns5.cloudflare.net. 172166 IN     A      162.159.60.254
ns5.cloudflare.net. 172166 IN     A      172.64.40.254
```

It is unclear if these additional IP addresses add to the resilience or the performance of the resolution of the name. If there is an error state that would trigger a resolver to go through the complete list of nameserver IP addresses in an effort to find a response, the larger IP address set would certainly cause a larger query collection, and a longer elapsed time to reach the point of returning an error condition back to the querier.

Conclusions

It does not make a whole lot of sense to provision DNS nameservers to optimise the *cold-start* scenario. That scenario is rarely encountered. A populated resolver cache will mask out many of overheads of poorly configured nameservers, as most of these issues are acutely visible only with a *cold-start* resolver state.

However, it is useful to be mindful of the load that is being placed on the DNS with diverse nameserver names, CNAME chains and *out-of-balliwick* nameservers. Designing a delegation framework that is efficient and robust irrespective of the current cache state of the resolver is a win for both the name server and the name user.

Disclaimer

The above views do not necessarily represent the views or positions of the Asia Pacific Network Information Centre.

Author

Geoff Huston AM, B.Sc., M.Sc., is the Chief Scientist at APNIC, the Regional Internet Registry serving the Asia Pacific region.

www.potaroo.net