How many DNS queries does it take to resolve one name?

A cold-cache tour through delegations, CNAMEs, and PTRs

Ondřej Surý, ISC

IETF 126 Vienna

One name. One resolver. Cold cache.

329

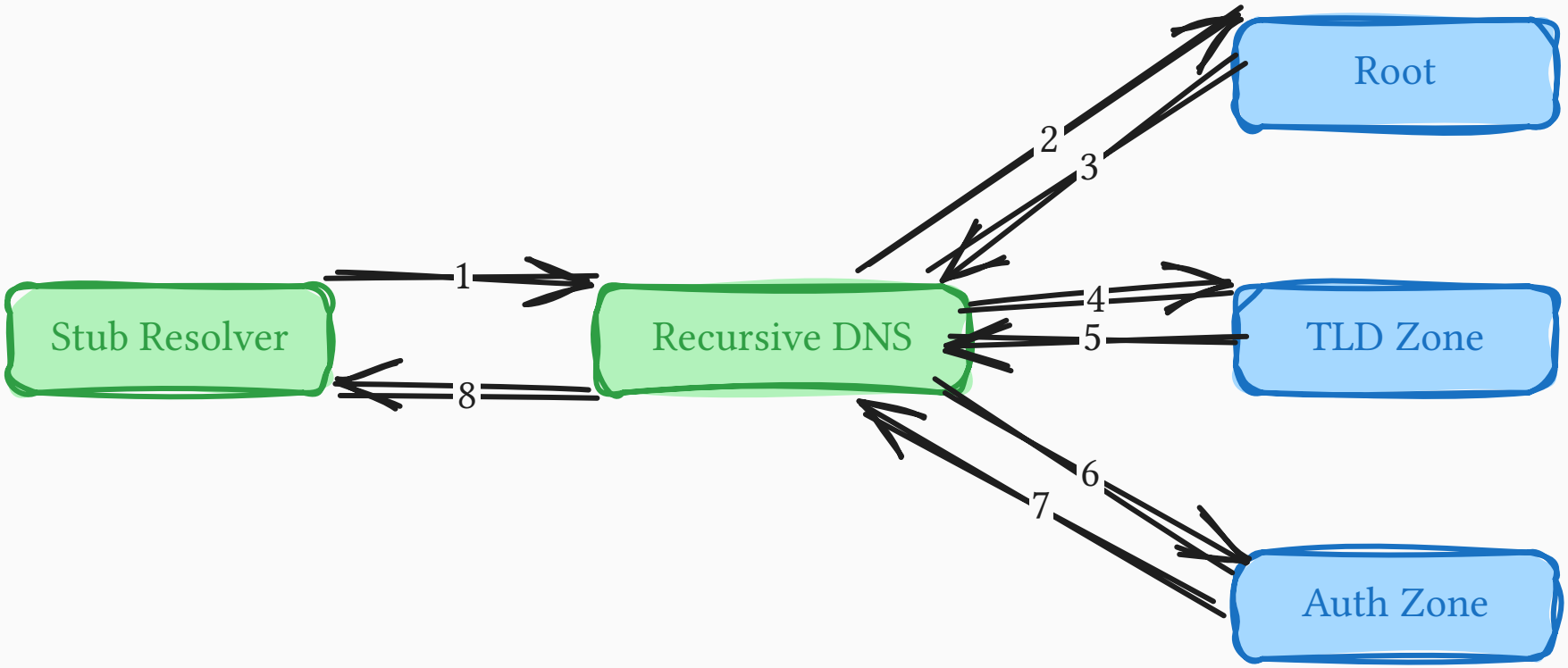
`dnssec-stats.ant.isi.edu. IPv6 PTR`

cold cache · single resolver · one name · BIND 9.18

What this talk does

- All numbers come from BIND 9 with cold cache
- Counts queries as we add **one source of complexity at a time**
- Three delegation types: **in-domain · in-bailiwick · out-of-bailiwick**
- **CNAME chains** and the CDN multiplier
- How much does **DNSSEC** cost you?
- The forgotten case: **PTR**

Recursive resolution in 30 seconds

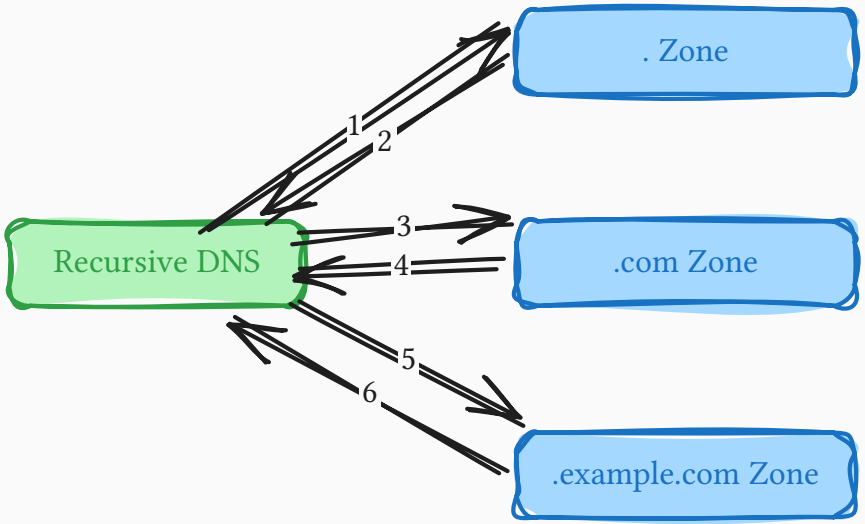


Every arrow on the resolver side is an RTT.

Cache hides most of them — until it doesn't.

Baseline – the textbook case

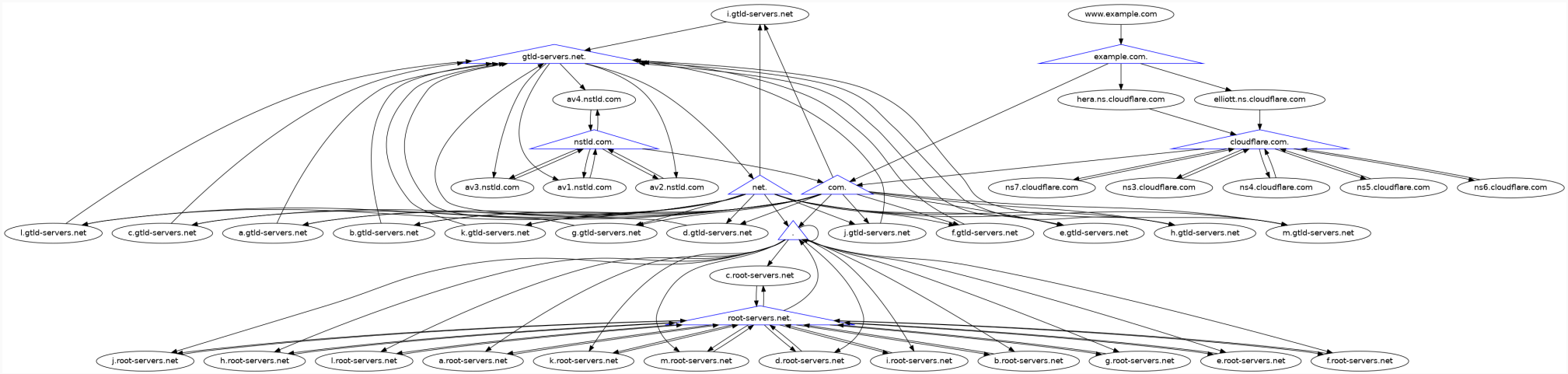
Resolve `www.example.com` — cold cache, in-bailiwick, no CNAME.



Σ = 3

This is our (imaginary, naive) floor.

Baseline – the real-world case



This is the real-world graph of domains involved in resolving www.example.com

trans-trust.verisignlabs.com/?z=www.example.com

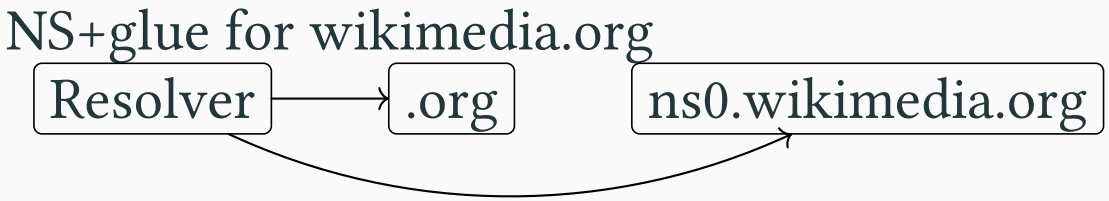
Delegation taxonomy

Type	**NS name lives in…**	**Glue?**	**Extra cost**
In-domain	the zone itself	required (mandatory)	**0**
In-bailiwick	parent of the zone	promotable	**0 to $N * 2$**
Out-of-bailiwick	an unrelated zone	not allowed	**$N * 2$**

The cost difference between row 1 and row 3 is the entire subject of this talk.

In-domain – strict glue (the cheap case)

`wikimedia.org` delegated to `ns0.wikimedia.org.`, …, `ns2.wikimedia.org.`
Glue **must** be in the parent (chicken-and-egg otherwise).



Σ = 3

(no change over baseline)

Couple of top-20 domain does this. There is a reason.

In-bailiwick – sibling glue

`ripe.net` ⟶ NS `ns4.apnic.net.`
`ripe.net` ⟶ NS `ns3.afrinic.net.`
`ripe.net` ⟶ NS `ns3.lacnic.net.`
`ripe.net` ⟶ NS `rirns.arin.net.`

Parent (`.net`) **can** ship glue — it's authoritative for the sibling.
Whether the resolver promotes that glue depends on policy.

- Glue trusted (was default before the CVE): **0 extra**
- Glue ignored / strict: small sub-resolution, still inside `.net`

$\Sigma = 3$ to 5

There is also a reason for this.

Out-of-bailiwick — no glue (the expensive case)

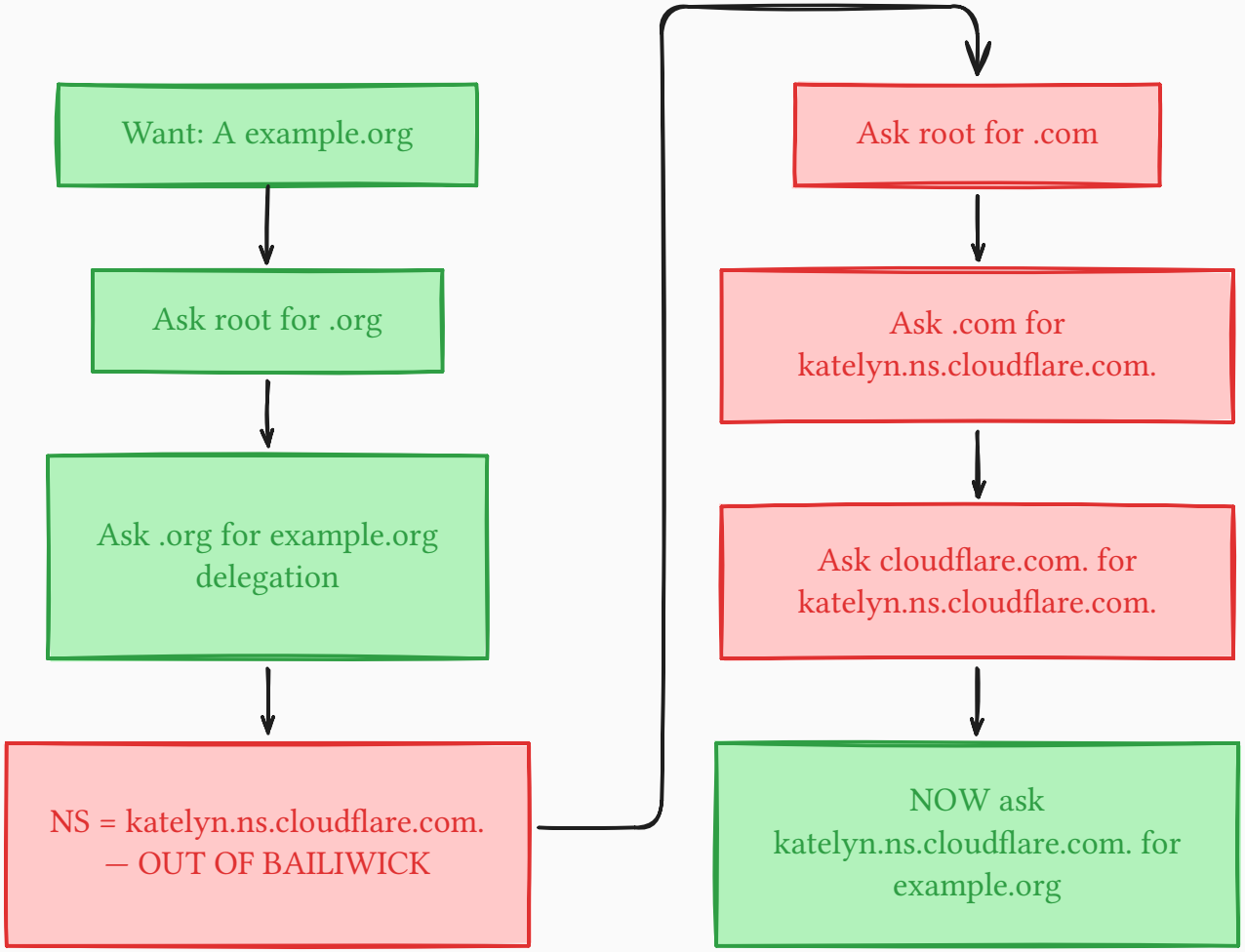
`example.org.` ⟶ NS `katelyn.ns.cloudflare.com.`
`example.org.` ⟶ NS `mitch.ns.cloudflare.com.`

Parent has **no authority** over `.com`. Glue, if present, is **untrusted**.
Resolver **must** resolve the NS name from scratch.

$$\Sigma = 3 + (3 \times N_{\text{OOB parents}}) \text{ per name}$$

Even in this case, there's also a reason for this.

Out-of-bailiwick — no glue (the expensive case)



What the spread costs you

Nothing in cache, resolving just *teams.microsoft.com/CNAME*:

Domain	DNSSEC	9.18	9.20	9.21
teams.microsoft.com	off	88 – 132	48 – 58	16 – 16
teams.microsoft.com	auto	85 – 132	52 – 62	25 – 26

- measuring log entries, 10x.

Σ = ≈ 16–132 cold-cache queries

Now the CNAME chain

`teams.microsoft.com` is not an A record. It's a CNAME. To another. To another.

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

Each CNAME target is a fresh resolution.

Each target zone may be OOB. Each NS-set may span multiple TLDs.

Multiplicative — not additive.

Counting the whole Microsoft Teams resolution

Domain	DNSSEC	9.18	9.20	9.21
teams.microsoft.com/A	off	141 – 180	85 – 102	30 – 30
teams.microsoft.com/AAAA	off	140 – 170	93 – 100	30 – 30
teams.microsoft.com/A	auto	139 – 182	91 – 110	42 – 43
teams.microsoft.com/AAAA	auto	135 – 182	100 – 107	42 – 43

Σ = ≈ 30-182 cold-cache queries

Fun with PTR

Reverse DNS is everywhere:

- mail server reputation checks
- SSH / sudo / syslog
- traceroute, NetFlow, monitoring
- web server access logs

Reverse zones routinely have:

- out-of-bailiwick delegations at every level (RIR → LIR → ISP → customer)
- separate infrastructure from the forward zone
- broken or missing glue
- TTLs nobody ever tuned

IPv6 PTR — 32 nibbles of pain

A reversed IPv6 address has **32 labels** under `ip6.arpa`:

`7.8.6.0.0.0.1.c.0.0.0.0.0.0.0.0.2.2.0.0.8.e.2.0.c.7.6.0.1.0.0.2.ip6.arpa.`

Level	Delegated by
`ip6.arpa.`	IANA → root ops
`2.ip6.arpa.` etc	IANA → RIR
`0.0.1.0.6.0.7.0...`	RIR → LIR
…`/48 reverse`	LIR → end-site
…`/56 or /64`	end-site → customer

Each level is potentially OOB. Each level has its own glue story.

PTR cold-cache trace

Trace for one IPv4 address (and we will need more slides):

Name	Nameservers	Response	Bailiwick
arpa/NS	root-servers.net	*.ns.arpa	strict
in-addr.arpa/NS	*.ns.arpa	*.in-addr-servers.arpa.	sibling
⟶ in-addr-servers.arpa/NS	*.ns.arpa	*.in-addr-servers.arpa.	strict
⟶ *.in-addr-servers.arpa/AAAA	*.in-addr-servers.arpa.	IPv6	n/a
212.in-addr.arpa.	*.in-addr-servers.arpa.	pri.authdns.ripe.net. ns4.apnic.net. ns3.lacnic.net. rirns.arin.net. ns3.afrinic.net.	no

PTR cold-cache trace

Trace for one IPv4 address (continued):

Name	Nameservers	Response	Bailiwick
⟶ net.	root-servers.net	*.gtld-servers.net.	strict
⟶ ripe.net.	*.gtld-servers.net.	ns4.apnic.net. ns3.afrinic.net. manus.authdns.ripe.net. ns3.lacnic.net. rirns.arin.net.	strict/sibling
⟶ lacnic.net.	*.gtld-servers.net.	…lacnic.net. …ripe.net. …apnic.net. …arin.net. **a.lactld.org.**	strict

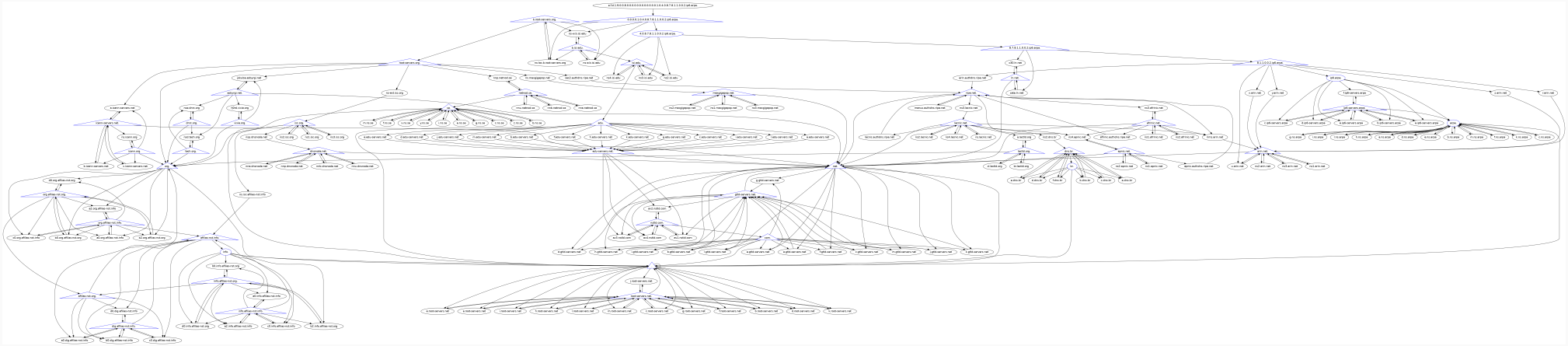
Σ = N, where N is uncomfortably large

And resolvers do this for every connection in your access logs.

Some more PTR examples

Domain	9.18	9.20	9.21
212.in-addr.arpa.	135 – 182	65 – 94	38 – 43
84.99.132.212.in-addr.arpa.	216 – 245	138 – 144	87 – 94
whois.db.ipv6.ripe.net./PTR	96 – 142	62 – 88	45 – 47
dnssec-stats.ant.isi.edu.(6)	295 – 329	193 – 210	102 – 128

PTR in pictures (dnssec-stats.ant.isi.edu.)



Two different approaches

Resolve everything:

- For every delegation level:
  - Store all nameserver names
  - Resolve both A and AAAA
- Explodes under attack

$$\Sigma = \text{level} * (1 + N * 2)$$

Resolve bare minimum:

- For every delegation level:
  - Pick one name
- Removes resiliency
  - Server might be slow or dead

$$\Sigma = \text{level} * (1 + 0 - 1)$$

Neither is viable, so something in between needs to be implemented:

- Limit the number of NS resolved (varies per delegation level)
- Limit the number of IP addresses
- Use sibling glue as much as possible

What went sideways and when?

- BIND 9.20.13 used “whatever was available”
- Then came CVE-2025-40778 (cache-poisoning via sibling glue)
- BIND 9.20.14 ignores sibling glue

Domain	9.20.13	9.20.14	9.20.latest
google.com	7	28	24
facebook.com	8	28	22
chatgpt.com	9	35	32
x.com	35	116	88
reddit.com	20	69	53
bing.com	21	45	54
wikipedia.org	7	31	28

What went well

BIND 9.21.20 switched to parent-centric delegation model:

Domain	Child-centric	Parent-centric
google.com	24	7
facebook.com	25	7
chatgpt.com	32	8
x.com	100	41
reddit.com	51	29
bing.com	51	23
wikipedia.org	34	7

What you can do

Domain operators:

- Prefer **in-domain** delegations where you can
- If you must use a managed DNS provider, pick **one or two**, not four each across different four TLDs
- Audit CNAME depth — every hop is a fresh chain
- Keep PTR delegations boring: in-domain/in-bailiwick, glued

Managed DNS providers:

- Do **in-domain** delegations for the client nameservers

Resolver operators:

- Enable aggressive NSEC if you haven't
- Deploy the latest versions of the resolvers

Takeaways

1. A single lookup is **not** “a few queries” — in real-world it’s **tens** or even **hundreds** of queries
2. Out-of-bailiwick NS and CNAME chains compound **multiplicatively**
3. PTR is the worst-case nobody talks about
4. The structural cost is in the topology — caching only **hides** it

329

One name

TL;DR DO THIS

www.example.com/AAAA

Domain	Response	Queries
com/NS	NS+GLUE [a-m].gtld-servers.net.	2
example.com/NS	NS+GLUE ns[1-3].dns.rocks.	2
www.example.com/AAAA	AAAA	2

3

6 with DNSSEC