What’s in a DNS?

Tour of an eventually consistent, globally distributed key value store

Outline

DNS history

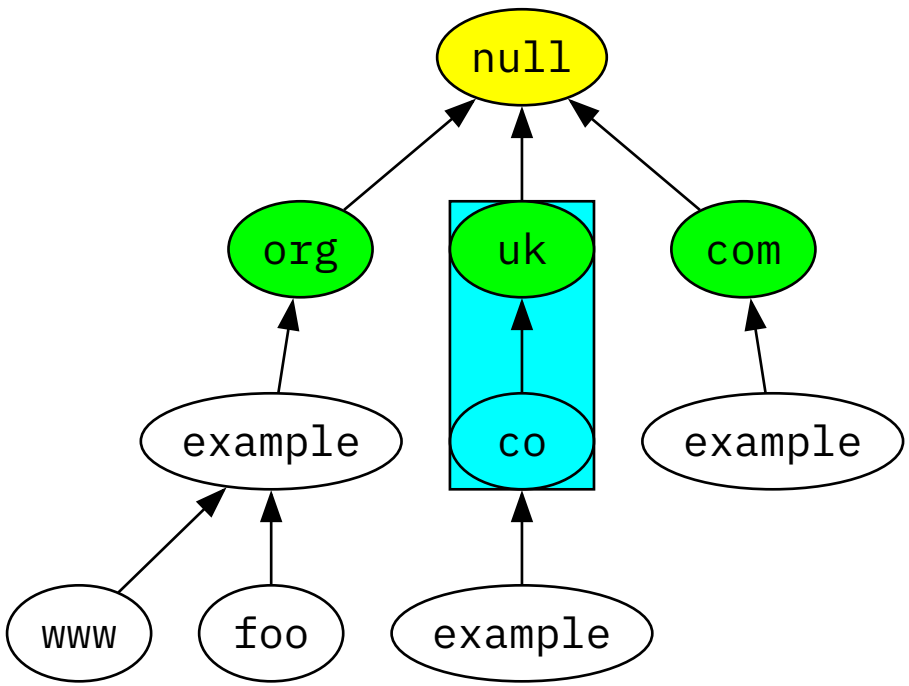
- Predates Internet: Used in ARPANET
- Initially a HOSTS.TXT file (similar to `/etc/hosts`)
  - Maintained by hand
  - Synchronized via phone
- “Domain names” introduced in RFC882 and RFC883
- First name server created in 1984 at UC Berkeley, *BIND*[1].
  - *BIND* is still one of the most popular name server today
- Modern specification of DNS in 1987, in RFC1034 and RFC1035. Extended later, but DNS today is still based on these two specifications.

[1] “Berkeley Internet Name Domain”

A short lexicon

- DNS: **D**omain **N**ame **S**ystem
- RR: **R**esource **R**ecord
- NS: **N**ame **S**erver
- Authority: A name server owning the *authoritative* data for a domain
- TLD: **T**op-**L**evel **D**omain
- Registrar: Organization handling the reservation of domain names and their mapping to IP addresses.

DNS architecture



- Tree structure
  - Nodes contains one *label* and one or more RRs
  - The root node is the *null label*
  - The direct children of the root are TLDs
- Some registrar don't allow specific domains to be registered directly under the TLDs. These reserved domains, such as `.co.uk` or `.us.com` are called effective top-level domains (eTLDs)[1].

[1] https://publicsuffix.org/

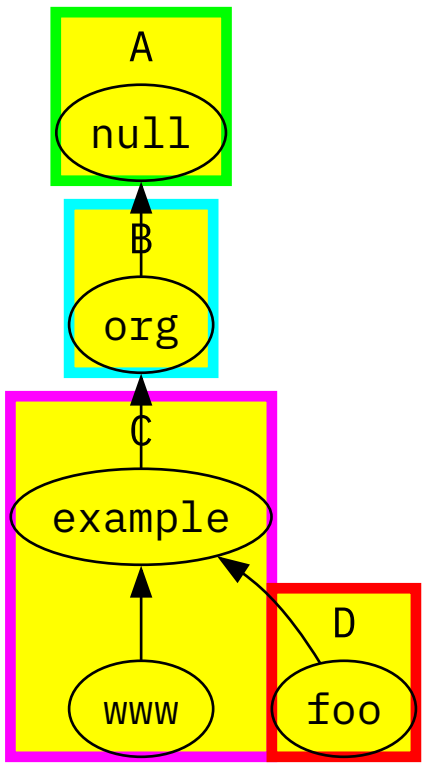
RRs

example.org.		
RR1	Type	A
	Class	IN
	TTL	86400
	RDATA	93.184.216.34
RR2	Type	NS
	Class	IN
	TTL	172800
	RDATA	a.iana-servers.net.

- Each label is associated with a set of RRs.
- Each RR is defined by a *type*, a *class*, a *time-to-live* (TTL) and *RDATA*.
  - Some of the more well-known *types* are A, AAAA, CNAME, NS, TXT and MX.
  - The *class* is almost always IN (Internet)[1].
  - The *TTL* indicates to name servers how long, in seconds, they're allowed to cache the RR.
  - Finally, the *RDATA* describes the resource. for an RR of type A for example, it would contain the IPv4 associated with the domain name.

[1] The other two classes are CH, for Chaosnet, and HS, for Hesiod.

Zones



Portions of the DNS space are grouped into zones.

- Allows for different entities to manage a subset of the DNS space
- The zone owner is the domain names authority: here, the owner of all name servers for `www.example.org.` is C. B is the owner of the authoritative servers for `org.`. This means B's name servers are the source of truth for any name under `org.` – they are the *authority* for this TLD.
- Not all children of a node have to belong to the same zone. Here `foo.example.org.` is managed by a different entity than `www.example.org.`.
- The zone at the top of the DNS space is the *root zone*.

Root Zone

The root zone contains all the authoritative name servers for the TLDs. There are 13[1] root server clusters, named `a.root-servers.net` to `m.root-servers.net`, with well-known IP addresses.

Any domain name can be found recursively from any name server in the root zone.

```
> dig NS . +noall +additional | sort
a.root-servers.net.     6859    IN      A       198.41.0.4
a.root-servers.net.     6859    IN      AAAA    2001:503:ba3e::2:30
b.root-servers.net.     6859    IN      A       170.247.170.2
...
l.root-servers.net.     6859    IN      AAAA    2001:500:9f::42
m.root-servers.net.     6859    IN      A       202.12.27.33
m.root-servers.net.     6859    IN      AAAA    2001:dc3::35
```

[1] https://www.iana.org/domains/root/servers

Name Servers

DNS relies on *authoritative* name servers to hold fragments of information about the DNS name space and *recursive* name servers (or recursive resolvers) to act as DNS clients, querying authoritative name servers and caching the result of their queries.

Effectively, this means an end-user only needs to query a single name server to get a response, instead of going through each level of the DNS space by itself.

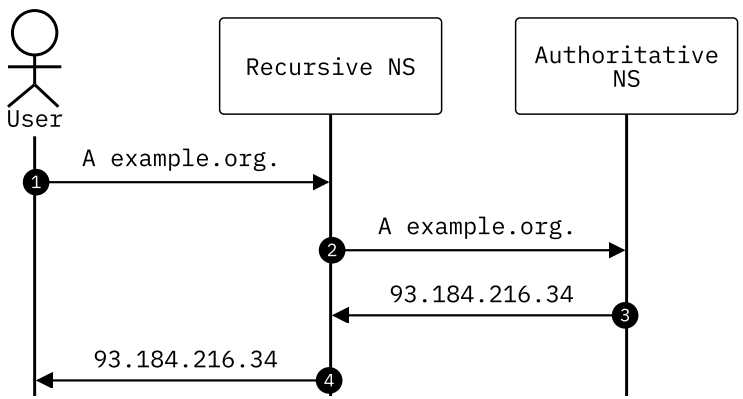
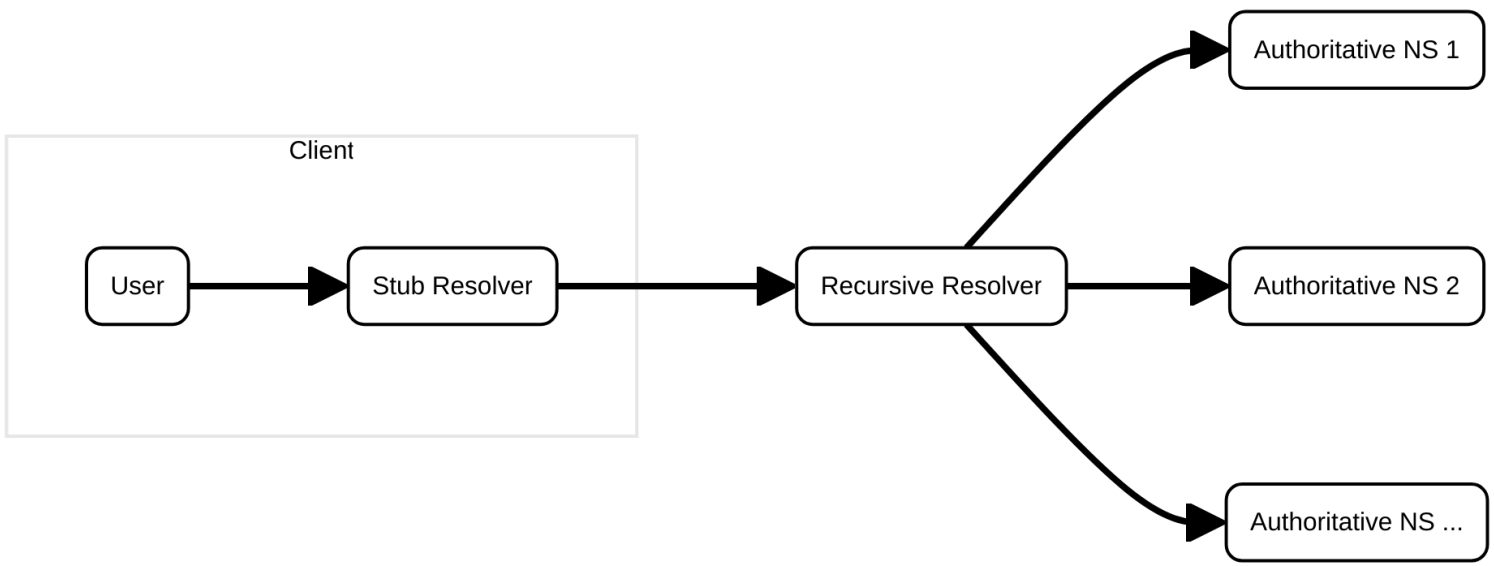


Figure 1: Simplified DNS query

Aside: *Stub Resolvers*

A *stub resolver* – also *DNS client* – is the part of the user's operating system that talks with the recursive server.



~~Security~~

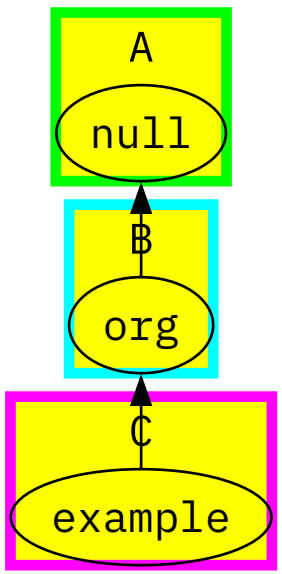
The primary tool created to help with securing DNS is DNSSEC[1], a suite of extensions to DNS. Its primary goal is *authenticating* DNS responses, using a chain of trust starting from the root zone.

1. It doesn't do anything to encrypt DNS traffic,
2. It doesn't prevent selective blocking / DNS interception,
3. Its deployment is still slow[2], the root were only signed in 2010,
4. It gives even more control to the root owner, as they now own the root of the chain of trust.

DNS over TLS (DoT) and *DNS over HTTPS* (DoH) resolve most of these issues, but they don't help with authenticating the entire hop sequence. They only authenticate the initial query from the client to the recursive resolver.

[1] https://datatracker.ietf.org/doc/html/rfc9364
[2] https://stats.labs.apnic.net/dnssec

Zone Delegation



To know which zone is the authority for a domain, DNS provides *Zone Delegation*.

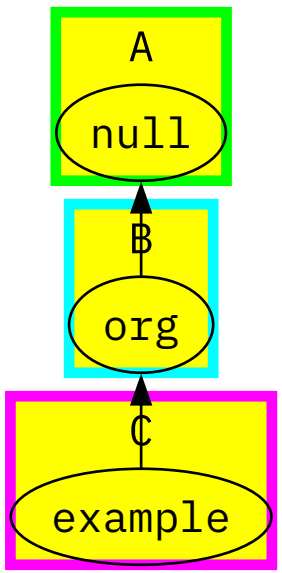
To delegate a zone to a child zone, the DNS in the parent zone returns a set of *NS records*.

In addition to the NS records, all zones are defined by a *Start Of Authority (SOA) record*[1]. Each zone must have one SOA record at its root.

Here, the root zone delegates its authority to a child zone B `org.`, and B delegate to another child zone C for `example.org.`. `.`, `org.` and `example.org.` must each have one SOA record.

[1] Introduced in RFC1034

NS Records



NS records are defined in two places: in the parent zone, for *delegation*, and at the root of the zone itself. Here, B and C should return a set of NS records for example.org. to identify the authoritative server for this domain.

Name Server	Query	Answer
a.root-servers.net	NS org.?	NS …afilias-nst.org.
…afilias-nst.org.	NS org.?[1]	NS …afilias-nst.org.
…afilias-nst.org.	NS example.org.?	NS a.iana-servers.net.
a.iana-servers.net.	NS example.org.?[2]	NS a.iana-servers.net.

Table 1: Finding the authoritative name servers for example.org.

[1] Again? We are asking the authority now, since a.root-servers.net isn't the authority for org.

[2] Same here

Aside: *Glue Records*

There's an issue with these NS records. We need them to find an authoritative name server, but how do we find an authoritative name server for these authoritative name servers?

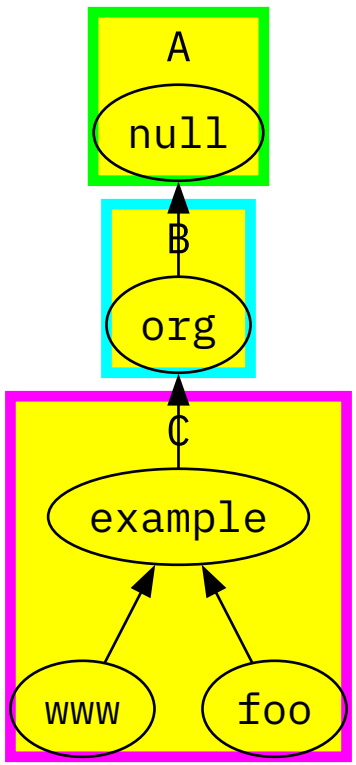
Query	**Answer**
NS org.?	NS b0.org.afilias-nst.org.

How can we find an IP for b0.org.afilias-nst.org.? It's part of org., so we would need to ask b0.org.afilias-nst.org. an IP address for itself! *Glue records* resolve this issue: they're additional records returned by name servers to help avoid this dependency cycle.

Query	**Answer**	**Additional Answers** (Glue records)
NS org.?	NS b0.org.afilias-nst.org.	A b0.org.afilias-nst.org. 199.19.54.1

We now have an IP for b0.org.afilias-nst.org., and can use it for any subsequent queries.

SOA Records

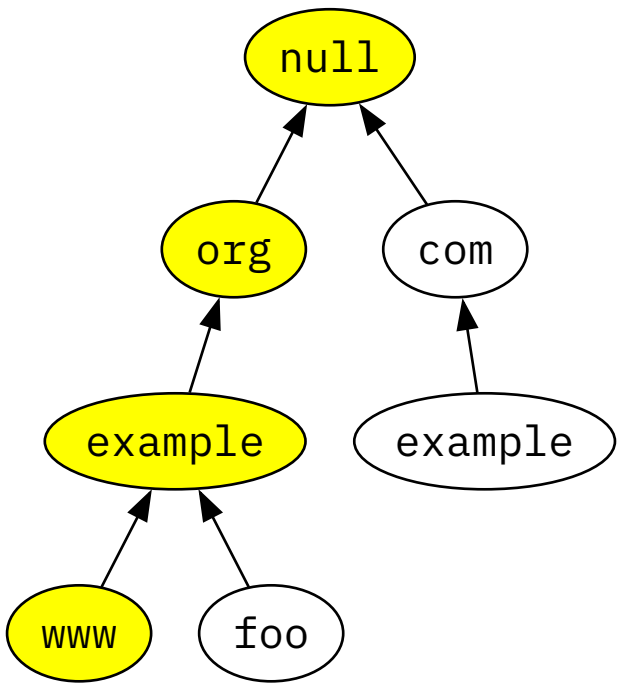


When registering a domain name, the registar creates a **S**tart **O**f **A**uthority record for the user at the start of the new zone for this domain. Here, C would have a single SOA record for the entire zone: example.org., www.example.org. and foo.example.org.. Exactly one SOA record and one or more NS records are required to define the start of a zone.

Some SOA fields can be used for zone transfers, but this is now mostly done internally by registrars. The remaining ones are:

- MNAME: One of the name server for the zone
- RNAME: Email address of the administrator for the zone
- MINIMUM: Used together with the TTL for negative caching, since RFC2308

Domain Names



Domain names are built by starting from a node (subdomain or domain), going up the tree and concatenating the parent label separated with dot (.).

Here, the highlighted path should be read as `www.example.org.` – note the final dot, and how we read from bottom to top, up to the `null` label, rendered as the empty string.

- Each label can be at most 63 characters (64 - 1 byte to encode its length)
- The full domain name can be at most 253 characters (255 - 2 bytes for the length)

DNS and the Internet

The Internet is very large[citation needed]. DNS survived the growth of the Internet mostly due to delegation, and caching and eventual consistency.

- Delegation: The 13 root server clusters don’t need to know the entire DNS tree. Instead, zones are delegated to authoritative name servers, which in turn delegate subtrees to other authoritative name servers. This mean each authoritative name server only needs to know about a small portion of the Internet.
- Caching and Eventual Consistency (or *Convergence*): DNS is never fully consistent. Instead, due to caching happening at many layers in the DNS tree, changes take time to propagate. However, without this, DNS wouldn’t work at the scale of the Internet.

Under a microscope

DNS is not tied to a transport layer and can be sent over UDP[1], TCP, HTTPS[2], QUIC[3], etc. A DNS packet is laid out as follows:

Header	Some useful details
Question	The question for the NS
Answer	RRs answering the question
Authority	RRs pointing toward an authority
Additional	RRs holding additional information

Table 2: DNS packet layout

[1] Limited to 512 octets
[2] DNS Queries over HTTPS (DoH), RFC8484
[3] DNS over Dedicated QUIC Connections, RFC9250

Under a microscope

The Header

DNS follows *network order* (big-endian)[1]. The header layout is as follows:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ID															
QR	Opcode				AA	TC	RD	RA	Z[2]	AD	CD	RCODE			
QDCOUNT															
ANCOUNT															
NSCOUNT															
ARCOUNT															

Table 3: DNS packet header

[1] RFC1035, RFC1700 and IEN137

[2] Reserved for future use. It used to be 3 bits, until AD and CD were introduced in RFC2535.

Under a microscope

The Header

Let’s focus on these first 12 bytes of the DNS packet, which constitute its entire header.

```
0020 <Ethernet, IP, UDP headers …> e5 7a 81 a0 00 01
0030 00 01 00 00 00 01 07 65 78 61 6d 70 6c 65 03 6f
0040 72 67 00 00 01 00 01 c0 0c 00 01 00 01 00 01 24
0050 03 00 04 5d b8 d8 22 00 00 29 04 d0 00 00 00 00
0060 00 00
```

Table 4: `dig @1.1.1.1 A example.org`

Under a microscope

The Header

```
0020 e5 7a 81 a0 00 01
0030 00 01 00 00 00 01 …
```

Table 5: `dig @1.1.1.1 A example.org`, header only

We start with the ID, followed by the list of flags:

1	0	0	0	0	0	0	1	1	0	1	0	0	0	0	0
QR	Opcode				AA	TC	RD	RA	Z	AD	CD	RCODE			

Table 6: `0x81a0` → `10000001 10100000`

This already gives us a lot of information:

- We're looking at a DNS response,
- The NS responding is not an authority,
- We asked for a recursive query, and the server supports it
- The data returned has been authenticated by the server.

Under a microscope

The Header

```
0020 e5 7a 81 a0 00 01
0030 00 01 00 00 00 01 …
```

Table 7: `dig @1.1.1.1 A example.org`, header only

- QDCOUNT indicates the number of questions. Since we know we're dealing with a response, this tells us the NS is answering to 1 of our questions.
- ANCOUNT tells us the number of answers we're getting. Here, the NS only has 1 answer for us.
- NSCOUNT is set to 0: there's no RR in the authority records section.
- ARCOUNT is set to 1: the NS sent us 1 additional record.

The Questions

We've looked at the header, let's move on to one of the most important piece of a DNS packet: the questions.

```
0020 <Ethernet, IP, UDP headers …> e5 7a 81 a0 00 01
0030 00 01 00 00 00 01 07 65 78 61 6d 70 6c 65 03 6f
0040 72 67 00 00 01 00 01 c0 0c 00 01 00 01 00 01 24
0050 03 00 04 5d b8 d8 22 00 00 29 04 d0 00 00 00 00
0060 00 00
```

Table 8: `dig @1.1.1.1 A example.org`

Even answers come with questions!

The Questions

```
0030 07 65 78 61 6d 70 6c 65 03 6f
0040 72 67 00 00 01 00 01 …
```

Table 9: `dig @1.1.1.1 A example.org`

The first part of a question is a length-prefixed sequence of ASCII[1] *labels*. Here, the labels are `example`, `org` and the *null* label for the root.

07	65	78	61	6d	70	6c	65	03	6f	72	67	00
7	e	x	a	m	p	l	e	3	o	r	g	0

The 4 remaining bytes tell us the type of the question (`A`) and its class (`IN`, for Internet).

[1] RFC5890 introduced the concept of *A-labels* and *U-labels* for Unicode domain names.

Under a microscope

The Answers

```
0020 <Ethernet, IP, UDP headers …> e5 7a 81 a0 00 01
0030 00 01 00 00 00 01 07 65 78 61 6d 70 6c 65 03 6f
0040 72 67 00 00 01 00 01 c0 0c 00 01 00 01 00 01 24
0050 03 00 04 5d b8 d8 22 00 00 29 04 d0 00 00 00 00
0060 00 00
```

Table 11: `dig @1.1.1.1 A example.org`

The Answers

```
0040 c0 0c 00 01 00 01 00 01 24
0050 03 00 04 5d b8 d8 22 …
```

Table 12: `dig @1.1.1.1 A example.org`

In order, we have:

- The name. Sometimes a sequence of labels but here an offset in octets into the DNS packet: the two highest bits are set to 1. `0x0cc0` is `11000000 00001100`. Our offset is `12`, putting us right at the beginning of the Question section where `example.org` is already defined.
- The type, `A`, and the class, `IN`.
- The Time-To-Live (TTL) in seconds, telling us how long we may cache the answer – the “DNS propagation time” has its roots right here.
- And finally the data length and the data itself. Here we requested an RR of type `A`, so the data is the 4 octets of the IPv4 for the `example.org` domain: `93.184.216.34`.

~~Additional Records~~

```
0020 <Ethernet, IP, UDP headers …> e5 7a 81 a0 00 01
0030 00 01 00 00 00 01 07 65 78 61 6d 70 6c 65 03 6f
0040 72 67 00 00 01 00 01 c0 0c 00 01 00 01 00 01 24
0050 03 00 04 5d b8 d8 22 00 00 29 04 d0 00 00 00 00
0060 00 00
```

Table 13: `dig @1.1.1.1 A example.org`

For `A` records, we don't usually need to look at the additional records. They're used to provide answers that could be useful to the client. Here it contains a pseudo-RR[1], allowing to introduce new flags that can't fit in the DNS header.

[1] RFC2671

mDNS: Almost DNS

Local DNS! *Bonjour* on MacOS, *Avahi* on Linux.

- The `m` stands for *multicast*
- Introduced in RFC6762
- Same structure as a DNS packet except for an additional flag (highest bit of QCLASS in the Question) to ask for a unicast response
- Uses port 5353, and `224.0.0.251` for IPv4, `ff02::fb` for IPv6.
- Implementations fairly immature on some newer OSes

```
$ tshark -f 'udp port 5353' &
$ ping mylaptop.local
2 0.000097587 192.168.1.13 → 224.0.0.251  MDNS 70 Standard query 0x0000 A mylaptop.local, "QM"
question
3 0.269215159 192.168.1.22 → 192.168.1.13 MDNS 96 Standard query response 0x0000 A
mylaptop.local, "QM" question A 192.168.1.22
```

Further Reading

- Internetworking with TCP/IP Volume 1, Chapter 23: The Domain Name System
- Development of the Domain Name System, Mockapetris and Dunlap 1988
- DNS Performance and the Effectiveness of Caching, Jung et al 2002
- RFC882, Domain Names - Concepts and Facilities and RFC883, Domain Names - Implementation and Specification and their modern counterparts RFC1034 and RFC1035
- RFC3833, Threat Analysis of the Domain Name System