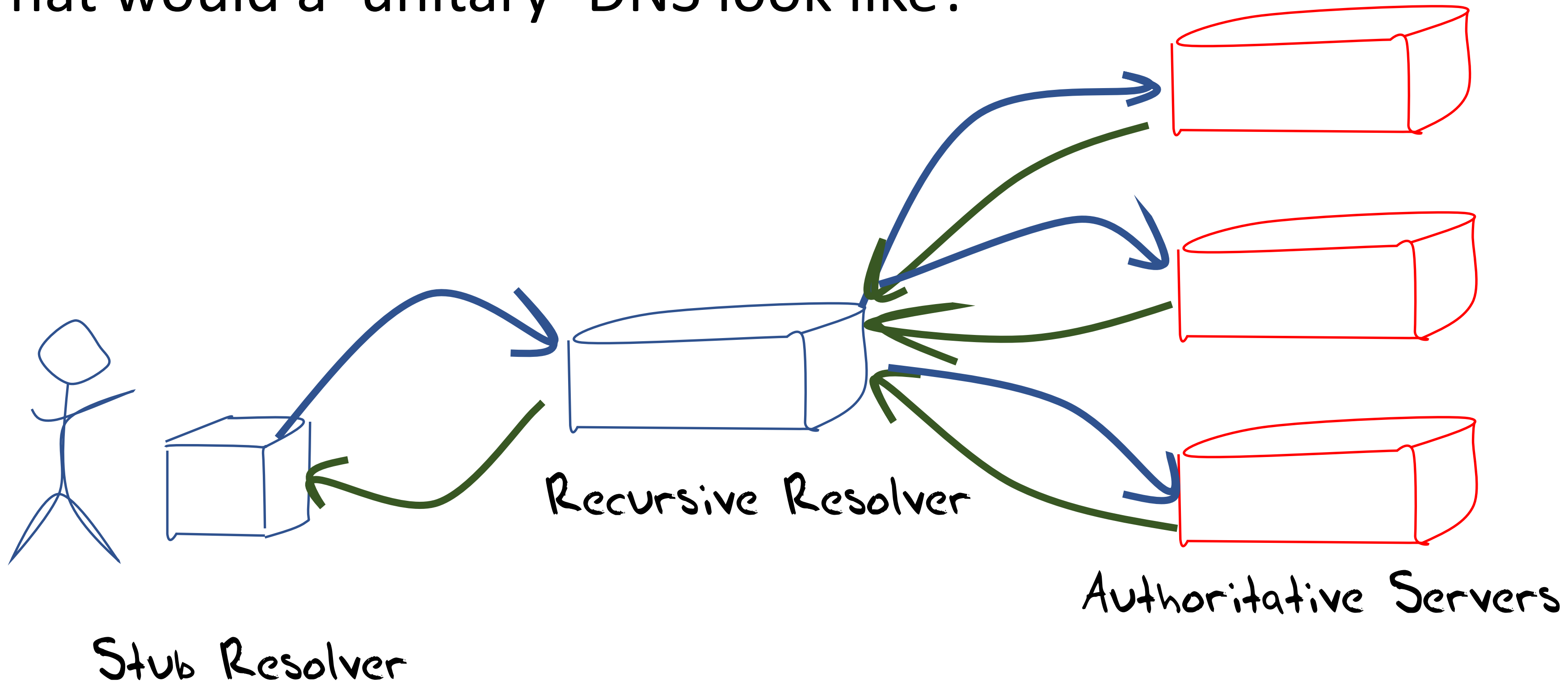
DNS Deep Dive, IETF 108

Resilience

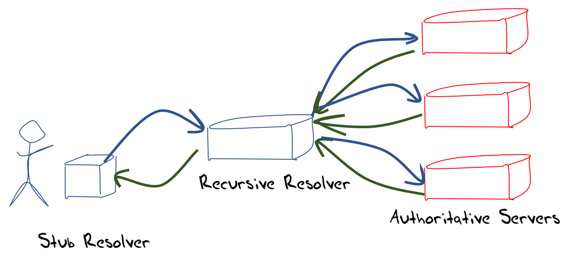
Resilience and Replication

One and Only One?

What would a ‘unitary’ DNS look like?



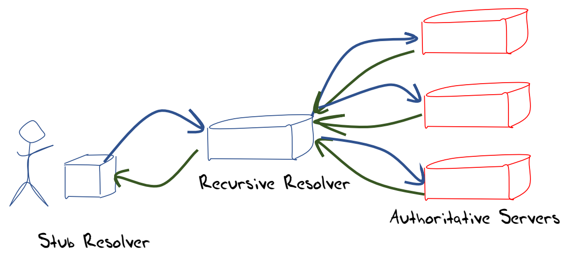
One and Only One?



What would a ‘unitary’ DNS look like?

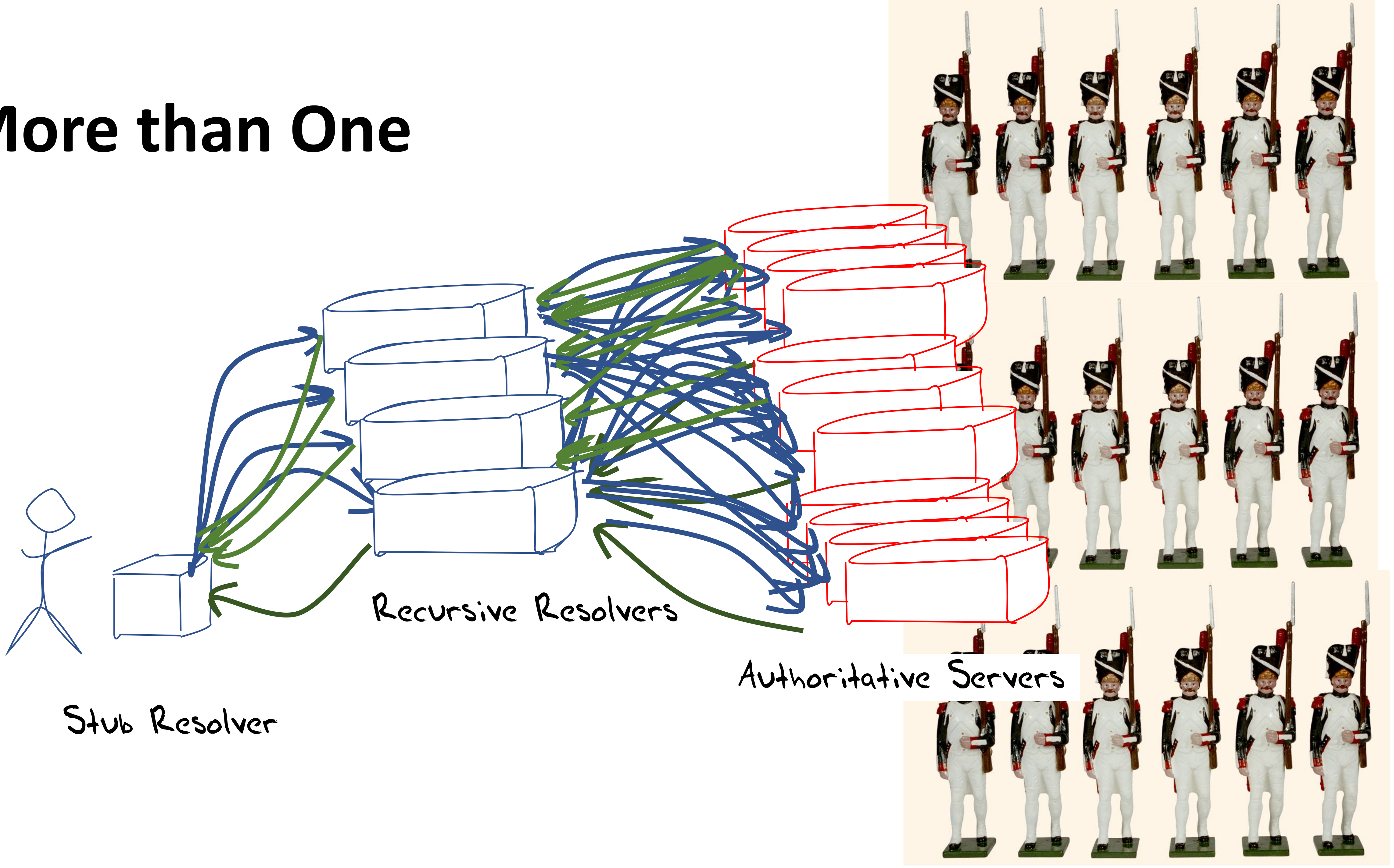
- Each stub resolver has a single address for a recursive resolver
- The stub resolver uses a single query for each name to be resolved
- Each domain is served by a single authoritative server with a single address
- Each recursive resolver uses a single query model

One and Only One = Fail!



- Obviously this model has multiple single points of failure
  - Recursive resolver failure
  - Authoritative server failure
  - UDP packet loss
- And each of these SPOFs becomes a point of vulnerability for hostile attack

More than One



More than One

Resolv.conf

- Stub resolvers are typically configured with up to three recursive resolvers
    - You can add more but…
- In a dual stack world each of these resolvers has both IPv4 and IPv6 addresses and queries are made in parallel via both protocols (1)
- The order is usually important (2), and subsequent resolvers are only queried if there is a UDP timeout from prior queries (3)

1. Unless the “single-request” resolv.conf option is enabled

2. Some measurement of behaviour of Chromium showed a round-robin pattern of use. This round-robin behaviour can be selected on some systems with the resolv.conf option “rotate”

3. Or the resolver receives a response with response code RCODE 2, (SRVFAIL)

More than One

Authoritative Servers

- DNS domains are typically configured with more than one authoritative server
- The root zone uses 13 different server names each with IPv4 and IPv6 addresses
- Between 2 and 4 authoritative servers seems to be common practice these days
- For most applications 5 or more services is probably excessive, and 1 is regarded as too few (1)
- The server list MAY round robin depending on the authoritative server code

1. A single unicast server is probably not enough, and a single anycast address with a set of servers using the same address may still not be enough to be resilient! Diversity is the key here, not mindless replication

Resilience in Transport – not!

UDP

- All bets are off!
- You have no idea if the query or the response was lost
- The UDP sender uses a timeout to determine a "no answer" condition
- The timeout value depends on context:
    - Stub resolvers are observed to use timeout values between 1 and 5 secs (1)
    - Recursive resolvers are seen to use use values between ~300ms and 1 sec
- Should you ask the same server again? Or maybe change the address/protocol? Or maybe ask a different server?
- When to just give up? The number of timeouts is typically limited (2)

1. resolv.conf sets the timeout to 5 seconds by default, with a minimum of 1 and a maximum of 30 configurable in resolv.conf
2. resolv.conf sets the number of attempts to 2 by default with a maximum of 5 configurable in resolv.conf

Resilience in Transport

TCP

- If you can reach the server then you can reasonably expect a response
  - Or an error code or some sort
- Asking the same server again is kinda pointless, particularly if you expected a different answer
  - But sometimes different recursives hide behind the same name or even the same protocol and protocol address
- Asking a different server might be helpful if you didn’t like some first answers (SERVFAIL or possibly REFUSED) but generally it’s not all that useful

Resilience in Transport++

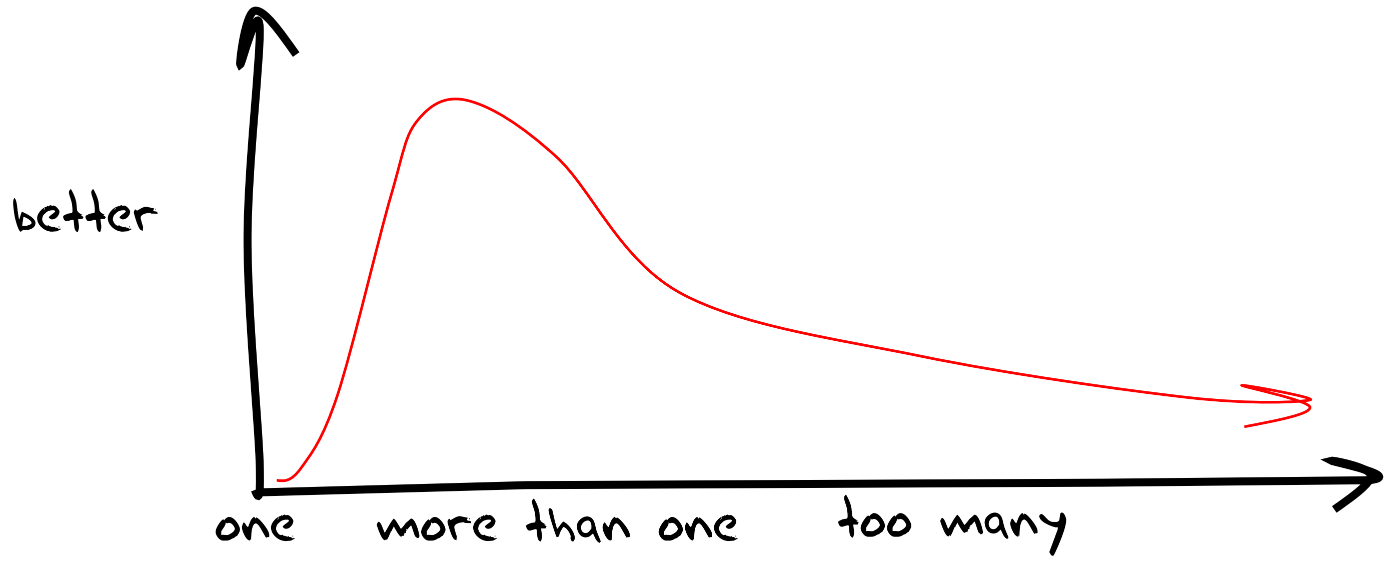
- DNS over TLS over TCP (DoT)
- DNS over HTTPS over TLS over TCP (DoH)
- DNS over QUIC over encrypted TCP-like transport (DoQ)

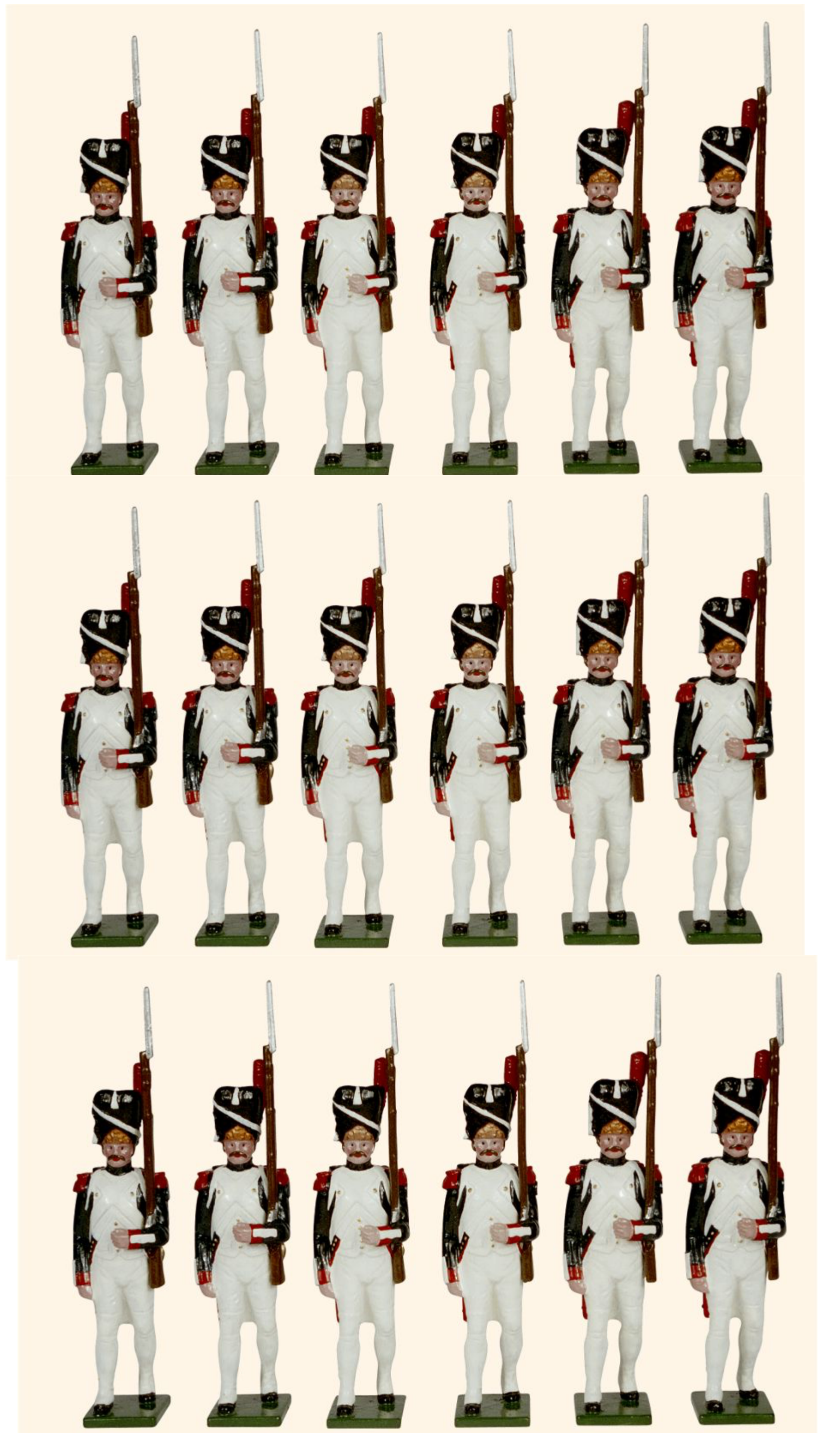
Yes (1)

(1) Session time is finite, this topic is not! Best left to another session with more time!

If more is better…

When is "more" too much?





Pick your answer!

- Replication has its limits as a means of improving service resilience
  - To avoid flooding clients typically use a serial query approach
  - Clients typically limit the number of queries and the overall elapsed time and then declare failure
- Adding more resolvers to resolv.conf, adding more name servers and adding more addresses and protocols to each service name has an initial benefit, but quickly degenerates in terms of added resilience
- How can we improve resilience?

Resilience Engineering

Replication is a coarse response to resilience - can we do this better?

Caching!

- DNS queries have a strong component of self similarity, and caching allows such queries to be answered by the caching resolver and not passed onward into the DNS name resolution infrastructure
  - “Here’s an answer I received earlier that matches your question”
- Without caching in recursive resolvers the DNS as we know it would probably collapse
- Caching is a balance between “freshness” and ”effectiveness”
  - The DNS resolution infrastructure tends to prefer longer cache times over freshness to improve resolution performance
  - Name publishers have an interest in preferring shorter cache times to allow rapid dissemination of changes to the zone contents

Caches

Who caches?

- stub resolvers
- recursive resolvers
- forwarding resolvers
- applications (typically browsers)

Who doesn’t cache?

- caching is optional

Caches

What is cached?

- resource records (not entire responses)

What should not be cached?

- resource records learned through additional sections in responses

Corner Cases:

- Responses with EDNS(0) Client Subnet (1) are cached with the Client Subnet value attached
- NS records from Child vs NS records from Parent
- resource records that fail local DNSSEC validation
  - cache them but only serve from cache if the query has the CD bit set
  - not clear what to do if the query has no EDNS(0) extensions
- DNSSEC NSEC records, which can be used to respond to a range of qnames. This exploits the property that the total name space is far larger than the occupied name space. So the "gaps" between the list of alphabetically sorted (2) names encompass a vastly greater pool of names than the occupied names, so NSEC caching stores these "gaps" to increase the leverage of caches

(1) Which should never have happened

(2) NSEC3 works too, but now we sort the hashes of names, not the names (3)

(3) NSSEC3 should never have happened

Cache TTLs

TTL determines the cache lifetime

- Well not really – TTLs **SUGGEST** a cache lifetime (in seconds) (1)
- The resolver may shorten or lengthen the cache time
- Popular caching resolvers have a “Max TTL” setting to cap TTL values
- There does not appear to be a universal minimum TTL above 0
    - And “0” is equivalent to “do not cache”
- And caches are finite-sized, so entries may be removed prior to TTL expiry under high load

(1) The resolver may (should) treat the value as a timer, but its just a suggestion, so it may play whatever sgames it wants with the TTL value!

Resilience Engineering

Replication is a coarse response to resilience - can we do this better?

Anycast

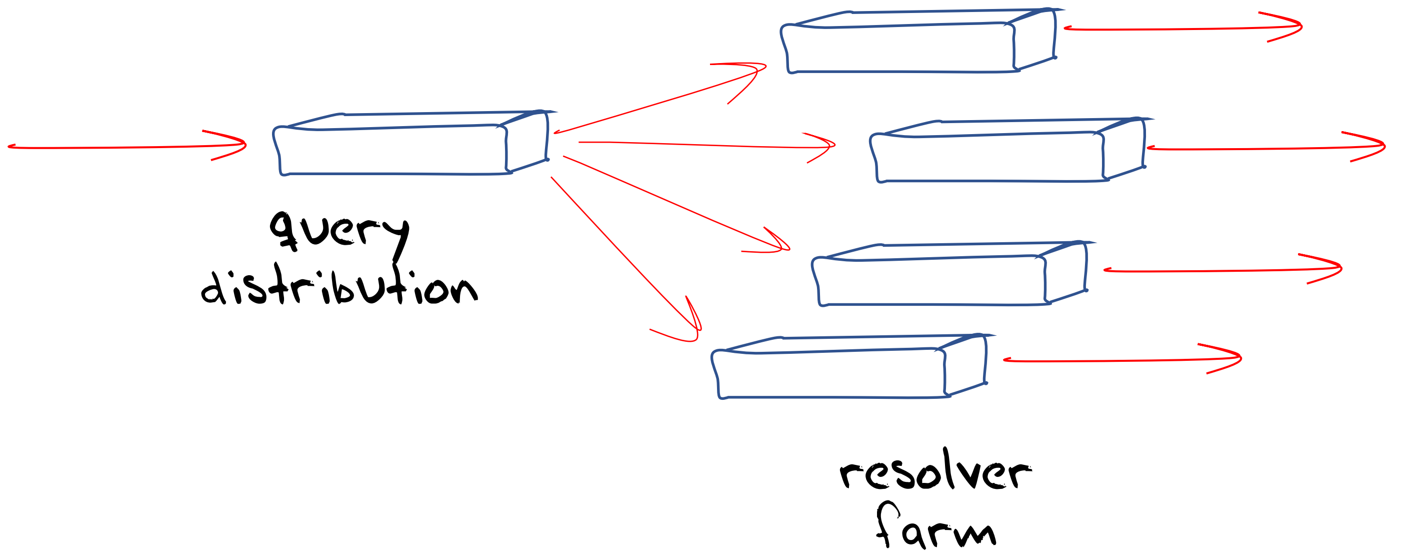
- Use the routing system to determine “closest” instance
- Service instances can be added or removed seamlessly
- Relieves the client of the overheads of serial enumeration
- Anycast Recusive Resolvers
  - Multiple recursive resolver service points that all respond to the same address
- Anycast Authoritative Servers
  - Multiple auth servers all on the same address

Resilience Engineering

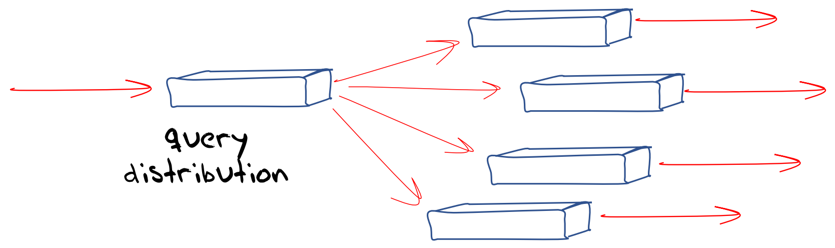
Replication is a coarse response to resilience - can we do this better?

Parallel Resolution!

- Increase resolver capacity though the use of resolver "farms"
  - Common front end / query distributor
  - Collection of resolver engines to form the back end



Resilience Engineering



Resolver "farms":

- How to distribute queries across the back end systems?
  - IP 5-tuple hash (possibly router-based)
  - IP source hash
  - DNS Qname hash
  - Hot caching (unbalanced engine load)
- Cache coherency in "farms"
  - Qname hashing is effective for positive caches, but not NSEC caches
  - IP source hashing is less efficient than 5-tuple hashing, but is coherent for caching
  - Shared caches across the resolver farm can create cache coherency irrespective of the query distribution mechanism, but are challenging to implement efficiently

On to part 3…