who-dis

Desmond Cheong
Brown University
desmond@desmondcheong.com

Nick Young
Brown University
nicholas_young@brown.edu

1. INTRODUCTION

`who-dis` is a DNS resolver that we built to better understand DNS, both from a user and nameserver perspective. Similar to `dig`, `who-dis` takes in a domain name and contacts name servers recursively, until it either finds an IP address to contact or bottoms out in some other way. We support both recursive and non-recursive requests, result caching, and can decode A/AAAA/CNAME records, with extensibility to support more record types in the future. In this write-up, we detail the implementation and functionality of `who-dis`, learning points, and potential for future work.

2. USAGE

Our application works by supplying a domain name to be resolved to an IP address. We provide two flags to explore the functionality of our DNS resolver:

```
usage:  ./who-dis [-trace] [-no-cache] <domain-name>
```

1. *-trace* enables a recursive DNS query. When *-trace* is set, and assuming no results have been cached, `who-dis` will query a root server, followed by the name server for the top-level domain, and so on until there are no additional name servers to query. This happens when we either find the A/AAAA/CNAME record for the requested domain name.
2. *-nocache* disables use of the cache for DNS responses. By default, caching is enabled, and maps domain names to an IP address, along with the TTL that came with the IP address, and the timestamp when the record was stored.

3. IMPLEMENTATION

3.1 Message Encoding and Decoding

Instead of using a package, in order to engage more deeply with the RFC (and also because who doesn't like building things all the way down the stack) we implemented encoding and decoding of DNS messages. In short, a DNS message has four parts: a header section, a question section, and an answers section.

The DNS header primarily contains information about the other sections, but also indicates whether the message is a request or a response, and includes a randomly generated request ID.

The question section typically only contains a single question, which specifies what you're looking for: the domain name, the record type, and the record class (typically internet).

Lastly, the answers section is divided into answers, authorities, and additional records, each of which is a list of resource records. Depending on the type of resource, the data contained could be a domain name, an IP address, or string; for example, A records will contain the IP address for the specified domain name.

3.2 Request Resolution

Resolving a request is simple; we decide on a nameserver to contact (we default to `8.8.8.8` if we aren't going to do the recursive resolution ourselves, `199.9.14.201` if we are) and then send them a message on the UDP DNS port, :53. Once we get a response, we decode it and react.

If we aren't doing recursive resolution, we are done at this point. Otherwise, we have to sus out what information was returned and whether or not we have to do more work. If the records that were returned are A or CNAME records for the requested domain name, then we are done. Otherwise, there should be information about which nameserver to contact next, in which case we start the process over again at that nameserver. Assuming we are able to bottom out, we will eventually find a nameserver that is aware of our desired domain name and returns a result to us.

3.3 Caching

When we receive a response from a DNS server, we always cache results. Results are stored in a persistent key-value store using `Bolt`[1], where the key is a domain name (such as "`www.brown.edu`", "`brown.edu`", or "`edu`"), and the value is an IP address, the TTL for the result, and the timestamp when the result was cached. For resource records in the

[1] github.com/boltdb/bolt

answer section, we store the IP address for A and AAAA records, along with the TTL and timestamp. For records in the authorities section, use the domain name as the key, then find a corresponding resource record from the additional section and store the IP address and TTL from that record.

When we make a request, we first check if we have any information that might help our query along. If possible, we find a nameserver that is authoritative for the longest suffix of the domain name we're looking to resolve, and begin our query there. If we're lucky enough that this longest suffix is the whole domain name, then we're done.

4. RESULTS

If *-trace* is not set, `who-dis` resolves the results from an external DNS server. For example:

```
$ ./who-dis www.brown.edu -trace=true
nochache = false; trace = false

### Response from 8.8.8.8 ###
;; QUESTION SECTION:
www.brown.edu                          IN        A

;; ANSWER SECTION:
www.brown.edu               3040        IN \
CNAME        www.brown.edu.cdn.cloudflare.net
www.brown.edu.cdn.cloudflare.net \
14        IN        A        104.18.3.173
www.brown.edu.cdn.cloudflare.net \
14        IN        A        104.18.2.173

;; AUTHORITY SECTION:

;; ADDITIONAL SECTION:
```

If we perform a recursive query, we get the following results:

```
$ ./who-dis -trace www.brown.edu
nochache = false; trace = true

### Response from 199.9.14.201 ###
;; QUESTION SECTION:
www.brown.edu                          IN        A

;; ANSWER SECTION:

;; AUTHORITY SECTION:
edu                172800        IN        NS \
c.edu-servers.net
edu                172800        IN        NS \
b.edu-servers.net
edu                172800        IN        NS \
f.edu-servers.net
edu                172800        IN        NS \
h.edu-servers.net
edu                172800        IN        NS \
k.edu-servers.net
edu                172800        IN        NS \
j.edu-servers.net
edu                172800        IN        NS \
m.edu-servers.net
edu                172800        IN        NS \
d.edu-servers.net
edu                172800        IN        NS \
g.edu-servers.net
edu                172800        IN        NS \
a.edu-servers.net
edu                172800        IN        NS \
e.edu-servers.net
edu                172800        IN        NS \
l.edu-servers.net
edu                172800        IN        NS \
i.edu-servers.net

;; ADDITIONAL SECTION:
m.edu-servers.net        172800        IN \
A        192.55.83.30
l.edu-servers.net        172800        IN \
A        192.41.162.30
k.edu-servers.net        172800        IN \
A        192.52.178.30
j.edu-servers.net        172800        IN \
A        192.48.79.30
i.edu-servers.net        172800        IN \
A        192.43.172.30
h.edu-servers.net        172800        IN \
A        192.54.112.30
g.edu-servers.net        172800        IN \
A        192.42.93.30
f.edu-servers.net        172800        IN \
A        192.35.51.30
e.edu-servers.net        172800        IN \
A        192.12.94.30
d.edu-servers.net        172800        IN \
A        192.31.80.30
c.edu-servers.net        172800        IN \
A        192.26.92.30
b.edu-servers.net        172800        IN \
A        192.33.14.30
a.edu-servers.net        172800        IN \
A        192.5.6.30
m.edu-servers.net        172800        IN \
A        2001:501:b1f9::30

### Response from 192.55.83.30 ###
;; QUESTION SECTION:
www.brown.edu                          IN        A

;; ANSWER SECTION:

;; AUTHORITY SECTION:
brown.edu                172800        IN \
NS        ns1.ucsb.edu
brown.edu                172800        IN \
NS        bru-ns1.brown.edu
brown.edu                172800        IN \
NS        bru-ns2.brown.edu
brown.edu                172800        IN \
NS        bru-ns3.brown.edu

;; ADDITIONAL SECTION:
```

```
ns1.ucsb.edu        172800        IN \
A        128.111.1.1
ns1.ucsb.edu        172800        IN \
A        2607:f378::1
bru-ns1.brown.edu        172800        IN \
A        128.148.248.11
bru-ns2.brown.edu        172800        IN \
A        128.148.248.12
bru-ns3.brown.edu        172800        IN \
A        128.148.2.13

### Response from 128.111.1.1 ###
;; QUESTION SECTION:
www.brown.edu                          IN        A

;; ANSWER SECTION:
www.brown.edu                3600        IN \
CNAME        www.brown.edu.cdn.cloudflare.net

;; AUTHORITY SECTION:

;; ADDITIONAL SECTION:
```

Performing the recursive query again, we get a cache hit for `brown.edu` and query `ns1.ucsb.edu` to get the following results:

```
$ ./who-dis -trace www.brown.edu
nochache = false; trace = true
### Cached response ###
brown.edu        128.111.1.1

### Response from 128.111.1.1 ###
;; QUESTION SECTION:
www.brown.edu                          IN        A

;; ANSWER SECTION:
www.brown.edu                3600        IN \
CNAME        www.brown.edu.cdn.cloudflare.net

;; AUTHORITY SECTION:

;; ADDITIONAL SECTION:
```

It's worth noting that when we try to perform the recursive resolution on `www.brown.edu.cdn.cloudflare.net`, we aren't able to get a response from Cloudflare's nameserver `ns1.cloudflare.net`. Potentially, this is because they are more selective with which clients they respond to with DNS information. As such, we decided to follow the behaviour of `dig +trace` and stop the recursive query once we receive the CNAME record for `www.brown.edu`.

5. FUTURE WORK

The following are implementation points that we wish we could have hit on, but didn't quite have the chance to:

1. Nameserver functionality - we can make requests, but we cannot respond to them. Presumably, it would be somewhat simple to host a set of domain names that resolve to (most likely toy) IP addresses. We could have also acted as a secondary name server for some existing foreign name server by acquiring a zone and playing around with the zone transfer protocol. However, we ran out of time before implementing this.
2. Supporting more record types - we have the machinary to do so, but didn't find it practical to support any but the main record types. However, being able to support the full gamut of resource types would improve the project.
3. Root server selection - we currently hard-code which nameserver we default to, but it would be nice to be able to run diagnostics to figure out which nameserver is closest to us / fastest to contact.
4. Smarter caching - currently, we only store one IP address per domain name / nameserver, it would be better to have a more robust caching system to better adapt to expiring TTLs.
5. Message compression - while we are able to read compressed messages, we don't compress them ourselves; this would be a nice optimization to make our application more robust.

6. CONCLUSION

Working on this project taught us many things about the inner workings of DNS. Here are some of the highlights:

6.1 Message Structure

DNS messages were challenging to parse because they consist of lists of variable-length questions and resource records, meaning that we had to parse each entry one by one to process the desired responses.

6.2 Message Compression

In the initial stages of our implementation, we could send DNS messages but had problems parsing them, because it appeared that resource records did not contain domain names. Upon further investigation, we discovered that section 4.1.4 of RFC 1035 explains the concept of message compression where domain names are either encoded as a sequence of labels, or a combination of labels and pointers to domain names elsewhere in the DNS message. This was an interesting compression technique, but also complicated the domain name decoding process.

6.3 Recursive Resolution

Querying some DNS servers, like Google's `8.8.8.8`, is nice since they will do all of the work for you, going off and resolving the name fully before returning a response to you. Authoritative nameservers, as well as the nameservers that Google is contacting, don't all do this work, and we learned how one inspects the DNS response to figure out who to ask next. It also helped us learn how DNS divides responsibility among nameservers, and better understand the hierarchical model that lets DNS scale.

6.4 And so...

Overall, the project was a joy to work on. Thank you for the semester!