Telematics Project 01

Freie Universität Berlin Winter Term 2021/22

Prof. Dr. Matthias Wählisch, Marcin Nawrocki, M.Sc. Deadline: 07/12/2021 – *graded*

In this project, you will get hands-on experiences with core services running on top of the Internet, DNS and HTTP. You will implement your own DNS system and a HTTP proxy that will use your DNS.

We will not focus on the exact implementation of standards but on proof-of-concepts of the underlying concepts.

Do not hesitate to initiate discussions and questions on our Slack channel.

Domain Name System

The goal is to implement basic functionalities of a stub resolver, a recursive resolver, as well as an authoritative DNS server, and to operate namespaces. Your approach should be distributed and scalable. It is worth noting that we do not assess the performance or code quality of your implementation but expect that you design your solution based on sound networking concepts.

We will provide a list of name records (see below). You have to decide, which name server is responsible for which zone.

To validate your solution, we will send name resolution requests to the recursive resolver. Each resolver and authoritative server should create a log file that provides insights about the number of requests sent and received (details see below).

1. **Background: Basic Implementation Approach**

   All of your resolver and server software runs on the same host, communication will be local. Each resolver and server will exchange packets, though, based on the loopback interface.

   To allow for multiple instances of your software, you need to bind the various servers to different IP addresses.

   To ease implementation, each name may belong to multiple resource records but for each name there exists only one resource record per resource record type.

2. **IP Addresses and Ports**

   Each DNS server should be assigned an IP address within the range `127.0.0.10` - `127.0.0.100`. Each server should listen to UDP Port `53053`.

   Background: The network `127.x.y.z` is an address range reserved for local (*i.e.,* on the host) communication, details see RFC 1122. This basically means that you can send and receive packets only within the host.

3. **Packet Format**

   DNS server and client should exchange UDP packets. The transport layer payload should be JSON strings, to ease implementation (*e.g.,* to be independent of byte ordering). The following keys be at least supported:

```
dns.flags.response
dns.flags.recdesired
dns.qry.name
dns.qry.type
dns.flags.rcode
dns.count.answers
dns.flags.authoritative
dns.a
dns.ns
dns.resp.ttl
```

For further details see `https://www.wireshark.org/docs/dfref/d/dns.html`.

4. **Recursive Resolver & Cache & Network Delay**

   The recursive resolver should listen to `127.0.0.10` and UDP Port `53053`. It supports a simple caching algorithm which respects the TTLs of the resource records, *i.e.,* an entry is removed after the TTL expires.

   In order to make the beneficial effects of caching measurable, we have to introduce an artificial network delay of at least 100ms. This means that each send operation must sleep for at least the duration of the delay and only then send the DNS message. Please note that this delay is required for all send operations including the stub and authoritative servers.

5. **Names**

```
switch.telematik
www.switch.telematik
mail.switch.telematik

router.telematik
news.router.telematik
shop.router.telematik

homework.fuberlin
easy.homework.fuberlin
hard.homework.fuberlin

pcpools.fuberlin
linux.pcpools.fuberlin
macos.pcpools.fuberlin
windows.pcpools.fuberlin
```

6. **Log Files**

   Each DNS server should produce a log file called `<ip address>.log`, where `<ip address>` denotes the IP address of the server. The log file should include number of requests sent and received and the number of replies sent and received. It should be formated as follow:

   `<Epoch Timestamp>|<IP address>|<#req snd>|<#req rcv>|<#resp snd>|<#resp rcv>`

7. **Milestones**

(a) Your stub resolver is able to (directly) request an `A` record from the authorative server.

(b) Your recursive resolver is able to discover the authoritative server of a name, and resolve the `A` record for this name.

(c) Your stub resolver is able to resolve any name in the list via the recursive resolver and profits from faster replies in the case of cache hits at the recursive resolver.

(d) Your DNS implementation is used by an application (see HTTP proxy below).

8. **Submission**

   - Please include a concise project description.
   - Document the setup and run commands.
   - Prepare a test script against your recursive resolver.

HTTP Proxy

A HTTP proxy server is an intermediary for HTTP requests from clients. In this exercise you will implement such proxy and interconnect it with your own DNS. You can reuse available HTTP libraries.

- Run a simple HTTP server serving a website for one of the sites above. This webserver needs to be reachable under the address which is stored in your DNS, *e.g.,* IP address `127.0.0.80` and port `8080`.
- Create a second HTTP server which acts as a proxy. Incoming `GET` requests are inspected and the proxy distinguishes between two cases:
  - Requests for names within your DNS (`.telematik, .fuberlin`) trigger a local name resolution. The website is than requested from the respective answer IP address and eventually forwarded to the client of the HTTP proxy.
  - Requests for all other names simply trigger a `GET` request for that name. This means that you do not need to resolve these names, usually this will be done automatically by your operating system. Again, the answer is forwarded to the client of the proxy.