Naming and Network Topology

Networked Systems (H) 2021-2022 – Laboratory Exercise 5
Dr Colin Perkins, School of Computing Science, University of Glasgow

1 Introduction

The laboratory exercises for Networked Systems (H) will introduce you to network programming in C using the Berkeley Sockets API, and help you understand the operation and structure of the network. The exercises will help you practice C programming, building on the Systems Programming (H) course, and introduce network programming in C. Other exercises will illustrate key points in the operation of the network. The laboratory exercises are intended to complement the material covered in the lectures. Some expand on the lectures to give you broader experience in a particular subject. Others exercises cover material, such as network programming in C, that's better taught by doing than by lecturing.

There are a mixture of formative and summative exercises. The formative exercises will give you practice in programming networked systems in C, and allow you to gain further experience in C programming. They are not assessed. The two summative exercises will focus on security and protocol ossification, and on understanding the network topology. They are assessed and together are worth a total of 20% of the marks for this course.

In this laboratory exercise you will investigate the mapping from DNS names to IP addresses, and explore the IPv4 and IPv6 router level topology of the Internet. **The second assessed exercise is based on this laboratory exercise.**

Part 1: IP Addresses of Popular Websites

The Internet separates naming from addressing. The network operates using IP addresses, that denote locations in the network, while users prefer to use readable domain names. The domain name system (DNS) is an application that runs of the network, and translates names into addresses.

Popular web sites and services are implemented using multiple servers, hosted at different locations in the network. This increases robustness of the sites to failures, and also allows for load balancing across different data centres and network connections. To support this, domain names often correspond to more than one IP address. These addresses can represent servers in different locations in the network, or perhaps connected using different protocols, such as IPv4 and IPv6.

Write a program, `dnslookup`, that takes a list of one or more DNS names on the command line, and performs a DNS lookup for each name (hint: use `getaddrinfo()` as in the laboratory exercise 1 client, replacing the `socket()` and `connect()` calls with a call to `inet_ntop()` to print the IP address). After each DNS lookup, your program should print out the list of IP addresses returned, prefixed with the DNS name and address type. For example, your program might print the following, if asked for the addresses of `www.google.com`:

```
$ ./dnslookup www.google.com
www.google.com IPv4 172.217.23.36
www.google.com IPv6 2a00:1450:4009:801::2004
$
```

The number and type of IP addresses returned for a given name might vary, depending on the date and location from which you perform the lookup. The format of the output from your program should match this example, with the host name, address type, and IP address on each line, separated by spaces.

Your program **must** be written as a single file, `dnslookup.c`. This **must** compile without errors or warnings, using the command `clang -W -Wall dnslookup.c -o dnslookup` run on the `stlinux` machines provided by the School (note that the letter W is capitalised in the two places it appears this command).

Use your `dnslookup` tool to find the IP addresses for a range of websites. These should be a mix of academic, commercial, social network, news, and personal websites, from a wide range of different countries. You should find the addresses of around 15-20 different websites. You should run your program on the `stlinux` machines, since they support both IPv4 and IPv6.

Part 2: The Router-level Network Topology

The `traceroute` tool can discover the network path used to reach a particular destination. For example, a traceroute from one of the `stlinux` machines to the IPv6 address of `www.google.com` previously found might show:

```
$ traceroute -6 -q 1 -n 2a00:1450:4009:801::2004
traceroute to 2a00:1450:4009:801::2004 (2a00:1450:4009:801::2004), 30 hops max, 80 byte packets
 1  2001:630:40:f00:e22f:6dff:fe2c:ed80  0.276 ms
 2  2001:630:40:20::11  0.298 ms
 3  2001:630:40:20::72  0.723 ms
 4  2001:630:0:900c::1  0.587 ms
 5  2001:630:0:10::185  1.016 ms
 6  2001:630:0:10::1fd  5.337 ms
 7  2001:630:0:10::1f1  7.400 ms
 8  2001:630:0:10::1dd  11.276 ms
 9  2001:630:0:10::1c9  11.742 ms
10  *
11  2001:4860:0:1::ca1  12.514 ms
12  2001:4860:0:1::2b3  12.918 ms
13  2a00:1450:4009:801::2004  12.234 ms
```

The `-6` option means trace to an IPv6 address (use `-4` to trace to an IPv4 address). The `-q 1` option means only probe each hop once, the `-n` option means don't try to look-up the DNS names for the routers. Each line identifies one router on the path from the source to the destination. It gives the distance in hops, the IP address of the router, and the time taken for the response. The first router (2001:630:40:f00:e22f:6dff:fe2c:ed80 in this example) is the device directly connected to the sender. That connects, in turn, to a router with IP address 2001:630:40:20::11, which connects to 2001:630:40:20::72, and so on. A * response (as in hop 10 of the example) indicates that no response was received. A long sequence of * responses, sometimes seen at the end of a trace, either indicates a firewall that is blocking traceroute requests at that point or that the destination host has been configured to not respond to traceroute requests.

Run `traceroute` to each of the IP addresses you discovered using your `dnslookup` tool in Part 1 of this exercise, redirecting the output into files, with separate files for IPv4 and IPv6 traceroute output. You might find it helpful to write a script to do this. This will give you a record of the IP addresses of the routers that data traverses to reach each of the chosen sites (or, if there's a firewall, until it reaches the firewall that blocks the traceroute).

Write a program, in a language of your choice, that reads the saved traceroute files, strips out any hops that didn't respond to traceroute (lines with * responses), and reformats them as a list of pairs of adjacent router addresses, in quotes, with each pair of addresses being separated by `--`. For example, if run on the traceroute output shown above, this program would give the following output:

```
"2001:630:40:f00:e22f:6dff:fe2c:ed80"  -- "2001:630:40:20::11"
"2001:630:40:20::11"  -- "2001:630:40:20::72"
"2001:630:40:20::72"  -- "2001:630:0:900c::1"
"2001:630:0:900c::1"  -- "2001:630:0:10::185"
"2001:630:0:10::185"  -- "2001:630:0:10::1fd"
"2001:630:0:10::1fd"  -- "2001:630:0:10::1f1"
"2001:630:0:10::1f1"  -- "2001:630:0:10::1dd"
"2001:630:0:10::1dd"  -- "2001:630:0:10::1c9"
"2001:630:0:10::1c9"  -- "2001:4860:0:1::ca1"
"2001:4860:0:1::ca1"  -- "2001:4860:0:1::2b3"
"2001:4860:0:1::2b3"  -- 2a00:1450:4009:801::2004"
```

Run your program on each of the saved traceroute files in turn, saving the results from each into a separate file (called a "processed traceroute file"). Then, concatenate the contents of all your processed traceroute files for IPv4 addresses into a single file called `router-topology-v4.dot`. Sort this file into order based on the first IP address on each line, then remove duplicate lines (hint: `cat ... | sort | uniq > ...` in the command shell). Then, add a line containing:

```
graph routertopology {
```

to the beginning, and a line containing just:

```
}
```

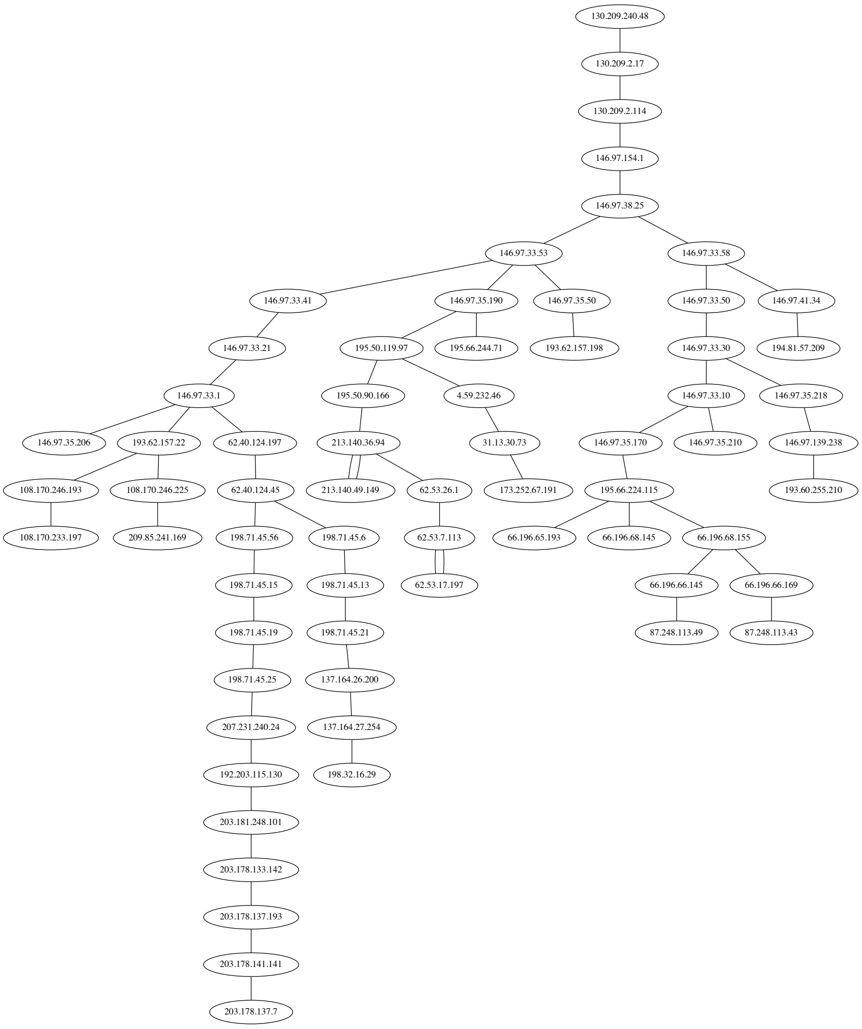

Figure 1: Sample router topology visualisation from a single site

at the end. The resulting `router-topology-v4.dot` file is a description of the router-level network topology, and should be a valid file in the *dot* graph description language.[1] Do the same with your saved IPv6 traceroute files, to produce `router-topology-v6.dot`.

Use the `dot` program (or one of its variants such as `neato`, `twopi`, or `circo`), installed on the `stlinux` machines and part of the Graphviz suite of tools[2] (or some other graph plotting program of your choice) to generate a router level map of the router level network topology. You should produce two PDF files, called `router-topology-v4.pdf` and `router-topology-v6.pdf`, from the IPv4 and IPv6 processed traceroute files. If you use `dot`, you should run a command of the form

```
dot -T pdf -o router-topology-v4.pdf router-topology-v4.dot
```

to produce the output. Figure 1 shows an example an IPv4 router level topology map generated using `dot`.

Part 3: Understanding Network Topology and the `traceroute` Tool

Review the IP addresses found for each site in Part 1 of this exercise, and the router level network topology maps you prepared in Part 2, and write up a short report, not to exceed two sides of A4 paper when printed using a 10pt font, that discusses the following points:

[1] https://en.wikipedia.org/wiki/DOT_(graph_description_language)

[2] http://www.graphviz.org/

- **IP addresses:** Looking at the addresses found for the different sites in Part 1, do some sites have more than one IP address, and what does it mean if they do? If you run your `dnslookup` program several times, do you always get the same IP addresses for a site, and if not, why might this be? Do you get the same IP addresses for a site if you run your `dnslookup` program from different locations, and if not, why? What proportion of sites have an IPv6 address?

- **Router-level Topology Maps:** Looking at the router level topology maps you produced in Part 2 of this exercise, what is the longest path you can find in the network? Are paths from different locations to the same destination disjoint or do they join-up somewhere before the destination? Are there multiple routes to some destinations? Can you infer anything about organisational boundaries (e.g., which parts of the network are operated by different ISPs) from changes in the IP address prefixes?

- **IPv4 and IPv6:** Do the IPv4 and IPv6 router level network topology maps match? The addresses will be different, of course, but do the topology maps have the similar structure? Think about whether you expect that they should have similar structure – is there any reason why the IPv4 and IPv6 networks should match?

- **The `traceroute` Tool:** How does the `traceroute` tool work? Do some background reading to understand how `traceroute` can find the IP addresses of the routers on the path to a destination. You should learn how the TTL is varied during a traceroute; what is ICMP, and how does it relate to IP, UDP, and TCP; and how traceroute uses ICMP.

Prepare this report using the word processing package of your choice, formatted for A4 size paper, and using the Times font in 10pt. Save a copy of your report in PDF format, under the filename `report.pdf`. You are encouraged to discuss the answers to these questions with the lecturer or lab demonstrators to make sure you understand the issues raised.

- + -