University of St Andrews School of Computer Science

CS2003 — Internet and the Web — 2020/21

Exercise 3: DNS and UDP

Date: 2020-10-01

As with all lab exercises, this exercise is not directly assessed. It is intended to complement the material presented in lectures and may turn up in the exam. You are encouraged to complete it in your own time if you do not finish during the scheduled exercise class time. *You are also encouraged to work with other people in the class to complete the exercises — you can also work on the exercise classes during lab sessions and use the relevant Teams channels to discuss things with each other or to ask for help from a demonstrator.*
The aim of this exercise class is to use the Linux command line interface (CLI) and some simple Java programs to explore some of the material covered in lectures on DNS and UDP. By now you should be familiar with logging in to the lab clients, so please do this before attempting any of the exercises.

1 DNS

`dig` is a command line tool that can send queries to the DNS. In this activity, you will use dig to observe the request-response behaviour of DNS.

1. Use `dig` to look up the IPv4 address for the host `www.caida.org`, by typing:

   ```
   $ dig www.caida.org
   ```

   In the output, the part marked '`;; ANSWER SECTION:`' contains an entry in the format:

   ```
   <hostname>  <TTL>   IN    A     <IPv4 address>
   ```

   Note the value of `<TTL>`, and the value of `Query time:`, at the end of the response.

2. Run the `dig` command again, ≈30s after the first run. Make a note of the `<TTL>` value, and the `Query time:` value, again.

3. Wait at least another 40s and run the `dig` command yet again, noting once more the value of both `<TTL>` and `Query time:`.

4. What do you think is happening? (You might not see exactly what you expect to see if other students are performing this same exercise at the same time as you – see if you can explain why!)

   Remember that you can type `man dig` to find out more about `dig`.

2 More DNS

For this section you may wish to refer to the textbook at https://book.systemsapproach.org/applications/infrastructure.html#name-service-dns

1. Use `dig` to look up the DNS records for `teams.microsoft.com`.

   a) What is a CNAME record? Why do you think these are used for Microsoft Teams?

2. Application users might sometimes wish to submit partial hostnames rather than FQDNs for simplicity. For instance if you are on the CS network, you might wish to just refer to `trenco`, rather than the FDQN `trenco.cs.st-andrews.ac.uk`.

   a) Do a DNS lookup for `trenco`:

   ```
   $ dig +search trenco
   ```

   (you can search the `dig` manual pages for “+[no]search” to see why I have used this command-line argument)

   and also search for the FQDN:

   ```
   $ dig +search trenco.cs.st-andrews.ac.uk
   ```

   b) Now try to search for the (non-existent) host `trenco2`:

   ```
   $ dig +search trenco2
   ```

   and:

   ```
   $ dig +search trenco2.cs.st-andrews.ac.uk
   ```

How does the output differ and why do you think this is? Examine the file `/etc/resolv.conf` and the manual page `man resolv.conf` to find out more.

3. Refer back to the DNS name resolution slide (slide 27 of https://studres.cs.st-andrews.ac.uk/CS2003/Lectures/cs2003_net-04_internet_protocol.pdf). Now look at `/etc/resolv.conf` on one of the host servers. What are the IP addresses of that host server's local resolvers? Why does the `/etc/resolv.conf` file require IP addresses for resolvers and not hostnames?

3 UDP

There is code on studres for some simple unicast UDP client-server applications.
Please examine the code in `CS2003/Examples/CS2003-Examples-wk03/code/udp/`, especially looking at the use of the Java classes `DatagramSocket` and `DatagramPacket`.
`UdpClient1` and `UdpServer1` simply demonstrate the transmission and reception of a UDP packet, with the server printing out relevant information about the UDP packet.
`UdpClient1a` and `UdpServer1a` extend this a little, with a trivial session protocol: using the string "quit" to terminate the session (as well as terminate the server).
`UdpSocket` is an abstraction that combines `DatagramSocket` and `DatagramPacket` to represent a UDP communication endpoint.
Note that there is not such a big difference in the 'client' and 'server' code, as there was for TCP. `UdpChat` demonstrates this, by showing how a single program can have both 'client' and 'server' roles, accepting messages and sending messages.

1. Compile the example code — you will need to edit the code appropriately to do this.
2. Run `UdpServer1`, and then run `UdpClient1` to communicate with the running server. Note and discuss your observations.
3. Without `UdpServer1` running, run `UdpClient1` to the same name/port as the server was running on before. Note and discuss your observations.
4. Run `UdpServer1a` , and then run `UdpClient1a` to communicate with the running server. Note and discuss your observations.
5. Run the `UdpChat` program with another student. Note and discuss your observations and discuss the operation of the program.

4 Supplementary

We have seen in the first practical that providing command-line parameters to applications can make it useful for both users and developers, since this allows run-time rather than compile-time configuration. There is some other example code on studres that you might find useful for future practicals. Please do look at the code in `CS2003/Examples/CS2003-Examples-wk03/code/supplementary/` to see how it works, and discuss it with others.

4.1 Configuration

It is often necessary to supply initial values for certain parameters for configuring an application, especially servers, but clients also. The Java class `Configuration` provides an example of how this can be done in Java. The Java approach is to use `Properties`, which are simply name/value bindings in a file. This is not the only way of doing this: it is possible to use other configuration file formats, e.g. JSON (https://json.org/) or YAML (https://yaml.org), or even something custom designed and built for a particular application.

4.2 LogFileWriter

The Java class `LogFileWriter` provides a very simple mechanism for writing information to a file from any program. Servers often need logs for various reasons, e.g. performance monitoring, measuring usage / load, etc. That said, logging is useful for many general purposes, including for debugging.

4.3 AddressInfo

The Java class `AddressInfo` is a simple demonstration of a `ServerSocket`, but also gives some information you might find useful. If you run the server, you can connect to it using the `nc` command-line program.

1. `TestConfiguration` shows a very simple example of the `Configuration` class. Try to adapt `UdpServer1` so that it can read in configuration information from `server.properties` and use a configured port number, print out `ServerName_` on startup, and log to a file.

Tristan Henderson, with thanks to Saleem Bhatti