Systems Integration

DNS Part 1: Building a DNS resolver

6th October 2021

In today's lab, we'll take a look at DNS, by building a simple DNS resolver. This resolver will allow us to lookup both IPv4 and IPv6 addresses.

Installing a Linux VM

You are, of course, free to run today's lab on your local machine. However, you will need a Linux server both for future labs, and for the assignment, so it's a good idea to set one up if you haven't already. If you already have a Linux server set up, you can skip this section.

Since you only need the command line for this course, you should only need to install a command-line based Linux distribution. The one that is recommended for this course is Ubuntu server. Download it from:

https://ubuntu.com/download/server

When you open the Ubuntu site, click 'Option 2 - Manual server installation'. Download the Ubuntu Server 20.04 ISO, and install it on a VM such as VirtualBox. You'll probably have done this in past modules, but I'll outline the process below in case anyone isn't sure. The installation should usually take about 30 to 40 minutes to complete.

Run VirtualBox, and click ‘New’ at the top to create a new VM. You will be asked to make a few choices when setting up the VM:

- Give the VM a name of your choice
- Set the type to 'Linux' and the version to 'Ubuntu (64-bit)'
- Unless you are using a very low-end computer, try to set about 1GB (1024MB) of memory for the VM
- Create a hard disk that's about 10GB

It's also a good idea to set your network settings before going any further -- click 'Settings' and then 'Network' and then set the following in the 'Adapter 1' tab:

- Make sure 'Enable network adapter' is checked
- 'Attached to' is set to 'Bridged adapter'
- 'Cable connected' is checked (you might need to click 'Advanced' to see this)

Save the settings and then run the VM by clicking ‘Start’ at the top of VirtualBox. The first time you run the VM, you will be given the option to choose an ISO file to boot from. Use this to select the Ubuntu ISO you downloaded earlier.

You can generally just use all of the default settings when installing, but there are a few things to look out for:

- Make sure you select the right keyboard layout – Ubuntu will default to an American keyboard, so make sure you select the Irish or UK layout (or whatever keyboard you have on your computer).

- Make sure you remember the name you gave the server, as well as the username and password you set up. Since you will be the only person using the VM pick a password you’ll remember instead of one that is secure! :-)
- The installation will give you the option to install an SSH server. Say yes to this, it will make life easier later on!

Once Ubuntu is installed, there's some software that you'll want to install that we'll be using in this lab. The command to install it is:

```
sudo apt install nano python-is-python3
```

As you can see, I'm installing Nano as a text editor. This is a personal preference: if you prefer vim or something else (Emacs if you're really brave!) feel free to use it instead.

Parsing DNS messages

If you check Brightspace, you'll see that I've uploaded a Python file called *dns.py*. This is a small library that I've written that provides basic DNS parsing. If you open it up, you'll see that it defines four Python dataclasses:

- `DNSHeader`
- `DNSQuestion`
- `DNSAnswer`
- `DNSDatagram`

These correspond to the DNS datagram formats discussed during the lecture. The file also contains functions that can convert raw Python bytes objects into instances of the `DNSDatagram` class. Each `DNSDatagram` contains a header, zero or more questions, and zero or more answers. Here's an example Python script which uses the dns module to parse an example DNS datagram into the objects and display it:

```
import dns


dns_message = b'\x00{\x81\x80\x00\x01\x00\x02\x00\x00\x00\x00\
x08tudublin\x02ie\x00\x00\x01\x00\x01\xc0\x0c\x00\x01\x00\x01\x00\
x00\x02X\x00\x044\x1e\xe0\x8f\xc0\x0c\x00\x01\x00\x01\x00\x00\
x02X\x00\x04?"\xf1\xcc'


print(dns.read_dns_datagram(dns_message))
```

There's also a function called `make_dns_datagram()`, which goes in the opposite direction: it takes a `DNSDatagram` object as input, and returns a bytes object containing the encoded DNS message. You will also see that `make_*()` and `read_*()` functions also exist for headers, questions, answers and also for the label encodings. Labels are stored inside the library as a list of strings, with one string for each label (so `tudublin.ie` becomes `['tudublin', 'ie']` inside the library).

Sending a DNS request

Let's make a DNS request, send it to a read DNS server, and parse the result we get back.

We start by creating a DNS question asking for the IPv4 address for the `tudublin.ie` domain:

```
import dns


question = dns.DNSQuestion(
    qname = ['tudublin', 'ie'],
    qtype = 1, # QTYPE = A
    qclass = 1
)
```

Now, add a header to the script:

```
header = dns.DNSHeader(
    ident = 1000, # You can pick any number you want for this!
    qr = 0, # Set QR to zero to represent a query
    opcode = 0,
    aa = 0,
    tc = 0,
    rd = 1, # Request recursion
    ra = 0,
    z = 0,
    rcode = 0,
    qdcount = 1, # We have one question!
    ancount = 0,
    nscount = 0,
    arcount = 0
)
```

Finally, create the entire datagram, containing the question and header:

```
datagram = dns.DNSDatagram(
    header = header,
    questions = [question],
    answers = []
)
```

The datagram can be converted into a Python bytes object with:

```
datagram_bytes = dns.make_dns_datagram(datagram)
```

Google maintain a DNS server at the IP address `8.8.8.8`. We can send our request to port 53 of this server this using Python's sockets API, and then use `read_dns_datagram()` to parse the results. Note that `8.8.8.8` seems to be blocked on the eduroam network used in TU Dublin, so you may need to use the nslookup command to find the IP for a recursive server that works in this context!

```
import socket


destination = ('8.8.8.8', 53)
connection = socket.socket(family=socket.AF_INET,
                           type=socket.SOCK_DGRAM)
connection.sendto(datagram_bytes, destination)


result = connection.recvfrom(4096)[0]


print(result)


print(dns.read_dns_datagram(result))
```

Modify the script to make the following three changes:

- Iterate through the answers attribute of the DNSDatagram object returned, and print out each of the IP addresses in the usual `a.b.c.d` format. Recall that the address will be found in the `rdata` attribute of each object (½ mark).
- Allow users to enter a domain as a command line argument, rather than relying on the hardcoded tudublin.ie domain. Recall that you'll have to split the string into a list at the dots for this: the `split()` function built in to Python strings can help you here! (½ mark)
- The `dns` module should have no problem parsing IPv6 DNS requests and responses. You can modify your code to make it support those too. Can you remember the things that need to change from the lecture? (1 mark)

Linux has a DNS resolver called `dig` built into it. It works the same as resolver you just built: for example you can type `dig tudublin.ie`. Try using it. Does it return the same IP addresses?

Submission

This lab is worth 2 marks. Make the three changes to the simple resolver described in the last section, and upload your finished resolver to Brightspace to complete the lab. You can upload the Python code (.py files) directly, or you can put you code in a Word or PDF document and upload it.

Upload to Brightspace by Friday 22nd October 2021.