UNIVERSITY OF TECHNOLOGY SYDNEY

49202: Communication Protocols
Laboratory Notes: Laboratory 2

Dr. Daniel R. Franklin

March 3, 2021

Contents

1 Introduction

This laboratory covers material from Module 2, covering the application layer. The main aims of this laboratory are to:

1. To observe some common application layer protocols in action in the laboratory using Wireshark;
2. To measure throughput from the perspective of an application; and
3. To implement some simple TCP/IP servers and clients in Python

2 Setup

Prior to starting the lab, start your virtual machine (refer to Lab 1 notes) and make sure you have the latest version of the lab scripts:

```
cd ~/comm-protocols-labs

git pull
```

Start your VM (as described in Lab 1), open a terminal window (note: you may always open more than one terminal window if desired using ctrl-shift-N).

For most of the labs in this subject, we will be using a packet capture application or "packet sniffer" called **Wireshark**, an open source utility program that can be obtained for Windows, Mac OS X and Linux operating systems for free (see `https://www.wireshark.org/`).

Today, we will be using a generic OSPF-based network topology with full Internet access, as shown in Figure 1.

All links are assigned an equal cost - therefore, traffic will always take the path with the smallest number of hops (the minimum total cost - hence *open shortest path first*). The core routers are denoted `r1-r8`; `r1` is also connected to a network address translation gateway `r9` which provides access to the Internet for all hosts in the network. In addition, `r1` is our DNS server - it translates hostnames into IP addresses for hosts in our network (and forwards requests for other hostnames onwards to your upstream DNS server). Each of the core routers has one host attached to it - these are denoted `h1-h8`, with the number corresponding to the connected router (for example, `h3` is connected to `r3`).

To start the Mininet emulated network with OSPF, enter the folder `~/comm-protocols-labs/ospf_ring` if you haven't already done so (see Lab 1 for instructions on how to change directory in a Linux system), and run the command

```
sudo ../start.py
```

Wait a few moments until you can see the **mininet** prompt (it should look like this: `mininet>`). This indicates that all hosts and services are running. At this point, can open a shell on any of the hosts.

You can open shells on multiple hosts and/or routers - for example, to open a shell on `h3` and `r1`, we use the `xterm` command at the mininet prompt:

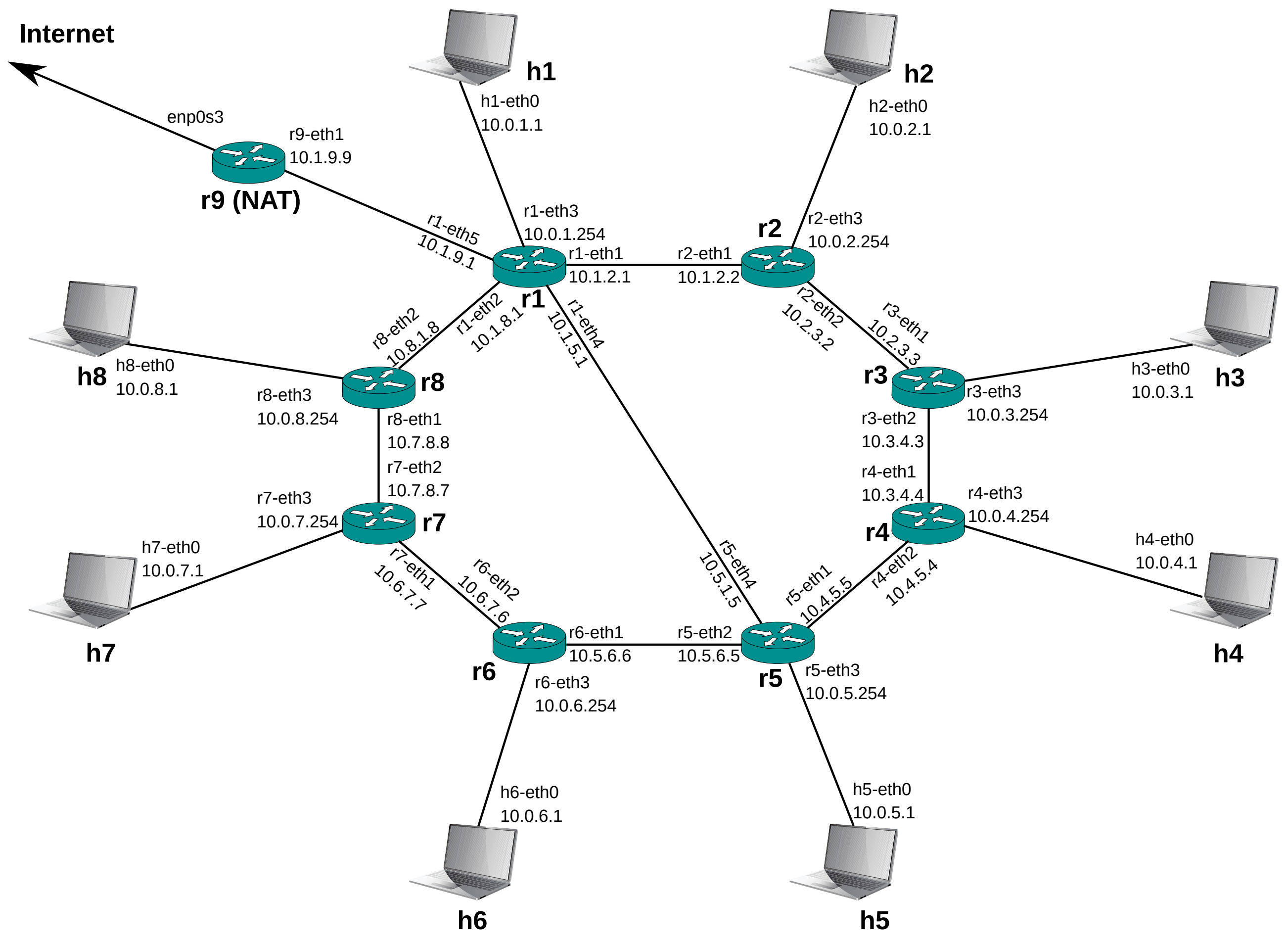


Figure 1: Network topology for Lab 2. The routing protocol is OSPF.

```
xterm h3

xterm r1
```

You can even use `xterm h3 r1` to open them both simultaneously if you want.

For this lab, you should open shells on at least `r1` plus one other router and its connected host. For the remainder of this document, it is assumed that you are primarily using the pair of `r3` and `h3` unless otherwise specified (but any host and its associated router can be used).

It will take about 30 seconds (after starting mininet) for the network to achieve full connectivity - this is the time needed for OSPF to fully converge across all routers in the network. You can test this using the `ping` command (more on this later). For example, running the following command on `h3` allows us to verify connectivity with `r3`:

```
ping -c 5 10.10.10.3
```

This command sends 5 ping packets to the router `r3` via its loopback system address `10.10.10.3`. You could also use one if its interface addresses (e.g. `10.0.3.254`). Try pinging some of the other nodes in the network, including the gateway node. If you don't see any replies, you may need to wait a few more seconds for OSPF to fully converge. Note: if you don't specify a ping count via the `-c` option, `ping` will continue until interrupted via `ctrl-c`.

You can also ping a host or router by its name, for example:

```
ping r1
```

```
ping google.com
```

NOTE: If you are on campus, you won't be able to ping `google.com` due to restrictions on the border gateway. The timetable server is hosted internally however - so from inside UTS you can try to ping `mytimetable.uts.edu.au`. Don't try to ping `www.uts.edu.au`, however, as this WILL NOT WORK from within the UTS network because the University hosts its public website on the Amazon cloud! It should work fine from your home network.

Identify and record all of the interface MAC addresses of `r1`, `r3` and `h3`, together with their IP addresses. There are several Linux command which you may use for this purpose, including:

```
ip addr
```

which will produce the following output (for example, on `h6`:

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: h6-eth0@if3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc htb state UP group default qlen 1000
    link/ether 52:c4:44:86:73:c6 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.0.6.1/24 brd 10.0.6.255 scope global h6-eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::50c4:44ff:fe86:73c6/64 scope link
       valid_lft forever preferred_lft forever
```

in this case, the MAC address of the first physical interface `h6-eth0` is `52:c4:44:86:73:c6`, and the IP address/netmask of the interface is `10.0.6.1/24`. The other interface (`lo`) is a loopback interface, and does not correspond to any physical hardware. It provides a mechanism for TCP/IP clients and servers to communicate with each other on a single machine, even if it has no physical network connection; the IP address associated with this interface is normally `127.0.0.1/8` and it can normally be accessed via the alias `localhost` (e.g. `ping localhost`).

You can also use the slightly more verbose but easier to read command:

```
ifconfig
```

3 Getting Started with Wireshark

3.1 Capturing and Analysing Protocol Data Units (PDUs)

Start `wireshark` on your chosen host (e.g. `h3`) by typing `wireshark` at the command prompt (in the xterm you have opened for `h3`). To start a capture, click the blue sharkfin icon on the left-hand side of the toolbar or go to **Capture** → **Start**.

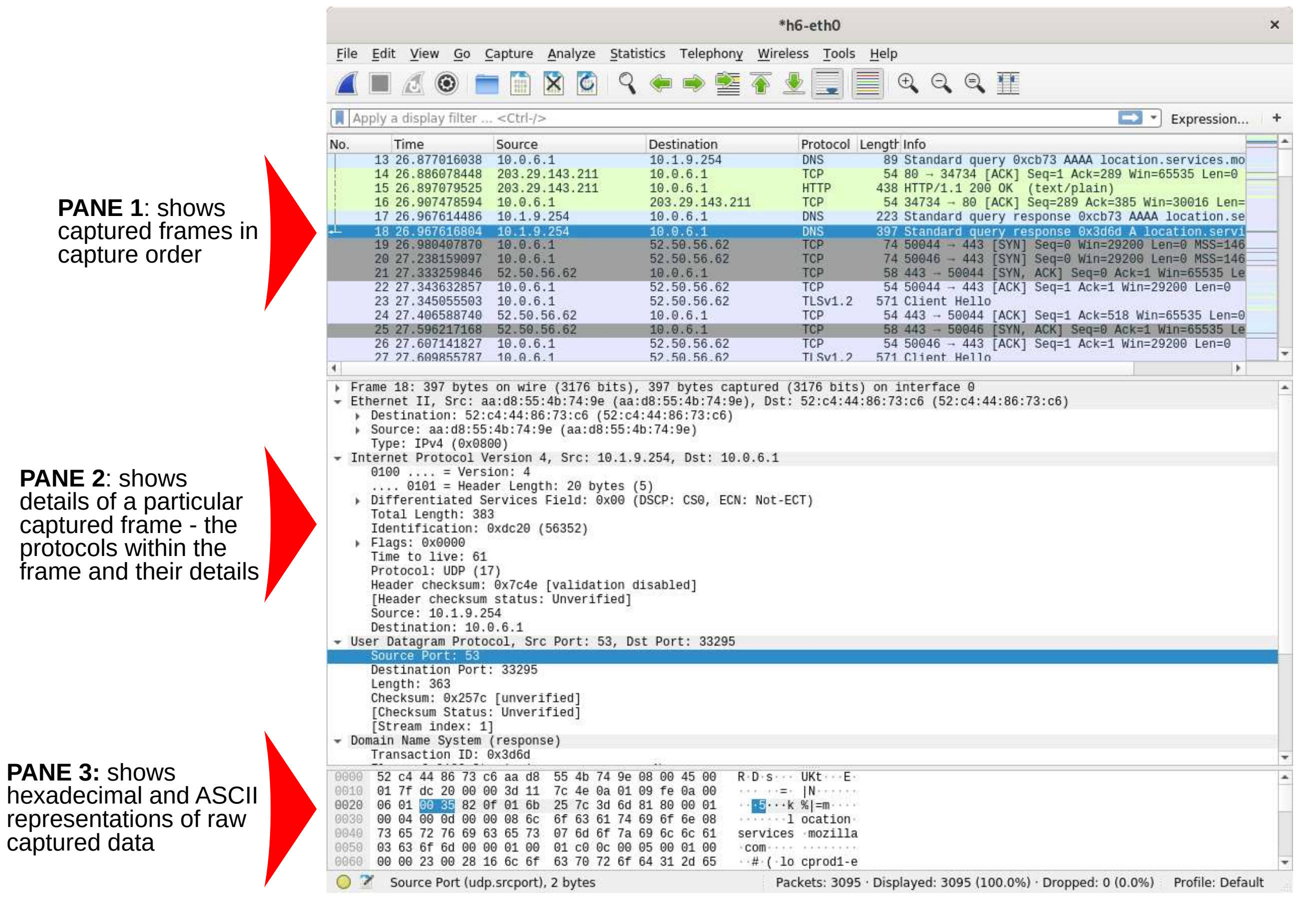
Next, open an instance of Firefox on the host - note that you MUST run this as the mininet user rather than root, otherwise it will not work:

```
sudo -u mininet firefox
```

Load a website such as `https://www.uts.edu.au/`. Once the website has finished loading, stop the capture (click the second-to-left icon on the toolbar). Now you can examine all packets which were exchanged during this session.

The screen is divided into 3 main areas (as shown in Figure 2:

1. The top pane shows the frames that have been captured on the selected interface in order of capture and some summary information about each frame;
2. The middle pane shows the details of a particular frame that has been highlighted (clicked on) in the top pane. In this middle pane, you are able to "drill down" into frames and see the details


PANE 1: shows captured frames in capture order
PANE 2: shows details of a particular captured frame - the protocols within the frame and their details
PANE 3: shows hexadecimal and ASCII representations of raw captured data
*h6-eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...

```
No.  Time          Source          Destination     Protocol Length Info
  13 26.877016038  10.0.6.1        10.1.9.254      DNS       89 Standard query 0xcb73 AAAA location.services.mo
  14 26.886078448  203.29.143.211  10.0.6.1        TCP       54 80 → 34734 [ACK] Seq=1 Ack=289 Win=65535 Len=0
  15 26.897079525  203.29.143.211  10.0.6.1        HTTP     438 HTTP/1.1 200 OK  (text/plain)
  16 26.907478594  10.0.6.1        203.29.143.211  TCP       54 34734 → 80 [ACK] Seq=289 Ack=385 Win=30016 Len=
  17 26.967614486  10.1.9.254      10.0.6.1        DNS      223 Standard query response 0xcb73 AAAA location.se
  18 26.967616804  10.1.9.254      10.0.6.1        DNS      397 Standard query response 0x3d6d A location.servi
  19 26.980407870  10.0.6.1        52.50.56.62     TCP       74 50044 → 443 [SYN] Seq=0 Win=29200 Len=0 MSS=146
  20 27.238159097  10.0.6.1        52.50.56.62     TCP       74 50046 → 443 [SYN] Seq=0 Win=29200 Len=0 MSS=146
  21 27.333259846  52.50.56.62     10.0.6.1        TCP       58 443 → 50044 [SYN, ACK] Seq=0 Ack=1 Win=65535 Le
  22 27.343632857  10.0.6.1        52.50.56.62     TCP       54 50044 → 443 [ACK] Seq=1 Ack=1 Win=29200 Len=0
  23 27.345055503  10.0.6.1        52.50.56.62     TLSv1.2  571 Client Hello
  24 27.406588740  52.50.56.62     10.0.6.1        TCP       54 443 → 50044 [ACK] Seq=1 Ack=518 Win=65535 Len=0
  25 27.596217168  52.50.56.62     10.0.6.1        TCP       58 443 → 50046 [SYN, ACK] Seq=0 Ack=1 Win=65535 Le
  26 27.607141827  10.0.6.1        52.50.56.62     TCP       54 50046 → 443 [ACK] Seq=1 Ack=1 Win=29200 Len=0
  27 27.609855787  10.0.6.1        52.50.56.62     TLSv1.2  571 Client Hello
```

```
▸ Frame 18: 397 bytes on wire (3176 bits), 397 bytes captured (3176 bits) on interface 0
▾ Ethernet II, Src: aa:d8:55:4b:74:9e (aa:d8:55:4b:74:9e), Dst: 52:c4:44:86:73:c6 (52:c4:44:86:73:c6)
  ▸ Destination: 52:c4:44:86:73:c6 (52:c4:44:86:73:c6)
  ▸ Source: aa:d8:55:4b:74:9e (aa:d8:55:4b:74:9e)
    Type: IPv4 (0x0800)
▾ Internet Protocol Version 4, Src: 10.1.9.254, Dst: 10.0.6.1
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▸ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 383
    Identification: 0xdc20 (56352)
  ▸ Flags: 0x0000
    Time to live: 61
    Protocol: UDP (17)
    Header checksum: 0x7c4e [validation disabled]
    [Header checksum status: Unverified]
    Source: 10.1.9.254
    Destination: 10.0.6.1
▾ User Datagram Protocol, Src Port: 53, Dst Port: 33295
    Source Port: 53
    Destination Port: 33295
    Length: 363
    Checksum: 0x257c [unverified]
    [Checksum Status: Unverified]
    [Stream index: 1]
▾ Domain Name System (response)
    Transaction ID: 0x3d6d
```

```
0000  52 c4 44 86 73 c6 aa d8  55 4b 74 9e 08 00 45 00   R·D·s··· UKt···E·
0010  01 7f dc 20 00 00 3d 11  7c 4e 0a 01 09 fe 0a 00   ··· ··=· |N······
0020  06 01 00 35 82 0f 01 6b  25 7c 3d 6d 81 80 00 01   ···5···k %|=m····
0030  00 04 00 0d 00 00 08 6c  6f 63 61 74 69 6f 6e 08   ·······l ocation·
0040  73 65 72 76 69 63 65 73  07 6d 6f 7a 69 6c 6c 61   services ·mozilla
0050  03 63 6f 6d 00 00 01 00  01 c0 0c 00 05 00 01 00   ·com···· ········
0060  00 00 23 00 28 16 6c 6f  63 70 72 6f 64 31 2d 65   ··#·(·lo cprod1-e
```

```
Source Port (udp.srcport), 2 bytes        Packets: 3095 · Displayed: 3095 (100.0%) · Dropped: 0 (0.0%)    Profile: Default
```

Figure 2: Main window of Wireshark, after capturing some packets.

of the encapsulated protocols and the fields and values within these protocols; and

3. The bottom pane shows the hexadecimal and ASCII text representations of the captured data. This is useful for unencrypted protocols, where you may be able to find text strings and other interesting data.

Looking at the top pane in more detail, we can see the following columns (left to right):

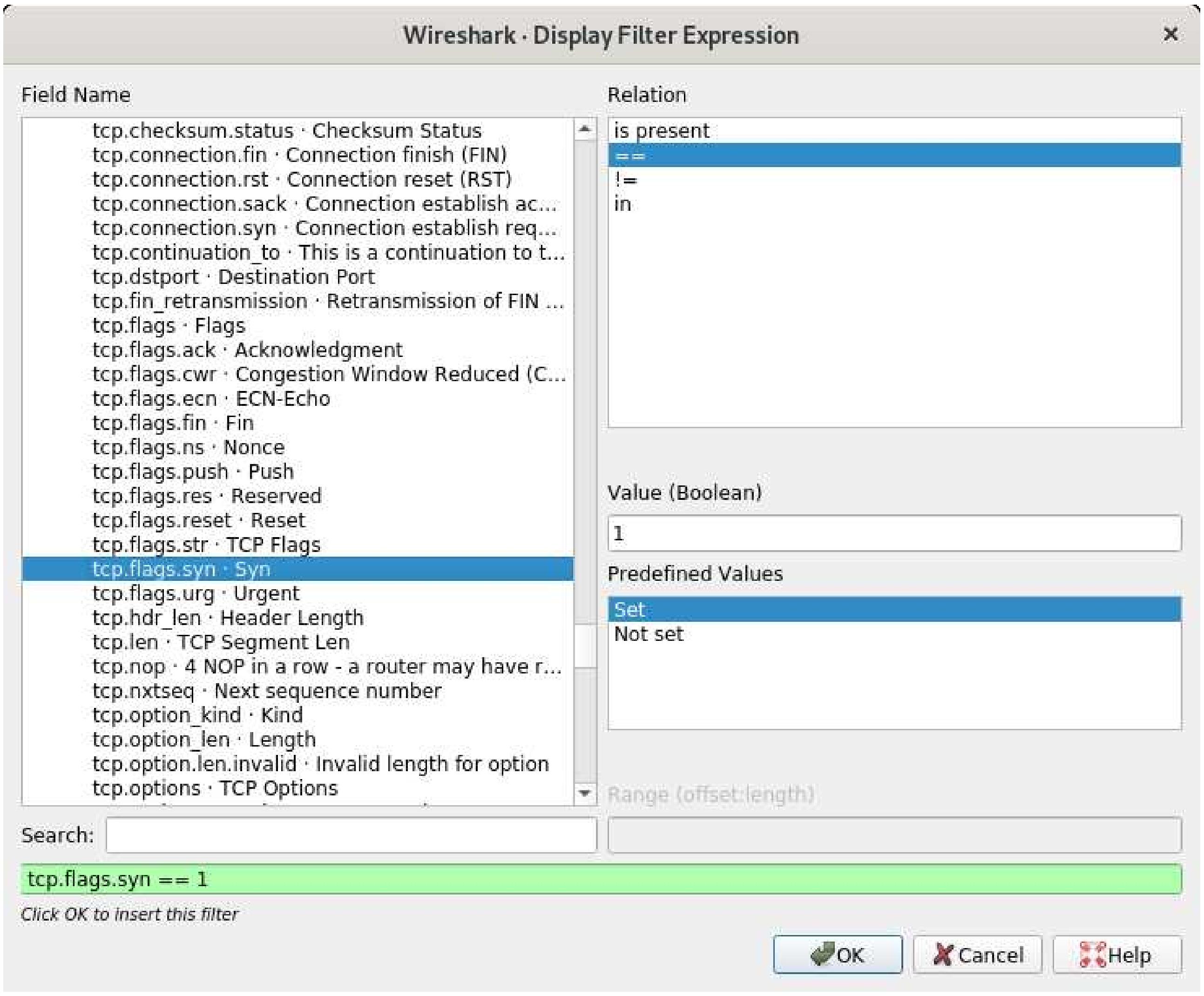
- Frame number - the frame number in the order of capture, relative to the first captured frame
- Capture time - Time in seconds that this packet was captured, relative to the start of the capture session
- Source and destination addresses - either Layer 3 (IP) or Layer 2 (MAC) addresses, depending on what type of frame it is (e.g. some Ethernet frames do not contain IP datagrams; these will just show the source and destination MAC addresses
- Protocol - Displays the highest-layer protocol that could be detected in this frame (there are probably multiple protocols at different layers)
- Length - Total number of bytes in this frame
- Summary Information - Brief human-readable summary of what this packet contains. This may include port numbers, TCP header flags, sequence numbers etc.

The middle pane breaks down the packet layer by layer. Clicking on the small triangles on the left-hand side of this pane expands each protocol section, revealing more expandable sub-sections in turn. For example, the frame shown in Figure 2 shows that Frame 18 contains an Ethernet II frame, with source MAC address aa:d8:55:4b::74:9e and destination 52:c4:44:86:73:c6. It contains an IPv4 datagram, with source IP address 10.1.9.254 and destination IP address 10.0.6.1. Inside this datagram there is a UDP datagram, source port 53 and destination port 33295. Finally in this UDP datagram, there is a DNS response (a reply to a previous DNS query). We can also scroll further down to see the details of the DNS response. You can see some of the text in the final pane, where the ASCII representation of the payload is shown on the right-hand side (indicating that this is a response to a query for the domain name `locationservices.mozilla.com`.

You can select a particular frame in Pane 1, then select a particular part of the frame in Pane 2. The corresponding raw data will then be highlighted, both in the ASCII and hexadecimal sections, in Pane 3; the total size of that particular field will be displayed at the very bottom of the Wireshark window. In this case, the UDP source port is highlighted, and you can see that this is a 2-byte field (a 16-bit unsigned big-endian integer (0x0035 (hex) = 53 (decimal)).

3.2 Display Filtering

In a busy network, you may capture a huge number of packets which are not of interest to you. By default, Wireshark displays ALL packets in their order of arrival - which may correspond to many simultaneous independent packet flows in the network. Wireshark provides a flexible mechanism to filter and display only the packets which we are interested in - the *display filter*. At the top of the window, you can enter a filter descriptor in the field immediately under the toolbar. For example, to only view packets being sent by or to your host, you may use a filter of the form



Wireshark · Display Filter Expression

Field Name

```
tcp.checksum.status · Checksum Status
tcp.connection.fin · Connection finish (FIN)
tcp.connection.rst · Connection reset (RST)
tcp.connection.sack · Connection establish ac...
tcp.connection.syn · Connection establish req...
tcp.continuation_to · This is a continuation to t...
tcp.dstport · Destination Port
tcp.fin_retransmission · Retransmission of FIN ...
tcp.flags · Flags
tcp.flags.ack · Acknowledgment
tcp.flags.cwr · Congestion Window Reduced (C...
tcp.flags.ecn · ECN-Echo
tcp.flags.fin · Fin
tcp.flags.ns · Nonce
tcp.flags.push · Push
tcp.flags.res · Reserved
tcp.flags.reset · Reset
tcp.flags.str · TCP Flags
tcp.flags.syn · Syn
tcp.flags.urg · Urgent
tcp.hdr_len · Header Length
tcp.len · TCP Segment Len
tcp.nop · 4 NOP in a row - a router may have r...
tcp.nxtseq · Next sequence number
tcp.option_kind · Kind
tcp.option_len · Length
tcp.option.len.invalid · Invalid length for option
tcp.options · TCP Options
```

Relation

```
is present
==
!=
in
```

Value (Boolean): 1

Predefined Values: Set, Not set

Range (offset:length)

Search:

```
tcp.flags.syn == 1
```

Click OK to insert this filter

Figure 3: Wireshark filter expression editor.

```
ip.addr == 10.0.3.1
```

(for the case of host `h3`). This will show any packet whose source or destination IP address is `10.0.3.1`. If you only want packets originating on your source, you can use `ip.src == 10.0.3.1`. **NOTE: you should press "Enter" after typing the filter expression to apply the display filter!**

Filter expressions can be negated using the `!` operator and logically combined using `&&` and `||` together with parentheses - for example `(ip.addr == 10.0.3.1) || ospf` would display any packet with source or destination address of `10.0.3.1` OR any OSPF packet. Some other interesting packet types include `arp`, `dns`, `udp`, and `icmp`. It is well worth spending some time playing around with the filter as it will save you a lot of time in the lab.

If you are unsure about Wireshark filter syntax (and it can get quite complex for some protocols), Wireshark has a syntax builder that you can use (see Figure 3. Click on the **Expression...** button to the right of the filter expression field. You may now select any one of a huge number of potential field names, add some conditions (e.g. "is present" or "==") with an optional value, and the editor will create an expression for you to express that rule, which will be inserted into the display filter field of the main Wireshark window. You can then edit this or combine it with other expressions before applying the filter to your packet capture. The example in Figure 3 displays any packet with the **TCP PUSH** flag set.

3.3 Measuring Time

A very common problem in network packet trace analysis is to work out how long some operation takes. Although you can manually calculate differences between packet capture times shown in Pane 1, a better way is to set a reference time. Select the packet which you consider to be the first packet (the "reference time"), and either press **ctrl-t** or use the menu **Edit** → **Set/Unset Time Reference**. Now, the arrival times of packets following this packet will be shown relative to the arrival time of this packet (which now has an arrival time of zero, displayed as ***REF***). Repeating the **ctrl-t** or menu operation will clear the reference time. You can have more than one reference time set if you wish; the reference time is simply reset to zero for any packet flagged as a timing reference.

3.4 Performance Analysis

Wireshark offers very detailed analysis of network throughput. It can both compute statistics and generate high quality graphs of different performance parameters. Go to **Statistics** → **I/O Graph** to plot a graph of I/O throughput as a function of time; you can modify the plotting style (e.g. changing from packets per second to bytes per second) via the options pane below the graph. Another interesting function is the **Flow graph**, which neatly illustrates the exchange of packets between different hosts. **TCP stream graphs** provide a useful visualisation of the TCP sequence number, round trip time, window size or throughput as a function of time.

You will find many other useful statistical analysis functions available via this menu - you should familiarise yourself with all of them!

3.5 Programming Exercise: Basic Socket Programming in Python

Note: this section does not need to be completed in one session - you may wish to spend some time learning about Python before you get started.

1. Go to the folder `comm-protocols-labs/bin`. Open the scripts `server.py` and `client.py` in a text editor (e.g. `gedit`):

   ```
   gedit server.py client.py
   ```

   Read through both scripts to understand approximately how they work. Make a note of the port that the server will listen on.

2. Start the `server.py` script on `h1` from inside the `comm-protocols-labs/bin` folder:

   ```
   cd ~/comm-protocols-labs/bin

   ./server.py
   ```

Table 1: Commands for your file transfer program

Command	Function	Response
GET filename	Request to download a file with the provided filename	SIZE XXXXXXXXXXX(newline)followed by binary data ERROR(newline) (if the file is not readable)
EXIT	Initiate connection teardown	GOODBYE(newline) followed by connection termination
HELP	Request a brief summary of usage	Very brief one-line usage summary(newline)

3. On `h2`, use the `nmap` port-scan utility to verify that you can see this port open on `h1`:

   ```
   nmap h1
   ```

   Use the `telnet` command to connect to the server from `h2` on the port you identified:

   ```
   telnet h5 portnum
   ```

   You should see a short response printed in the `telnet` session, before it is terminated.

4. Now run the client on `h2`:

   ```
   cd ~/comm-protocols-labs/bin
   ./client.py h1
   ```

   Now re-read the python scripts and try to follow the logic (your demonstrator will go through this in the lab.

5. Python programming exercise (optional - for bonus marks!): Using your knowledge of basic socket programming, convert the supplied client and server programs into a simple file transfer program. The server will sit and wait for incoming connections. When a client connects, it will send one of a number of commands to the server, which will respond as appropriate. The commands are as listed in Table 1:

   Your client should be able to request a file for downloading, interpret the SIZE response, create a file with the requested name, read the binary data from the server, before terminating. You can test the server using telnet before writing the client.

References

[1] James F. Kurose and Keith W. Ross. *Computer Networking A Top-Down Approach.* 8th edition, 2020.

[2] Behrouz A. Forouzan. *TCP/IP Protocol Suite.* 4th edition, 2017.