Sys4 Assessment

Y3891128, Desk 59

Lab 1

Q1

Propagation delay = Distance / Max signal speed = 2000 / (2 x 10^8) = 1 x 10^(-5)
Transmission delay = Size / Bandwidth

Where transmission delay = propagation delay, transmission delay = 1x10^(-5) secs
1 x 10^(-5) = 200 *(bytes)* / Bandwidth
Bandwidth = 200 / (1 x 10^(-5)) = 20 000 000 = **2 x 10^7 bytes**

Q2

(i)

(a)

134744072 → 00001000.00001000.00001000.00001000 → **8.8.8.8**

(b)

00001010.100000000.00000001.00000011 → **10.128.0.129**

(ii)

```
H:\>ping www.cs.uaf.edu

Pinging www.cs.uaf.edu [137.229.113.44] with 32 bytes of data:
Reply from 137.229.113.44: bytes=32 time=187ms TTL=36
Reply from 137.229.113.44: bytes=32 time=186ms TTL=36
Reply from 137.229.113.44: bytes=32 time=186ms TTL=36
Reply from 137.229.113.44: bytes=32 time=186ms TTL=36

Ping statistics for 137.229.113.44:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 186ms, Maximum = 187ms, Average = 186ms

H:\>
```

Average latency: (187 + 186 + 186 + 186) / 4 = **186.25ms**

Theoretical latency

Assumptions:

1. The packet will not be queued in transit, s.t. we can show the fastest theoretical time.
2. Bandwidth is significantly larger than the data transferred, with a quick google the current maximum bandwidth of fibre optics is 100Gbps, much larger than our packet size of 32 bytes
3. By definition, latency is one-way delay

Transmission delay = Packet Size / Bandwidth
Transmission delay = 32 / Bandwidth
As Bandwidth $\rightarrow \infty$, Transmission delay $\rightarrow 0$
From assumption (2), we assume that bandwidth is significantly larger than 32, so we will assume that transmission delay is negligible.
Transmission delay ≈ 0

From assumption (1), we can assume queuing delay = 0

From assumption (3), distance = 6,594 km (6,594,000 m)

Max signal speed = Speed of light / index of refraction = 299,792,458 / 1.5 = 199,861,638.7 m/s
Propagation delay = distance / max signal speed = 6,594,000 / 199,861,638.7 = 0.03299282465
= 33 ms (2.d.p)

Latency = Propagation delay + Transmission delay (0) + Queuing delay (0) = **33 ms**

Q3

```
H:\>tracert www.cs.uaf.edu

Tracing route to www.cs.uaf.edu [137.229.113.44]
over a maximum of 30 hops:

  1     1 ms    <1 ms    <1 ms  gw-pc-student.cs.york.ac.uk [144.32.179.254]
  2     1 ms    <1 ms    <1 ms  bsdcstr0-2418.ospf.york.ac.uk [10.16.35.173]
  3     2 ms     1 ms     2 ms  bsdcfab2-2401.ospf.york.ac.uk [10.16.23.129]
  4     1 ms    <1 ms    <1 ms  sentry-659.york.ac.uk [144.32.65.69]
  5     1 ms     1 ms    <1 ms  janetrouter1.york.ac.uk [144.32.255.226]
  6     1 ms     1 ms    <1 ms  ae31-391.yorktd-rbr1.ja.net [146.97.150.89]
  7     2 ms     1 ms     1 ms  ae6.leedaq-rbr1.ja.net [146.97.71.149]
  8     2 ms     3 ms     2 ms  ae0.leedlu-rbr1.ja.net [146.97.71.125]
  9     4 ms     4 ms     7 ms  ae26.manckh-sbr2.ja.net [146.97.36.221]
 10     7 ms     6 ms     7 ms  ae29.erdiss-sbr2.ja.net [146.97.33.41]
 11    29 ms    10 ms    10 ms  ae31.londpg-sbr2.ja.net [146.97.33.21]
 12    11 ms    11 ms    10 ms  ae29.londhx-sbr1.ja.net [146.97.33.1]
 13    11 ms    10 ms    11 ms  janet.mx1.lon.uk.geant.net [62.40.124.197]
 14    86 ms    86 ms    86 ms  internet2-gw.mx1.lon.uk.geant.net [62.40.124.45]
 15   150 ms   148 ms   150 ms  fourhundredge-0-0-0-0.4079.core2.ashb.net.internet2.edu [163.253.1.120]
 16   150 ms   150 ms   149 ms  fourhundredge-0-0-0-1.4079.core2.clev.net.internet2.edu [163.253.1.139]
 17   151 ms   150 ms   150 ms  fourhundredge-0-0-0-2.4079.core2.eqch.net.internet2.edu [163.253.2.17]
 18   149 ms   149 ms   148 ms  fourhundredge-0-0-0-2.4079.core2.chic.net.internet2.edu [163.253.2.18]
 19   150 ms   150 ms   150 ms  fourhundredge-0-0-0-1.4079.core1.kans.net.internet2.edu [163.253.1.245]
 20   148 ms   151 ms   150 ms  fourhundredge-0-0-0-1.4079.core1.denv.net.internet2.edu [163.253.1.242]
 21   149 ms   150 ms   154 ms  fourhundredge-0-0-0-3.4079.core1.salt.net.internet2.edu [163.253.1.171]
 22   149 ms   149 ms   150 ms  fourhundredge-0-0-0-1.4079.core1.seat.net.internet2.edu [163.253.1.157]
 23   147 ms   147 ms   147 ms  64.57.28.54
 24   147 ms   147 ms   147 ms  209.124.181.217
 25   185 ms   185 ms   185 ms  swf-mx480-1.ne.alaska.edu [137.229.255.233]
 26   186 ms   186 ms   186 ms  uaf.vpn.alaska.edu [137.229.252.163]
 27     *        *        *     Request timed out.
 28     *        *        *     Request timed out.
 29   187 ms   187 ms   193 ms  www.cs.uaf.edu [137.229.113.44]

Trace complete.

H:\>_
```

First hop in America: **Hop 15**, as it was a large increase in latency, from 86ms to 150ms, and hop 14 still has a lon.uk address.

Q4

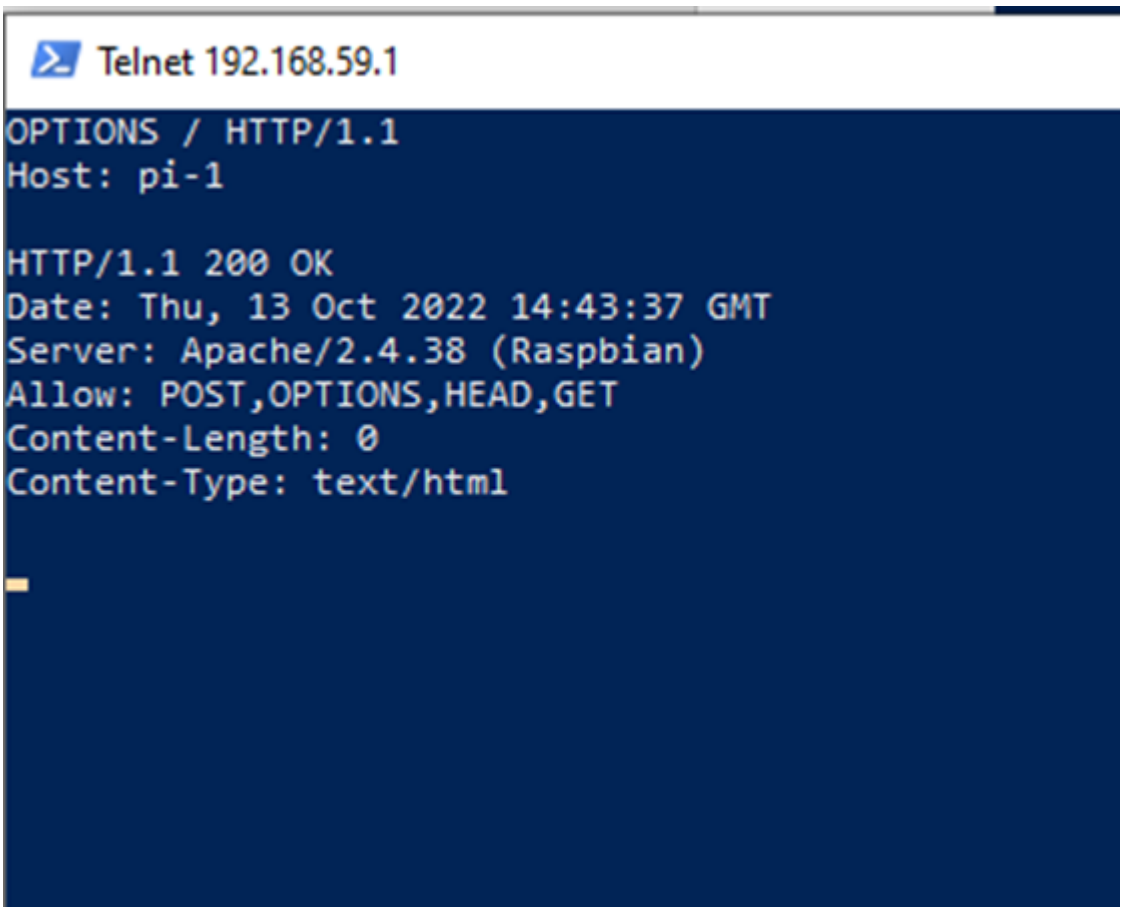
```
pi@pi-1:~ $ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.16.59.5  netmask 255.255.255.252  broadcast 172.16.59.7
        inet6 fe80::dea6:32ff:fef5:4e9f  prefixlen 64  scopeid 0x20<link>
        ether dc:a6:32:f5:4e:9f  txqueuelen 1000  (Ethernet)
        RX packets 3  bytes 180 (180.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 177  bytes 14458 (14.1 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.59.1  netmask 255.255.0.0  broadcast 192.168.255.255
        inet6 fe80::826d:97ff:fe10:daca  prefixlen 64  scopeid 0x20<link>
        ether 80:6d:97:10:da:ca  txqueuelen 1000  (Ethernet)
        RX packets 3337  bytes 217250 (212.1 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 3571  bytes 381649 (372.7 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 88  bytes 9166 (8.9 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 88  bytes 9166 (8.9 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo:1: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 10.59.1.1  netmask 255.255.255.0
        loop  txqueuelen 1000  (Local Loopback)

pi@pi-1:~ $
```

By address of network, I am assuming that means the CIDR subnet address address range

- eth0
    - Network address: 172.16.59.5
    - Broadcast address: 172.16.59.7
    - Address of network: 172.16.59.5/30
- eth1
    - Network address: 192.168.59.1
    - Broadcast address: 192.168.255.255
    - Address of network: 192.168.59.1/16
- lo
    - Network address: 127.0.0.1
    - Broadcast address: 255.0.0.0
    - Address of network: 127.0.0.1/8
- lo:1
    - Network address: 10.59.1.1
    - Broadcast address: 255.255.255.0
    - Address of network: 10.59.1.1/24

Lab 2

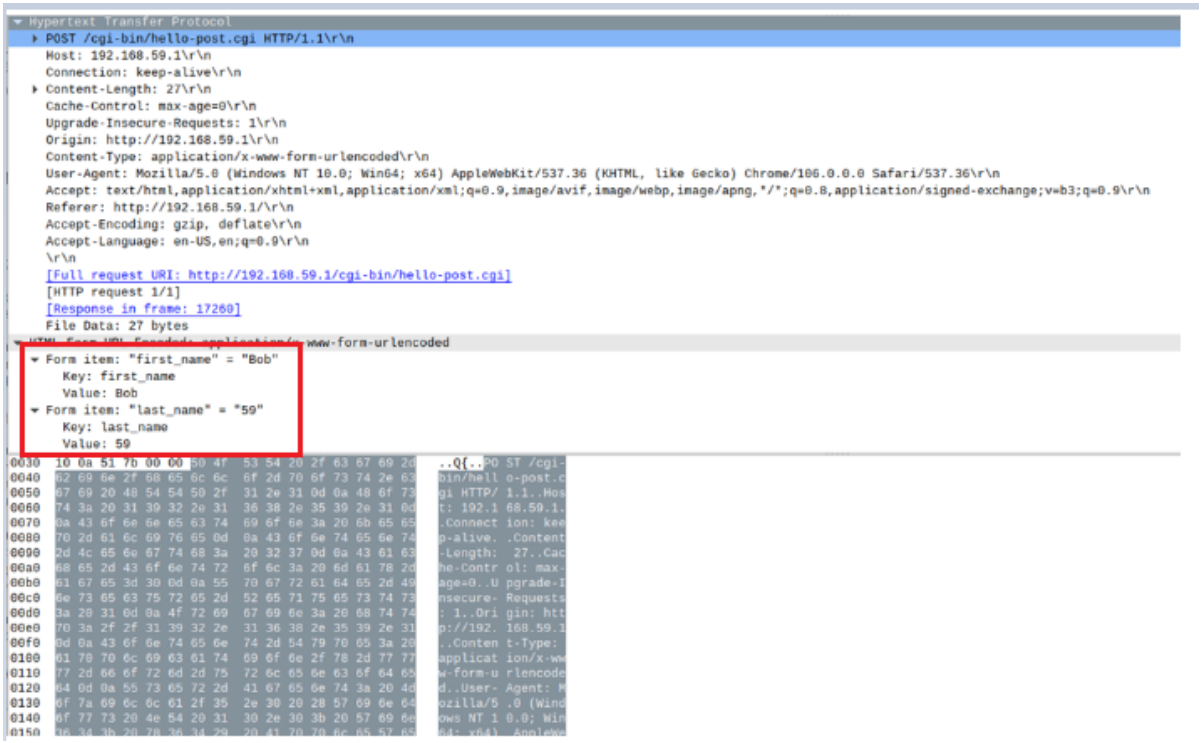
Q1


Telnet 192.168.59.1
OPTIONS / HTTP/1.1
Host: pi-1
HTTP/1.1 200 OK
Date: Thu, 13 Oct 2022 14:43:37 GMT
Server: Apache/2.4.38 (Raspbian)
Allow: POST,OPTIONS,HEAD,GET
Content-Length: 0
Content-Type: text/html

Q2

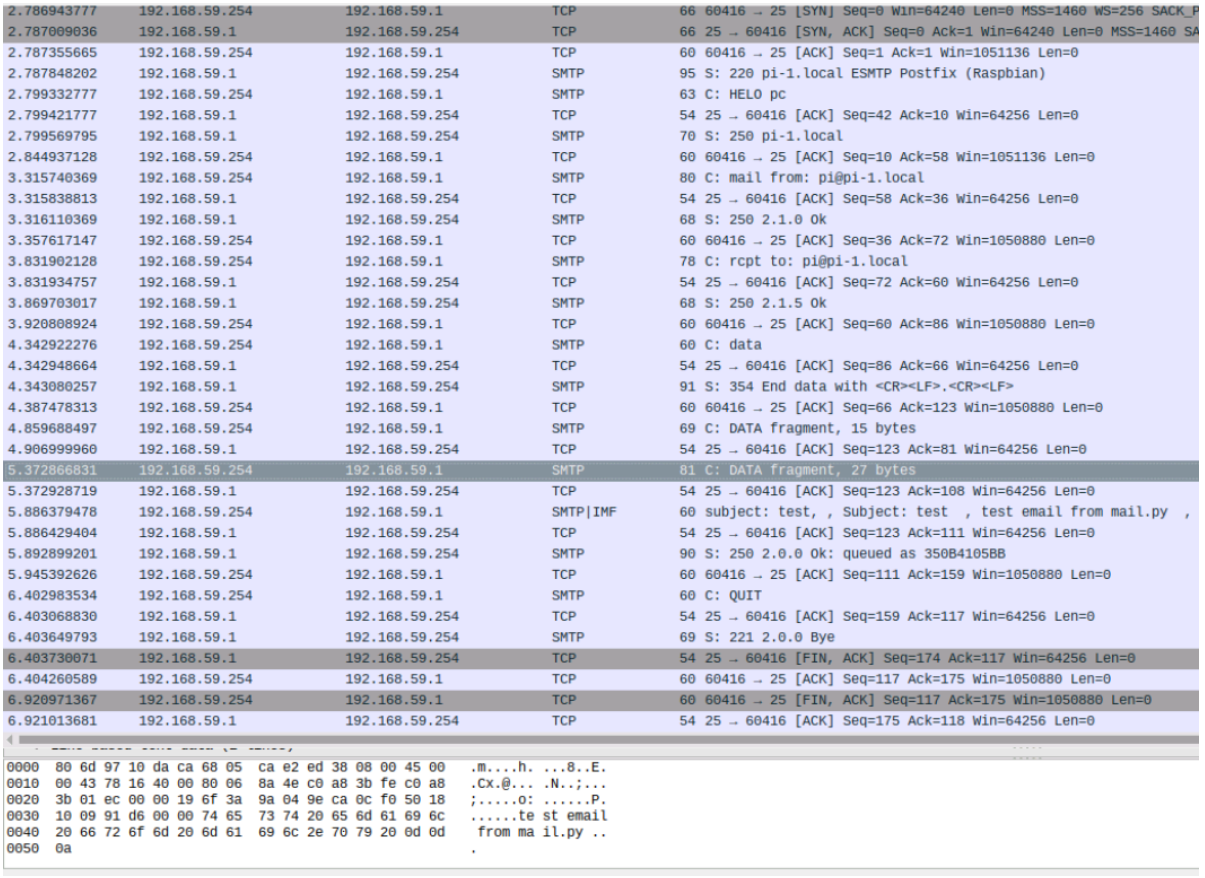
```
H:\sys4>python fortune.py
HTTP/1.1 200 OK
Date: Fri, 09 Dec 2022 15:44:26 GMT
Server: Apache/2.4.38 (Raspbian)
Vary: Accept-Encoding
Transfer-Encoding: chunked
Content-Type: text/html

416

<html>
<head>
<title>Hello - Program</title>
</head>
<body>
<h2>Hello Bob 59</h2>
<p><font face="courier" size="+1">
 ______________________________________
/ To refuse praise is to seek praise \
\ twice.                               /
 --------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||

 </font></p>
</body>
</html>



H:\sys4>
```

Q3



Q4

Sending mail to the local mail server, with message, “test email from mail.py”



2.786943777	192.168.59.254	192.168.59.1	TCP	66 60416 → 25 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_P
2.787009036	192.168.59.1	192.168.59.254	TCP	66 25 → 60416 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SA
2.787355665	192.168.59.254	192.168.59.1	TCP	60 60416 → 25 [ACK] Seq=1 Ack=1 Win=1051136 Len=0
2.787848202	192.168.59.1	192.168.59.254	SMTP	95 S: 220 pi-1.local ESMTP Postfix (Raspbian)
2.799332777	192.168.59.254	192.168.59.1	SMTP	63 C: HELO pc
2.799421777	192.168.59.1	192.168.59.254	TCP	54 25 → 60416 [ACK] Seq=42 Ack=10 Win=64256 Len=0
2.799569795	192.168.59.1	192.168.59.254	SMTP	70 S: 250 pi-1.local
2.844937128	192.168.59.254	192.168.59.1	TCP	60 60416 → 25 [ACK] Seq=10 Ack=58 Win=1051136 Len=0
3.315740369	192.168.59.254	192.168.59.1	SMTP	80 C: mail from: pi@pi-1.local
3.315838813	192.168.59.1	192.168.59.254	TCP	54 25 → 60416 [ACK] Seq=58 Ack=36 Win=64256 Len=0
3.316110369	192.168.59.1	192.168.59.254	SMTP	68 S: 250 2.1.0 Ok
3.357617147	192.168.59.254	192.168.59.1	TCP	60 60416 → 25 [ACK] Seq=36 Ack=72 Win=1050880 Len=0
3.831902128	192.168.59.254	192.168.59.1	SMTP	78 C: rcpt to: pi@pi-1.local
3.831934757	192.168.59.1	192.168.59.254	TCP	54 25 → 60416 [ACK] Seq=72 Ack=60 Win=64256 Len=0
3.869703017	192.168.59.1	192.168.59.254	SMTP	68 S: 250 2.1.5 Ok
3.920808924	192.168.59.254	192.168.59.1	TCP	60 60416 → 25 [ACK] Seq=60 Ack=86 Win=1050880 Len=0
4.342922276	192.168.59.254	192.168.59.1	SMTP	60 C: data
4.342948664	192.168.59.1	192.168.59.254	TCP	54 25 → 60416 [ACK] Seq=86 Ack=66 Win=64256 Len=0
4.343080257	192.168.59.1	192.168.59.254	SMTP	91 S: 354 End data with <CR><LF>.<CR><LF>
4.387478313	192.168.59.254	192.168.59.1	TCP	60 60416 → 25 [ACK] Seq=66 Ack=123 Win=1050880 Len=0
4.859688497	192.168.59.254	192.168.59.1	SMTP	69 C: DATA fragment, 15 bytes
4.906999960	192.168.59.1	192.168.59.254	TCP	54 25 → 60416 [ACK] Seq=123 Ack=81 Win=64256 Len=0
5.372866831	192.168.59.254	192.168.59.1	SMTP	81 C: DATA fragment, 27 bytes
5.372928719	192.168.59.1	192.168.59.254	TCP	54 25 → 60416 [ACK] Seq=123 Ack=108 Win=64256 Len=0
5.886379478	192.168.59.254	192.168.59.1	SMTP|IMF	60 subject: test, , Subject: test , test email from mail.py ,
5.886429404	192.168.59.1	192.168.59.254	TCP	54 25 → 60416 [ACK] Seq=123 Ack=111 Win=64256 Len=0
5.892899201	192.168.59.1	192.168.59.254	SMTP	90 S: 250 2.0.0 Ok: queued as 350B4105BB
5.945392626	192.168.59.254	192.168.59.1	TCP	60 60416 → 25 [ACK] Seq=111 Ack=159 Win=1050880 Len=0
6.402983534	192.168.59.254	192.168.59.1	SMTP	60 C: QUIT
6.403068830	192.168.59.1	192.168.59.254	TCP	54 25 → 60416 [ACK] Seq=159 Ack=117 Win=64256 Len=0
6.403649793	192.168.59.1	192.168.59.254	SMTP	69 S: 221 2.0.0 Bye
6.403730071	192.168.59.1	192.168.59.254	TCP	54 25 → 60416 [FIN, ACK] Seq=174 Ack=117 Win=64256 Len=0
6.404260589	192.168.59.254	192.168.59.1	TCP	60 60416 → 25 [ACK] Seq=117 Ack=175 Win=1050880 Len=0
6.920971367	192.168.59.254	192.168.59.1	TCP	60 60416 → 25 [FIN, ACK] Seq=117 Ack=175 Win=1050880 Len=0
6.921013681	192.168.59.1	192.168.59.254	TCP	54 25 → 60416 [ACK] Seq=175 Ack=118 Win=64256 Len=0

```
0000  80 6d 97 10 da ca 68 05  ca e2 ed 38 08 00 45 00   .m....h. ...8..E.
0010  00 43 78 16 40 00 80 06  8a 4e c0 a8 3b fe c0 a8   .Cx.@... .N..;...
0020  3b 01 ec 00 00 19 6f 3a  9a 04 9e ca 0c f0 50 18   ;.....o: ......P.
0030  10 09 91 d6 00 00 74 65  73 74 20 65 6d 61 69 6c   ......te st email
0040  20 66 72 6f 6d 20 6d 61  69 6c 2e 70 79 20 0d 0d    from ma il.py ..
0050  0a                                                .
```

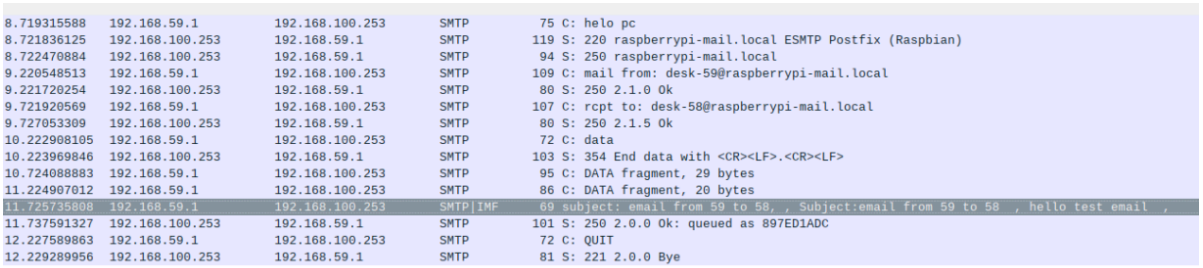
Retrieving email from mail server

2.069278573	192.168.59.254	192.168.59.1	TCP	66 60435 → 110 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
2.069369147	192.168.59.1	192.168.59.254	TCP	66 110 → 60435 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM=1 WS=128
2.069677943	192.168.59.254	192.168.59.1	TCP	60 60435 → 110 [ACK] Seq=1 Ack=1 Win=1051136 Len=0
2.090070703	192.168.59.254	192.168.59.1	POP	63 C: user pi
2.090155758	192.168.59.1	192.168.59.254	TCP	54 110 → 60435 [ACK] Seq=1 Ack=10 Win=64256 Len=0
2.099895869	192.168.59.1	192.168.59.254	POP	85 S: +OK Dovecot (Raspbian) ready.
2.152301165	192.168.59.254	192.168.59.1	TCP	60 60435 → 110 [ACK] Seq=10 Ack=32 Win=1051136 Len=0
2.152370980	192.168.59.1	192.168.59.254	POP	59 S: +OK
2.199758073	192.168.59.254	192.168.59.1	TCP	60 60435 → 110 [ACK] Seq=10 Ack=37 Win=1051136 Len=0
2.610562980	192.168.59.254	192.168.59.1	POP	66 C: pass 12345
2.610650073	192.168.59.1	192.168.59.254	TCP	54 110 → 60435 [ACK] Seq=37 Ack=22 Win=64256 Len=0
2.706159758	192.168.59.1	192.168.59.254	POP	70 S: +OK Logged in.
2.762085517	192.168.59.254	192.168.59.1	TCP	60 60435 → 110 [ACK] Seq=22 Ack=53 Win=1051136 Len=0
3.128311276	192.168.59.254	192.168.59.1	POP	60 C: list
3.128354128	192.168.59.1	192.168.59.254	TCP	54 110 → 60435 [ACK] Seq=53 Ack=28 Win=64256 Len=0
3.128472443	192.168.59.1	192.168.59.254	POP	81 S: +OK 1 messages:
3.170318832	192.168.59.254	192.168.59.1	TCP	60 60435 → 110 [ACK] Seq=28 Ack=80 Win=1050880 Len=0
3.647712053	192.168.59.254	192.168.59.1	POP	62 C: retr 1
3.647788498	192.168.59.1	192.168.59.254	TCP	54 110 → 60435 [ACK] Seq=80 Ack=36 Win=64256 Len=0
3.649479665	192.168.59.1	192.168.59.254	POP	363 S: +OK 290 octets
3.705050461	192.168.59.254	192.168.59.1	TCP	60 60435 → 110 [ACK] Seq=36 Ack=389 Win=1050624 Len=0
4.169078053	192.168.59.254	192.168.59.1	POP	60 C: QUIT
4.169981757	192.168.59.1	192.168.59.254	POP	72 S: +OK Logging out.

```
    TCP payload (309 bytes)
▾ Post Office Protocol
  ▸ +OK 290 octets\r\n
    Return-Path: <pi@pi-1.local>\r\n
    X-Original-To: pi@pi-1.local\r\n
    Delivered-To: pi@pi-1.local\r\n
    Received: from pc (pc.local [192.168.59.254])\r\n
    \tby pi-1.local (Postfix) with SMTP id 350B4105BB\r\n
    \tfor <pi@pi-1.local>; Fri, 18 Nov 2022 15:54:15 +0000 (GMT)\r\n
    Subject: test\r\n
    \r\n
    test email from mail.py \r\n
    .\r\n
```

Q5

Email subject: “email from 59 to 58”, message: “hello test email”



8.719315588	192.168.59.1	192.168.100.253	SMTP	75	C: helo pc
8.721836125	192.168.100.253	192.168.59.1	SMTP	119	S: 220 raspberrypi-mail.local ESMTP Postfix (Raspbian)
8.722470884	192.168.100.253	192.168.59.1	SMTP	94	S: 250 raspberrypi-mail.local
9.220548513	192.168.59.1	192.168.100.253	SMTP	109	C: mail from: desk-59@raspberrypi-mail.local
9.221720254	192.168.100.253	192.168.59.1	SMTP	80	S: 250 2.1.0 Ok
9.721920569	192.168.59.1	192.168.100.253	SMTP	107	C: rcpt to: desk-58@raspberrypi-mail.local
9.727053309	192.168.100.253	192.168.59.1	SMTP	80	S: 250 2.1.5 Ok
10.222908105	192.168.59.1	192.168.100.253	SMTP	72	C: data
10.223969846	192.168.100.253	192.168.59.1	SMTP	103	S: 354 End data with <CR><LF>.<CR><LF>
10.724088883	192.168.59.1	192.168.100.253	SMTP	95	C: DATA fragment, 29 bytes
11.224907012	192.168.59.1	192.168.100.253	SMTP	86	C: DATA fragment, 20 bytes
11.725735808	192.168.59.1	192.168.100.253	SMTP|IMF	69	subject: email from 59 to 58, , Subject:email from 59 to 58 , hello test email ,
11.737591327	192.168.100.253	192.168.59.1	SMTP	101	S: 250 2.0.0 Ok: queued as 897ED1ADC
12.227589863	192.168.59.1	192.168.100.253	SMTP	72	C: QUIT
12.229289956	192.168.100.253	192.168.59.1	SMTP	81	S: 221 2.0.0 Bye

```
▸ [Timestamps]
  TCP payload (3 bytes)
▾ Simple Mail Transfer Protocol
  C: .
▸ [2 DATA fragments (49 bytes): #2573(29), #2577(20)]
▾ Internet Message Format
  Subject: email from 59 to 58
▾ Message-Text
    Subject:email from 59 to 58
    hello test email
```

Lab 3

Q1

http://www.google.com/index.html

- Domain name: google.com
- Top level domain: .com
- Sub-domain: www.

```
Domain Name: google.com
Registry Domain ID: 2138514_DOMAIN_COM-VRSN
Registrar WHOIS Server: whois.markmonitor.com
Registrar URL: http://www.markmonitor.com
Updated Date: 2019-09-09T15:39:04+0000
Creation Date: 1997-09-15T07:00:00+0000
Registrar Registration Expiration Date: 2028-09-13T07:00:00+0000
Registrar: MarkMonitor, Inc.
Registrar IANA ID: 292
Registrar Abuse Contact Email: abusecomplaints@markmonitor.com
Registrar Abuse Contact Phone: +1.2083895770
Domain Status: clientUpdateProhibited (https://www.icann.org/epp#clientUpdateProhibited)
Domain Status: clientTransferProhibited (https://www.icann.org/epp#clientTransferProhibited)
Domain Status: clientDeleteProhibited (https://www.icann.org/epp#clientDeleteProhibited)
Domain Status: serverUpdateProhibited (https://www.icann.org/epp#serverUpdateProhibited)
Domain Status: serverTransferProhibited (https://www.icann.org/epp#serverTransferProhibited)
Domain Status: serverDeleteProhibited (https://www.icann.org/epp#serverDeleteProhibited)
Registrant Organization: Google LLC
Registrant State/Province: CA
Registrant Country: US
Registrant Email: Select Request Email Form at https://domains.markmonitor.com/whois/google.co
Admin Organization: Google LLC
Admin State/Province: CA
Admin Country: US
Admin Email: Select Request Email Form at https://domains.markmonitor.com/whois/google.com
Tech Organization: Google LLC
Tech State/Province: CA
Tech Country: US
Tech Email: Select Request Email Form at https://domains.markmonitor.com/whois/google.com
Name Server: ns1.google.com
Name Server: ns3.google.com
Name Server: ns2.google.com
Name Server: ns4.google.com
DNSSEC: unsigned
URL of the ICANN WHOIS Data Problem Reporting System: http://wdprs.internic.net/
>>> Last update of WHOIS database: 2022-11-30T12:23:41+0000 <<<
```

IANA ID: **292**

Q2

(a)

The closest root server to Aberdeen, according to https://root-servers.org/, is Dundee, GB


Dundee, GB
Operator ICANN
IPv4 199.7.83.42
ASN 20144
Perth
Stirling
Glasgow
Carlisle
Newcastle upon Tyne
Sunderland
L
J
E
D
F


Inverness
Aberdeen
Dundee
Perth
Stirling
Glasgow
L
J
E
D
F

(b)

www.york.ac.uk is an alias of w2f5www.york.ac.uk using the CNAME record, with a TTL of 900 seconds. The alias w2f5www.york.ac.uk has an A record itself that has a TTL of 86400 seconds.

I will assume the question is querying for the A record rather than the CNAME alias: **86400 seconds.**

```
H:\>dig +noall +answer www.york.ac.uk
www.york.ac.uk.          900     IN      CNAME   w2f5www.york.ac.uk.
w2f5www.york.ac.uk.      86400   IN      A       144.32.128.93
```

The TTL for www.google.co.uk is **232 seconds.**

```
H:\>dig +noall +answer www.google.co.uk
www.google.co.uk.        232     IN      A       216.58.212.227
```

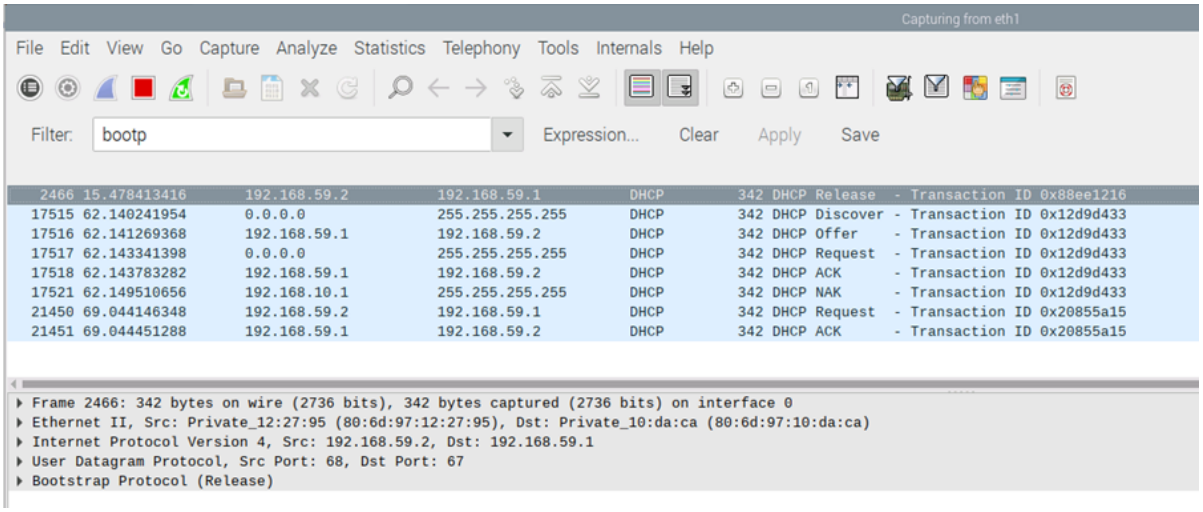
The google DNS is decremented every second, and once it reaches 0 the DNS record will be refreshed. However, the www.york.ac.uk DNS record TTL does not decrease, always staying at 900 & 86400 seconds. This is because the PC is on the york.ac.uk network, so has the DNS record as static.

(c)

144.32.50.12 = xmasopenday.york.ac.uk.

```
144.32.50.10 = pc182vm.cs.york.ac.uk.
144.32.50.11 = videowallcam.cs.york.ac.uk.
144.32.50.12 = xmasopenday.york.ac.uk.
144.32.50.13 = csereception.cs.york.ac.uk.
```

Q3



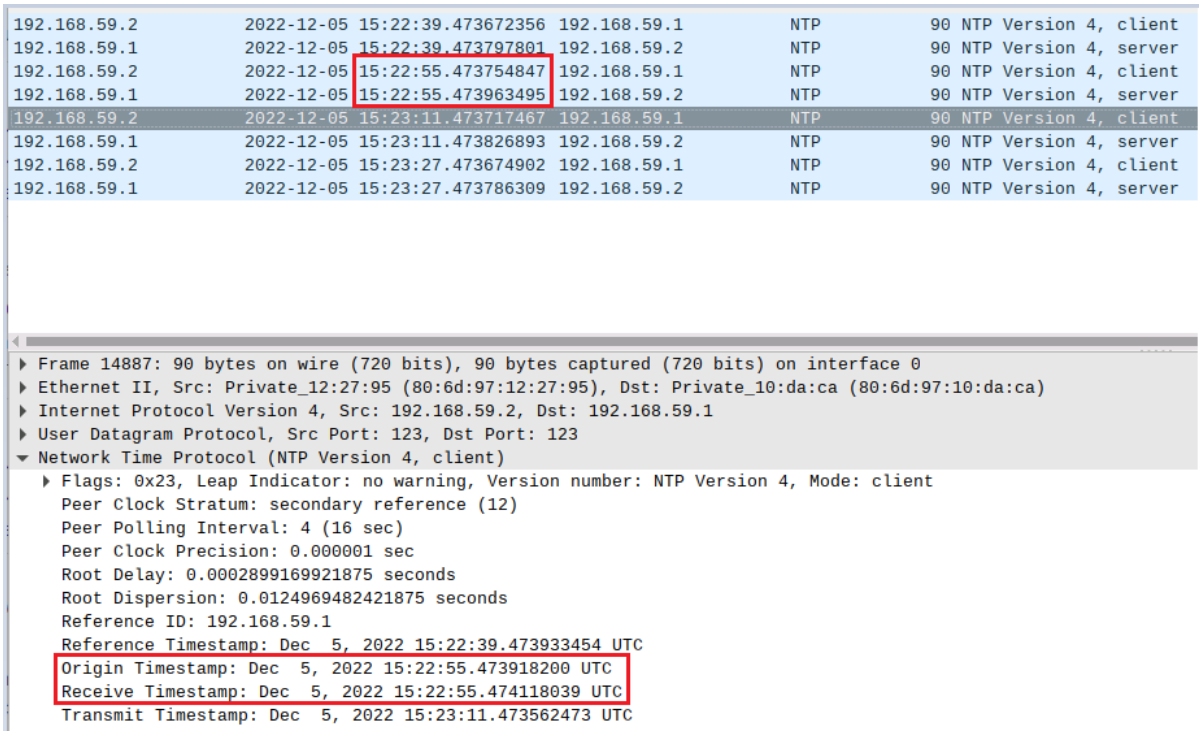
Q4

NTP timestamp process:

Client		Server
t1	→	t2
		(Process time before sending packet back)
t4	←	t3

From these four timestamps we can calculate the time offset, round trip delay, and root dispersion.

Offset = [(t2 + t3) – (t4 + t1)]/2
Delay = (t4 -t1) – (t3 – t2)
Dispersion = DR * (t4 – t1) + timestamping errors



- Pi-1 to Pi-2 send: 55.473754847 (t1)
- Pi-2 to Pi-1 send back: 55.473963495 (t3)
- Origin timestamp: 55.473918200 (t2)
- Receive timestamp: 55.474118039 (t4)

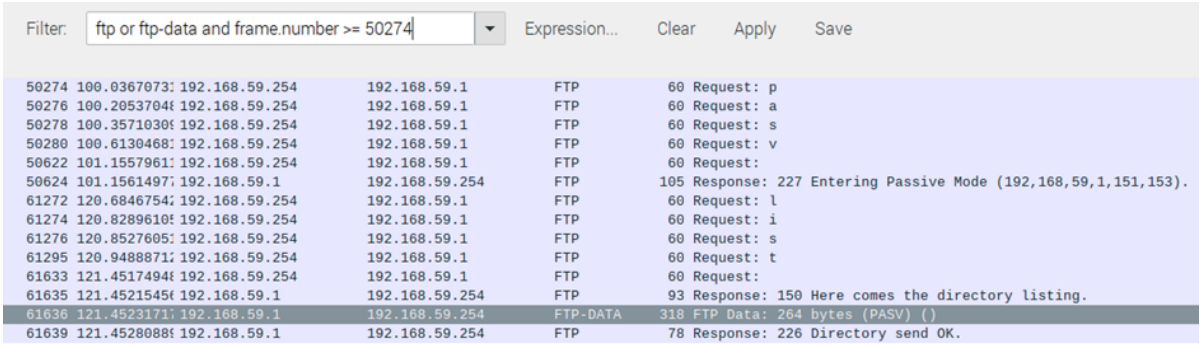
Delay = (55.474118039 - 55.473754847) - (55.473963495 - 55.473918200) =
0.00031789699 seconds

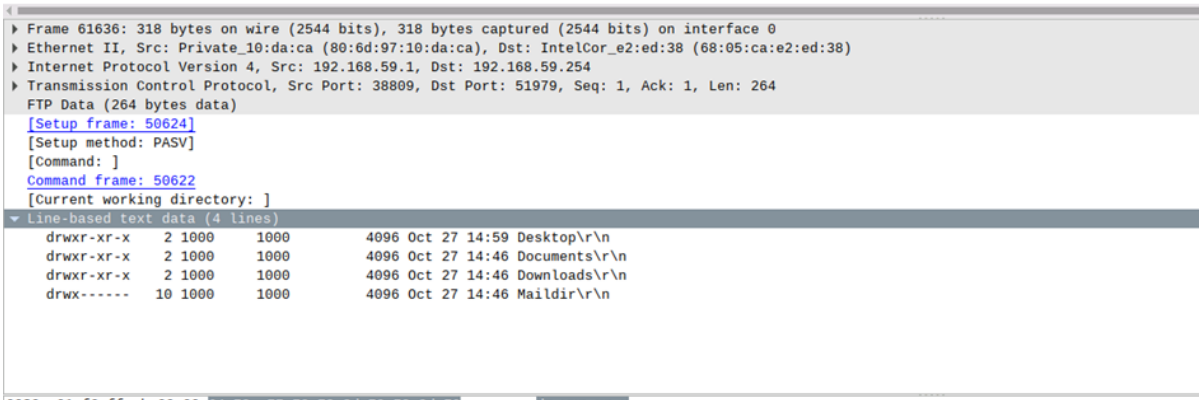
This is very close to the root delay calculated by Wireshark, I believe the difference is a rounding error from the timestamps shown in the list of packets is not to full precision, as the wireshark root delay is precise to 16 d.p., while the timestamps shown are only accurate to 9 d.p.

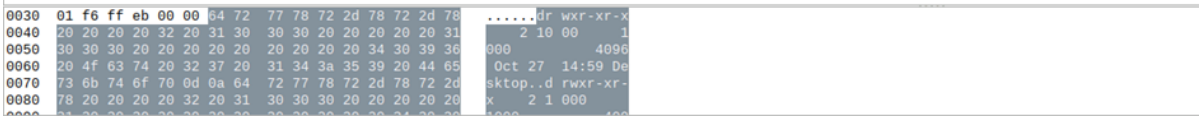
Time offset = [(55.473918200 + 55.473963495) - (55.474118039 + 55.473754847)] / 2 = **0.0000044045 seconds**

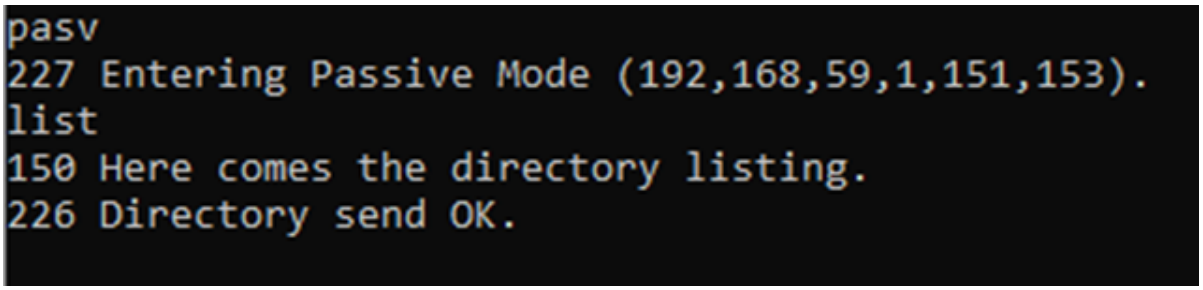
Lab 4

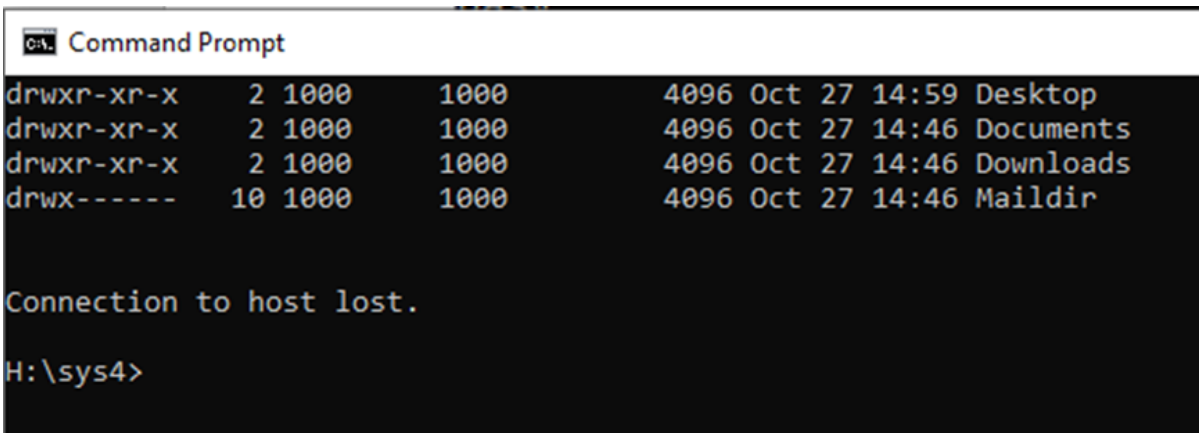
Q1









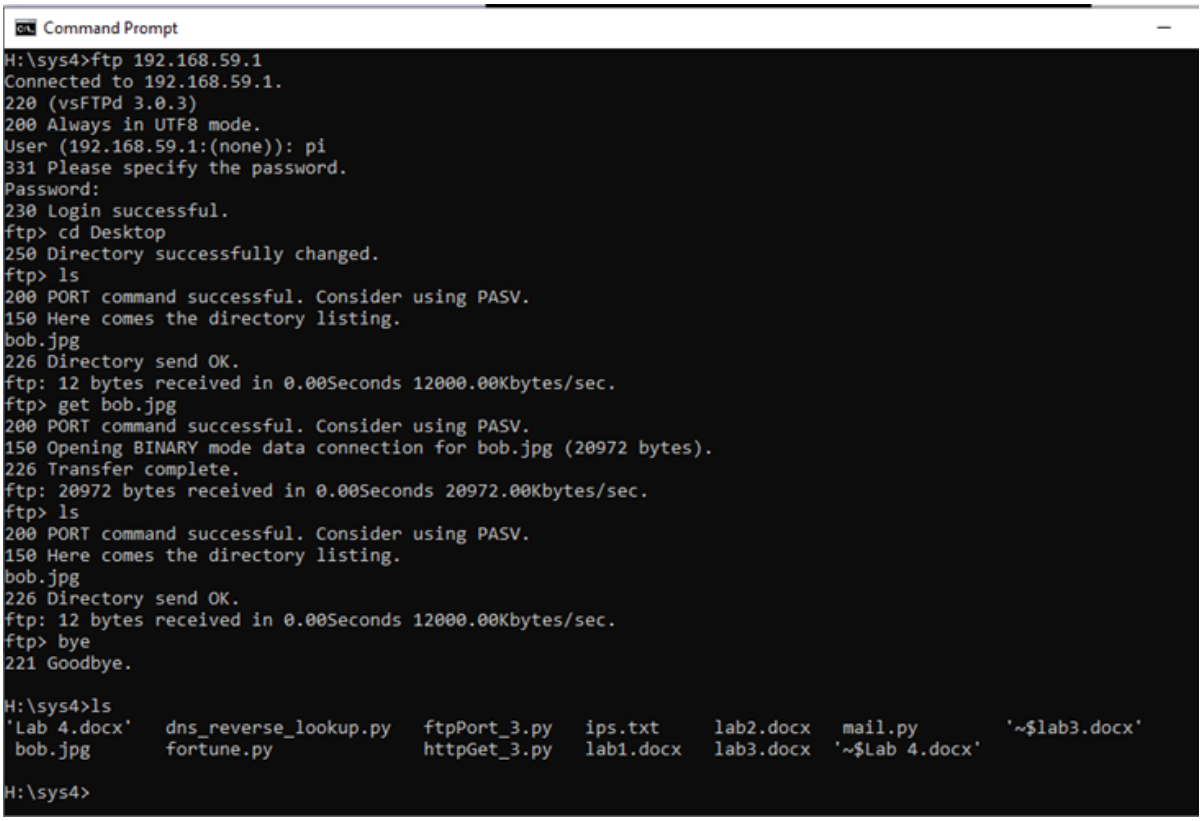


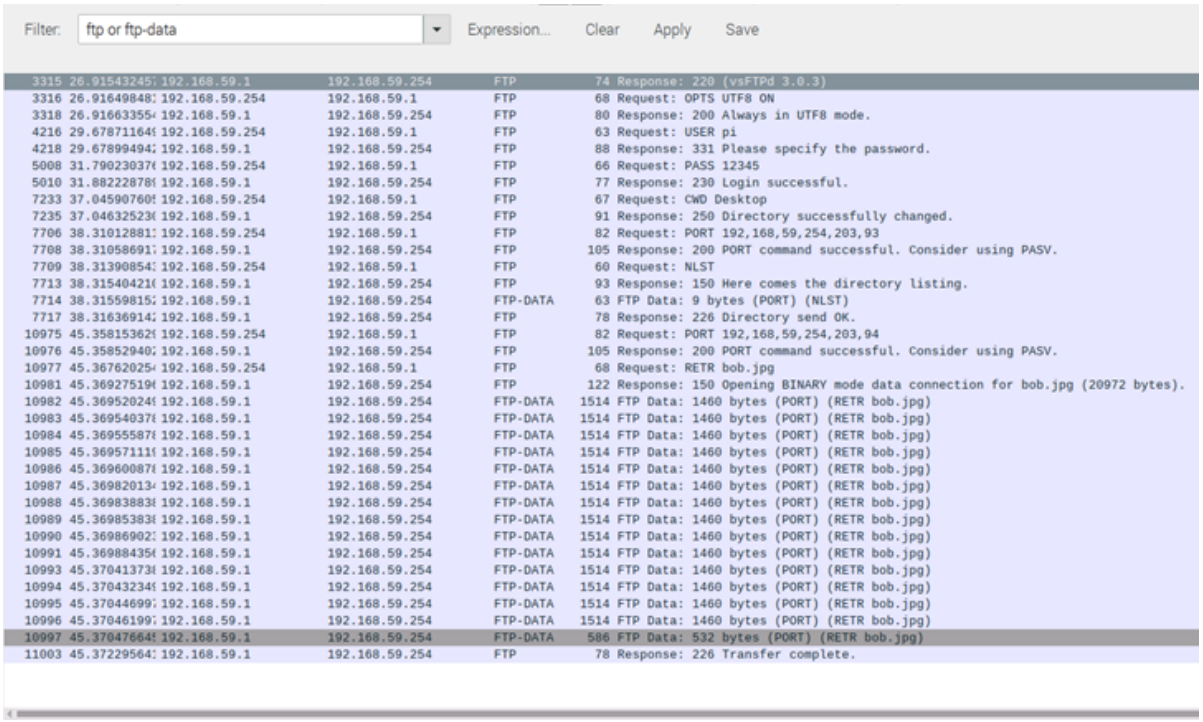
Passive mode: (192,168,59,1,151,153)
Address: 192.168.59.1
Data Port: 256 * 151 + 153 = **38809**
Control Port: **21**

Why is passive mode used more than active mode?
Passive mode is useful as both control and data connections can be initiated by the client, which alleviates some responsibility, as the client does not have to configure their firewall to allow for inbound connections.

Q2

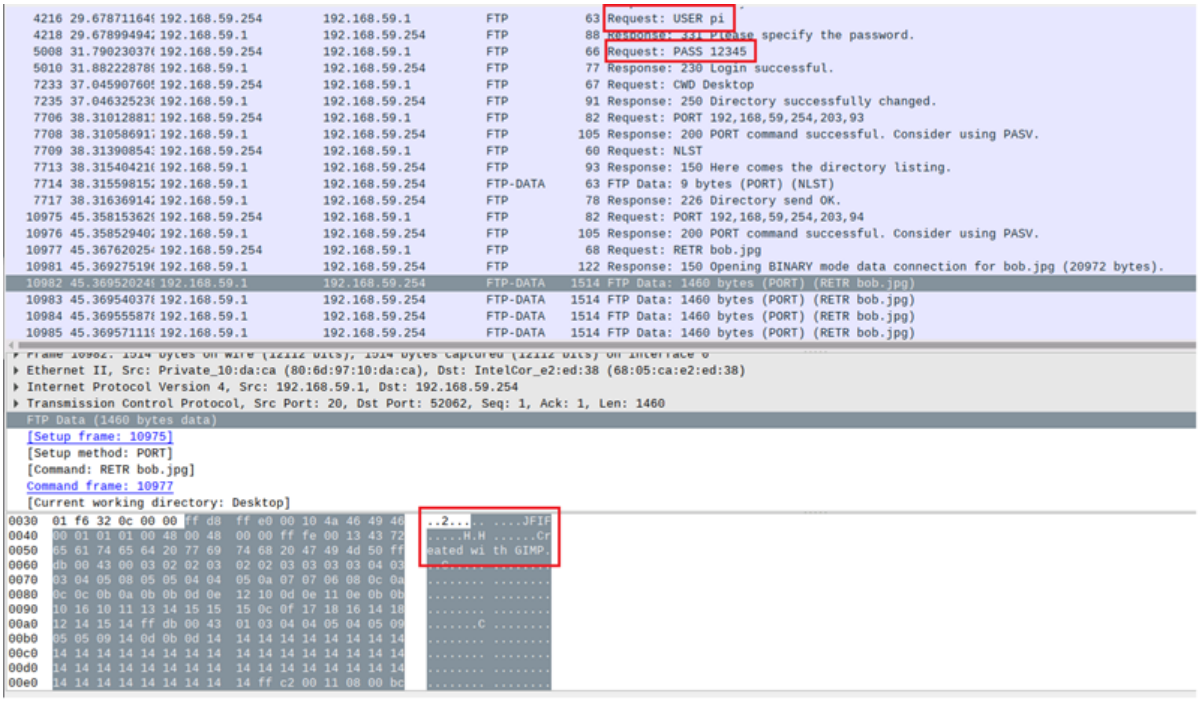
Transferring bob.jpg





Is data encrypted in the FTP transfer?

No, the username and password are sent in plaintext as is the bob.jpg data, and can be intercepted by Wireshark, as shown in the image below.



Q3

Filter: ftp or ftp-data or ssh | Expression... | Clear | Apply | Save

4	0.010427669	192.168.59.1	192.168.59.1	FTP	86 Response: 220 (vsFTPd 3.0.3)
6	0.013003471	192.168.59.1	192.168.59.1	FTP	80 Request: OPTS UTF8 ON
8	0.013192376	192.168.59.1	192.168.59.1	FTP	92 Response: 200 Always in UTF8 mode.
10	2.215354774	192.168.59.1	192.168.59.1	FTP	75 Request: USER pi
12	2.215534883	192.168.59.1	192.168.59.1	FTP	100 Response: 331 Please specify the password.
14	3.768679189	192.168.59.1	192.168.59.1	FTP	78 Request: PASS 12345
16	3.854158404	192.168.59.1	192.168.59.1	FTP	89 Response: 230 Login successful.
18	14.67176145	192.168.59.1	192.168.59.1	FTP	90 Request: PORT 127,0,0,1,206,216
20	14.67219208	192.168.59.1	192.168.59.1	FTP	93 Response: 500 Illegal PORT command.
22	14.67551000	192.168.59.1	192.168.59.1	FTP	72 Request: NLST
23	14.67579762	192.168.59.1	192.168.59.1	FTP	95 Response: 425 Use PORT or PASV first.

```
H:\>ftp
ftp> open 127.0.0.1 8000
Connected to 127.0.0.1.
220 (vsFTPd 3.0.3)
200 Always in UTF8 mode.
User (127.0.0.1:(none)): pi
331 Please specify the password.
Password:
230 Login successful.
ftp> ls
500 Illegal PORT command.
425 Use PORT or PASV first.
```

After login it attempts to bind data port to 127.0.0.1:52952 (256*206 + 216) However, this port is not available to be forwarded through the SSH tunnel.

In this example, port 21 on the 192.168.59.1 is mapped to port 8000 on localhost. We then connect to FTP by 127.0.0.1 8000, and FTP in active mode attempts to set up a random data port back to the client at 127.0.0.1:52952, however this is not forwarded over SSH so is unavailable to be connected to.

A way to fix this problem would be to start the FTP client in PASSIVE mode, and set up another SSH tunnel to the data port on the raspberry pi.

Q4

- Host: 192.168.59.254
- Pi-1: 192.168.59.1
- Pi-2: 192.168.59.2

Our host machine is connecting to Pi-2's FTP server using Pi-1 as a proxy. The proxy connection between Host and Pi-1 is using SSH and encrypted, however the messages are

then sent by Pi-1 as cleartext to Pi-2: username, password and FTP data are seen clearly by intercepting the packets between Pi-1 and Pi-2.

Pi-2 then sends response messages (in plaintext) back to Pi-1, which encrypts the response and sends it encrypted over SSH back to our host.

- Host to Pi-1: Encrypted
- Pi-1 to Pi-2: Not encrypted

Can Pi-2 tell that data is being transferred to the PC?
Pi-2 only talks to Pi-1, which acts as a middle man between the host and Pi-2. No messages are ever sent from Host to Pi-2 or vice versa, and Pi-2 cannot tell the data is being transferred to the Host.

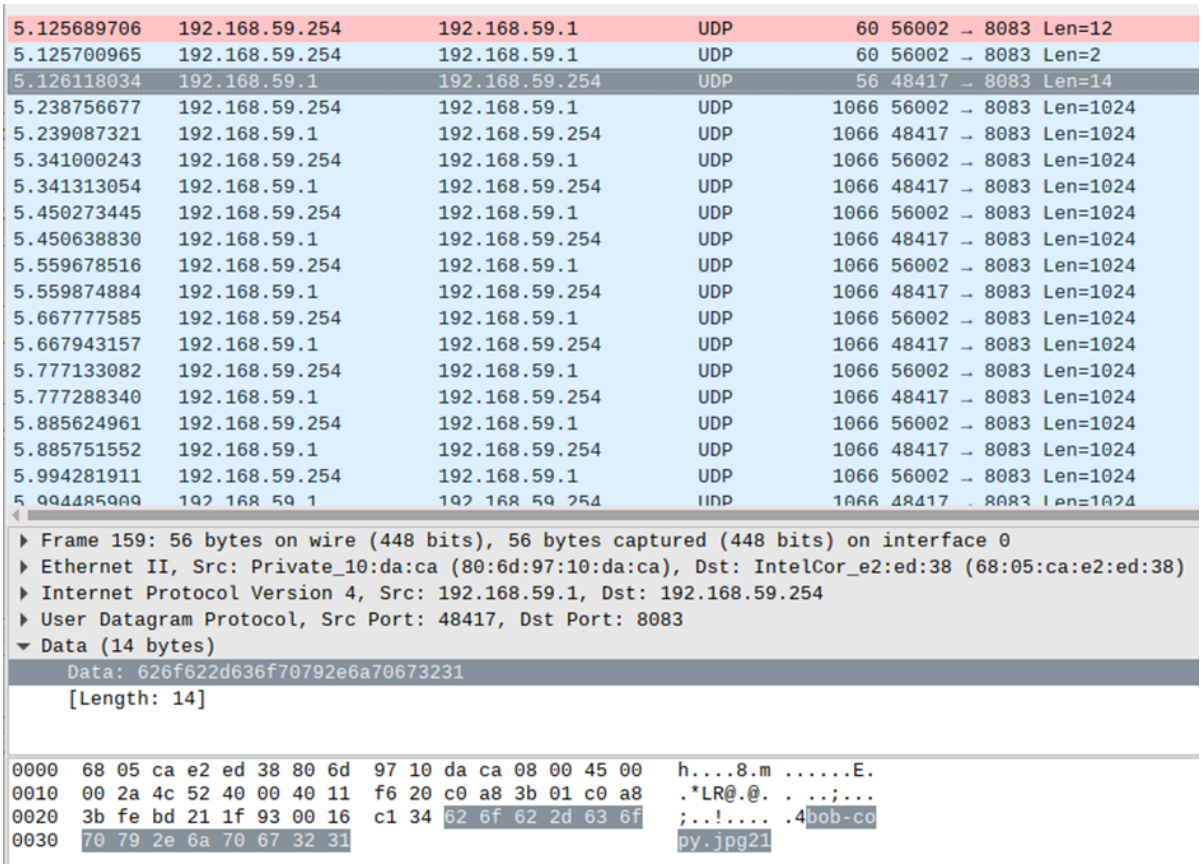
19.916136137	192.168.59.254	192.168.59.1	SSH	90	Client: Encrypted packet (len=36)
19.916760277	192.168.59.1	192.168.59.254	SSH	126	Server: Encrypted packet (len=72)
19.916992219	192.168.59.254	192.168.59.1	SSH	90	Client: Encrypted packet (len=36)
19.923254050	192.168.59.254	192.168.59.1	SSH	146	Client: Encrypted packet (len=92)
19.923736970	192.168.59.1	192.168.59.254	SSH	98	Server: Encrypted packet (len=44)
19.934076122	192.168.59.2	192.168.59.1	FTP	86	Response: 220 (vsFTPd 3.0.3)
19.969772815	192.168.59.1	192.168.59.254	SSH	114	Server: Encrypted packet (len=60)
19.970463047	192.168.59.254	192.168.59.1	SSH	98	Client: Encrypted packet (len=44)
19.970562139	192.168.59.1	192.168.59.2	FTP	76	Request: AUTH TLS
19.971048892	192.168.59.2	192.168.59.1	FTP	104	Response: 530 Please login with USER and PASS.
19.971135984	192.168.59.1	192.168.59.254	SSH	130	Server: Encrypted packet (len=76)
19.971996288	192.168.59.254	192.168.59.1	SSH	98	Client: Encrypted packet (len=44)
19.972072565	192.168.59.1	192.168.59.2	FTP	76	Request: AUTH SSL
19.972462116	192.168.59.2	192.168.59.1	FTP	104	Response: 530 Please login with USER and PASS.
19.972541874	192.168.59.1	192.168.59.254	SSH	130	Server: Encrypted packet (len=76)
19.974372166	192.168.59.254	192.168.59.1	SSH	98	Client: Encrypted packet (len=44)
19.974446110	192.168.59.1	192.168.59.2	FTP	75	Request: USER pi
19.974840420	192.168.59.2	192.168.59.1	FTP	100	Response: 331 Please specify the password.
19.974921604	192.168.59.1	192.168.59.254	SSH	122	Server: Encrypted packet (len=68)
19.975386765	192.168.59.254	192.168.59.1	SSH	106	Client: Encrypted packet (len=52)
19.975461653	192.168.59.1	192.168.59.2	FTP	78	Request: PASS 12345
20.059805715	192.168.59.2	192.168.59.1	FTP	89	Response: 230 Login successful.
20.059901010	192.168.59.1	192.168.59.254	SSH	114	Server: Encrypted packet (len=60)
20.064217217	192.168.59.254	192.168.59.1	SSH	98	Client: Encrypted packet (len=44)
20.064309179	192.168.59.1	192.168.59.2	FTP	71	Request: PWD
20.064633564	192.168.59.2	192.168.59.1	FTP	107	Response: 257 "/home/pi" is the current directory
20.064713174	192.168.59.1	192.168.59.254	SSH	130	Server: Encrypted packet (len=76)

Q5

(a)

With no sleep on line 17, the host sends both the file name “bob-copy.jpg” and the file length “21” very quickly after each other, causing Pi-1 to receive both packets at *almost* the same time, which causes Pi-1 to package them into a single packet for efficiency when sending it back to the host.

When the PC receives the packets, the program expects two individual packets: one with the file name, and one with the number of segments. It instead receives “bob-copy.jpg21” as a single message, which it processes as a filename, and attempts to interpret the next packet as the number of segments. However the next packet is instead the first segment of the jpeg data in binary format, which causes the python program to crash as the binary data is not able to be interpreted as an integer.



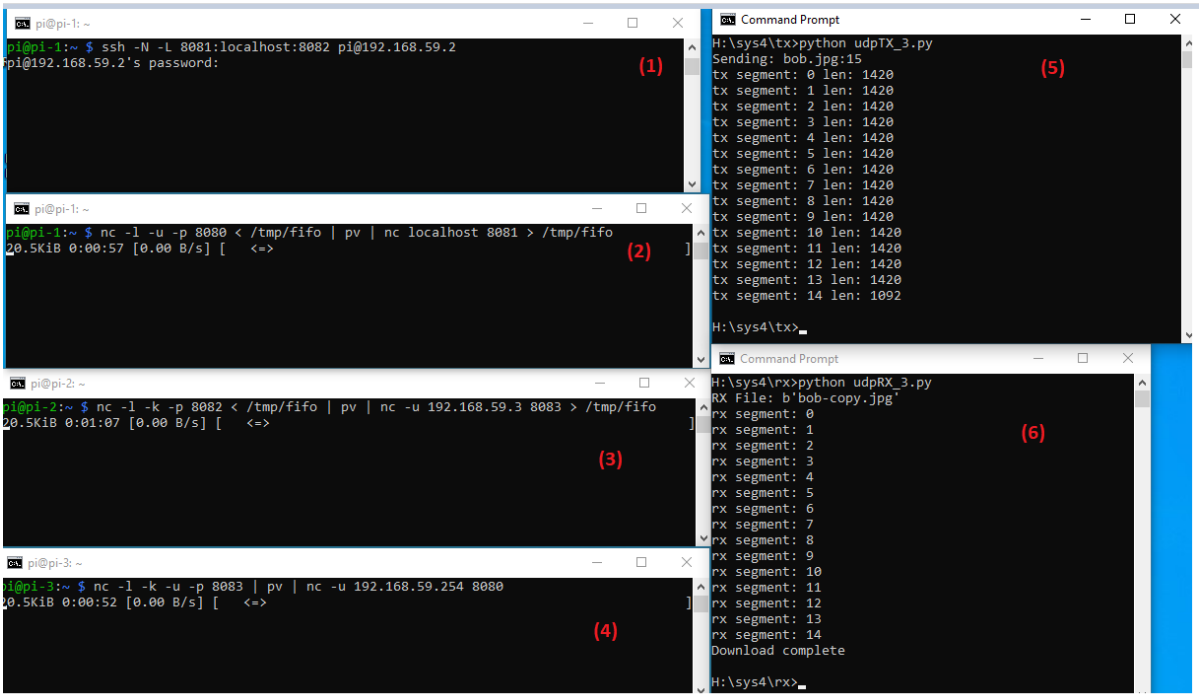

```
H:\sys4\rx>python udpRX_3.py
Traceback (most recent call last):
  File "H:\sys4\rx\udpRX_3.py", line 15, in <module>
    number_of_segments = int(data)
ValueError: invalid literal for int() with base 10: b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x01\x01\x01\x00H\x00H\x00\x00\xff
\xfe\x00\x13Created with GIMP\xff\xdb\x00C\x00\x03\x02\x02\x03\x02\x02\x03\x03\x03\x04\x03\x03\x04\x05\x08\x05\x05\x
04\x04\x05\n
```

(b)

To transmit UDP data through a TCP port we must use a special type of file, called a FIFO to initiate two-way communication between the channels. This is required over a standard pipe as a pipe would only communicate from the left of the pipe to the right, and not the other way, which is required by the TCP protocol.

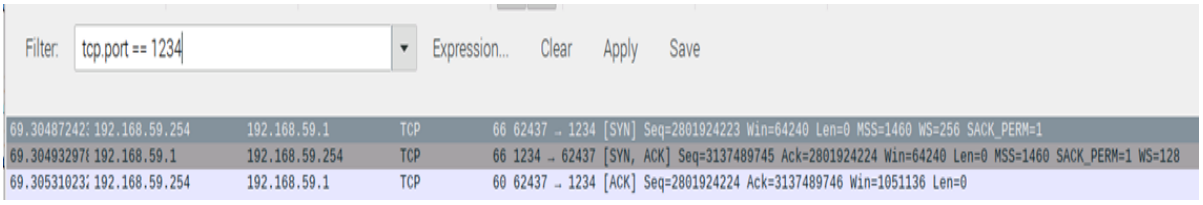
1. Open a (*TCP*) forward port through an ssh tunnel. This forwards all data from pi-1 8081 to pi-2 8082 over ssh, but only uses tcp packets as ssh is a tcp protocol.
2. Open a UDP listener on port 8080 for the PC to connect to, using FIFO for the two way communication required for TCP. The UDP data is sent through the TCP and through the ssh tunnel on 8081
3. On the other side of the ssh tunnel, it is the same process except we are converting the TCP data back to UDP and forwarding it to pi-3 on port 8083
4. Pi-3 receives the UDP data on port 8083 and forwards it back to the PC port 8080.
5. The PC runs the python script to send bob over UDP. The only change was the port to 8080.
6. The PC runs the python script to receive bob over UDP. Only change was the receive port to 8080.

As you can see from the image below, Bob's 20.5KiB was sent through the pipeline.



Lab 5

Q1

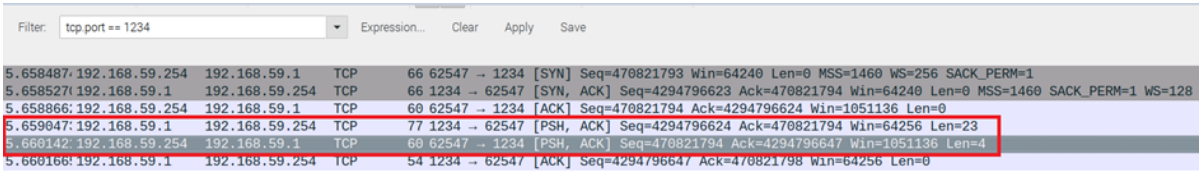


Destination port: 192.168.59.1
Source port: 192.168.59.254
Sequence number: 28019224223
Acknowledgment number: 0
Length: 0
Flags: SYN
Data: N/A

Destination port: 192.168.59.254
Source port: 192.168.59.1
Sequence number: 3137489745
Acknowledgment number: 28019224224
Length: 0
Flags: SYN, ACK
Data: N/A

Destination port: 192.168.59.1
Source port: 192.168.59.254
Sequence number: 28019224224
Acknowledgment number: 3137489746
Length: 0
Flags: ACK
Data: N/A

Q2



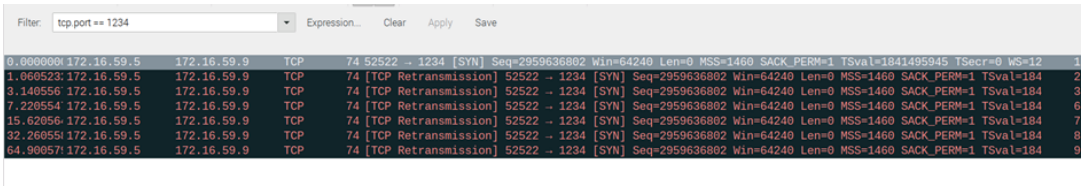
- First packet: Welcome message
- Second packet: helo message

Next SEQ = Current SEQ + LEN

Server to client – initial welcome message
SEQ: 4294796624
LEN: 23
Next SEQ: 4294796624 + 23 = 4294796647

Client to server - helo message
SEQ: 470821794
LEN: 4
Next SEQ: 470821794 + 4 = 470821798

Q3



How does RTO vary?
At each TCP retransmission, the RTO doubles. Eventually, the telnet command timeouts and quits, giving the message: “telnet: Unable to connect to remote host: Connection timed out”.

Why did the client give up?
This is because it did not receive a “SYN, ACK” response from 172.16.59.9 in its timeout period, which is shorter than the network delay of 100 seconds.

The telnet client sent 1 SYN packet and 6 SYN retransmission packets.

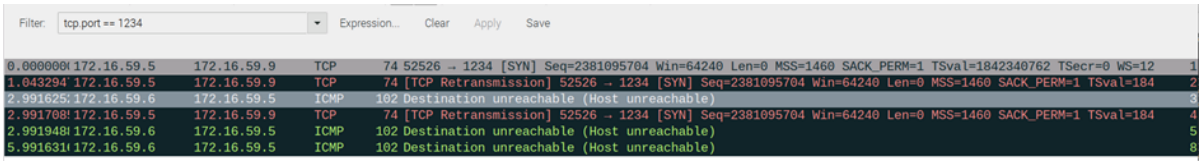
How does the cowsay server distinguish between retransmission SYN packets?
The cowsay server can detect a retransmission packet if the sequence number is less than what is expected. If so, it is a duplicate of a packet it has received earlier.

How does the cowsay server distinguish a new connection from a retransmission?
The client can send a RST control message if the server is not in a synchronised state, such as in the below example where an old duplicate packet disrupts the three way handshake in message 2. In message 3 the client receives a SYN,ACK with an incorrect ACK (given 6 when it should be 11), so the client sends a RST message in 4 to reset the server back to a listen state. From here the client can retry the three-way handshake.

	Client	Cowsay server
0	CLOSED	LISTEN
1	→ SYN (SEQ=10)	SYN-RECEIVED
2	(duplicate) SYN (SEQ=5)	SYN-RECEIVED
3	← SYN ,ACK (SEQ=X, ACK=6)	SYN-RECEIVED
4	→ RST (SEQ=6)	LISTEN

—

On replaying the command, the router at this point has noticed that no responses are coming from 172.16.59.9 on our previous transmission, and sends its own ICMP packets to our client on 172.16.59.5, telling us that the host is unreachable. On receiving these packets, telnet gives up and no more retransmission packets are sent.



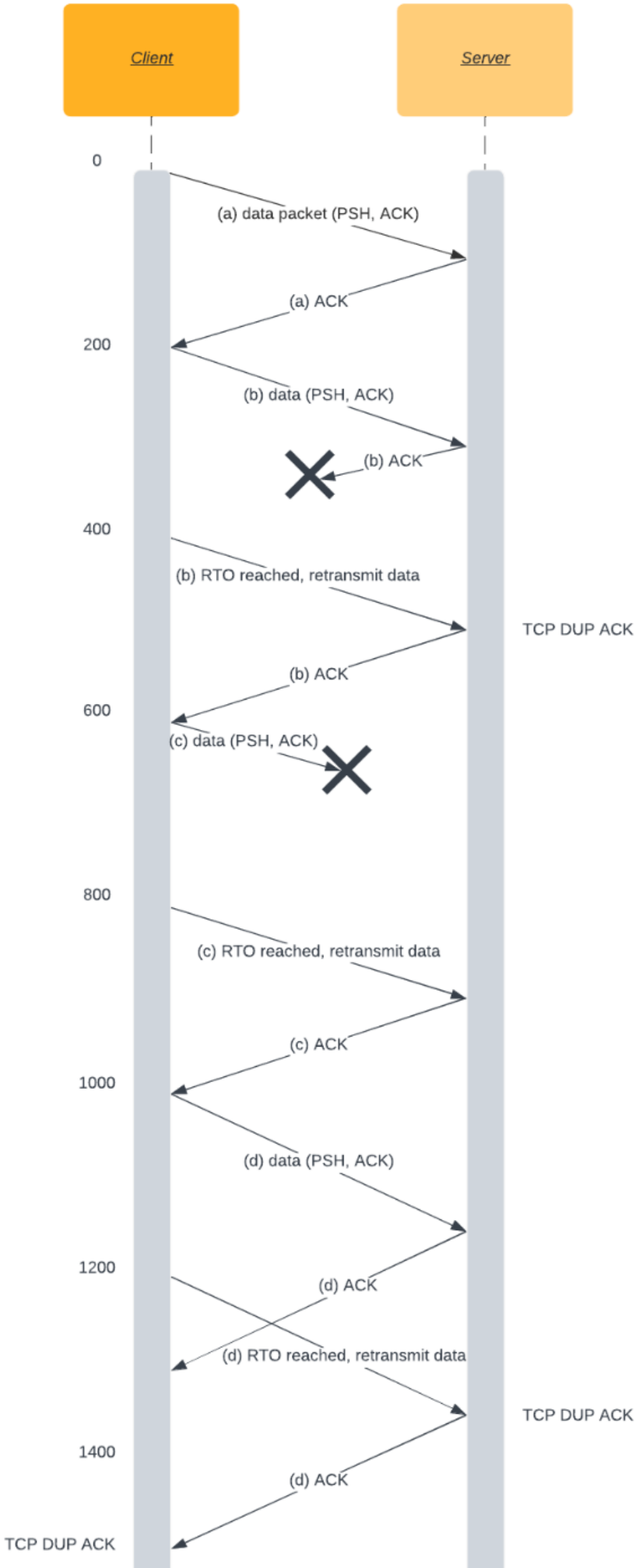
Q4

Client closing a connection with a server:

1. The client closes the connection with a FIN packet
2. Server responds with an ACK, and tells the application to close. The client then waits to receive a FIN packet from server
3. When the server is ready to close it sends a FIN back, and waits for an ACK packet from the client to close the connection
    a. These packets may be sent together as a (FIN, ACK) packet
4. The client acknowledges the FIN packet by sending an ACK packet back, waits for double the *maximum segment lifetime*, which is twice as long as a TCP packet is expected to exist in the network. This allows the client to resend the final ACK if the first ACK is lost.
5. Once the server has received the ACK packet, it closes the connection

Once the client reconnects to the server, this will be after double the maximum segment lifetimes, so all delayed packets from the previous connection will be flushed through the system by the time a new connection is established.

Q5


Client
Server
0
(a) data packet (PSH, ACK)
(a) ACK
200
(b) data (PSH, ACK)
(b) ACK
400
(b) RTO reached, retransmit data
TCP DUP ACK
(b) ACK
600
(c) data (PSH, ACK)
800
(c) RTO reached, retransmit data
(c) ACK
1000
(d) data (PSH, ACK)
1200
(d) ACK
(d) RTO reached, retransmit data
TCP DUP ACK
1400
(d) ACK
TCP DUP ACK

Lab 6

Q1

Size of data from packet headers

TCP
We can calculate the size of the data by looking in the IP header to find the “Total Length” field which gives the total size of the packet, with both the IP and TCP headers included.

We can then find the size of the IP header by the “IP Header Length” field in the IP header. We can find the size of the TCP header by the “Data Offset” field in the TCP header. These give the size of their respective headers in 32-bit words, so must be multiplied by 4 to get a result in bytes.

The total size of the of the data can be calculated by:
*Data size (bytes) = IP Header Length – (IP Header length + Data Offset) * 4*

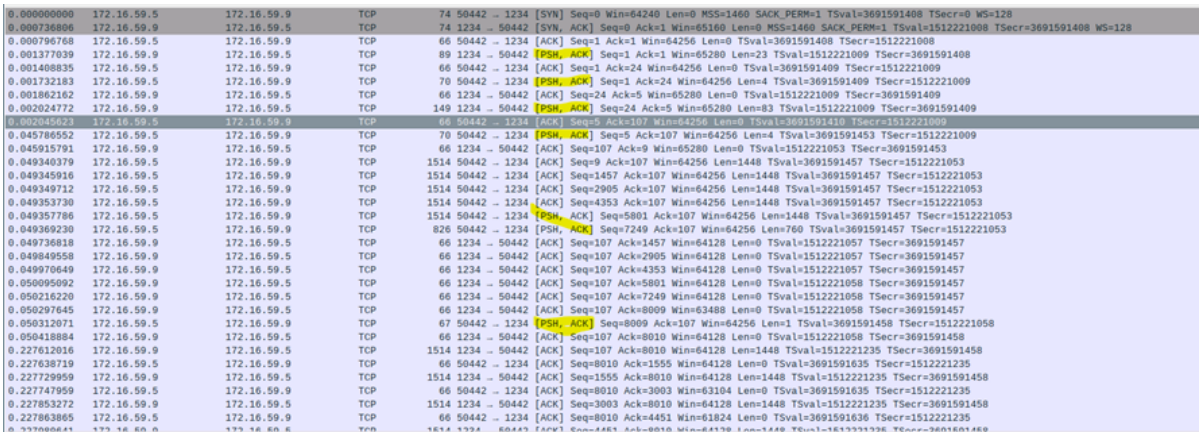
UDP
The “Length” field from the UDP header specifies the length in bytes of the UDP header and UDP data. As all UDP headers are 8 bytes in length, the size of the data can be calculated by:

Data size (bytes) = UDP Length – 8

What determines when the PSH flag is set?

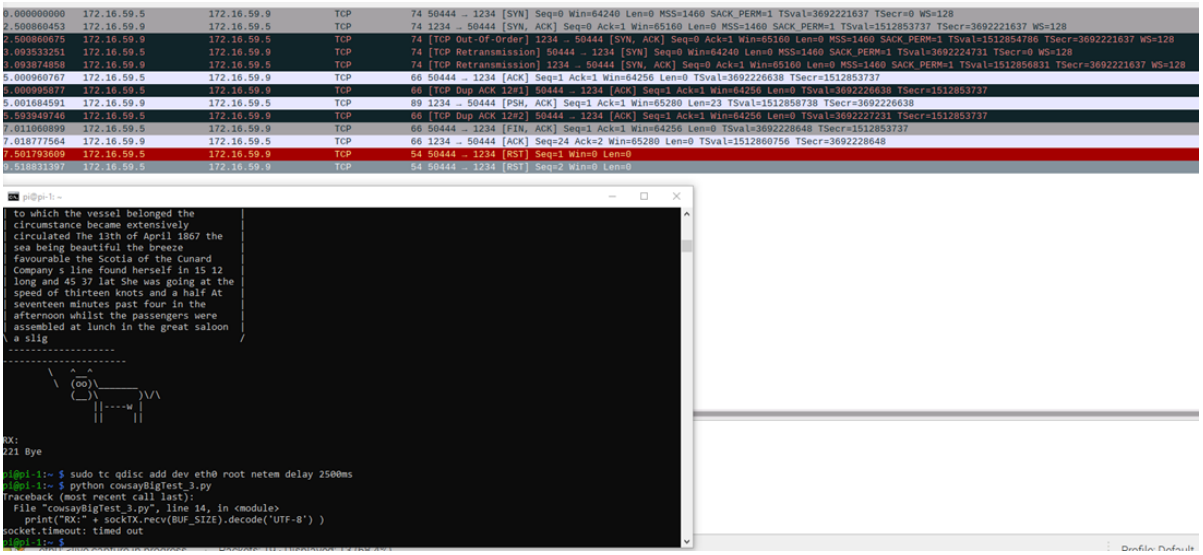
The PSH flag allows the sending application to start sending the data and not to wait for more packets, even when the buffer is not full. This is sent at the end of transmission so the receiver won’t sit around waiting for data to fill the buffer when it isn’t coming.

So in this scenario, a PSH flag is telling the receiver that they have finished sending their message, and the receiver can start sending a response. The double PSH, ACK in the middle of the transmission is an exception to this, but I believe is due to external factors.



What scenario will the RST flag be set?

This was set when the connection timed out and the program exited. This is to signify an “Abrupt connection release” which forces the TCP connection to close.



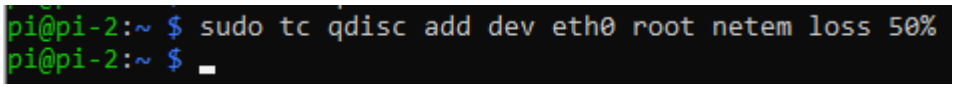
Q2

MSS

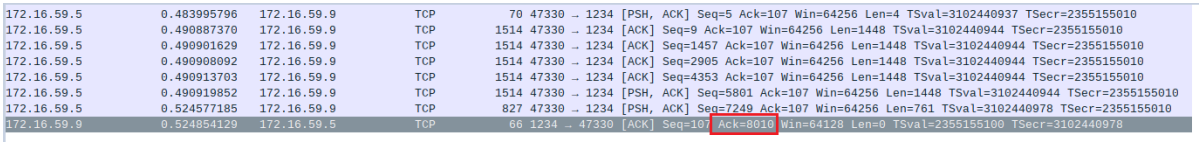
The MSS is sent in the first two packets, [SYN, (SYN, ACK)], which dictate the largest segment the host will accept. In this example, both pi-1 and pi-2 set it to 1460 bytes.

Cumulative ACK

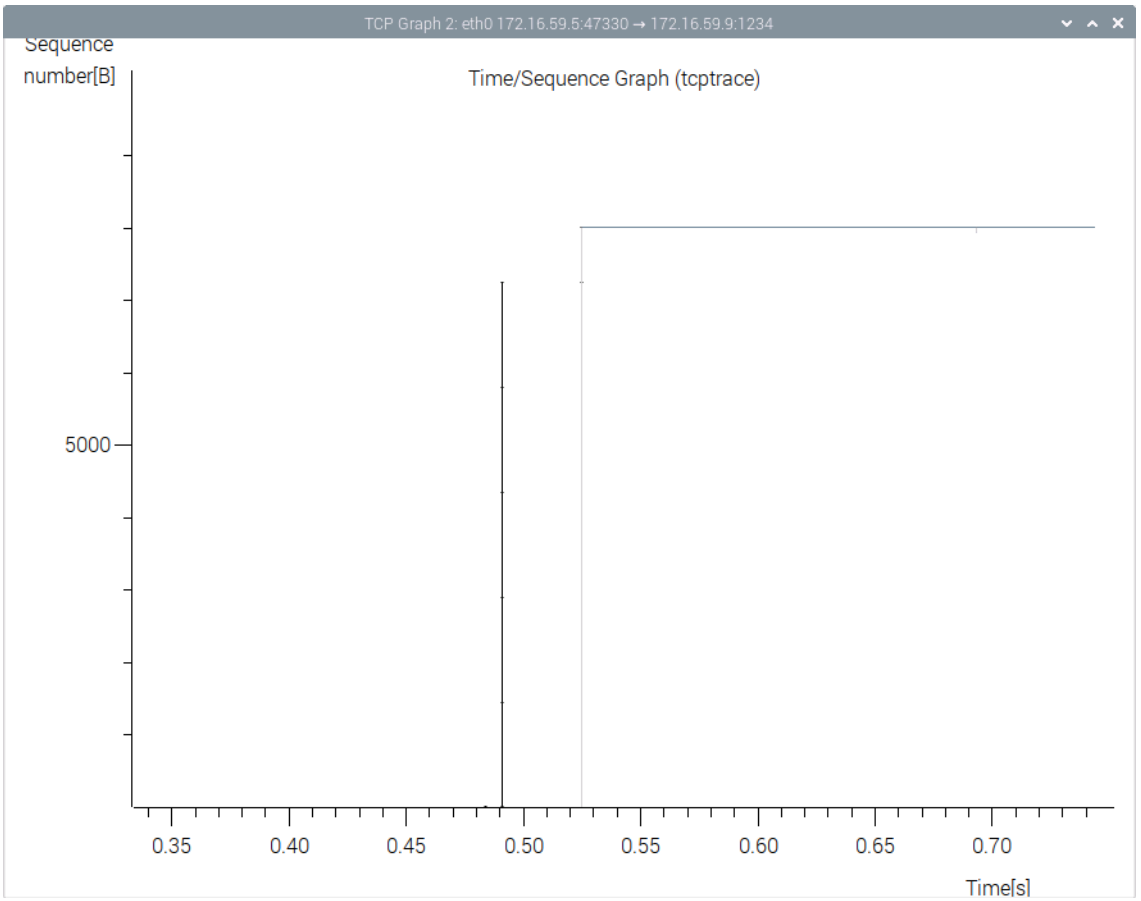
To create a scenario for cumulative ACK packets when sending a large amount of data, I set the loss rate to 50%, to maximise the chance of pi-1 not receiving the ACK packets in sequence and instead a large cumulative ACK packet.



From the screenshot below, we can see from pi-1 perspective that it sends 7 packets with 8010 bytes of data, and then receives a single “cumulative” ACK packet in return acknowledging that pi-2 has received all packets up to this point in a single packet.



In the TCP trace you can clearly see the 8000 bytes sent at 0.49 and the *single* cumulative ACK received at 0.52.



Q3

A TCP checksum is calculated by one's complement of the sum of all the packet data. So, if for any given input to a one's complement calculation the output is 0xFFFF, it is possible for a packet to have the checksum 0xFFFF.

There is a possible packet with the TCP checksum of 0xFFFF, as the value 131071 (among others) gives an output of 0xFFFF.

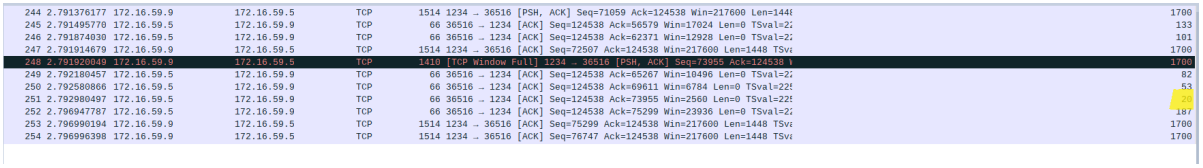
```
def ones_complement(x: int):
    x = (x >> 16) + (x & 0xffff) # Creates unsigned 32 bit
    x = ~x & 0xffff
    return x

print(ones_complement(0x1ffff) == 0xffff)
```

0x1ffff >> 16 + 131071 & 0xffff = 0x10000
~0x10000 = 0xfffffffffffefffff

0xffffffffffefffff & 0xffff = **0xffff**

Q4

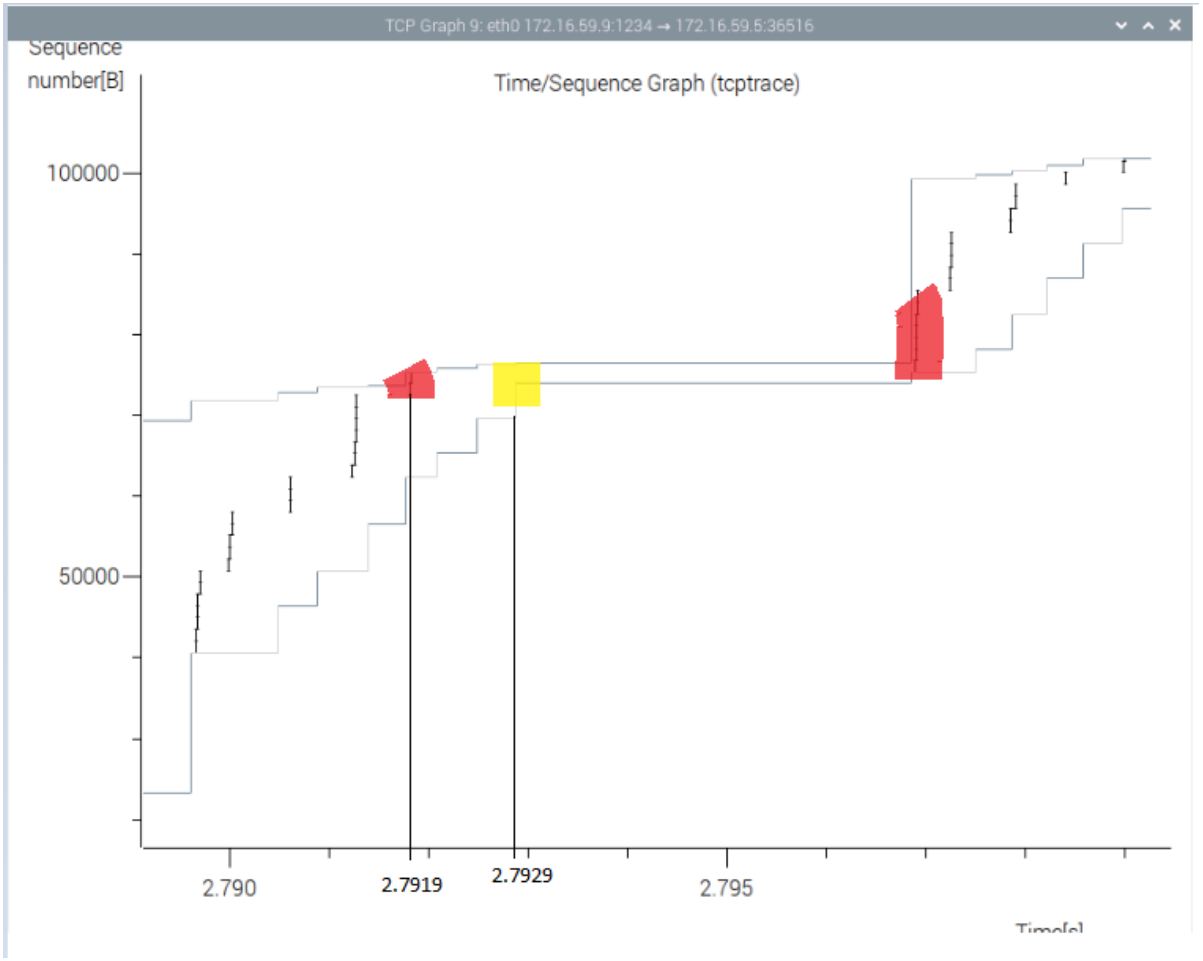
Window size reaches zero



This wireshark packet trace shows when pi-2 window size reaches 0, and pi-2 sends a TCP window full packet at *2.7919* to inform pi-1 to stop sending packets (there's a line in the red highlight).

However, pi-1 does not respond in time, and the window size maxes out, resulting in lost packets. As we can see from the packet trace above, the last packet was received by pi-2 at 2.7929, with a remaining window size of 20. Any packets sent after were lost.

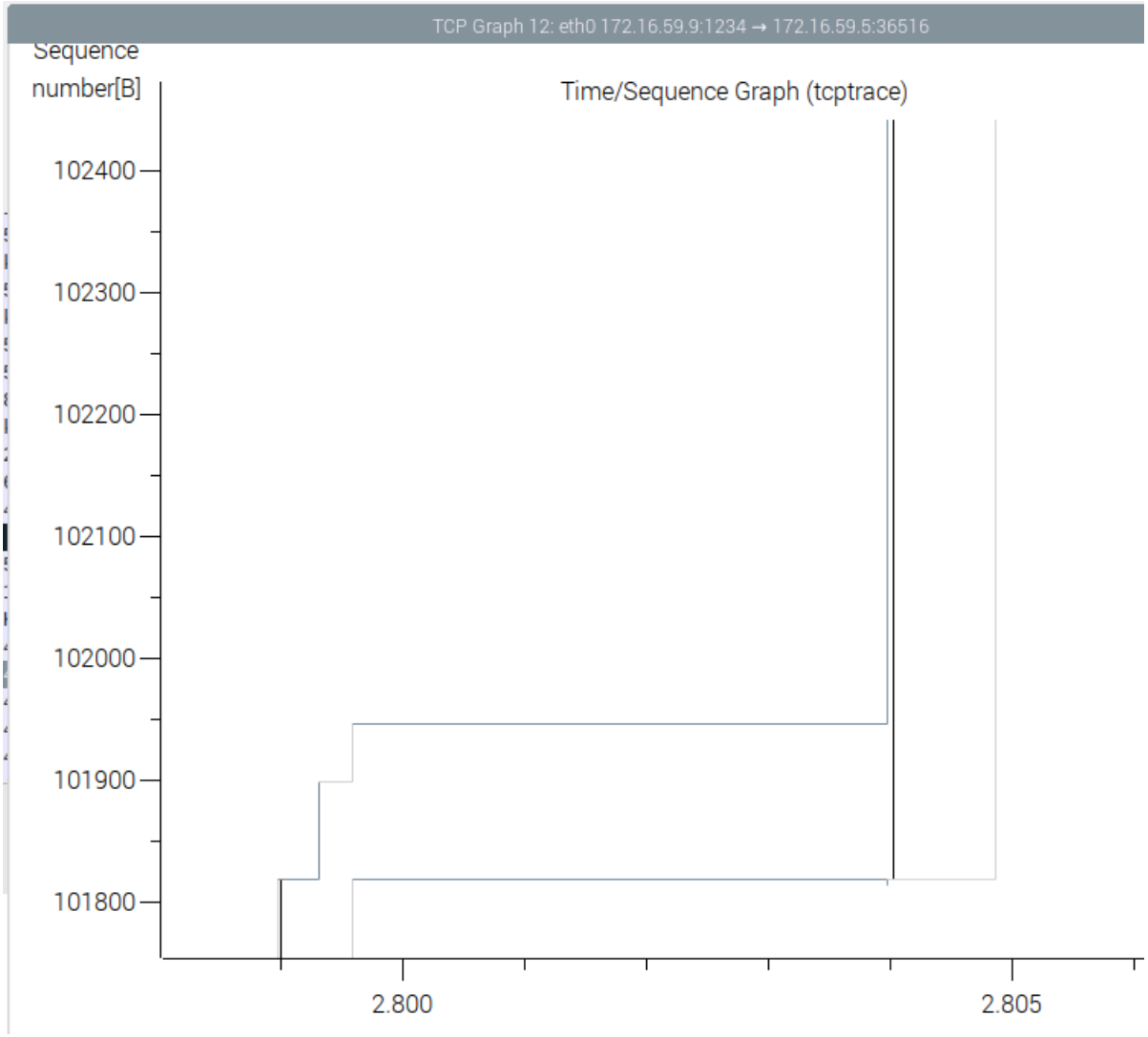
Finally, at 2.9796, pi-2 starts sending ACKs back to pi-1 to inform it that the window size buffer has recovered and it can start sending packets again. Unfortunately, the window size is still not large enough and it fills quickly up to 2.799.



Window size variations – TCP Window Update

The TCP window updates at 2.804, which causes the “ceiling” of the TCP trace to drastically increase, as indicated by the leading light blue bar. However, this lets pi-1 to restart sending packets again now there is more space for them (immediately filling the window again) as indicated by the black bar.

273	2.798506940	172.16.59.9	172.16.59.5	TCP	1514 1234 → 36516 [PSH, ACK] Seq=98571 Ack=124538 Win=217600 Len=144
274	2.798679829	172.16.59.5	172.16.59.9	TCP	66 36516 → 1234 [ACK] Seq=124538 Ack=91227 Win=10496 Len=0 TSval=2
275	2.799061997	172.16.59.5	172.16.59.9	TCP	66 36516 → 1234 [ACK] Seq=124538 Ack=95675 Win=6144 Len=0 TSval=22
276	2.799086478	172.16.59.9	172.16.59.5	TCP	1514 1234 → 36516 [ACK] Seq=100019 Ack=124538 Win=217600 Len=1448 TS
277	2.799091497	172.16.59.9	172.16.59.5	TCP	418 [TCP Window Full] 1234 → 36516 [PSH, ACK] Seq=101467 Ack=124538
278	2.799401868	172.16.59.5	172.16.59.9	TCP	66 36516 → 1234 [ACK] Seq=124538 Ack=98571 Win=3328 Len=0 TSval=22
279	2.799672814	172.16.59.5	172.16.59.9	TCP	66 36516 → 1234 [ACK] Seq=124538 Ack=101819 Win=128 Len=0 TSval=22
280	2.804072383	172.16.59.5	172.16.59.9	TCP	66 [TCP Window Update] 36516 → 1234 [ACK] Seq=124538 Ack=101819 Wi
281	2.804123864	172.16.59.9	172.16.59.5	TCP	1514 1234 → 36516 [ACK] Seq=101819 Ack=124538 Win=217600 Len=1448 TS
282	2.804130290	172.16.59.9	172.16.59.5	TCP	1514 1234 → 36516 [ACK] Seq=103267 Ack=124538 Win=217600 Len=1448 TS
283	2.804133938	172.16.59.9	172.16.59.5	TCP	1514 1234 → 36516 [ACK] Seq=104715 Ack=124538 Win=217600 Len=1448 TS
284	2.804137364	172.16.59.9	172.16.59.5	TCP	1514 1234 → 36516 [ACK] Seq=106163 Ack=124538 Win=217600 Len=1448 TS
285	2.804140920	172.16.59.9	172.16.59.5	TCP	1514 1234 → 36516 [ACK] Seq=107611 Ack=124538 Win=217600 Len=1448 TS


TCP Graph 12: eth0 172.16.59.9:1234 → 172.16.59.5:36516
Sequence number[B]
Time/Sequence Graph (tcptrace)
102400
102300
102200
102100
102000
101900
101800
2.800
2.805

Lab 7

Q1

4.690397071	192.168.59.254	192.168.59.1	UDP	60	62672 → 8000 Len=12
4.690735608	192.168.59.1	192.168.59.254	UDP	54	44117 → 8000 Len=12
4.803718275	192.168.59.254	192.168.59.1	UDP	60	62672 → 8000 Len=1
4.804085386	192.168.59.1	192.168.59.254	UDP	43	44117 → 8000 Len=1
4.929214608	192.168.59.254	192.168.59.1	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=0, ID=c5e0) [Reassembled in #710]
4.929338793	192.168.59.254	192.168.59.1	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=1480, ID=c5e0) [Reassembled in #710]
4.929453108	192.168.59.254	192.168.59.1	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=2960, ID=c5e0) [Reassembled in #710]
4.929566756	192.168.59.254	192.168.59.1	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=4440, ID=c5e0) [Reassembled in #710]
4.929692405	192.168.59.254	192.168.59.1	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=5920, ID=c5e0) [Reassembled in #710]
4.930019682	192.168.59.254	192.168.59.1	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=7400, ID=c5e0) [Reassembled in #710]
4.930029516	192.168.59.254	192.168.59.1	UDP	162	62672 → 8000 Len=9000
4.930730423	192.168.59.1	192.168.59.254	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=0, ID=eef8) [Reassembled in #718]
4.930762738	192.168.59.1	192.168.59.254	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=1480, ID=eef8) [Reassembled in #718]
4.930786108	192.168.59.1	192.168.59.254	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=2960, ID=eef8) [Reassembled in #718]
4.930883442	192.168.59.1	192.168.59.254	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=4440, ID=eef8) [Reassembled in #718]
4.930942923	192.168.59.1	192.168.59.254	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=5920, ID=eef8) [Reassembled in #718]
4.930966553	192.168.59.1	192.168.59.254	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=7400, ID=eef8) [Reassembled in #718]
4.931025071	192.168.59.1	192.168.59.254	UDP	162	44117 → 8000 Len=9000
5.038919330	192.168.59.254	192.168.59.1	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=0, ID=c5e9) [Reassembled in #754]
5.039032645	192.168.59.254	192.168.59.1	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=1480, ID=c5e9) [Reassembled in #754]
5.039152979	192.168.59.254	192.168.59.1	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=2960, ID=c5e9) [Reassembled in #754]
5.039282905	192.168.59.254	192.168.59.1	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=4440, ID=c5e9) [Reassembled in #754]
5.039398849	192.168.59.254	192.168.59.1	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=5920, ID=c5e9) [Reassembled in #754]
5.039533645	192.168.59.254	192.168.59.1	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=7400, ID=c5e9) [Reassembled in #754]
5.039542108	192.168.59.254	192.168.59.1	UDP	162	62672 → 8000 Len=9000
5.039991164	192.168.59.1	192.168.59.254	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=0, ID=ef02) [Reassembled in #761]
5.040019405	192.168.59.1	192.168.59.254	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=1480, ID=ef02) [Reassembled in #761]
5.040037423	192.168.59.1	192.168.59.254	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=2960, ID=ef02) [Reassembled in #761]
5.040067738	192.168.59.1	192.168.59.254	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=4440, ID=ef02) [Reassembled in #761]
5.040134164	192.168.59.1	192.168.59.254	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=5920, ID=ef02) [Reassembled in #761]
5.040152516	192.168.59.1	192.168.59.254	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=7400, ID=ef02) [Reassembled in #761]
5.040198997	192.168.59.1	192.168.59.254	UDP	162	44117 → 8000 Len=9000
5.148408608	192.168.59.254	192.168.59.1	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=0, ID=c5f5) [Reassembled in #801]
5.148535553	192.168.59.254	192.168.59.1	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=1480, ID=c5f5) [Reassembled in #801]
5.148540293	192.168.59.254	192.168.59.1	UDP	60	62672 → 8000 Len=2972
5.148729293	192.168.59.1	192.168.59.254	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=0, ID=ef05) [Reassembled in #804]
5.148744219	192.168.59.1	192.168.59.254	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=1480, ID=ef05) [Reassembled in #804]
5.148755127	192.168.59.1	192.168.59.254	UDP	54	44117 → 8000 Len=2972

The image is sent fragmented through 6 different packets.

The fragment offset is used to indicate the start of the data in the fragment in *relation to the start of the data in the original packet*.

The fragment offset field is calculated by *previous fragment offset + size of the data in the last packet*. The size of this data is calculated by *total packet length - header size*.

All fragments of the packet except the last one will have the “More fragments” field set. This is to identify that there are more fragments being sent after this one.

Q2

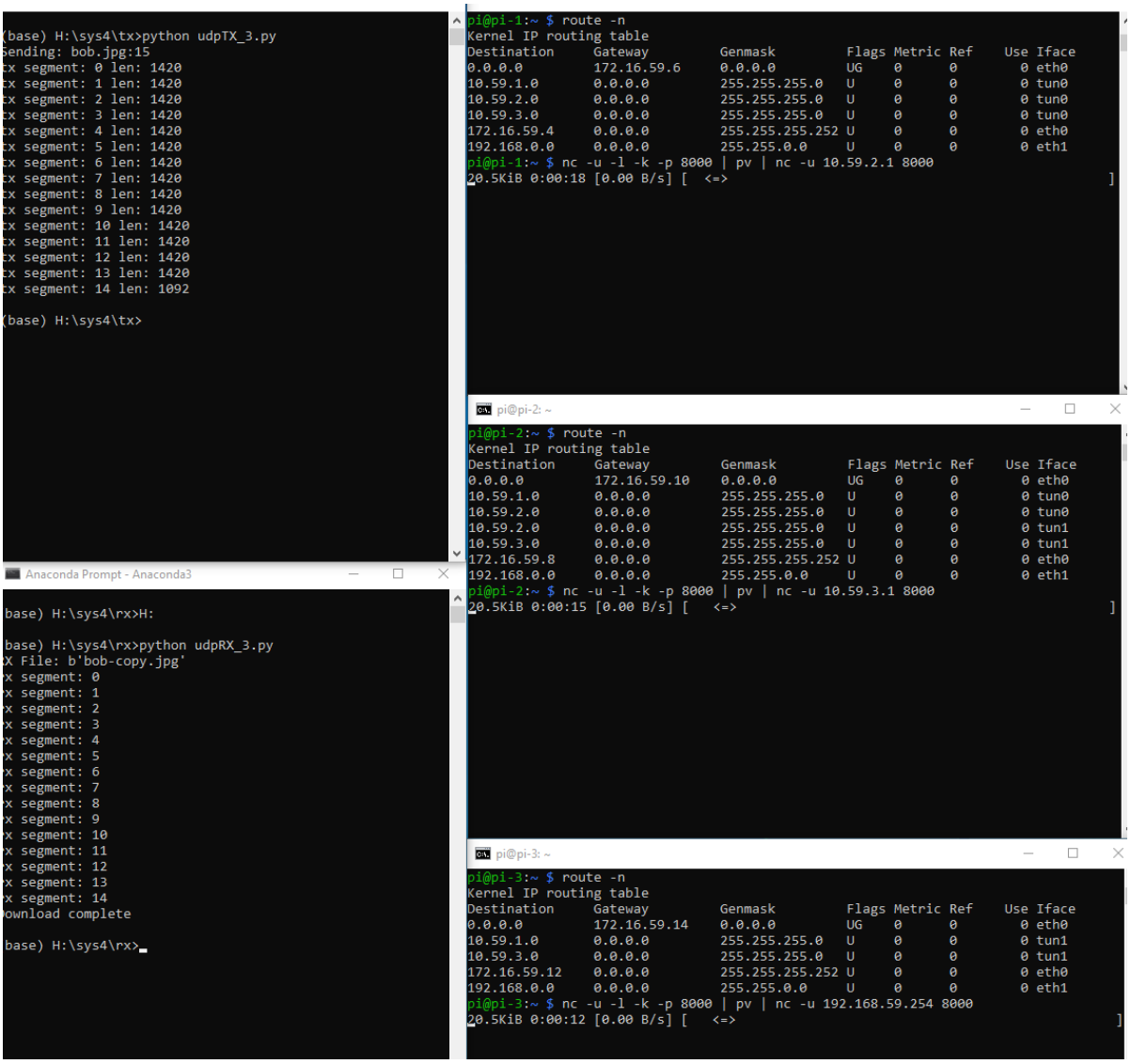
PC ping 10.59.1.1 & 172.16.59.5

```
IPv4 Route Table
===========================================================================
Active Routes:
Network Destination        Netmask          Gateway       Interface  Metric
          0.0.0.0          0.0.0.0   144.32.179.254    144.32.179.22     25
        127.0.0.0        255.0.0.0         On-link         127.0.0.1    331
        127.0.0.1  255.255.255.255         On-link         127.0.0.1    331
  127.255.255.255  255.255.255.255         On-link         127.0.0.1    331
     144.32.178.0    255.255.254.0         On-link     144.32.179.22    281
    144.32.179.22  255.255.255.255         On-link     144.32.179.22    281
   144.32.179.255  255.255.255.255         On-link     144.32.179.22    281
      192.168.0.0      255.255.0.0         On-link    192.168.59.254    291
   192.168.59.254  255.255.255.255         On-link    192.168.59.254    291
  192.168.255.255  255.255.255.255         On-link    192.168.59.254    291
        224.0.0.0        240.0.0.0         On-link         127.0.0.1    331
        224.0.0.0        240.0.0.0         On-link    192.168.59.254    291
        224.0.0.0        240.0.0.0         On-link     144.32.179.22    281
  255.255.255.255  255.255.255.255         On-link         127.0.0.1    331
  255.255.255.255  255.255.255.255         On-link    192.168.59.254    291
  255.255.255.255  255.255.255.255         On-link     144.32.179.22    281
===========================================================================
Persistent Routes:
  None
```

IP Routes for PC

10.59.1.1 and 172.16.59.5 are not in the PC's route table, so these pings will get directed to the default interface, 144.32.179.22. The interface is what is responsible for reaching the gateway router, at 144.32.179.254, which is the department router that can direct traffic to the outside internet. However, 10.0.0.0/8 and 172.16.0.0/12 are private addresses, so the router will instead route the ping request to machines on the york network, if a machine with that address exists. If not, the router will respond with a ICMP packet telling the PC that no host exists with that address and drop the packet.

Transferring bob.jpg across the tunnels



Q3

The IP checksum field is calculated using the TTL field, so will need to be updated every time TTL is updated.

I use the command: “sudo sysctl net.ipv4.ip_forward=1” so that the Pi’s forward packets to other destinations.

Pi-1
sudo ip route add 10.200.0.0/24 via 10.59.2.2 dev tun0

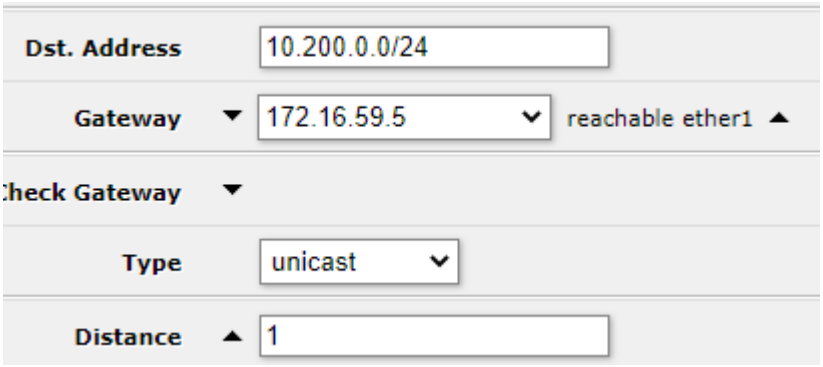
```
pi@pi-1:~ $ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         172.16.59.6     0.0.0.0         UG    0      0        0 eth0
10.59.1.0       0.0.0.0         255.255.255.0   U     0      0        0 tun0
10.59.2.0       0.0.0.0         255.255.255.0   U     0      0        0 tun0
10.200.0.0      10.59.2.2       255.255.255.0   UG    0      0        0 tun0
172.16.59.4     0.0.0.0         255.255.255.252 U     0      0        0 eth0
192.168.0.0     0.0.0.0         255.255.0.0     U     0      0        0 eth1
pi@pi-1:~ $
```

Pi-2

sudo ip route add 10.200.0.0/24 via 172.16.59.10 dev eth0

```
pi@pi-2:~ $ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         172.16.59.10    0.0.0.0         UG    0      0        0 eth0
10.59.1.0       0.0.0.0         255.255.255.0   U     0      0        0 tun0
10.59.2.0       0.0.0.0         255.255.255.0   U     0      0        0 tun0
10.200.0.0      172.16.59.10    255.255.255.0   UG    0      0        0 eth0
172.16.59.8     0.0.0.0         255.255.255.252 U     0      0        0 eth0
192.168.0.0     0.0.0.0         255.255.0.0     U     0      0        0 eth1
pi@pi-2:~ $
```

Router



The idea was for the ping request from pi-1 to be routed through tun0 to pi-2 (via 10.59.2.2 dev tun0), which would then route the packet through the router gateway 172.16.59.10. The router would then forward the packet back to pi-1, which would then send the packet through the tunnel again.

The captured packets below is from a single ping request, and we can see it circulating the system 3 times across the different interfaces before the packet seems to be dropped. I'm not sure why the packet was dropped. The ping request hangs as the TTL is decremented but never reaches zero before it is dropped.

Pi-1

```
10.59.1.2 9.551579346  10.200.0.1  ICMP   118 Echo (ping) request  id=0x06bb, seq=1/256, ttl=64 (no response found!)  eth0
10.59.1.2 9.551990587  10.200.0.1  ICMP    98 Echo (ping) request  id=0x06bb, seq=1/256, ttl=62 (no response found!)  eth0
10.59.1.2 9.551533161  10.200.0.1  ICMP    84 Echo (ping) request  id=0x06bb, seq=1/256, ttl=64 (no response found!)  tun0
```

Pi-2

```
832 7.318785535 10.59.1.2 10.200.0.1  ICMP   84 Echo (ping) request  id=0x06bb, seq=1/256, ttl=64 (no response found!) tun0
833 7.318785535 10.59.1.2 10.200.0.1  ICMP  118 Echo (ping) request  id=0x06bb, seq=1/256, ttl=64 (no response found!) eth0
834 7.318860109 10.59.1.2 10.200.0.1  ICMP   98 Echo (ping) request  id=0x06bb, seq=1/256, ttl=63 (no response found!) eth0
```

Lab 8

Q1

Direct broadcast of eth0 for Pi-2

```
pi@pi-2:~ $ ip route
default via 172.16.59.10 dev eth0 onlink
172.16.59.8/30 dev eth0 proto kernel scope link src 172.16.59.9
192.168.0.0/16 dev eth1 proto kernel scope link src 192.168.59.2
pi@pi-2:~ $
```

172.16.59.8/30 address.
CIDR /30 => 255.255.255.252
Number of bits for subnetting: 6
Number of bits left to host: 8 - 6 = 2
Value of last bit used for subnet masking: 2^2 = 4
As our subnet starts on 172.16.59.8 - 172.16.59.11 (4 addresses)
Therefore our direct broadcast on eth0 is **172.16.59.11**

Ping direct & local broadcast

Direct broadcast, on the 192.168.0.0/16 subnet. This receives replies from pi-1 and pi-3 as all pi's are on this subnet. We can see the responses on eth1 (router network).

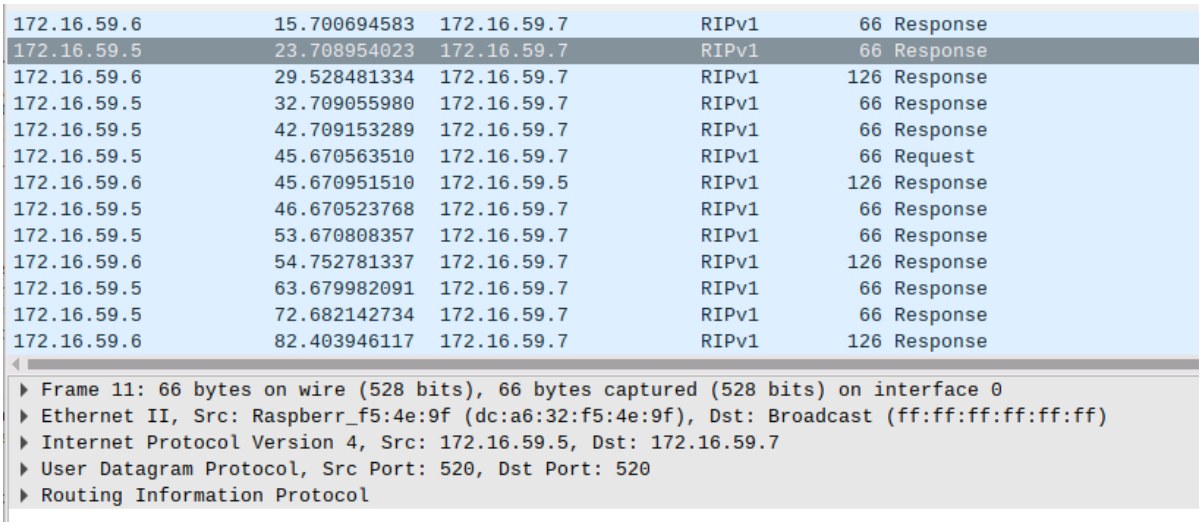
```
3304 7.875160936 192.168.59.2  192.168.255.255  ICMP   98 Echo (ping) request  id=0x0682, seq=1/256, ttl=64 (no response f
3307 7.875538808 192.168.59.3  192.168.59.2      ICMP   98 Echo (ping) reply    id=0x0682, seq=1/256, ttl=64
3308 7.875555456 192.168.59.1  192.168.59.2      ICMP   98 Echo (ping) reply    id=0x0682, seq=1/256, ttl=64
3555 8.884530673 192.168.59.2  192.168.255.255  ICMP   98 Echo (ping) request  id=0x0682, seq=2/512, ttl=64 (no response f
3557 8.884827397 192.168.59.3  192.168.59.2      ICMP   98 Echo (ping) reply    id=0x0682, seq=2/512, ttl=64
3558 8.884874286 192.168.59.1  192.168.59.2      ICMP   98 Echo (ping) reply    id=0x0682, seq=2/512, ttl=64
3953 9.924534192 192.168.59.2  192.168.255.255  ICMP   98 Echo (ping) request  id=0x0682, seq=3/768, ttl=64 (no response f
3955 9.924865397 192.168.59.3  192.168.59.2      ICMP   98 Echo (ping) reply    id=0x0682, seq=3/768, ttl=64
3956 9.924913897 192.168.59.1  192.168.59.2      ICMP   98 Echo (ping) reply    id=0x0682, seq=3/768, ttl=64
4183 10.96453050 192.168.59.2  192.168.255.255  ICMP   98 Echo (ping) request  id=0x0682, seq=4/1024, ttl=64 (no response
4185 10.96488139 192.168.59.3  192.168.59.2      ICMP   98 Echo (ping) reply    id=0x0682, seq=4/1024, ttl=64
4186 10.96500358 192.168.59.1  192.168.59.2      ICMP   98 Echo (ping) reply    id=0x0682, seq=4/1024, ttl=64
4730 12.00462382 192.168.59.2  192.168.255.255  ICMP   98 Echo (ping) request  id=0x0682, seq=5/1280, ttl=64 (no response
4731 12.00501927 192.168.59.3  192.168.59.2      ICMP   98 Echo (ping) reply    id=0x0682, seq=5/1280, ttl=64
4732 12.00502923 192.168.59.1  192.168.59.2      ICMP   98 Echo (ping) reply    id=0x0682, seq=5/1280, ttl=64
6364 15.39266705 192.168.59.2  192.168.59.254    ICMP  570 Destination unreachable (Port unreachable)
```

Local broadcast - will be broadcast through the local network on the switch, which is equivalent to the 172.16.59.8/30 network, which pi-1 and pi-3 are not a part of. Therefore the broadcast goes through eth0 (switch network). As no other devices are part of this network, we get no response

```
1 0.000000000 172.16.59.9  255.255.255.255  ICMP   98 Echo (ping) request  id=0x068b, seq=1/256, ttl=64 (broadcast)
2 1.039640974 172.16.59.9  255.255.255.255  ICMP   98 Echo (ping) request  id=0x068b, seq=2/512, ttl=64 (broadcast)
3 2.079643560 172.16.59.9  255.255.255.255  ICMP   98 Echo (ping) request  id=0x068b, seq=3/768, ttl=64 (broadcast)
4 3.119646185 172.16.59.9  255.255.255.255  ICMP   98 Echo (ping) request  id=0x068b, seq=4/1024, ttl=64 (broadcast)
5 4.159683811 172.16.59.9  255.255.255.255  ICMP   98 Echo (ping) request  id=0x068b, seq=5/1280, ttl=64 (broadcast)
```

Q2

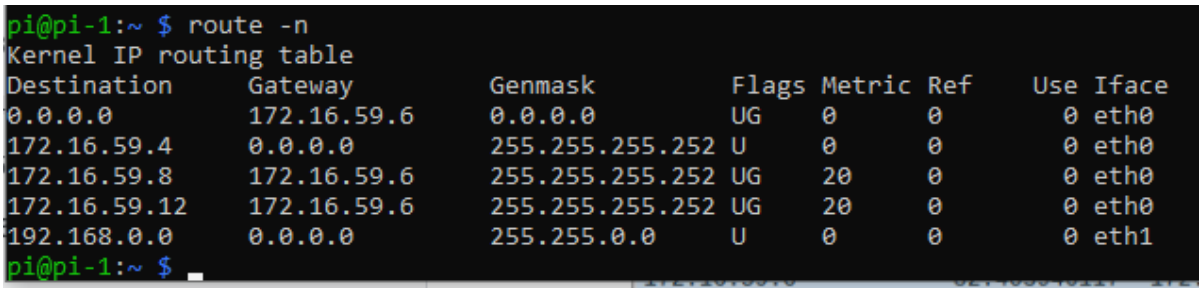
RIPv1 uses port 520, with a broadcast address of 172.16.59.7, which is the broadcast address of pi-1 on the eth0 network.



172.16.59.6	15.700694583	172.16.59.7	RIPv1	66	Response
172.16.59.5	23.708954023	172.16.59.7	RIPv1	66	Response
172.16.59.6	29.528481334	172.16.59.7	RIPv1	126	Response
172.16.59.5	32.709055980	172.16.59.7	RIPv1	66	Response
172.16.59.5	42.709153289	172.16.59.7	RIPv1	66	Response
172.16.59.5	45.670563510	172.16.59.7	RIPv1	66	Request
172.16.59.6	45.670951510	172.16.59.5	RIPv1	126	Response
172.16.59.5	46.670523768	172.16.59.7	RIPv1	66	Response
172.16.59.5	53.670808357	172.16.59.7	RIPv1	66	Response
172.16.59.6	54.752781337	172.16.59.7	RIPv1	126	Response
172.16.59.5	63.679982091	172.16.59.7	RIPv1	66	Response
172.16.59.5	72.682142734	172.16.59.7	RIPv1	66	Response
172.16.59.6	82.403946117	172.16.59.7	RIPv1	126	Response

```
▸ Frame 11: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
▸ Ethernet II, Src: Raspberr_f5:4e:9f (dc:a6:32:f5:4e:9f), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
▸ Internet Protocol Version 4, Src: 172.16.59.5, Dst: 172.16.59.7
▸ User Datagram Protocol, Src Port: 520, Dst Port: 520
▸ Routing Information Protocol
```

All the pis have interacted over the switch using the rip protocol. This has created new routes, with a metric of 20, signifying unroutable by pi-1 as RIPv1 is limited to 15 hops.

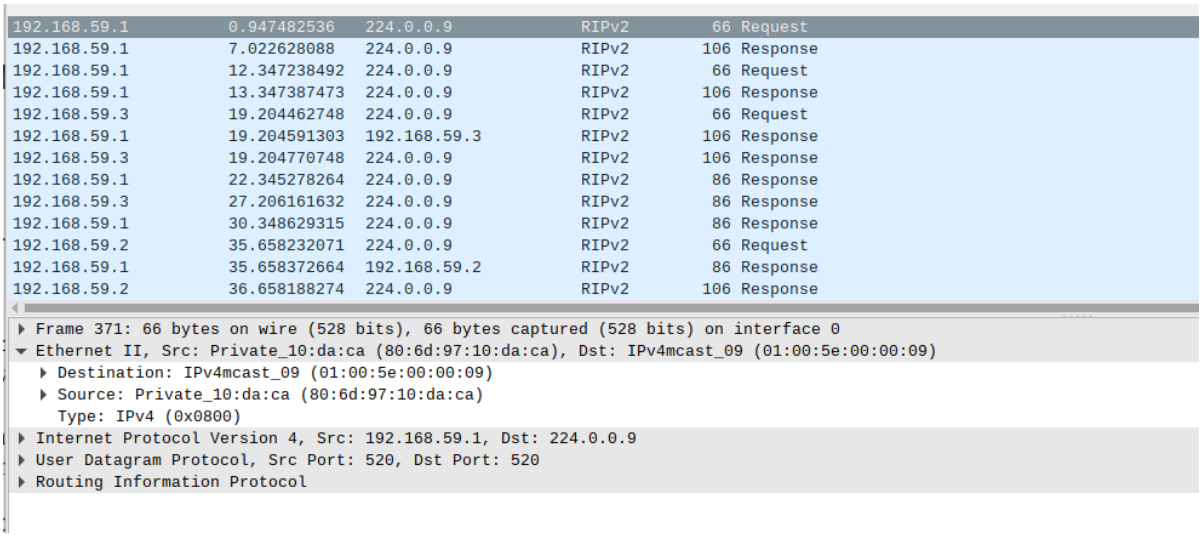


```
pi@pi-1:~ $ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         172.16.59.6     0.0.0.0         UG    0      0        0 eth0
172.16.59.4     0.0.0.0         255.255.255.252 U     0      0        0 eth0
172.16.59.8     172.16.59.6     255.255.255.252 UG    20     0        0 eth0
172.16.59.12    172.16.59.6     255.255.255.252 UG    20     0        0 eth0
192.168.0.0     0.0.0.0         255.255.0.0     U     0      0        0 eth1
pi@pi-1:~ $
```

The routes that have been added (172.16.59.8, 172.16.59.12), are subnets for each of the pi networks on the switch, communicated by each of the raspberry pi's acting as routers. Pi-1 can access its own subnet, but cannot reach Pi-2 or Pi-3 subnet so these are marked with a metric of 20, signifying that they are unroutable by pi-1 as RIPv1 is limited to 15 hops. The gateway is still marked with 172.16.59.6, which is pi-1's router gateway as this is the IP address that is communicated on RIPv1.

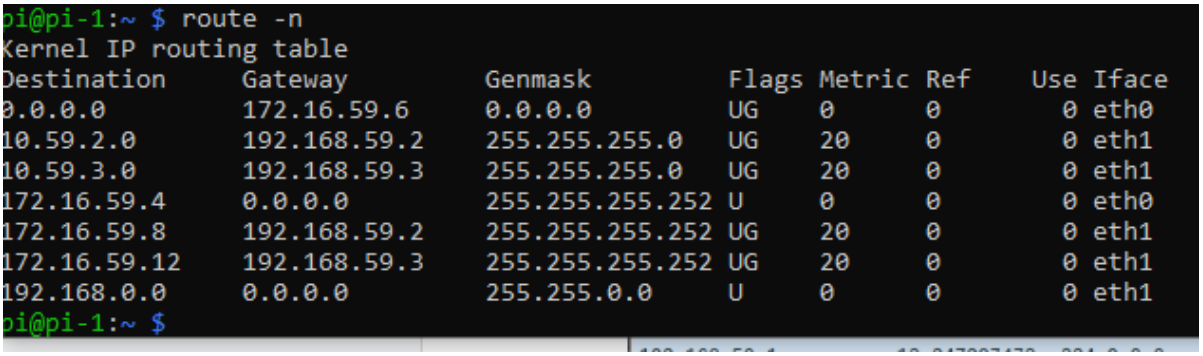
Q3

Pi-1 uses 224.0.0.9 multicast address over eth1. This has added 10.59.2.0 and 10.59.3.0 routes to pi-1. As with Q2, these have metrics 20 signifying unroutable. This is for the same reason as before, that these are private networks for pi-2 and pi-3 and are unreachable from the switch.



192.168.59.1	0.947482536	224.0.0.9	RIPv2	66	Request
192.168.59.1	7.022628088	224.0.0.9	RIPv2	106	Response
192.168.59.1	12.347238492	224.0.0.9	RIPv2	66	Request
192.168.59.1	13.347387473	224.0.0.9	RIPv2	106	Response
192.168.59.3	19.204462748	224.0.0.9	RIPv2	66	Request
192.168.59.1	19.204591303	192.168.59.3	RIPv2	106	Response
192.168.59.3	19.204770748	224.0.0.9	RIPv2	106	Response
192.168.59.1	22.345278264	224.0.0.9	RIPv2	86	Response
192.168.59.3	27.206161632	224.0.0.9	RIPv2	86	Response
192.168.59.1	30.348629315	224.0.0.9	RIPv2	86	Response
192.168.59.2	35.658232071	224.0.0.9	RIPv2	66	Request
192.168.59.1	35.658372664	192.168.59.2	RIPv2	86	Response
192.168.59.2	36.658188274	224.0.0.9	RIPv2	106	Response

```
▸ Frame 371: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
▾ Ethernet II, Src: Private_10:da:ca (80:6d:97:10:da:ca), Dst: IPv4mcast_09 (01:00:5e:00:00:09)
    ▸ Destination: IPv4mcast_09 (01:00:5e:00:00:09)
    ▸ Source: Private_10:da:ca (80:6d:97:10:da:ca)
      Type: IPv4 (0x0800)
▸ Internet Protocol Version 4, Src: 192.168.59.1, Dst: 224.0.0.9
▸ User Datagram Protocol, Src Port: 520, Dst Port: 520
▸ Routing Information Protocol
```



```
pi@pi-1:~ $ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         172.16.59.6     0.0.0.0         UG    0      0        0 eth0
10.59.2.0       192.168.59.2    255.255.255.0   UG    20     0        0 eth1
10.59.3.0       192.168.59.3    255.255.255.0   UG    20     0        0 eth1
172.16.59.4     0.0.0.0         255.255.255.252 U     0      0        0 eth0
172.16.59.8     192.168.59.2    255.255.255.252 UG    20     0        0 eth1
172.16.59.12    192.168.59.3    255.255.255.252 UG    20     0        0 eth1
192.168.0.0     0.0.0.0         255.255.0.0     U     0      0        0 eth1
pi@pi-1:~ $
```

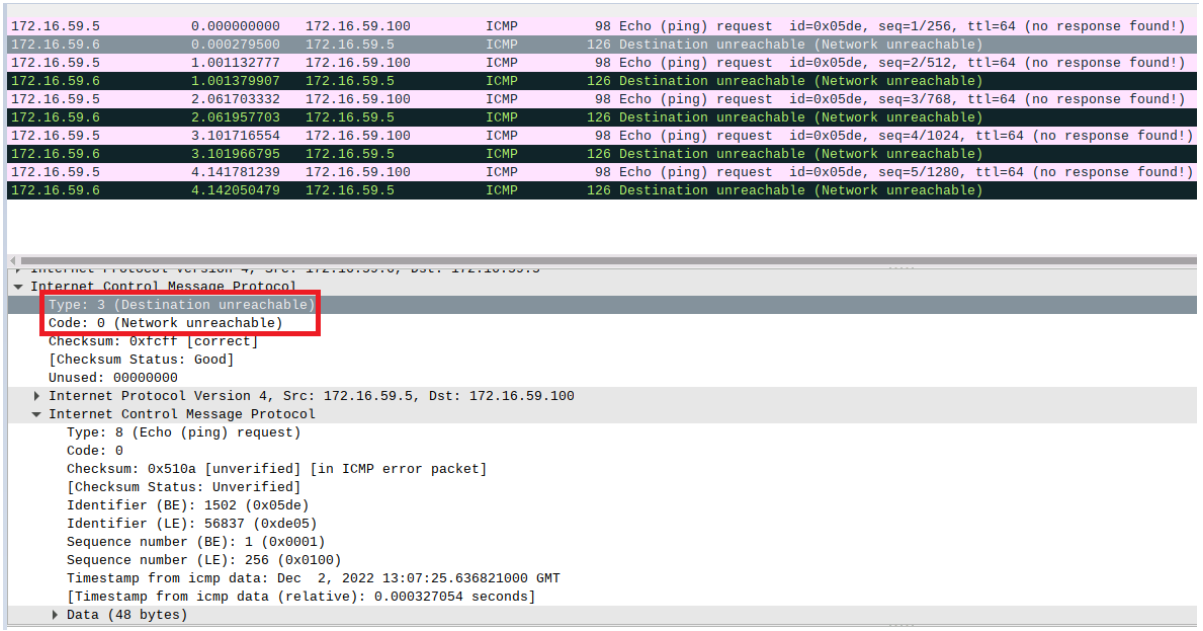
Q4

:(

Lab 9

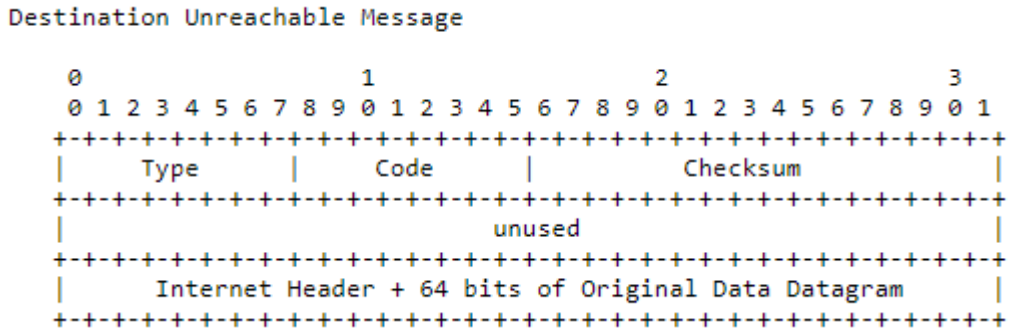
Q1

ICMP Destination/Network Unreachable



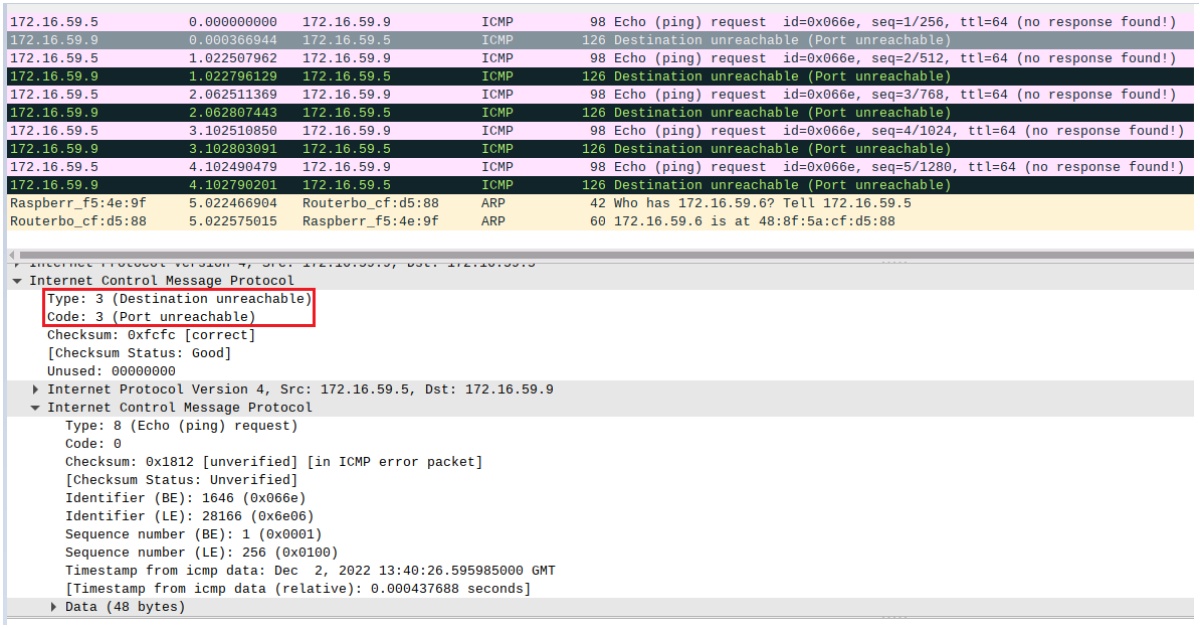
The first section is the ICMP response packet, which has a type of 3 (destination unreachable), and code 0 (network is unreachable).

The second section is the original ICMP ping request, as described in https://www.rfc-editor.org/rfc/rfc792



ICMP Port Unreachable

First section is the ICMP response packet, which has a type of 3 (destination unreachable) and code 0 (port unreachable).



To do this I added an entry to *iptables* to reject all ICMP requests (type 8) with error *icmp-port-unreachable*.

```
pi@pi-2:~ $ sudo iptables -S
-P INPUT ACCEPT
-P FORWARD ACCEPT
-P OUTPUT ACCEPT
-A INPUT -p icmp -m icmp --icmp-type 8 -j REJECT --reject-with icmp-port-unreachable
pi@pi-2:~ $
```

From a security point of view, why is a switch better than a hub?

Switches are more secure than hubs because they operate at the data link layer of the OSI model, while hubs operate at the physical layer. This means that switches are able to inspect the data packets passing through them and make forwarding decisions based on the MAC addresses of the devices on the network, while hubs are simply able to broadcast data to all devices connected to them. This can prevent an attacker intercepting packets meant for another device on the network.

A switch also allows you to segment your network into different virtual subnets. This provides an additional layer of security by isolating traffic between VLANs.

Q2

Data: 0001 0011 0101
=> 1 + 0x + x**2 + 0x**3 + x**4 + x**5 + 0x**6 + 0x**7 + x**8 + 0x**9 + 0x**10 + 0x**11 = x**8 + x**5 + x**4 + x**2 + 1

Key: 1001 => x**3 + 0x**2 + 0x + 1 = x**3 + 1

Message is padded by the largest polynomial in the key (in this case 3):

x**3 (x**8 + x**5 + x**4 + x**2 +1) = x**11 + x**8 + x**7 + x**5 + x**3 = 100110101000

We now perform division of the previous answer with the key until the result is less than the key. Leading zeros are trimmed.

100110101000 / 1001:

—-
100110101000
1001

000010101000

10101000
1001

00111000

111000
1001

011100

—--
11100
1001

01110

—-
1110
1001

0111

= 111

Transmitted value: Message + remainder
Transmitted value = 000100110101111

Q3

If two devices are connected to a local network with the same IP address, they will receive and respond to the same network requests.

For example, when another device sends an ARP requests for that IP, both devices will reply with their own MAC address, which will have unpredictable results based on which response arrives first.

The router CAM and SRAM tables will also constantly be updated, whenever either device sends or responds to a message over the switch, the router will observe the packet and update the IP-MAC mapping in the CAM / SRAM tables accordingly. If both devices are making requests / responses, then the tables will constantly be switching between the two devices MAC addresses.

In a real life scenario, the proper way to resolve this conflict is that both devices disconnect then reconnect to the network, and following reconnection request a new IP address from DHCP if not done so already.

Q4

4B Sequence: 1110 0101 0000

According to the 4B/5B Encoding table:
1110 → 11100
0101 → 01011
0000 → 11110

Which transforms our input into:
111000101111110

Assuming an -V start, this 5B encoded output will result in the bit stream:

	1	1	1	0	0	0	1	0	1	1	1	1	1	1	0
-V	0	+V	0	0	0	0	-V	-V	0	+V	0	-V	0	+V	+V