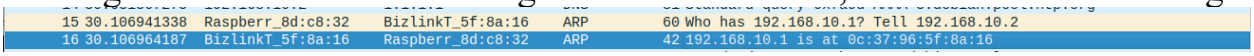
Exercise 1

Traffic capture

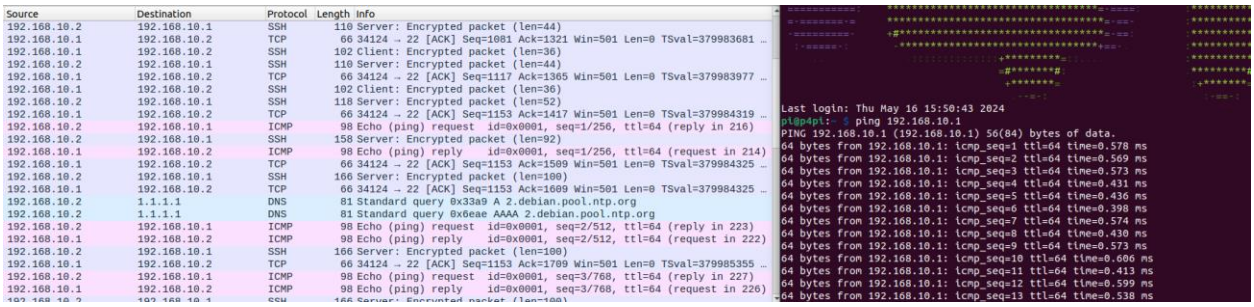
The ethernet connection is associated with 0c:37:96:5f:8a:16 (from the perspective of the Linux machine) under *enx0c37965f8a16*. On the raspberry pi, the connection is labelled with the much nicer *eth0* (if you like Minecraft this is also the name of a popular youtuber).

Starting Wireshark and looking at this interface, we see this message:



which matches the MAC address that we took earlier.

If we use the *ping* command,

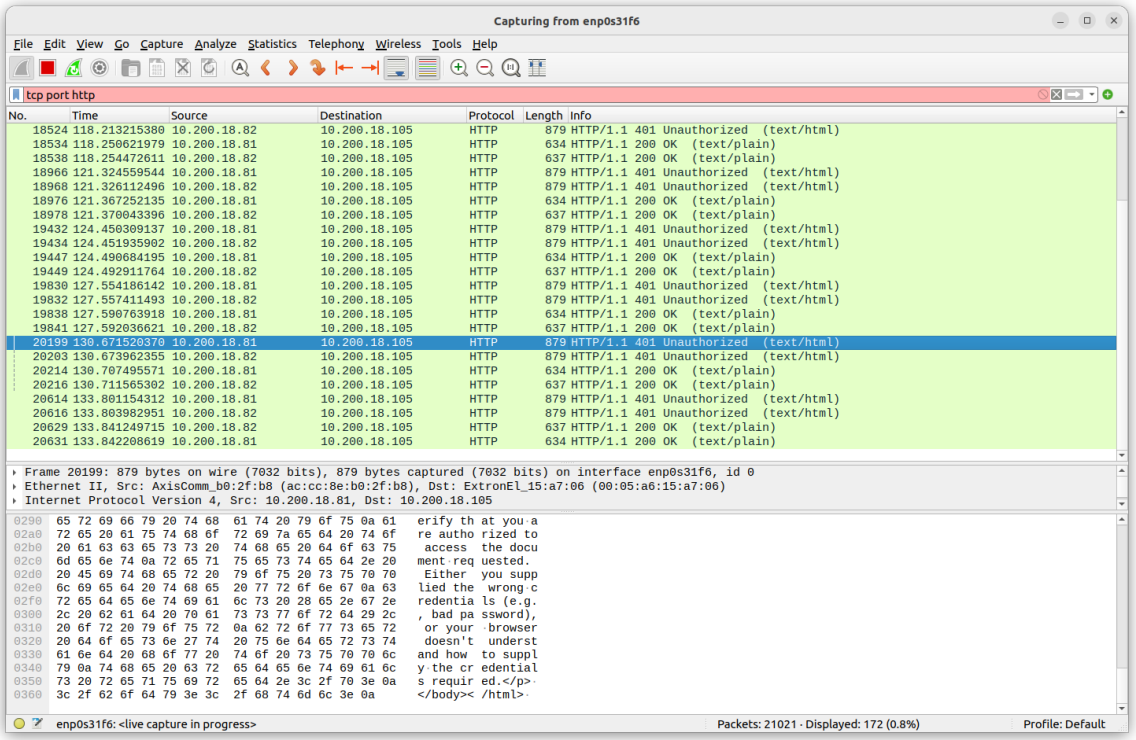


Here we see that the raspberry pi (ip: 192.168.10.2) sends a ping request, 64 bytes long, to the Linux machine (ip:192.168.10.1) which returns a reply to the raspberry pi.

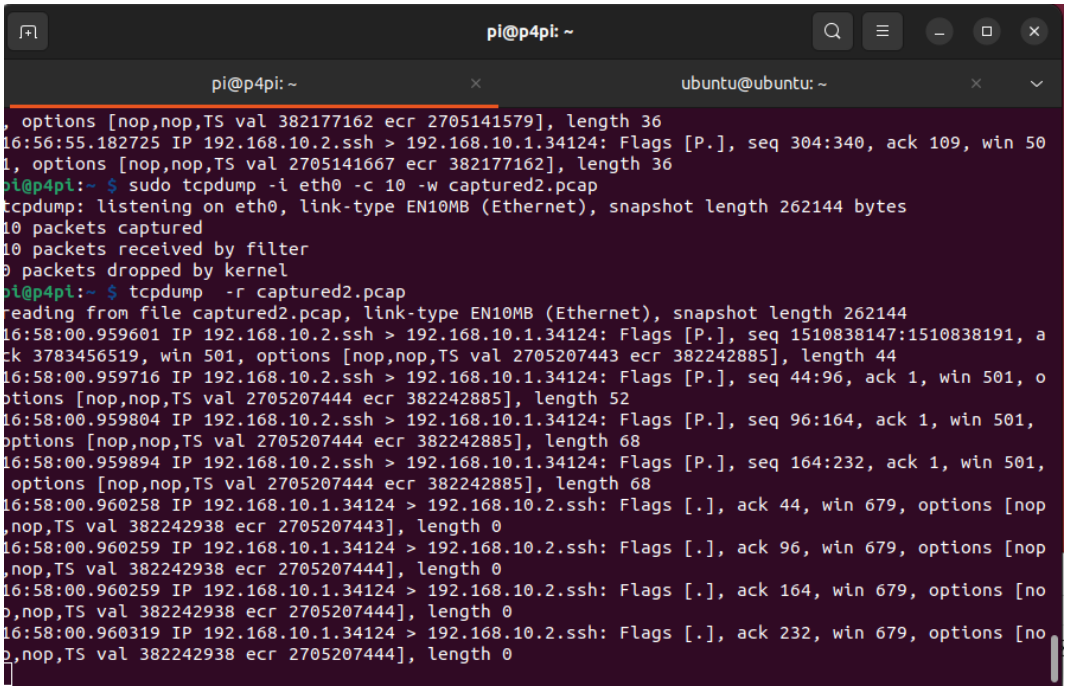
Over 560 pings, the raspberry pi reports a mean time between request and response of 0.516 ms with standard deviation 0.075 ms. The packet loss reported is zero (all 560 requests received a reply), so the ethernet cable seems to be working well.

Switching to the departmental connection (ip: 10.200.17.151/22), the network is much busier. We see many different protocols appearing, such as TCP, ARP, GVCP and STP.

To filter for *http* requests, we first lookup the filter under the *Capture* tab and find that we need the filter expression *tcp port http*. Here is a view of the result:



We can see that the packet's payload contains a message formatted in HTML. I'm not entirely sure where it comes from though.

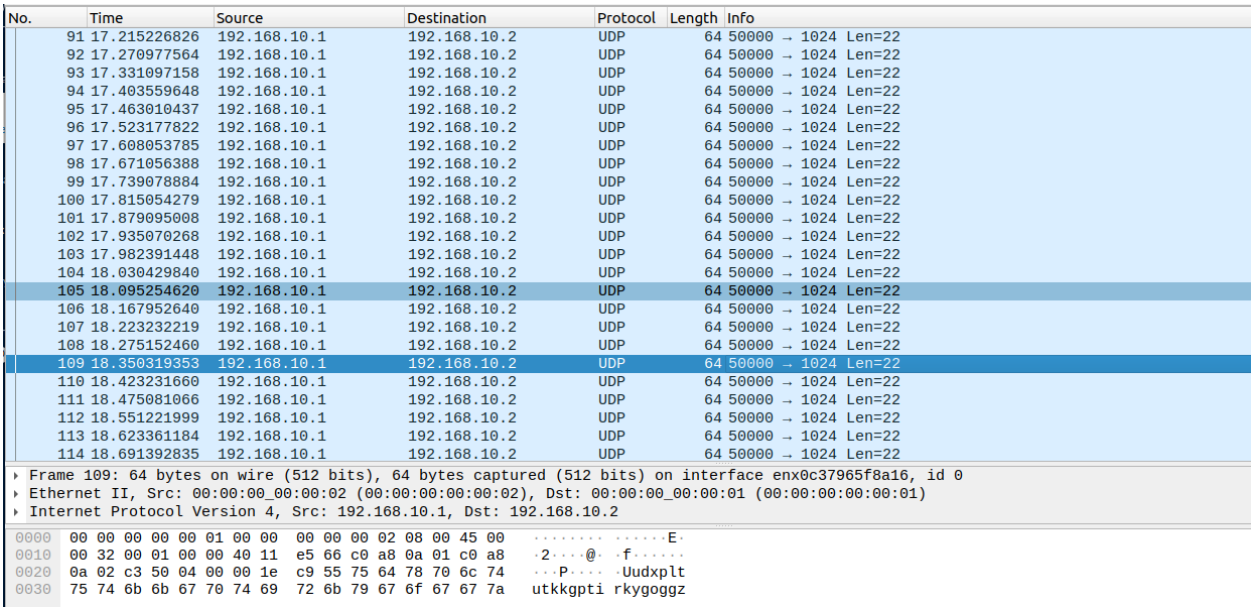


Here we capture 10 packets on the raspberry pi using *tcpdump*. I'm not sure what to say about these, either.

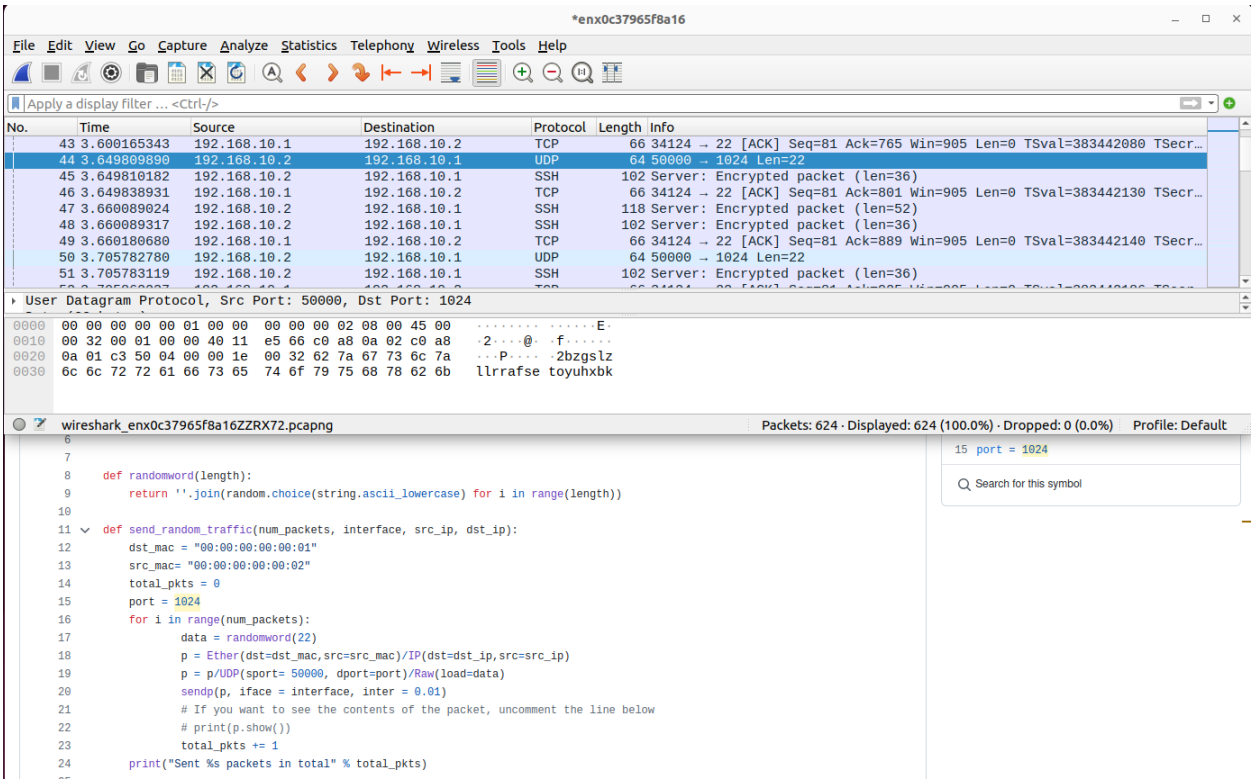
Sending traffic

Using the script provided in the Github repo, we may send 100 packets with the command :

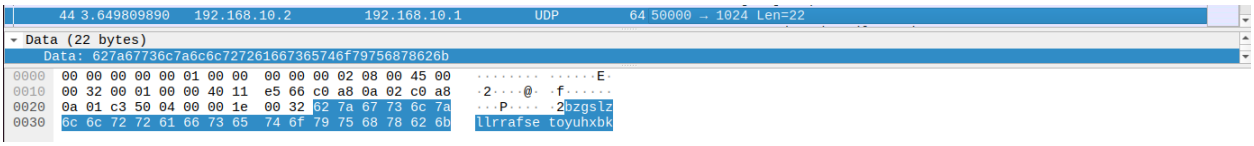
sudo python3 send.py 100 enx0c37965f8a16 192.168.10.1 192.168.10.2



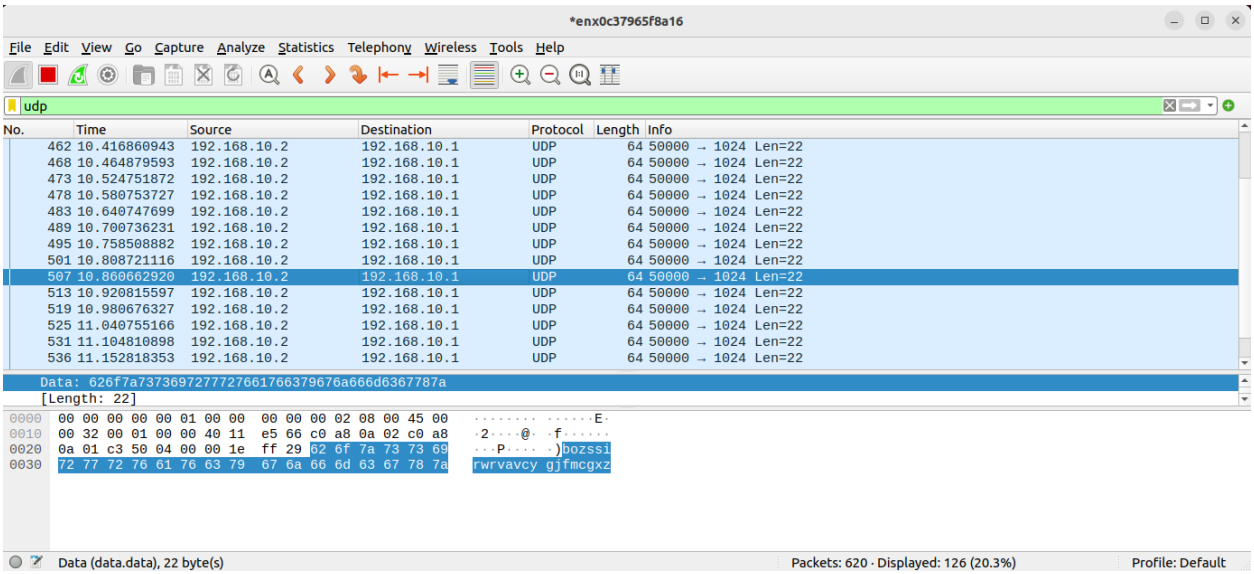
We can repeat the same with our duplicated repo on the raspberry pi (switching the IP addresses and changing the connection name to *eth0*:



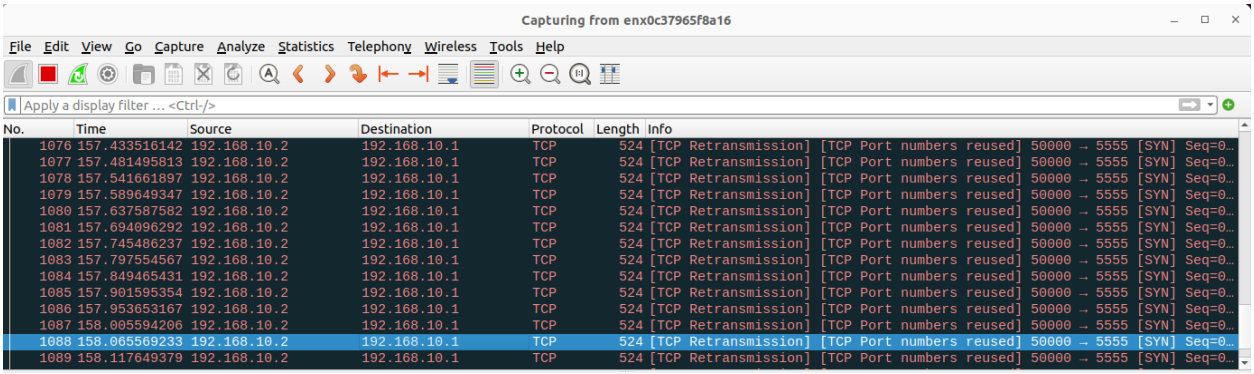
Looking at the code, we see that the protocol used to send the strings generated in python is UDP. Indeed, the number of bytes in the payload of each UDP packet is 22, matching the value in line 17 of the python code:



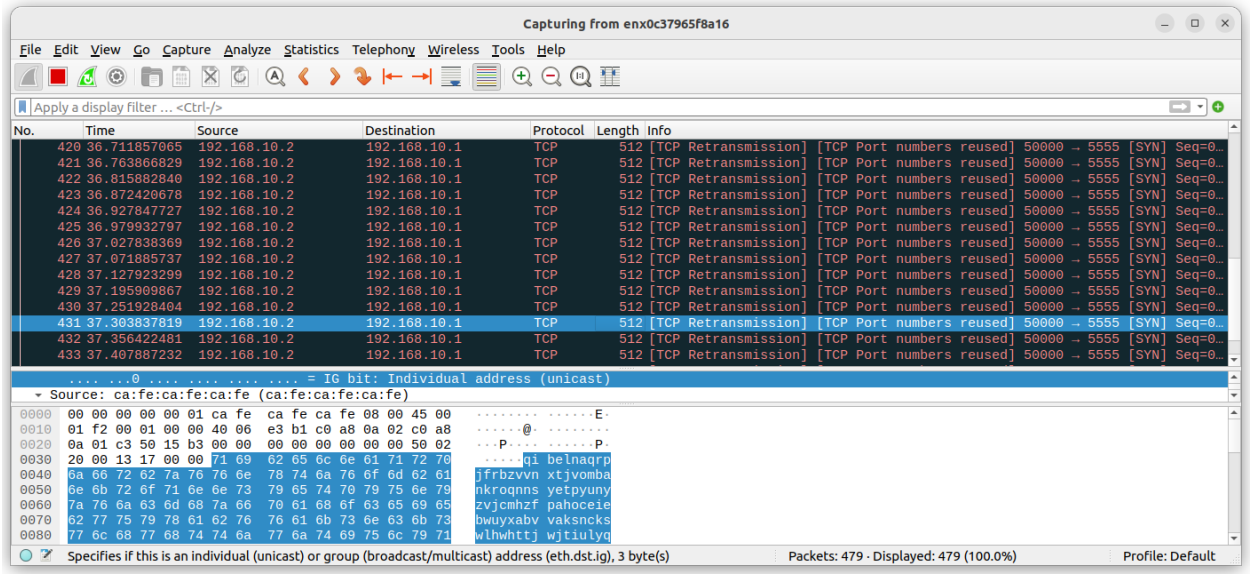
Hence, the relevant filter expression in WireShark is *udp*.



The total packet size is 64 bytes, since there are 64 hexadecimal digit pairs. Then we know that the header takes up 64 – 22 = 42 bytes. So initially I used a string length of 512-42=470 bytes, but I forgot that TCP is a different protocol and hence might have a different header! At a string length of 470 this yielded packets of size 524, so I actually needed a string length of 458.



The erroneous packets



The correct packet size!

Edited python code

send.py
~/CWM-ProgNets/assignment1

```python
#!/usr/bin/python

from scapy.all import Ether, IP, sendp, get_if_hwaddr, get_if_list, TCP, Raw, UDP
import sys
import random, string


def randomword(length):
    return ''.join(random.choice(string.ascii_lowercase) for i in range(length))

def send_random_traffic(num_packets, interface, src_ip, dst_ip):
    dst_mac = "00:00:00:00:00:01"
    src_mac= "CA:FE:CA:FE:CA:FE"
    total_pkts = 0
    port = 5555
    for i in range(num_packets):
            data = randomword(458) #TCP has a header size 54 so we need 512-54=458B
            p = Ether(dst=dst_mac,src=src_mac)/IP(dst=dst_ip,src=src_ip)
            p = p/TCP(sport= 50000, dport=port)/Raw(load=data)
            sendp(p, iface = interface, inter = 0.01)
            # If you want to see the contents of the packet, uncomment the line below
            # print(p.show())
            total_pkts += 1
    print("Sent %s packets in total" % total_pkts)

if __name__ == '__main__':
    if len(sys.argv) < 5:
        print("Usage: python send.py number_of_packets interface_name src_ip_address
dst_ip_address")
        sys.exit(1)
    else:
        num_packets = sys.argv[1]
        interface = sys.argv[2]
        src_ip = sys.argv[3]
        dst_ip = sys.argv[4]
        send_random_traffic(int(num_packets), interface, src_ip, dst_ip)
```

Python 2 · Tab Width: 8 · Ln 2, Col 1 · INS