

## Traffic Capture

### First Test

No packets are captured because there's no pinging between the interfaces (no connection, so no packets sent).

However, if we use P4Pi-9 to ping the ethernet interface (192.168.10.1), we can see multiple packets captured on wireshark:

SSH - Client: Encrypted packet

SSH - Server: Encrypted packet

TCP - 49924 -> 22 [ACK]

ICMP - Echo (ping) request

ICMP - Echo (ping) reply

ARP - Who has 192.168.10.2? Tell 192.168.10.1

ARP - 192.168.10.2 is at e4:5f:01:87:60:14

Due to remote operation using the Lab computer, we can see the SSH and TCP Protocol packets very often, almost like a background constant.

The ICMP and ARP Protocol packets exist due to the pinging that is being performed.

### Second Test

By filtering for HTTP and using Firefox on the Lab computer, we are able to detect HTTP Protocol packets.

Wireshark packet capture showing HTTP traffic. The packet list shows two HTTP GET requests. The packet details pane shows the structure of the first packet, including Ethernet II, Internet Protocol Version 4, Transmission Control Protocol, and Hypertext Transfer Protocol layers. The packet bytes pane shows the raw data in hexadecimal and ASCII.

No.	Time	Source	Destination	Protocol	Length	Info
5506	45.456110137	10.200.17.145	185.125.190.18	HTTP	153	GET / HTTP/1.1
5507	45.460233502	185.125.190.18	10.200.17.145	HTTP	255	HTTP/1.1 204 No Content

Frame 5506: 153 bytes on wire (1224 bits), 153 bytes captured (1224 bits) on interface enp0s31f6, id 0  
Ethernet II, Src: Dell\_02:e1:e0 (50:9a:4c:02:e1:e0), Dst: Cisco\_a2:1a:f1 (00:9a:d2:a2:1a:f1)  
Internet Protocol Version 4, Src: 10.200.17.145, Dst: 185.125.190.18  
Transmission Control Protocol, Src Port: 54504, Dst Port: 80, Seq: 1, Ack: 1, Len: 87  
Hypertext Transfer Protocol

```
0000  00 9a d2 a2 1a f1 50 9a 4c 02 e1 e0 08 00 45 00  ....P...L....E-
0010  00 8b cf 0f 40 00 40 06 d7 74 0a c8 11 91 b9 7d  ....@...t....}
0020  be 12 d4 e8 00 50 03 56 70 24 71 4c 37 c5 80 18  ....P..VpSQL7...
0030  01 f6 94 66 00 00 01 01 08 0a 7a a6 01 af 41 49  ....f....z...AI
0040  0b 0c 47 45 54 20 2f 20 48 54 54 50 2f 31 2e 31  ..lGET / HTTP/1.1
0050  0d 0a 48 6f 73 74 3a 20 63 6f 6e 6e 65 63 74 69  ..Host: connect1
0060  76 69 74 79 2d 63 68 65 63 6b 2e 75 62 75 6e 74  vity-che ck.ubunt
0070  75 2e 63 6f 6d 0d 0a 41 63 63 65 70 74 3a 20 2a  u.com..A ccept: *
0080  2f 2a 0d 0a 43 6f 6e 6e 65 63 74 69 6f 6e 3a 20  /*..Conn ection:
0090  63 6c 6f 73 65 0d 0a 0d 0a                      close...
```

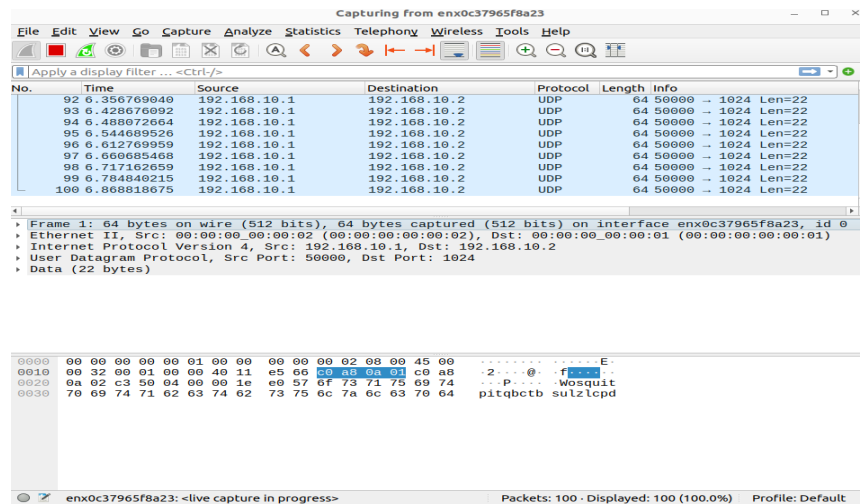
### Third Test

After waiting a short while, the interfaces began communicating, and the tcpdump output shows that below:

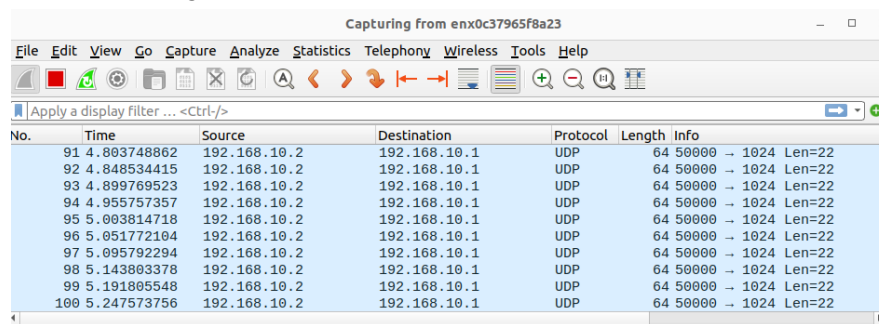
```
pi@p4pi:~$ sudo tcpdump -i eth0 -c 10 -w captured.pcap
tcpdump: listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
10 packets captured
14 packets received by filter
0 packets dropped by kernel
pi@p4pi:~$ tcpdump -r captured.pcap
reading from file captured.pcap, link-type EN10MB (Ethernet), snapshot length 262144
17:50:36.515365 IP 192.168.10.2.ssh > 192.168.10.1.49924: Flags [P.], seq 1606188336:1606188380, ack 2662221343, win 501, options [nop,nop,TS val 3129888895 ecr 188335524], length 44
17:50:36.515494 IP 192.168.10.2.ssh > 192.168.10.1.49924: Flags [P.], seq 44:96, ack 1, win 501, options [nop,nop,TS val 3129888896 ecr 188335524], length 52
17:50:36.515589 IP 192.168.10.2.ssh > 192.168.10.1.49924: Flags [P.], seq 96:164, ack 1, win 501, options [nop,nop,TS val 3129888896 ecr 188335524], length 68
17:50:36.515687 IP 192.168.10.2.ssh > 192.168.10.1.49924: Flags [P.], seq 164:232, ack 1, win 501, options [nop,nop,TS val 3129888896 ecr 188335524], length 68
17:50:36.515811 IP 192.168.10.1.49924 > 192.168.10.2.ssh: Flags [.], ack 44, win 2268, options [nop,nop,TS val 188335567 ecr 3129888895], length 0
17:50:36.515812 IP 192.168.10.1.49924 > 192.168.10.2.ssh: Flags [.], ack 96, win 2268, options [nop,nop,TS val 188335567 ecr 3129888896], length 0
17:50:36.515913 IP 192.168.10.1.49924 > 192.168.10.2.ssh: Flags [.], ack 164, win 2268, options [nop,nop,TS val 188335567 ecr 3129888896], length 0
17:50:36.515913 IP 192.168.10.1.49924 > 192.168.10.2.ssh: Flags [.], ack 232, win 2268, options [nop,nop,TS val 188335567 ecr 3129888896], length 0
17:50:45.956714 IP 192.168.10.1.49924 > 192.168.10.2.ssh: Flags [P.], seq 1:37, ack 232, win 2268, options [nop,nop,TS val 188345008 ecr 3129888896], length 36
17:50:45.956991 IP 192.168.10.2.ssh > 192.168.10.1.49924: Flags [P.], seq 232:268, ack 37, win 501, options [nop,nop,TS val 3129898337 ecr 188345008], length 36
pi@p4pi:~$
```

## Sending Traffic

Sending packets from the Lab machine to P4Pi-9, we see 100 UDP Protocol packets being detected on Wireshark.



The opposite gives the same result: 100 UDP Protocol packets being detected on Wireshark.



If a filter is necessary, it would be the UDP Protocol filter. In this case, there was no need to apply a filter since all the packets detected were UDP Protocol ones. Each packet is 64 bytes with 22 bytes worth dedicated to the payload. Based on what is observed, the protocol used is UDP.

## Script Modification

The following is a screenshot of the modified python script:

```
#!/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 = "0c:37:96:5f:8a:23"
    total_pkts = 0
    port = 1024
    for i in range(num_packets):
        data = randomword(458)
        p = Ether(dst=dst_mac,src=src_mac)/IP(dst=dst_ip,src=src_ip)
        p = p/TCP(sport= 5555, 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)
```

This yields the following result:

The screenshot shows the Wireshark network protocol analyzer interface. The main pane displays a list of captured packets, all of which are TCP retransmissions. The packet list is as follows:

No.	Time	Source	Destination	Protocol	Length	Info
92	6.081359066	192.168.10.1	192.168.10.2	TCP	512	[TCP Retransmission] [TCP
93	6.148716320	192.168.10.1	192.168.10.2	TCP	512	[TCP Retransmission] [TCP
94	6.224812437	192.168.10.1	192.168.10.2	TCP	512	[TCP Retransmission] [TCP
95	6.284902788	192.168.10.1	192.168.10.2	TCP	512	[TCP Retransmission] [TCP
96	6.360973238	192.168.10.1	192.168.10.2	TCP	512	[TCP Retransmission] [TCP
97	6.420912943	192.168.10.1	192.168.10.2	TCP	512	[TCP Retransmission] [TCP
98	6.497849758	192.168.10.1	192.168.10.2	TCP	512	[TCP Retransmission] [TCP
99	6.556970833	192.168.10.1	192.168.10.2	TCP	512	[TCP Retransmission] [TCP
100	6.613096068	192.168.10.1	192.168.10.2	TCP	512	[TCP Retransmission] [TCP

The bottom pane shows the details of the selected packet (Frame 8). The information is as follows:

- Frame 8: 512 bytes on wire (4096 bits), 512 bytes captured (4096 bits) on interface enx0c37965f8a23, id
- Ethernet II, Src: BizlinkT\_5f:8a:23 (0c:37:96:5f:8a:23), Dst: 00:00:00\_00:00:01 (00:00:00:00:00:01)
- Internet Protocol Version 4, Src: 192.168.10.1, Dst: 192.168.10.2
- Transmission Control Protocol, Src Port: 5555, Dst Port: 1024, Seq: 0, Len: 458