Assignment 6
Mateusz Zorga
CWM ProgNet TT22

1. Program description

The program firewall_final.p4 is a basic routing program with firewall functionality.

Features:

- IPv4 and IPv6 forwarding support

Code will forward both types of traffic (as long as no firewall modules below block it)

- Blacklisting addresses

IPv4 and IPv6 addresses can be added to their respective blacklist tables. Any packets from those addresses will be dropped. Since those are LPM tables, ranges of IPs can also be defined.

- Limiting a number of packets accepted through the network

Upon receiving a packet code will start counting the number of packets arriving within a set timeframe. If a maximum number of packets is exceeded all further traffic until the end of the timeframe will be dropped. This prevents flooding of the network, but can be exploited to stop all traffic crossing the switch through DDoS-style attacks (although the network itself is protected and will only receive a fraction of such traffic).

- Specifying behaviour for different IP protocols

For every IPv4/6 packet the code will check its protocol and decide whether to drop or forward it. By default only ICMP, TCP and UDP are defined and everything else is dropped by default, but the code can easily be expanded to allow more protocols.

GitHub link: https://github.com/mat-jakis/CWM-ProgNets/blob/main/assignment6/firewall_final.p4

2. Code description

The code is based on my Exercise 5 submission with additional functionality built on top. To show more information *'–log-console -L info'* can be added to the *simple_switch* command.

i. Definitions and variables

At the beginning of the code, a number of additional variables and definitions are introduced:

Type definitions: Ethernet type values for various types of traffic (only IPv4 and IPv6 used)
Protocol definitions: IP protocol values for various types of traffic (ICMP, TCP and UDP)
Temporal limiting variables: variables and registers used in packet limiting, explained later
Header declarations: declarations for Ethernet, IPv4 and IPv6

ii. Parser

At the beginning of the parser a custom error flag *WrongPacketType* is declared – it will be raised when a packet has a wrong Ethernet type and will prevent it from being processed further.

In *state start* the Ethernet header is extracted and its type checked. If it matches either IPv4 or IPv6 the relevant header will be extracted and parsing will end, otherwise the custom error flag is raised.

iii. Ingress

In *apply* code first checks if any flags were raised during parsing and exit ingress if so.

Further, code will perform either IPv4 or IPv6 processing depending on which header is valid. Processing is almost identical for both, with just different tables, variable names and types.

Processing starts with checking the blacklist. If any of the addresses in the blacklist table match the packet source, a blacklist flag will be raised and packet consequentially dropped.

Afterwards, code checks the protocol of the packet using the protocol table. Afterwards, a packet can either be sent to forwarding or dropped (default).

Further, the code checks the LPM table and tries to find the relevant address. If it succeeds, the packet will be forwarded using code from Exercise 5 (and a modified variant for IPv6).

Finally, the code will compare the current ingress timestamp with the one saved in the *regPrevTime* register. If the current timestamp is less than previous (overflow protection) or time difference is more than the value of *deltaTime*, the current timestamp will be written to the register and the *regPacketCount* register will be reset (start of new timeframe).
Otherwise, code needs to check if there weren't too many packets sent in the current timeframe by comparing the packet count register to *packetLimit.* If the register value is larger, the packet will be dropped.

iv. Deparser

Standard deparser, emits the Ethernet, IPv4 and IPv6 headers (only one of the last two will be valid)

3. Testing

Testing setup: standard PC-RasPi connection using USB-Ethernet cable.

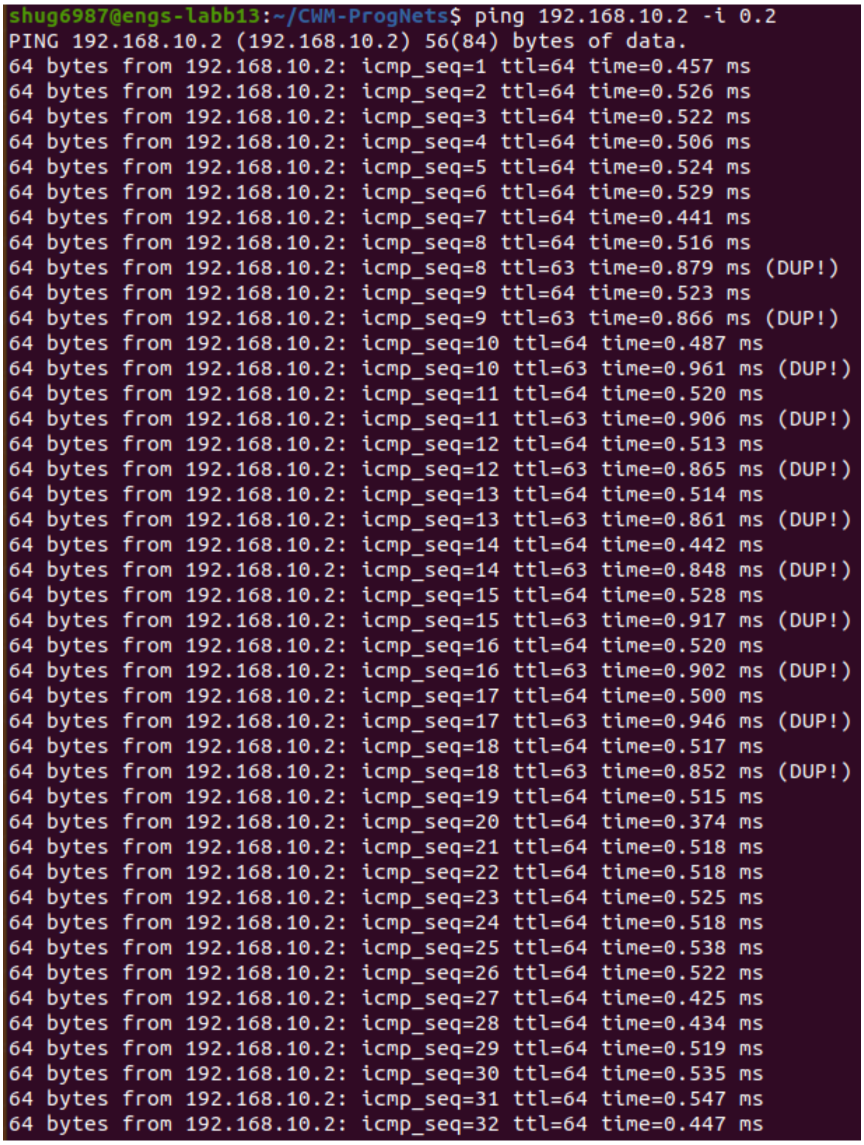
PC: v4:	192.168.10.1
v6:	fe80::6c1b:c07b:e79c:f73e
MAC:	0c:37:96:5f:89:e8
RasPI: v4:	192.168.10.2
v6	fe80::e65f:1ff:fe84:8c86
MAC:	e4:5f:01:84:8c:86

i. Port forwarding

After launching the program and turning on CLI, a command is used to add an entry to the LPM table (forwarding)

```
table_add MyIngress.ipv4_lpm MyIngress.ipv4_forward 192.168.10.1/32 => 0c:37:96:5f:89:e8 0
```

After sending this command, all valid traffic from RasPi to the PC will be duplicated. This can be seen by using a *ping* command:

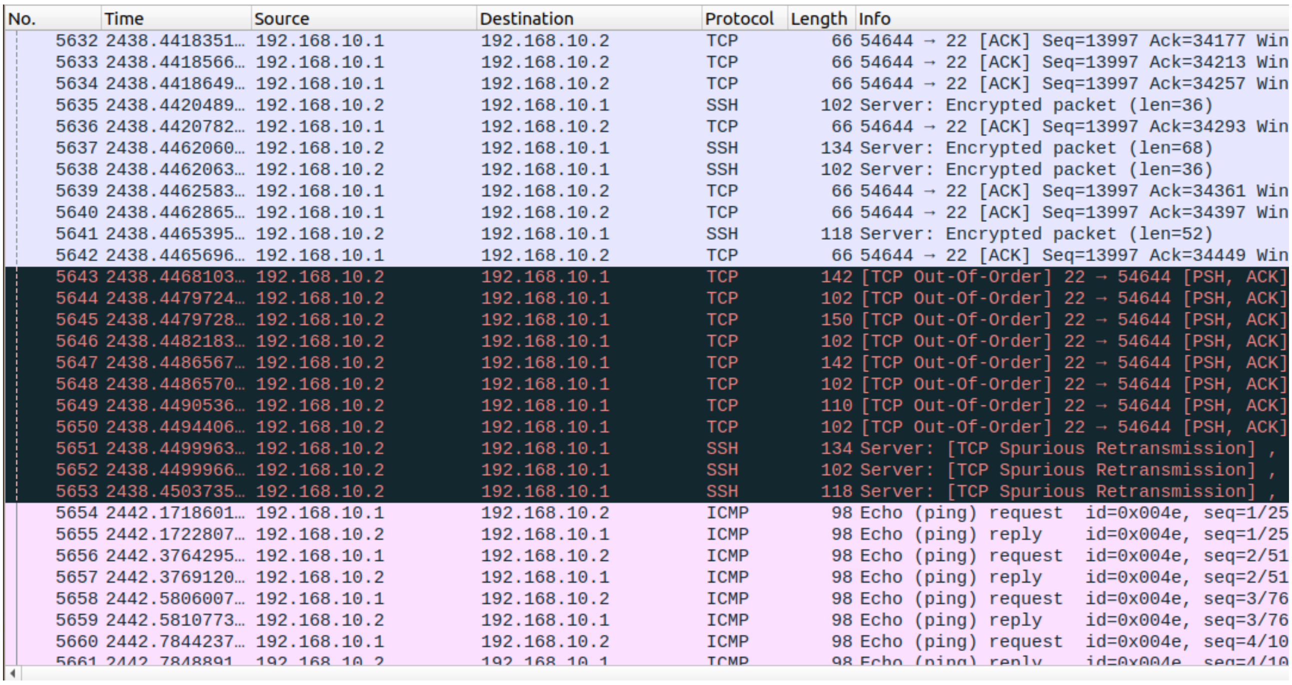


```
shug6987@engs-labb13:~/CWM-ProgNets$ ping 192.168.10.2 -i 0.2
PING 192.168.10.2 (192.168.10.2) 56(84) bytes of data.
64 bytes from 192.168.10.2: icmp_seq=1 ttl=64 time=0.457 ms
64 bytes from 192.168.10.2: icmp_seq=2 ttl=64 time=0.526 ms
64 bytes from 192.168.10.2: icmp_seq=3 ttl=64 time=0.522 ms
64 bytes from 192.168.10.2: icmp_seq=4 ttl=64 time=0.506 ms
64 bytes from 192.168.10.2: icmp_seq=5 ttl=64 time=0.524 ms
64 bytes from 192.168.10.2: icmp_seq=6 ttl=64 time=0.529 ms
64 bytes from 192.168.10.2: icmp_seq=7 ttl=64 time=0.441 ms
64 bytes from 192.168.10.2: icmp_seq=8 ttl=64 time=0.516 ms
64 bytes from 192.168.10.2: icmp_seq=8 ttl=63 time=0.879 ms (DUP!)
64 bytes from 192.168.10.2: icmp_seq=9 ttl=64 time=0.523 ms
64 bytes from 192.168.10.2: icmp_seq=9 ttl=63 time=0.866 ms (DUP!)
64 bytes from 192.168.10.2: icmp_seq=10 ttl=64 time=0.487 ms
64 bytes from 192.168.10.2: icmp_seq=10 ttl=63 time=0.961 ms (DUP!)
64 bytes from 192.168.10.2: icmp_seq=11 ttl=64 time=0.520 ms
64 bytes from 192.168.10.2: icmp_seq=11 ttl=63 time=0.906 ms (DUP!)
64 bytes from 192.168.10.2: icmp_seq=12 ttl=64 time=0.513 ms
64 bytes from 192.168.10.2: icmp_seq=12 ttl=63 time=0.865 ms (DUP!)
64 bytes from 192.168.10.2: icmp_seq=13 ttl=64 time=0.514 ms
64 bytes from 192.168.10.2: icmp_seq=13 ttl=63 time=0.861 ms (DUP!)
64 bytes from 192.168.10.2: icmp_seq=14 ttl=64 time=0.442 ms
64 bytes from 192.168.10.2: icmp_seq=14 ttl=63 time=0.848 ms (DUP!)
64 bytes from 192.168.10.2: icmp_seq=15 ttl=64 time=0.528 ms
64 bytes from 192.168.10.2: icmp_seq=15 ttl=63 time=0.917 ms (DUP!)
64 bytes from 192.168.10.2: icmp_seq=16 ttl=64 time=0.520 ms
64 bytes from 192.168.10.2: icmp_seq=16 ttl=63 time=0.902 ms (DUP!)
64 bytes from 192.168.10.2: icmp_seq=17 ttl=64 time=0.500 ms
64 bytes from 192.168.10.2: icmp_seq=17 ttl=63 time=0.946 ms (DUP!)
64 bytes from 192.168.10.2: icmp_seq=18 ttl=64 time=0.517 ms
64 bytes from 192.168.10.2: icmp_seq=18 ttl=63 time=0.852 ms (DUP!)
64 bytes from 192.168.10.2: icmp_seq=19 ttl=64 time=0.515 ms
64 bytes from 192.168.10.2: icmp_seq=20 ttl=64 time=0.374 ms
64 bytes from 192.168.10.2: icmp_seq=21 ttl=64 time=0.518 ms
64 bytes from 192.168.10.2: icmp_seq=22 ttl=64 time=0.518 ms
64 bytes from 192.168.10.2: icmp_seq=23 ttl=64 time=0.525 ms
64 bytes from 192.168.10.2: icmp_seq=24 ttl=64 time=0.518 ms
64 bytes from 192.168.10.2: icmp_seq=25 ttl=64 time=0.538 ms
64 bytes from 192.168.10.2: icmp_seq=26 ttl=64 time=0.522 ms
64 bytes from 192.168.10.2: icmp_seq=27 ttl=64 time=0.425 ms
64 bytes from 192.168.10.2: icmp_seq=28 ttl=64 time=0.434 ms
64 bytes from 192.168.10.2: icmp_seq=29 ttl=64 time=0.519 ms
64 bytes from 192.168.10.2: icmp_seq=30 ttl=64 time=0.535 ms
64 bytes from 192.168.10.2: icmp_seq=31 ttl=64 time=0.547 ms
64 bytes from 192.168.10.2: icmp_seq=32 ttl=64 time=0.447 ms
```

Some pings are marked with *(DUP!)*, which means they were successfully routed.

ii. Temporal limiting

In the above example only a few packets are marked with *(DUP!)*. This is due to the second firewall feature – temporal limiting. For demonstration purposes the timeframe has been set to 5 seconds and packet limit to 10. The first few packets are not forwarded due to the code forwarding TCP and SSH messages, which can be seen on Wireshark (black messages).

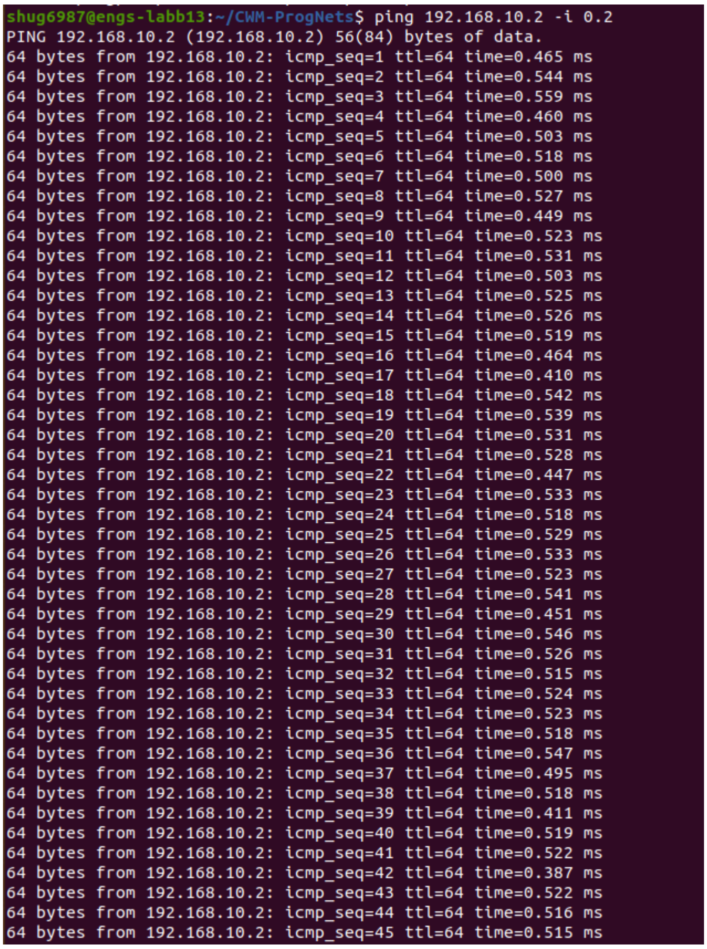


No.	Time	Source	Destination	Protocol	Length	Info
5632	2438.4418351…	192.168.10.1	192.168.10.2	TCP	66	54644 → 22 [ACK] Seq=13997 Ack=34177 Win
5633	2438.4418566…	192.168.10.1	192.168.10.2	TCP	66	54644 → 22 [ACK] Seq=13997 Ack=34213 Win
5634	2438.4418649…	192.168.10.1	192.168.10.2	TCP	66	54644 → 22 [ACK] Seq=13997 Ack=34257 Win
5635	2438.4420489…	192.168.10.2	192.168.10.1	SSH	102	Server: Encrypted packet (len=36)
5636	2438.4420782…	192.168.10.1	192.168.10.2	TCP	66	54644 → 22 [ACK] Seq=13997 Ack=34293 Win
5637	2438.4462060…	192.168.10.2	192.168.10.1	SSH	134	Server: Encrypted packet (len=68)
5638	2438.4462063…	192.168.10.2	192.168.10.1	SSH	102	Server: Encrypted packet (len=36)
5639	2438.4462583…	192.168.10.1	192.168.10.2	TCP	66	54644 → 22 [ACK] Seq=13997 Ack=34361 Win
5640	2438.4462865…	192.168.10.1	192.168.10.2	TCP	66	54644 → 22 [ACK] Seq=13997 Ack=34397 Win
5641	2438.4465395…	192.168.10.2	192.168.10.1	SSH	118	Server: Encrypted packet (len=52)
5642	2438.4465696…	192.168.10.1	192.168.10.2	TCP	66	54644 → 22 [ACK] Seq=13997 Ack=34449 Win
5643	2438.4468103…	192.168.10.2	192.168.10.1	TCP	142	[TCP Out-Of-Order] 22 → 54644 [PSH, ACK]
5644	2438.4479724…	192.168.10.2	192.168.10.1	TCP	102	[TCP Out-Of-Order] 22 → 54644 [PSH, ACK]
5645	2438.4479728…	192.168.10.2	192.168.10.1	TCP	150	[TCP Out-Of-Order] 22 → 54644 [PSH, ACK]
5646	2438.4482183…	192.168.10.2	192.168.10.1	TCP	102	[TCP Out-Of-Order] 22 → 54644 [PSH, ACK]
5647	2438.4486567…	192.168.10.2	192.168.10.1	TCP	142	[TCP Out-Of-Order] 22 → 54644 [PSH, ACK]
5648	2438.4486570…	192.168.10.2	192.168.10.1	TCP	102	[TCP Out-Of-Order] 22 → 54644 [PSH, ACK]
5649	2438.4490536…	192.168.10.2	192.168.10.1	TCP	110	[TCP Out-Of-Order] 22 → 54644 [PSH, ACK]
5650	2438.4494406…	192.168.10.2	192.168.10.1	TCP	102	[TCP Out-Of-Order] 22 → 54644 [PSH, ACK]
5651	2438.4499963…	192.168.10.2	192.168.10.1	SSH	134	Server: [TCP Spurious Retransmission] ,
5652	2438.4499966…	192.168.10.2	192.168.10.1	SSH	102	Server: [TCP Spurious Retransmission] ,
5653	2438.4503735…	192.168.10.2	192.168.10.1	SSH	118	Server: [TCP Spurious Retransmission] ,
5654	2442.1718601…	192.168.10.1	192.168.10.2	ICMP	98	Echo (ping) request id=0x004e, seq=1/25
5655	2442.1722807…	192.168.10.2	192.168.10.1	ICMP	98	Echo (ping) reply id=0x004e, seq=1/25
5656	2442.3764295…	192.168.10.1	192.168.10.2	ICMP	98	Echo (ping) request id=0x004e, seq=2/51
5657	2442.3769120…	192.168.10.2	192.168.10.1	ICMP	98	Echo (ping) reply id=0x004e, seq=2/51
5658	2442.5806007…	192.168.10.1	192.168.10.2	ICMP	98	Echo (ping) request id=0x004e, seq=3/76
5659	2442.5810773…	192.168.10.2	192.168.10.1	ICMP	98	Echo (ping) reply id=0x004e, seq=3/76
5660	2442.7844237…	192.168.10.1	192.168.10.2	ICMP	98	Echo (ping) request id=0x004e, seq=4/10
5661	2442.7848891…	192.168.10.2	192.168.10.1	ICMP	98	Echo (ping) reply id=0x004e, seq=4/10

iii. Blacklisting

Now, adding an entry with the address of RasPi to the blacklist table will stop all forwarding, since the code will no longer forward any packets from this address.

table_add MyIngress.ipv4_blacklist MyIngress.ipv4_block 192.168.10.2/32 =>



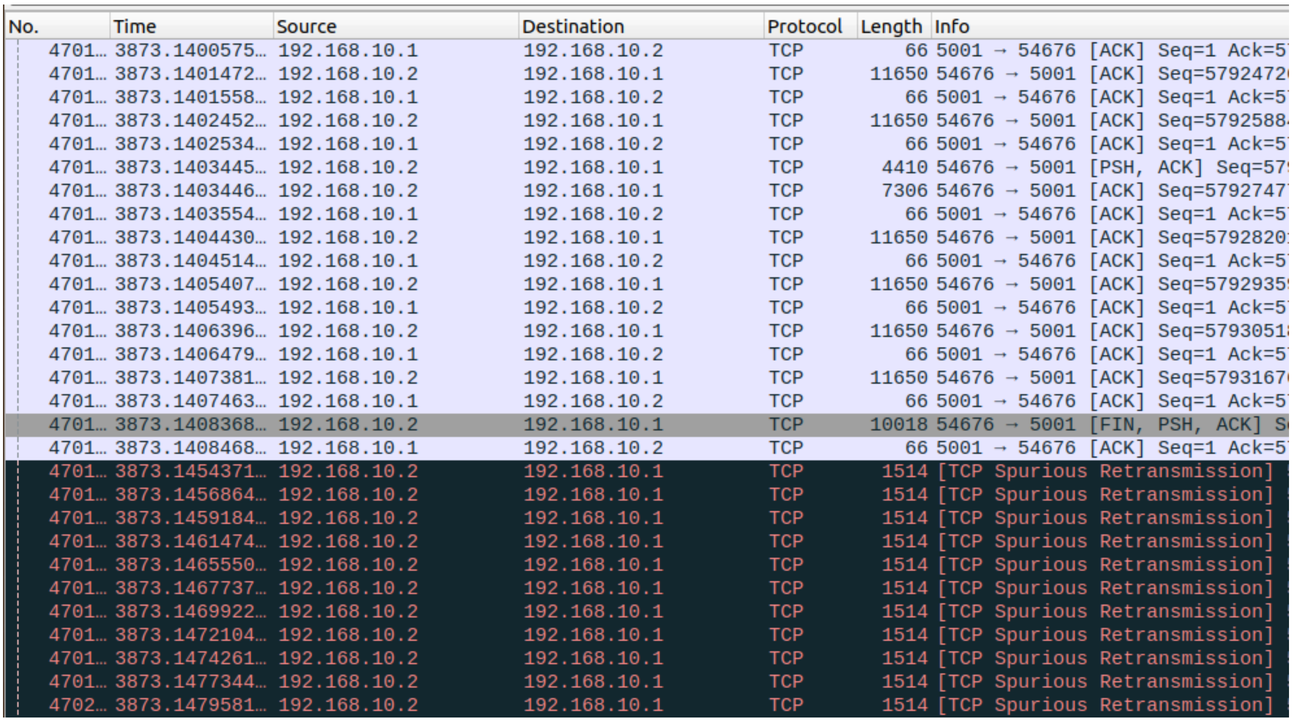
```
shug6987@engs-labb13:~/CWM-ProgNets$ ping 192.168.10.2 -i 0.2
PING 192.168.10.2 (192.168.10.2) 56(84) bytes of data.
64 bytes from 192.168.10.2: icmp_seq=1 ttl=64 time=0.465 ms
64 bytes from 192.168.10.2: icmp_seq=2 ttl=64 time=0.544 ms
64 bytes from 192.168.10.2: icmp_seq=3 ttl=64 time=0.559 ms
64 bytes from 192.168.10.2: icmp_seq=4 ttl=64 time=0.460 ms
64 bytes from 192.168.10.2: icmp_seq=5 ttl=64 time=0.503 ms
64 bytes from 192.168.10.2: icmp_seq=6 ttl=64 time=0.518 ms
64 bytes from 192.168.10.2: icmp_seq=7 ttl=64 time=0.500 ms
64 bytes from 192.168.10.2: icmp_seq=8 ttl=64 time=0.527 ms
64 bytes from 192.168.10.2: icmp_seq=9 ttl=64 time=0.449 ms
64 bytes from 192.168.10.2: icmp_seq=10 ttl=64 time=0.523 ms
64 bytes from 192.168.10.2: icmp_seq=11 ttl=64 time=0.531 ms
64 bytes from 192.168.10.2: icmp_seq=12 ttl=64 time=0.503 ms
64 bytes from 192.168.10.2: icmp_seq=13 ttl=64 time=0.525 ms
64 bytes from 192.168.10.2: icmp_seq=14 ttl=64 time=0.526 ms
64 bytes from 192.168.10.2: icmp_seq=15 ttl=64 time=0.519 ms
64 bytes from 192.168.10.2: icmp_seq=16 ttl=64 time=0.464 ms
64 bytes from 192.168.10.2: icmp_seq=17 ttl=64 time=0.410 ms
64 bytes from 192.168.10.2: icmp_seq=18 ttl=64 time=0.542 ms
64 bytes from 192.168.10.2: icmp_seq=19 ttl=64 time=0.539 ms
64 bytes from 192.168.10.2: icmp_seq=20 ttl=64 time=0.531 ms
64 bytes from 192.168.10.2: icmp_seq=21 ttl=64 time=0.528 ms
64 bytes from 192.168.10.2: icmp_seq=22 ttl=64 time=0.447 ms
64 bytes from 192.168.10.2: icmp_seq=23 ttl=64 time=0.533 ms
64 bytes from 192.168.10.2: icmp_seq=24 ttl=64 time=0.518 ms
64 bytes from 192.168.10.2: icmp_seq=25 ttl=64 time=0.529 ms
64 bytes from 192.168.10.2: icmp_seq=26 ttl=64 time=0.533 ms
64 bytes from 192.168.10.2: icmp_seq=27 ttl=64 time=0.523 ms
64 bytes from 192.168.10.2: icmp_seq=28 ttl=64 time=0.541 ms
64 bytes from 192.168.10.2: icmp_seq=29 ttl=64 time=0.451 ms
64 bytes from 192.168.10.2: icmp_seq=30 ttl=64 time=0.546 ms
64 bytes from 192.168.10.2: icmp_seq=31 ttl=64 time=0.526 ms
64 bytes from 192.168.10.2: icmp_seq=32 ttl=64 time=0.515 ms
64 bytes from 192.168.10.2: icmp_seq=33 ttl=64 time=0.524 ms
64 bytes from 192.168.10.2: icmp_seq=34 ttl=64 time=0.523 ms
64 bytes from 192.168.10.2: icmp_seq=35 ttl=64 time=0.518 ms
64 bytes from 192.168.10.2: icmp_seq=36 ttl=64 time=0.547 ms
64 bytes from 192.168.10.2: icmp_seq=37 ttl=64 time=0.495 ms
64 bytes from 192.168.10.2: icmp_seq=38 ttl=64 time=0.518 ms
64 bytes from 192.168.10.2: icmp_seq=39 ttl=64 time=0.411 ms
64 bytes from 192.168.10.2: icmp_seq=40 ttl=64 time=0.519 ms
64 bytes from 192.168.10.2: icmp_seq=41 ttl=64 time=0.522 ms
64 bytes from 192.168.10.2: icmp_seq=42 ttl=64 time=0.387 ms
64 bytes from 192.168.10.2: icmp_seq=43 ttl=64 time=0.522 ms
64 bytes from 192.168.10.2: icmp_seq=44 ttl=64 time=0.516 ms
64 bytes from 192.168.10.2: icmp_seq=45 ttl=64 time=0.515 ms
```

iv. Different protocol behaviour

By default, the program will only forward ICMP (v4, v6) and TCP (v4). UDP is not forwarded, which can be seen on wireshark for example by using iperf.

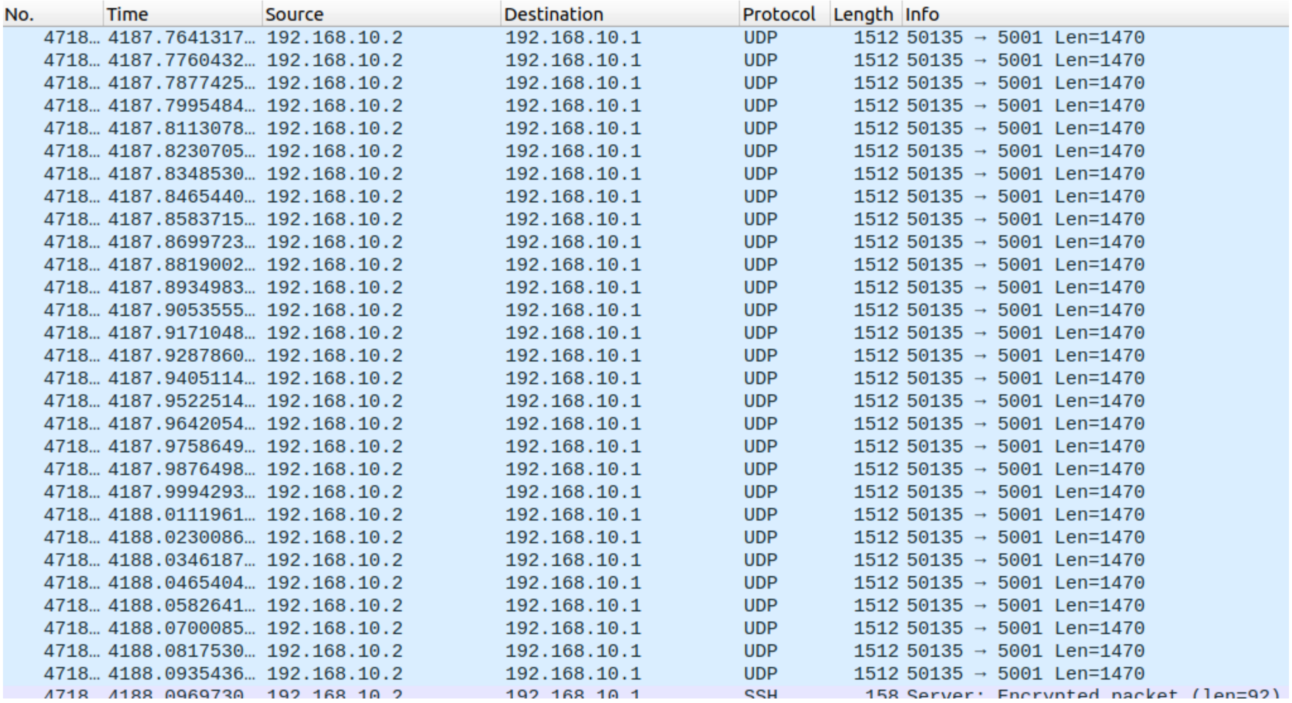
Setting up an iperf server on the PC and performing a TCP test will give duplicate packets, while UDP is not affected.

TCP:



No.	Time	Source	Destination	Protocol	Length	Info
4701…	3873.1400575…	192.168.10.1	192.168.10.2	TCP	66	5001 → 54676 [ACK] Seq=1 Ack=57
4701…	3873.1401472…	192.168.10.2	192.168.10.1	TCP	11650	54676 → 5001 [ACK] Seq=5792472
4701…	3873.1401558…	192.168.10.1	192.168.10.2	TCP	66	5001 → 54676 [ACK] Seq=1 Ack=57
4701…	3873.1402452…	192.168.10.2	192.168.10.1	TCP	11650	54676 → 5001 [ACK] Seq=5792588
4701…	3873.1402534…	192.168.10.1	192.168.10.2	TCP	66	5001 → 54676 [ACK] Seq=1 Ack=57
4701…	3873.1403445…	192.168.10.2	192.168.10.1	TCP	4410	54676 → 5001 [PSH, ACK] Seq=57
4701…	3873.1403446…	192.168.10.2	192.168.10.1	TCP	7306	54676 → 5001 [ACK] Seq=5792747
4701…	3873.1403554…	192.168.10.1	192.168.10.2	TCP	66	5001 → 54676 [ACK] Seq=1 Ack=57
4701…	3873.1404430…	192.168.10.2	192.168.10.1	TCP	11650	54676 → 5001 [ACK] Seq=5792820
4701…	3873.1404514…	192.168.10.1	192.168.10.2	TCP	66	5001 → 54676 [ACK] Seq=1 Ack=57
4701…	3873.1405407…	192.168.10.2	192.168.10.1	TCP	11650	54676 → 5001 [ACK] Seq=5792935
4701…	3873.1405493…	192.168.10.1	192.168.10.2	TCP	66	5001 → 54676 [ACK] Seq=1 Ack=57
4701…	3873.1406396…	192.168.10.2	192.168.10.1	TCP	11650	54676 → 5001 [ACK] Seq=5793051
4701…	3873.1406479…	192.168.10.1	192.168.10.2	TCP	66	5001 → 54676 [ACK] Seq=1 Ack=57
4701…	3873.1407381…	192.168.10.2	192.168.10.1	TCP	11650	54676 → 5001 [ACK] Seq=5793167
4701…	3873.1407463…	192.168.10.1	192.168.10.2	TCP	66	5001 → 54676 [ACK] Seq=1 Ack=57
4701…	3873.1408368…	192.168.10.2	192.168.10.1	TCP	10018	54676 → 5001 [FIN, PSH, ACK] S
4701…	3873.1408468…	192.168.10.1	192.168.10.2	TCP	66	5001 → 54676 [ACK] Seq=1 Ack=57
4701…	3873.1454371…	192.168.10.2	192.168.10.1	TCP	1514	[TCP Spurious Retransmission]
4701…	3873.1456864…	192.168.10.2	192.168.10.1	TCP	1514	[TCP Spurious Retransmission]
4701…	3873.1459184…	192.168.10.2	192.168.10.1	TCP	1514	[TCP Spurious Retransmission]
4701…	3873.1461474…	192.168.10.2	192.168.10.1	TCP	1514	[TCP Spurious Retransmission]
4701…	3873.1465550…	192.168.10.2	192.168.10.1	TCP	1514	[TCP Spurious Retransmission]
4701…	3873.1467737…	192.168.10.2	192.168.10.1	TCP	1514	[TCP Spurious Retransmission]
4701…	3873.1469922…	192.168.10.2	192.168.10.1	TCP	1514	[TCP Spurious Retransmission]
4701…	3873.1472104…	192.168.10.2	192.168.10.1	TCP	1514	[TCP Spurious Retransmission]
4701…	3873.1474261…	192.168.10.2	192.168.10.1	TCP	1514	[TCP Spurious Retransmission]
4701…	3873.1477344…	192.168.10.2	192.168.10.1	TCP	1514	[TCP Spurious Retransmission]
4702…	3873.1479581…	192.168.10.2	192.168.10.1	TCP	1514	[TCP Spurious Retransmission]

UDP:



No.	Time	Source	Destination	Protocol	Length	Info
4718…	4187.7641317…	192.168.10.2	192.168.10.1	UDP	1512	50135 → 5001 Len=1470
4718…	4187.7760432…	192.168.10.2	192.168.10.1	UDP	1512	50135 → 5001 Len=1470
4718…	4187.7877425…	192.168.10.2	192.168.10.1	UDP	1512	50135 → 5001 Len=1470
4718…	4187.7995484…	192.168.10.2	192.168.10.1	UDP	1512	50135 → 5001 Len=1470
4718…	4187.8113078…	192.168.10.2	192.168.10.1	UDP	1512	50135 → 5001 Len=1470
4718…	4187.8230705…	192.168.10.2	192.168.10.1	UDP	1512	50135 → 5001 Len=1470
4718…	4187.8348530…	192.168.10.2	192.168.10.1	UDP	1512	50135 → 5001 Len=1470
4718…	4187.8465440…	192.168.10.2	192.168.10.1	UDP	1512	50135 → 5001 Len=1470
4718…	4187.8583715…	192.168.10.2	192.168.10.1	UDP	1512	50135 → 5001 Len=1470
4718…	4187.8699723…	192.168.10.2	192.168.10.1	UDP	1512	50135 → 5001 Len=1470
4718…	4187.8819002…	192.168.10.2	192.168.10.1	UDP	1512	50135 → 5001 Len=1470
4718…	4187.8934983…	192.168.10.2	192.168.10.1	UDP	1512	50135 → 5001 Len=1470
4718…	4187.9053555…	192.168.10.2	192.168.10.1	UDP	1512	50135 → 5001 Len=1470
4718…	4187.9171048…	192.168.10.2	192.168.10.1	UDP	1512	50135 → 5001 Len=1470
4718…	4187.9287860…	192.168.10.2	192.168.10.1	UDP	1512	50135 → 5001 Len=1470
4718…	4187.9405114…	192.168.10.2	192.168.10.1	UDP	1512	50135 → 5001 Len=1470
4718…	4187.9522514…	192.168.10.2	192.168.10.1	UDP	1512	50135 → 5001 Len=1470
4718…	4187.9642054…	192.168.10.2	192.168.10.1	UDP	1512	50135 → 5001 Len=1470
4718…	4187.9758649…	192.168.10.2	192.168.10.1	UDP	1512	50135 → 5001 Len=1470
4718…	4187.9876498…	192.168.10.2	192.168.10.1	UDP	1512	50135 → 5001 Len=1470
4718…	4187.9994293…	192.168.10.2	192.168.10.1	UDP	1512	50135 → 5001 Len=1470
4718…	4188.0111961…	192.168.10.2	192.168.10.1	UDP	1512	50135 → 5001 Len=1470
4718…	4188.0230086…	192.168.10.2	192.168.10.1	UDP	1512	50135 → 5001 Len=1470
4718…	4188.0346187…	192.168.10.2	192.168.10.1	UDP	1512	50135 → 5001 Len=1470
4718…	4188.0465404…	192.168.10.2	192.168.10.1	UDP	1512	50135 → 5001 Len=1470
4718…	4188.0582641…	192.168.10.2	192.168.10.1	UDP	1512	50135 → 5001 Len=1470
4718…	4188.0700085…	192.168.10.2	192.168.10.1	UDP	1512	50135 → 5001 Len=1470
4718…	4188.0817530…	192.168.10.2	192.168.10.1	UDP	1512	50135 → 5001 Len=1470
4718…	4188.0935436…	192.168.10.2	192.168.10.1	UDP	1512	50135 → 5001 Len=1470
4718…	4188.0969730…	192.168.10.2	192.168.10.1	SSH	158	Server: Encrypted packet (len=92)