

POLITECNICO DI TORINO

01NWDBH - Mobile and Sensor Networks

Lab 03

Linux network programming



Group ID:
05

Students:

Valerio Collina	333919
Ammar Hussein	329829
Md Ismail Hossain	342704
Md. Ataur Rabby	347363

Academic year 2025/2026

Basic communication with UDP sockets

```
ubuntu@ubuntu:~$ ifconfig
eno1: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
    ether b0:22:7a:f2:bc:b1 txqueuelen 1000 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    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 3772 bytes 336444 (336.4 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 3772 bytes 336444 (336.4 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

wlo1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.1.134 netmask 255.255.255.0 broadcast 192.168.1.255
    inet6 fe80::339d:5f82:c805:1b71 prefixlen 64 scopeid 0x20<link>
    ether d4:1b:81:bb:48:bd txqueuelen 1000 (Ethernet)
    RX packets 75896 bytes 96664611 (96.6 MB)
    RX errors 0 dropped 813 overruns 0 frame 0
    TX packets 12645 bytes 5102711 (5.1 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Figure 1: IP of Device 1 (client) from ifconfig

```
ismail@ismail-TP410URR:~$ ifconfig
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 1373 bytes 115272 (115.2 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 1373 bytes 115272 (115.2 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

wlp2s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.1.99 netmask 255.255.255.0 broadcast 192.168.1.255
    inet6 fe80::bb83:eb1c:d36e:faec prefixlen 64 scopeid 0x20<link>
    ether 74:e5:f9:54:e9:85 txqueuelen 1000 (Ethernet)
    RX packets 8747 bytes 8716216 (8.7 MB)
    RX errors 0 dropped 3 overruns 0 frame 0
    TX packets 3848 bytes 1412968 (1.4 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Figure 2: IP of Device 2 (server) from ifconfig

Figure 1 shows that Device 1 (client) has IP 192.168.1.134, while Figure 2 shows Device 2 (server) with IP 192.168.1.99. Both addresses were obtained using ifconfig.

```
1 import socket
2 import time
3
4 UDP_IP = "192.168.1.99"
5 UDP_PORT = 47999
6 MESSAGE_STRING = "UDP_experiment"
7
8 print("UDP server IP: %s" % UDP_IP)
9 print("UDP destination port:", UDP_PORT)
10
11 count = 0
12 sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
13
14 while count < 30:
15     count += 1
16     message = MESSAGE_STRING + "," + str(count) + "$"
17     print("sending message:%s" % message)
18     sock.sendto(message.encode(), (UDP_IP, UDP_PORT))
19     time.sleep(0.2)
20
```

Listing 1: UDP client script sending 30 packets

```
1 import socket
2
3 UDP_IP = "192.168.1.99"
4 UDP_PORT = 47999
5
6 sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
7 sock.bind((UDP_IP, UDP_PORT))
8
9 count = 0
10 while True:
11     data, addr = sock.recvfrom(512)
12     count += 1
13     fields = data.decode().split(",")
14     print("received message: %s, %s, %d" % (fields[0], fields[1], count))
15
```

Listing 2: UDP server script receiving packets

Figure 3: Comparison of UDP client and server Python scripts

The **client** sets the server's IP and port, creates a UDP socket, and sends 30 messages using `sendto()` with a 200 ms delay between packets. Each message contains a fixed string and an incrementing counter.

The **server** binds a UDP socket to its local IP and port, then listens with `recvfrom()`. It decodes each message and prints the content along with a local counter.

```

ismail@ismail-TP410URR:~$ cd Lab_03
ismail@ismail-TP410URR:~/Lab_03$ L3_UDP_Server.py
L3_UDP_Server.py: command not found
ismail@ismail-TP410URR:~/Lab_03$ python3 L3_UDP_Server.py
Server is listening on IP 192.168.1.99, port 47999
received message: UDP_experiment, 1$, 1
received message: UDP_experiment, 2$, 2
received message: UDP_experiment, 3$, 3
received message: UDP_experiment, 4$, 4
received message: UDP_experiment, 5$, 5
received message: UDP_experiment, 6$, 6
received message: UDP_experiment, 7$, 7
received message: UDP_experiment, 8$, 8
received message: UDP_experiment, 9$, 9
received message: UDP_experiment, 10$, 10
received message: UDP_experiment, 11$, 11
received message: UDP_experiment, 12$, 12
received message: UDP_experiment, 13$, 13
received message: UDP_experiment, 14$, 14
received message: UDP_experiment, 15$, 15
received message: UDP_experiment, 16$, 16
received message: UDP_experiment, 17$, 17
received message: UDP_experiment, 18$, 18
received message: UDP_experiment, 19$, 19
received message: UDP_experiment, 20$, 20
received message: UDP_experiment, 21$, 21
received message: UDP_experiment, 22$, 22
received message: UDP_experiment, 23$, 23
received message: UDP_experiment, 24$, 24
received message: UDP_experiment, 25$, 25
received message: UDP_experiment, 26$, 26
received message: UDP_experiment, 27$, 27
received message: UDP_experiment, 28$, 28
received message: UDP_experiment, 29$, 29
received message: UDP_experiment, 30$, 30

```

Figure 4: Server receiving UDP packets on port 47999

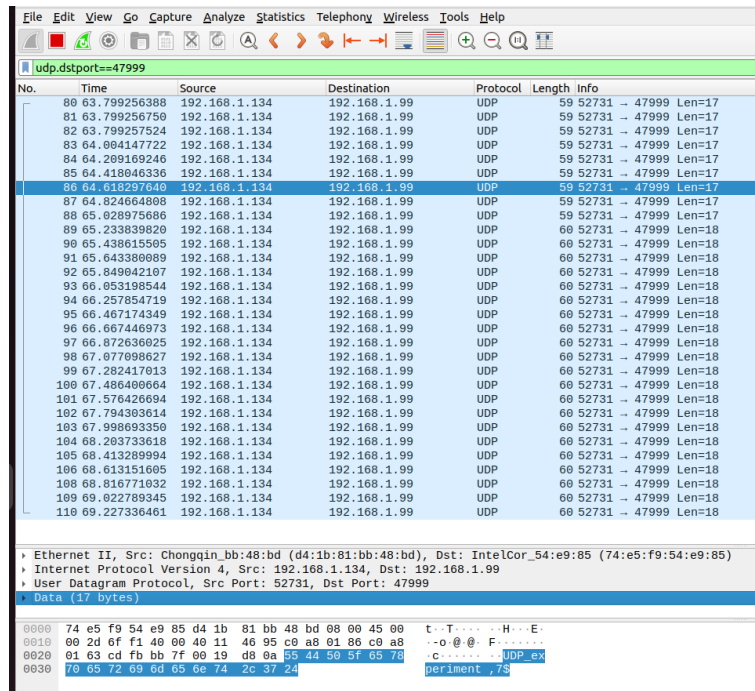


Figure 5: Wireshark capture on port 47999

Figure 4 shows the server received 30 UDP packets from the client. Wireshark (Figure 5) confirms reception on port 47999.

Wireshark monitored UDP communication using the filter

```
udp.port == 47999
```

to show only packets sent to the server's port 47999.

One captured packet shows:

- **Ethernet II layer:**
 - Source MAC: d4:1b:81:bb:48:bd
 - Destination MAC: 74:e5:f9:54:e9:85
- **IPv4 layer:**
 - Source IP: 192.168.1.134
 - Destination IP: 192.168.1.99

These match the addresses from `ifconfig` on the devices.

The payload in the Data section shows hex values representing ASCII text.

It contains the string:

```
UDP_experiment,7$
```

matching the client message and confirming correct data transfer.

The destination port is 47999, set by the client and used by the server. The source port is random.

If the client changes the destination port, the server stops receiving packets, though Wireshark still sees them. This is because the server listens only on port 47999, ignoring others.

Basic communication with TCP sockets

```
1 import socket
2 import time
3
4 TCP_IP = "192.168.1.99"
5 TCP_PORT = 48999
6 MESSAGE_STRING = "TCPexperiment"
7
8 print("TCP server IP: %s" % TCP_IP)
9 print("TCP destination port: %s" % TCP_PORT)
10
11 count = 0
12 sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
13 sock.connect((TCP_IP, TCP_PORT))
14
15 while count < 30:
16     count += 1
17     message = MESSAGE_STRING + "," + str(count) + "$"
18     print("sending message: %s" % message)
19     sock.send(message.encode())
20     time.sleep(0.2)
21
22 sock.close()
23
```

Listing 3: TCP client script sending 30 packets

```
1 import socket
2
3 TCP_IP = "192.168.1.99"
4 TCP_PORT = 48999
5
6 sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
7 sock.bind((TCP_IP, TCP_PORT))
8 sock.listen(1)
9 print("Server listening on port", TCP_PORT)
10
11 conn, addr = sock.accept()
12 print("Connection from:", addr)
13
14 count = 0
15
16 while True:
17     data = conn.recv(1024)
18     if not data:
19         break
20     messages = data.decode().split('$')
21     for message in messages[:-1]:
22         count += 1
23         print("Received message: %s, %d" % (message, count))
24         conn.send(message.encode())
25
26 conn.close()
```

Listing 4: TCP server script printing received messages and count

Figure 6: Comparison of TCP client and server Python scripts

```
lsnall@lsnall-TP410URR:~/Lab_05$ python3 L3_TCP_Server.py
Server listening on 192.168.1.99:48999
Connection accepted from: ('192.168.1.134', 59416)
Received message: TCPexperiment,1,1
Received message: TCPexperiment,2,2
Received message: TCPexperiment,3,3
Received message: TCPexperiment,4,4
Received message: TCPexperiment,5,5
Received message: TCPexperiment,6,6
Received message: TCPexperiment,7,7
Received message: TCPexperiment,8,8
Received message: TCPexperiment,9,9
Received message: TCPexperiment,10,10
Received message: TCPexperiment,11,11
Received message: TCPexperiment,12,12
Received message: TCPexperiment,13,13
Received message: TCPexperiment,14,14
Received message: TCPexperiment,15,15
Received message: TCPexperiment,16,16
Received message: TCPexperiment,17,17
Received message: TCPexperiment,18,18
Received message: TCPexperiment,19,19
Received message: TCPexperiment,20,20
Received message: TCPexperiment,21,21
Received message: TCPexperiment,22,22
Received message: TCPexperiment,23,23
Received message: TCPexperiment,24,24
Received message: TCPexperiment,25,25
Received message: TCPexperiment,26,26
Received message: TCPexperiment,27,27
Received message: TCPexperiment,28,28
Received message: TCPexperiment,29,29
Received message: TCPexperiment,30,30
lsnall@lsnall-TP410URR:~/Lab_05$
```

Figure 7: TCP server output on port 48999

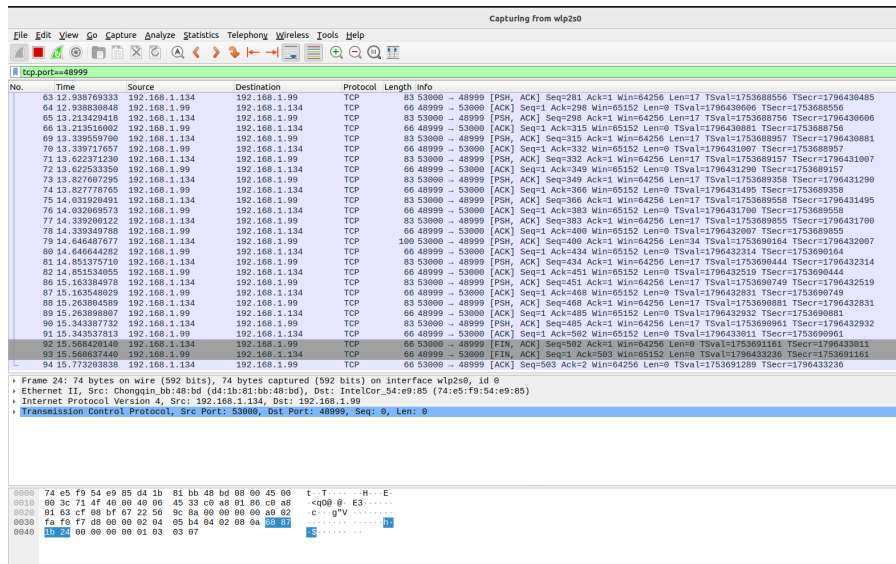


Figure 8: TCP packets on port 48999 highlighting the connection termination via FIN, ACK exchange

No.	Time	Source	Destination	Protocol	Length	Info
381	0.048626612	192.168.1.134	192.168.1.99	TCP	74	59226 → 48999 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=1753283337 TSecr=0 WS=128
382	0.048698845	192.168.1.99	192.168.1.134	TCP	74	48999 → 59226 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=1796025472 TSecr=1753283337 WS=128
383	0.048618741	192.168.1.134	192.168.1.99	TCP	66	59226 → 48999 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=1753283736 TSecr=1796025472
384	0.048619375	192.168.1.134	192.168.1.99	TCP	82	59226 → 48999 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=16 TSval=1753283737 TSecr=1796025472
385	0.048621605	192.168.1.99	192.168.1.134	TCP	66	48999 → 59226 [ACK] Seq=1 Ack=1 Win=65152 Len=0 TSval=1796026086 TSecr=1753283737
386	0.063311194	192.168.1.134	192.168.1.99	TCP	82	59226 → 48999 [PSH, ACK] Seq=17 Ack=1 Win=64256 Len=16 TSval=1753283937 TSecr=1796025882
387	0.063375589	192.168.1.99	192.168.1.134	TCP	66	48999 → 59226 [ACK] Seq=1 Ack=33 Win=65152 Len=0 TSval=1796026086 TSecr=1753283937
388	0.067724730	192.168.1.134	192.168.1.99	TCP	82	59226 → 48999 [PSH, ACK] Seq=33 Ack=1 Win=64256 Len=16 TSval=1753284151 TSecr=1796026086
389	0.067788180	192.168.1.99	192.168.1.134	TCP	66	48999 → 59226 [ACK] Seq=1 Ack=49 Win=65152 Len=0 TSval=1796026291 TSecr=1753284151
393	0.072435150	192.168.1.134	192.168.1.99	TCP	82	59226 → 48999 [PSH, ACK] Seq=49 Ack=1 Win=64256 Len=16 TSval=1753284453 TSecr=1796026291
394	0.072519550	192.168.1.99	192.168.1.134	TCP	66	48999 → 59226 [ACK] Seq=1 Ack=85 Win=65152 Len=0 TSval=1796026496 TSecr=1753284453
395	0.081576554	192.168.1.134	192.168.1.99	TCP	82	59226 → 48999 [PSH, ACK] Seq=85 Ack=1 Win=64256 Len=16 TSval=1753284538 TSecr=1796026496
396	0.08157708000	192.168.1.99	192.168.1.134	TCP	66	48999 → 59226 [ACK] Seq=1 Ack=81 Win=65152 Len=0 TSval=1796026581 TSecr=1753284538
397	0.08384772957	192.168.1.134	192.168.1.99	TCP	82	59226 → 48999 [PSH, ACK] Seq=81 Ack=1 Win=64256 Len=16 TSval=1753284738 TSecr=1796026581
398	0.08384830719	192.168.1.99	192.168.1.134	TCP	66	48999 → 59226 [ACK] Seq=1 Ack=97 Win=65152 Len=0 TSval=1796026688 TSecr=1753284738
399	0.0838484895	192.168.1.134	192.168.1.99	TCP	82	59226 → 48999 [PSH, ACK] Seq=97 Ack=1 Win=64256 Len=16 TSval=1753285072 TSecr=1796026688
400	0.083849060	192.168.1.99	192.168.1.134	TCP	66	48999 → 59226 [ACK] Seq=1 Ack=113 Win=65152 Len=0 TSval=1796027212 TSecr=1753285072
401	0.08384915527	192.168.1.134	192.168.1.99	TCP	98	59226 → 48999 [PSH, ACK] Seq=113 Ack=1 Win=64256 Len=32 TSval=1753285477 TSecr=1796027212
402	0.0838492548	192.168.1.99	192.168.1.134	TCP	66	48999 → 59226 [ACK] Seq=1 Ack=145 Win=65152 Len=0 TSval=1796027623 TSecr=1753285477
403	0.08384935037	192.168.1.134	192.168.1.99	TCP	100	59226 → 48999 [PSH, ACK] Seq=145 Ack=1 Win=64256 Len=34 TSval=1753285787 TSecr=1796027623
404	0.08384945576	192.168.1.99	192.168.1.134	TCP	66	48999 → 59226 [ACK] Seq=1 Ack=179 Win=65152 Len=0 TSval=1796027929 TSecr=1753285787
405	0.08384955924	192.168.1.134	192.168.1.99	TCP	83	59226 → 48999 [PSH, ACK] Seq=179 Ack=1 Win=64256 Len=17 TSval=1753285941 TSecr=1796027929
406	0.08384965188	192.168.1.99	192.168.1.134	TCP	66	48999 → 59226 [ACK] Seq=1 Ack=196 Win=65152 Len=0 TSval=1796027983 TSecr=1753285941
407	0.0838497506	192.168.1.134	192.168.1.99	TCP	83	59226 → 48999 [PSH, ACK] Seq=196 Ack=1 Win=64256 Len=17 TSval=1753286141 TSecr=1796027983
408	0.08384984610	192.168.1.99	192.168.1.134	TCP	66	48999 → 59226 [ACK] Seq=1 Ack=213 Win=65152 Len=0 TSval=1796028237 TSecr=1753286141
409	0.08384994180	192.168.1.134	192.168.1.99	TCP	83	59226 → 48999 [PSH, ACK] Seq=213 Ack=1 Win=64256 Len=17 TSval=1753286342 TSecr=1796028237
410	0.08384994560	192.168.1.99	192.168.1.134	TCP	66	48999 → 59226 [ACK] Seq=1 Ack=230 Win=65152 Len=0 TSval=1796028441 TSecr=1753286342

* Frame 385: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface wlp2s8, id 0
 * Ethernet II, Src: IntelCor.54:e9:85 (74:65:f9:54:e9:85), Dst: Chongqin.bb:48:bd (d4:1b:01:bb:48:bd)
 * Internet Protocol Version 4, Src: 192.168.1.99, Dst: 192.168.1.134
 * Transmission Control Protocol, Src Port: 48999, Dst Port: 59226, Seq: 1, Ack: 17, Len: 0

```

0000  d4 1b 01 bb 48 bd 74 65 f9 54 e9 85 00 00 45 00  ...H.t.Y...E.
0010  00 34 c8 e1 40 00 40 06 ee 28 c0 a8 01 63 c0 a8  4 a0 @ . (...c
0020  01 06 bf 67 e7 5a d0 25 66 58 c8 1e 36 fe 80 10  ...g.Z.%fx.6...
0030  01 fd 15 e7 00 00 01 01 08 0a 0b 0d 2e 1a 68 00  .......k...h...
0040  fc 99
  
```

Figure 9: Detailed TCP handshake packets: client SYN and server SYN, ACK

We used Wireshark on device 2 and captured TCP traffic filtered by:

`tcp.port == 48999`

This shows only packets on port 48999, used by our application.

Handshake: The first packets captured are [PSH, ACK], meaning the handshake (SYN, SYN-ACK, ACK) was likely completed before capture started. However, Figure 9 shows the detailed SYN and SYN-ACK packets initiating the connection.

Data exchange: Multiple [PSH, ACK] packets carry 17 bytes of data, each followed by an [ACK] confirming receipt (see Figure 8).

Termination: Figure 9 shows the connection close sequence: client sends FIN, ACK, server replies with FIN, ACK, then client sends final ACK, completing the TCP four-step termination.

No.	Time	Source	Destination	Protocol	Length	Info
92	23.769353455	192.168.1.134	192.168.1.99	TCP	74	57764 → 55555 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=1755833450 TSecr=0 WS=128
93	23.760402638	192.168.1.99	192.168.1.134	TCP	54	55555 → 57764 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

Figure 10: TCP server output showing multiple messages received together on port 55555

We changed the client's destination port to 55555, but the server was still listening on 48999.

With TCP, the connection failed. Wireshark shows the client sending a [SYN] to port 55555, but the server replies with [RST, ACK] (connection refused). This means the server was not listening on that port.

No data exchange or handshake occurred, and the server received nothing.

With UDP, packets would still be sent even if the port is wrong, but the server would ignore them. UDP does not require a connection.

After restoring the client to the correct port (48999), the TCP connection and communication worked as expected, confirming the proper operation of our TCP client-server programs.

Communication in unreliable networks with packet loss

No.	Time	Source	Destination	Protocol	Length	Info
875	385.239504189	192.168.1.134	192.168.1.99	UDP	59	39456 → 65000 Len=17
876	385.239642788	192.168.1.134	192.168.1.134	ICMP	87	Destination unreachable (Port unreachable)
877	385.449442651	192.168.1.134	192.168.1.99	UDP	59	39456 → 65000 Len=17
878	385.449514157	192.168.1.99	192.168.1.134	ICMP	87	Destination unreachable (Port unreachable)
881	385.649526594	192.168.1.134	192.168.1.99	UDP	59	39456 → 65000 Len=17
882	385.649595740	192.168.1.99	192.168.1.134	ICMP	87	Destination unreachable (Port unreachable)
883	385.854892517	192.168.1.134	192.168.1.99	UDP	59	39456 → 65000 Len=17
884	385.854179548	192.168.1.99	192.168.1.134	ICMP	87	Destination unreachable (Port unreachable)
886	386.058917353	192.168.1.134	192.168.1.99	UDP	59	39456 → 65000 Len=17
887	386.263993081	192.168.1.134	192.168.1.99	UDP	59	39456 → 65000 Len=17
888	386.468826249	192.168.1.134	192.168.1.99	UDP	60	39456 → 65000 Len=18
889	386.673529214	192.168.1.134	192.168.1.99	UDP	60	39456 → 65000 Len=18
890	386.878461348	192.168.1.134	192.168.1.99	UDP	60	39456 → 65000 Len=18
891	386.878519594	192.168.1.99	192.168.1.134	ICMP	88	Destination unreachable (Port unreachable)
893	387.007625334	192.168.1.134	192.168.1.99	UDP	60	39456 → 65000 Len=18
894	387.287562906	192.168.1.134	192.168.1.99	UDP	60	39456 → 65000 Len=18
895	387.492647789	192.168.1.134	192.168.1.99	UDP	60	39456 → 65000 Len=18
896	387.697569344	192.168.1.134	192.168.1.99	UDP	60	39456 → 65000 Len=18
897	387.906993339	192.168.1.134	192.168.1.99	UDP	60	39456 → 65000 Len=18
898	387.907641529	192.168.1.99	192.168.1.134	ICMP	88	Destination unreachable (Port unreachable)
899	388.001385540	192.168.1.134	192.168.1.99	UDP	60	39456 → 65000 Len=18
900	388.209888493	192.168.1.134	192.168.1.99	UDP	60	39456 → 65000 Len=18
901	388.414371561	192.168.1.134	192.168.1.99	UDP	60	39456 → 65000 Len=18
902	388.619148709	192.168.1.134	192.168.1.99	UDP	60	39456 → 65000 Len=18
903	388.823943200	192.168.1.134	192.168.1.99	UDP	60	39456 → 65000 Len=18
904	388.824081282	192.168.1.99	192.168.1.134	ICMP	88	Destination unreachable (Port unreachable)
905	389.028841851	192.168.1.134	192.168.1.99	UDP	60	39456 → 65000 Len=18
907	389.238266926	192.168.1.134	192.168.1.99	UDP	60	39456 → 65000 Len=18
909	389.443059538	192.168.1.134	192.168.1.99	UDP	60	39456 → 65000 Len=18
910	389.643297896	192.168.1.134	192.168.1.99	UDP	60	39456 → 65000 Len=18
911	389.848074901	192.168.1.134	192.168.1.99	UDP	60	39456 → 65000 Len=18
912	389.848162636	192.168.1.99	192.168.1.134	ICMP	88	Destination unreachable (Port unreachable)
917	390.001259226	192.168.1.134	192.168.1.99	UDP	60	39456 → 65000 Len=18
918	390.258033454	192.168.1.134	192.168.1.99	UDP	60	39456 → 65000 Len=18
919	390.466997375	192.168.1.134	192.168.1.99	UDP	60	39456 → 65000 Len=18

Figure 11 shows that if the client sends messages to the wrong port, the UDP server does not receive any data. This highlights that UDP requires correct socket configuration to function and offers no feedback in case of errors.

Figure 11: UDP server output when client uses port 65000 (port mismatch)

```
ismail@ismail-TP410URR:~/Lab_03$ python3 L3_UDP_Server.py
Server is listening on IP 192.168.155.20, port 47999
^CTraceback (most recent call last):
  File "/home/ismail/Lab_03/L3_UDP_Server.py", line 19, in <module>
    data, addr = sock.recvfrom(512)
KeyboardInterrupt

ismail@ismail-TP410URR:~/Lab_03$ python3 L3_UDP_Server.py
Server is listening on IP 192.168.155.20, port 47999
received message: UDP_experiment, 25, 1
received message: UDP_experiment, 35, 2
received message: UDP_experiment, 45, 3
received message: UDP_experiment, 65, 4
received message: UDP_experiment, 75, 5
received message: UDP_experiment, 85, 6
received message: UDP_experiment, 125, 7
received message: UDP_experiment, 135, 8
received message: UDP_experiment, 145, 9
received message: UDP_experiment, 155, 10
received message: UDP_experiment, 165, 11
received message: UDP_experiment, 175, 12
received message: UDP_experiment, 185, 13
received message: UDP_experiment, 195, 14
received message: UDP_experiment, 205, 15
received message: UDP_experiment, 225, 16
received message: UDP_experiment, 235, 17
received message: UDP_experiment, 245, 18
received message: UDP_experiment, 255, 19
received message: UDP_experiment, 265, 20
received message: UDP_experiment, 275, 21
received message: UDP_experiment, 285, 22
received message: UDP_experiment, 295, 23
received message: UDP_experiment, 305, 24
```

Figure 12: UDP server output under 20% packet loss

```
ubuntu@ubuntu: ~/LAB03
ubuntu@ubuntu:~$ sudo tc qdisc add dev wlo1 root netem loss 20%
ubuntu@ubuntu:~$ cd LAB03
ubuntu@ubuntu:~/LAB03$ python3 UDP_clientloss.py
UDP server IP: 192.168.155.20
UDP destination port: 47999
sending message: UDP_experiment,15$
sending message: UDP_experiment,25$
sending message: UDP_experiment,35$
sending message: UDP_experiment,45$
sending message: UDP_experiment,55$
sending message: UDP_experiment,65$
sending message: UDP_experiment,75$
sending message: UDP_experiment,85$
sending message: UDP_experiment,95$
sending message: UDP_experiment,105$
sending message: UDP_experiment,115$
sending message: UDP_experiment,125$
sending message: UDP_experiment,135$
sending message: UDP_experiment,145$
sending message: UDP_experiment,155$
sending message: UDP_experiment,165$
sending message: UDP_experiment,175$
sending message: UDP_experiment,185$
sending message: UDP_experiment,195$
sending message: UDP_experiment,205$
sending message: UDP_experiment,215$
sending message: UDP_experiment,225$
sending message: UDP_experiment,235$
sending message: UDP_experiment,245$
sending message: UDP_experiment,255$
sending message: UDP_experiment,265$
sending message: UDP_experiment,275$
sending message: UDP_experiment,285$
sending message: UDP_experiment,295$
sending message: UDP_experiment,305$
ubuntu@ubuntu:~/LAB03$
```

Figure 13: Command to simulate 20% packet loss using tc

In Figure 12, we observe missing sequence numbers in the UDP server output. This confirms that packets are indeed dropped under network loss conditions, and that UDP does not attempt any retransmission. Figure 13 shows the command used to apply 20% packet loss using `tc qdisc`.

```

ismail@ismail-TP410URR:~/Lab_03$ python3 L3_UDP_Server.py
Server is listening on IP 192.168.155.20, port 47999
received message: UDP_experiment, 3$, 1
received message: UDP_experiment, 4$, 2
received message: UDP_experiment, 5$, 3
received message: UDP_experiment, 6$, 4
received message: UDP_experiment, 8$, 5
received message: UDP_experiment, 9$, 6
received message: UDP_experiment, 10$, 7
received message: UDP_experiment, 11$, 8
received message: UDP_experiment, 12$, 9
received message: UDP_experiment, 16$, 10
received message: UDP_experiment, 19$, 11
received message: UDP_experiment, 20$, 12
received message: UDP_experiment, 21$, 13
received message: UDP_experiment, 22$, 14
received message: UDP_experiment, 23$, 15
received message: UDP_experiment, 24$, 16
received message: UDP_experiment, 25$, 17
received message: UDP_experiment, 26$, 18
received message: UDP_experiment, 27$, 19
received message: UDP_experiment, 28$, 20

```

Figure 14: UDP server output under 30% packet loss

Under 30% simulated packet loss (group 5), the UDP server received only 24 out of 30 packets (Figure 14). This results in an estimated loss of:

$$\frac{30 - 24}{30} \times 100 \approx 20\%$$

The actual loss is lower than the configured value, as expected due to randomness in simulation. As before, UDP does not recover missing packets.

We attempted to test TCP under the same conditions but could not establish a working connection. Therefore, a comparison is not possible from our tests, though TCP would typically recover lost packets through retransmission at the cost of added latency.

Writing a latency tester for a Wi-Fi sensor network

```

1 import socket
2 import time
3 import argparse
4
5 parser = argparse.ArgumentParser()
6 parser.add_argument("--num_pkt", help="number
    of packets to be sent to the server")
7 args = parser.parse_args()
8 num_pkt = int(args.num_pkt)
9
10 TCP_IP = "192.168.155.48"
11 TCP_PORT = 55555
12 TEST_ID = "test_MSN. 1"
13
14 sock = socket.socket(socket.AF_INET, socket.
    SOCK_STREAM)
15 sock.connect((TCP_IP, TCP_PORT))
16
17 for i in range(num_pkt):
18     t1 = time.time()
19     message = f"{TEST_ID},{i+1},{t1}"
20     sock.send(message.encode())
21     data = sock.recv(1024).decode()
22     t2 = time.time()
23
24     parts = data.split(",")
25     sent_time = float(parts[2])
26     rtt = (t2 - sent_time) * 1000
27
28     print(f"RTT for packet {i+1}: {rtt:.2f}
    ms")
29     time.sleep(0.2)
30
31 sock.close()
32

```

Listing 5: TCP client with RTT measurement

```

1 import socket
2
3 TCP_IP = "192.168.1.48"
4 TCP_PORT = 55555
5
6 sock = socket.socket(socket.AF_INET, socket.
    SOCK_STREAM)
7 sock.bind((TCP_IP, TCP_PORT))
8 sock.listen(1)
9
10 print("Server listening on port", TCP_PORT)
11 conn, addr = sock.accept()
12 print("Connection from:", addr)
13
14 while True:
15     data = conn.recv(1024)
16     if not data:
17         break
18     print("Received:", data.decode())
19     conn.send(data)
20
21 conn.close()
22

```

Listing 6: TCP server echoing received packets

```

ubuntu@ubuntu:~/LAB03$ python3 RTT_client.py --num_pkt 10
RTT for packet 1: 168.17 ms
RTT for packet 2: 49.42 ms
RTT for packet 3: 404.74 ms
RTT for packet 4: 11.73 ms
RTT for packet 5: 198.11 ms
RTT for packet 6: 9.32 ms
RTT for packet 7: 107.42 ms
RTT for packet 8: 514.97 ms
RTT for packet 9: 128.45 ms
RTT for packet 10: 97.32 ms
ubuntu@ubuntu:~/LAB03$

```

```

ismail@ismail-TP410URR:~/Lab_03$ python3 rtt_server.py
Server listening on port 55555
Connection from: ('192.168.155.48', 36714)
Received: test_MSN.1,1,1748453930.934327
Received: test_MSN.1,2,1748453931.1450899
Received: test_MSN.1,3,1748453931.5935357
Received: test_MSN.1,4,1748453931.8455024
Received: test_MSN.1,5,1748453932.1506252
Received: test_MSN.1,6,1748453932.4639308
Received: test_MSN.1,7,1748453932.767245
Received: test_MSN.1,8,1748453933.2180536
Received: test_MSN.1,9,1748453933.4580562
Received: test_MSN.1,10,1748453933.6892836
ismail@ismail-TP410URR:~/Lab_03$

```

Figure 15: TCP client and server scripts with round-trip time (RTT) measurement

The client and server scripts are written in Python using TCP sockets. The client sends a packet with a test identifier, packet number, and the current timestamp. After receiving the reply, it calculates the Round-Trip Time (RTT).

The server listens on port 55555 and echoes back each received message.

The buffer size is set to 1024 bytes, which is enough for our small packets. The client waits 200 ms between each packet to avoid congestion.

```

ismail@ismail-TP410URR:~/Lab_03$ python3 rtt_server.py
Server listening on port 55555
Connection from: ('192.168.155.48', 36714)
Received: test_MSN.1,1,1748453930.934327
Received: test_MSN.1,2,1748453931.1450899
Received: test_MSN.1,3,1748453931.5935357
Received: test_MSN.1,4,1748453931.8455024
Received: test_MSN.1,5,1748453932.1506252
Received: test_MSN.1,6,1748453932.4639308
Received: test_MSN.1,7,1748453932.767245
Received: test_MSN.1,8,1748453933.2180536
Received: test_MSN.1,9,1748453933.4580562
Received: test_MSN.1,10,1748453933.6892836
ismail@ismail-TP410URR:~/Lab_03$

```

Figure 16: RTT server echo output

```

ubuntu@ubuntu:~/LAB03$ python3 RTT_client.py --num_pkt 20
RTT for packet 1: 104.10 ms
RTT for packet 2: 347.59 ms
RTT for packet 3: 8.85 ms
RTT for packet 4: 272.89 ms
RTT for packet 5: 8.63 ms
RTT for packet 6: 9.26 ms
RTT for packet 7: 10.20 ms
RTT for packet 8: 7.10 ms
RTT for packet 9: 10.24 ms
RTT for packet 10: 86.36 ms
RTT for packet 11: 106.54 ms
RTT for packet 12: 7.23 ms
RTT for packet 13: 6.84 ms
RTT for packet 14: 6.93 ms
RTT for packet 15: 105.94 ms
RTT for packet 16: 14.06 ms
RTT for packet 17: 91.80 ms
RTT for packet 18: 204.86 ms
RTT for packet 19: 211.41 ms
RTT for packet 20: 10.85 ms

```

Figure 17: Ping results for RTT comparison

```

ubuntu@ubuntu:~$ ping 192.168.155.20
PING 192.168.155.20 (192.168.155.20) 56(84) bytes of data:
64 bytes from 192.168.155.20: icmp_seq=1 ttl=64 time=210 ms
64 bytes from 192.168.155.20: icmp_seq=2 ttl=64 time=8.12 ms
64 bytes from 192.168.155.20: icmp_seq=3 ttl=64 time=54.8 ms
64 bytes from 192.168.155.20: icmp_seq=4 ttl=64 time=76.0 ms
64 bytes from 192.168.155.20: icmp_seq=5 ttl=64 time=8.73 ms
64 bytes from 192.168.155.20: icmp_seq=6 ttl=64 time=533 ms
64 bytes from 192.168.155.20: icmp_seq=7 ttl=64 time=148 ms
64 bytes from 192.168.155.20: icmp_seq=8 ttl=64 time=580 ms
64 bytes from 192.168.155.20: icmp_seq=9 ttl=64 time=709 ms
64 bytes from 192.168.155.20: icmp_seq=10 ttl=64 time=423 ms
64 bytes from 192.168.155.20: icmp_seq=11 ttl=64 time=33.3 ms
64 bytes from 192.168.155.20: icmp_seq=12 ttl=64 time=672 ms
64 bytes from 192.168.155.20: icmp_seq=13 ttl=64 time=95.9 ms
64 bytes from 192.168.155.20: icmp_seq=14 ttl=64 time=719 ms
64 bytes from 192.168.155.20: icmp_seq=15 ttl=64 time=537 ms
64 bytes from 192.168.155.20: icmp_seq=16 ttl=64 time=144 ms
64 bytes from 192.168.155.20: icmp_seq=17 ttl=64 time=203 ms
64 bytes from 192.168.155.20: icmp_seq=18 ttl=64 time=191 ms
64 bytes from 192.168.155.20: icmp_seq=19 ttl=64 time=251 ms
^C
--- 192.168.155.20 ping statistics ---
20 packets transmitted, 19 received, 5% packet loss, time 19022ms
rtt min/avg/max/mdev = 8.120/294.609/719.098/246.869 ms

```

Figure 18: Ping executed from the client.