

Praktikum Jaringan Komputer



NRP : 3223600017
Nama : Muhammad Alfarrel Arya Mahardika
Materi : Socket Programming TCP dan UDP Echo Client/Server
Tanggal : 27 Februari 2025

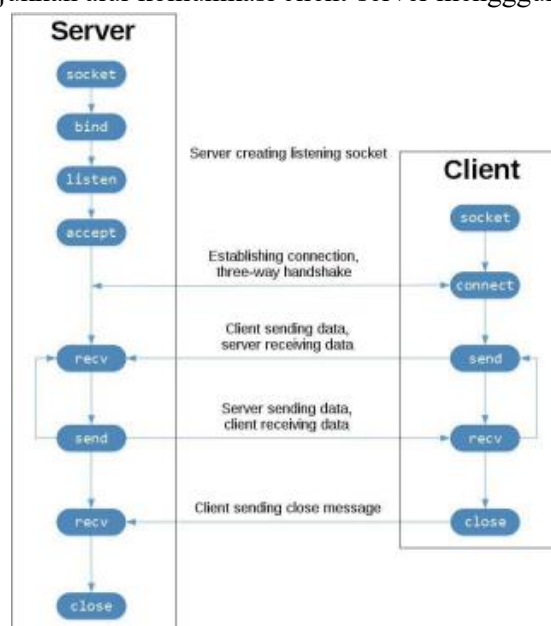
1. Tujuan

- 1.1. Mahasiswa dapat menjelaskan konsep jaringan berbasis client-server
- 1.2. Mahasiswa dapat menjelaskan konsep pemrograman socket berbasis TCP
- 1.3. Mahasiswa dapat menjelaskan konsep pemrograman socket berbasis UDP
- 1.4. Mahasiswa dapat menjelaskan cara kerja program TCP echo client/server
- 1.5. Mahasiswa dapat menjelaskan cara kerja program UDP echo client/server

2. Dasar Teori

Transport layer dalam model TCP/IP bertanggung jawab atas pengiriman data antar perangkat dalam suatu jaringan. Protokol utama yang bekerja pada lapisan ini adalah Transmission Control Protocol (TCP) dan User Datagram Protocol (UDP). Meskipun keduanya berfungsi untuk mengirimkan data, masing-masing memiliki karakteristik dan kegunaan yang berbeda.

Transmission Control Protocol (TCP) merupakan protokol pada transport layer berbasis connection-oriented. TCP menjamin realibilitas pengiriman data. Selain itu, TCP memiliki mekanisme three-way handshake sebelum client-server dapat saling mengirimkan data. Pada gambar 1 berikut menunjukkan alur komunikasi client-server menggunakan TCP.

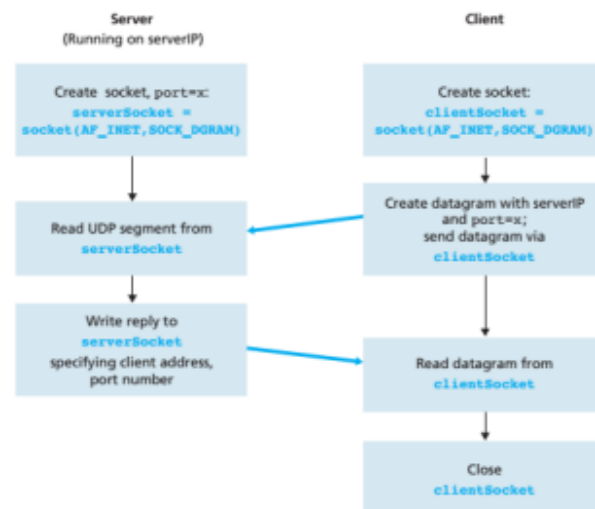


Pada gambar menunjukkan bahwa di sisi server terdapat beberapa socket API yang digunakan agar dapat berkomunikasi pada jaringan. Socket API pada TCP server yang bertugas untuk membuat listening socket yaitu: socket(), bind(), listen(), accept(). Proses listening socket merupakan saat dimana server menunggu koneksi dari client. Jadi, saat client menghubungkan ke server dengan connect(), hal ini mengawali proses connection establishment. Kemudian server akan menerima koneksi dari client melalui accept(), dan mekanisme three-way handshake selesai. Selanjutnya, client dan server dapat saling berkomunikasi atau bertukar data menggunakan send() dan recv().

Sedangkan untuk UDP atau User Datagram protocol merupakan protokol yang bersifat connectionless. UDP tidak dapat menjamin pengiriman maupun urutan paket. Selain itu juga tidak ada mekanisme handshaking. Sehingga, reliabilitas UDP tidak sebaik TCP. UDP menyediakan

checksum untuk integritas data dan port number sebagai pengalamatan. UDP diperlukan untuk aplikasi yang membutuhkan komunikasi efisien dan cepat tanpa mempermasalahkan packet loss.

Pada gambar dibawah ini menunjukkan proses pengiriman data untuk komunikasi menggunakan protokol UDP. Bila dilihat pada gambar ataupun pada prosedur percobaan, pada sisi server tidak ada proses listen dan accept seperti pada TCP. Namun, hanya ada proses bind untuk menghubungkan socket dengan port dan proses menunggu pesan yang dikirim dari client. Sedangkan pada sisi client menggunakan sendto() untuk mengirimkan pesan secara langsung ke server, dan recvfrom() untuk menerima pesan balasan dari server.



3. Prosedur

Protokol TCP

3.1. Buatlah program echo server.

```
#!/usr/bin/env python3

import socket
import sys
import argparse

host = 'localhost'
data_payload = 2048
backlog = 5

def echo_server(port):
    # Membuat socket object dengan TCP
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
```

```

server_address = (host, port)

print("Starting up echo server on %s port %s" % server_address)

# Bind the socket, Menentukan port yang digunakan pada host tujuan
sock.bind(server_address)

# Listen for incoming connection from clients, Nilai backlog menentukan jumlah
    maksimum antrian koneksi
sock.listen(backlog)

while True:
    print("Waiting to receive message from client")
    # Establish a connection, menunggu koneksi dari client
    client, address = sock.accept()

    # Server menerima data dari client
    data = client.recv(data_payload)

    if data:
        print("Data: %s" % data.decode('utf-8'))

        # Server mengirim data ke client
        client.send(data)
        print("sent %s bytes back to %s" % (data.decode('utf-8'), address))

    # Koneksi berakhir
    client.close()

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Socket Server Example')
    parser.add_argument('--port', action="store", dest="port", type=int, required=True)
    given_args = parser.parse_args()
    port = given_args.port
    server(port)

```

3.2. Buatlah program echo client.

```
#!/usr/bin/env python3

import socket
import sys
import argparse

host = 'localhost'

def echo_client(port):
    # Membuat socket object dengan TCP
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_address = (host, port)

    print("Connecting to %s port %s" % server_address)

    # Mengawali koneksi dengan server
    sock.connect(server_address)

    try:
        # Mengirim data ke server
        message = "Hello World!!! This will be echoed"
        print("Sending %s" % message)
        sock.sendall(message.encode('utf-8'))

        # Melihat respon
        amount_received = 0
        amount_expected = len(message)

        while amount_received < amount_expected:
            # Menerima data, balasan dari server
            data = sock.recv(16)
            amount_received += len(data)
            print("Received: %s" % data.decode('utf-8'))
```

```

except socket.error as e:
    print("Socket error: %s" % str(e))

except Exception as e:
    print("Other exception: %s" % str(e))

finally:
    print("Closing connection to the server")
    sock.close()

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Socket Client Example')
    parser.add_argument('--port', action="store", dest="port", type=int, required=True)
    given_args = parser.parse_args()
    port = given_args.port
    client(port)

```

- 3.3. Jalankan program echo server terlebih dulu. Tampilkan hasil pengujian anda.

```

# python3 tcpechoserver.py --port=3456
Starting up echo server on localhost port 3456
Waiting to receive message from client

```

Catatan: port tersebut adalah port server, dimana client nantinya akan menggunakan port tersebut agar terhubung ke server, gunakan non privileged ports yaitu port > 1023

- 3.4. Jalankan program echo client secara terpisah terminal atau command prompt dari program echo server. Tampilkan hasil pengujian anda.

```

# python3 tcpechoclient.py --port=3456
Connecting to localhost port 3456
Sending Hello World!!! This will be echoed
Received: Hello World!!! T
Received: his will be echo
Received: ed
Closing connection to the server

```

Sedangkan pada server akan tampil seperti berikut :

```
Data: Hello World!!! This will be echoed
sent Hello World!!! This will be echoed bytes back to ('127.0.0.1', 48486)
Waiting to receive message from client
```

- 3.5. Pada pengujian di prosedur 3 dan 4 menggunakan localhost, anda dapat juga menggunakan host yang berbeda dengan menyesuaikan IP address tujuan.
- 3.6. Buatlah program echo client/server, dimana anda bisa memberikan inputan melalui program client kemudian server akan membalas dengan mengirimkan ulang ke client apapun yang anda kirimkan sebelumnya. Program client dapat membaca inputan dari keyboard dan anda dapat mengirimkan data ke server beberapa kali selama koneksi dari client belum diakhiri.
- 3.7. Lampirkan program echo client/server yang anda buat untuk prosedur no. 6 pada laporan anda. Tampilkan hasil setelah anda menjalankan program yang anda buat tersebut.
- 3.8. Analisa program echo client/server pada prosedur 1 dan 2 serta pada tugas di prosedur 6. Jelaskan bagaimana penerapan mekanisme TCP pada socket programming dan perintah-perintah yang digunakan pada program socket yang telah dibuat.

Protokol UDP

- 3.9. Buatlah program echo server.

```
#!/usr/bin/env python3

import socket
import sys
import argparse

host = 'localhost'
data_payload = 2048

def echo_server(port):
    # Membuat socket object menggunakan UDP
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    server_address = (host, port)

    print("Starting up echo server on %s port %s" % server_address)
```

```

# Bind the socket, menentukan port yang digunakan pada host tujuan
sock.bind(server_address)

while True:
    # Menunggu menerima pesan dari client
    print("Waiting to receive message from client")
    data, address = sock.recvfrom(data_payload)
    print("received %s bytes from %s" % (len(data), address))
    print("Data: %s" % data.decode('utf-8'))

    if data:
        # Server mengirim pesan ke client
        sent = sock.sendto(data, address)
        print("sent %s bytes back to %s" % (sent, address))

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Socket Server Example')
    parser.add_argument('--port', action="store", dest="port", type=int, required=True)
    given_args = parser.parse_args()
    port = given_args.port
    echo_server(port)

```

3.10. Buatlah program echo client.

```

#!/usr/bin/env python3

import socket
import sys
import argparse

host = 'localhost'
data_payload = 2048

def echo_client(port):
    # Membuat socket object menggunakan UDP
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    server_address = (host, port)

```



```

print("Connecting to %s port %s" % server_address)
message = 'This is the message. It will be repeated.'

try:
    # Mengirim pesan ke server
    message = "Good Morning!!! This will be echoed"
    print("Sending %s" % message)
    sent = sock.sendto(message.encode('utf-8'), server_address)

    # Menerima balasan dari server
    data, server = sock.recvfrom(data_payload)
    print("received %s" % data.decode('utf-8'))

finally:
    print("Closing connection to the server")
    sock.close()

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Socket Client Example')
    parser.add_argument('--port', action="store", dest="port", type=int, required=True)
    given_args = parser.parse_args()
    port = given_args.port
    echo_client(port)

```

3.11. Jalankan program echo server terlebih dulu. Tampilkan hasil pengujian anda.

```

# python3 udpechoserver.py --port=8765
Starting up echo server on localhost port 8765
Waiting to receive message from client

```

3.12. Jalankan program echo client secara terpisah terminal atau command prompt dari program echo server. Tampilkan hasil pengujian anda.

```

# python3 udpechoclient.py --port=8765
Connecting to localhost port 8765
Sending Good Morning!!! This will be echoed

```

```
received Good Morning!!! This will be echoed
Closing connection to the server
```

Sedangkan pada server akan tampil berikut:

```
received 35 bytes from ('127.0.0.1', 55559)
Data: Good Morning!!! This will be echoed
sent 35 bytes back to ('127.0.0.1', 55559)
```

- 3.13. Pada pengujian di prosedur 3 dan 4 menggunakan localhost, anda dapat juga menggunakan host yang berbeda dengan menyesuaikan IP address tujuan.
- 3.14. Lakukan packet analysis menggunakan aplikasi Wireshark untuk mengamati pengiriman data menggunakan protokol UDP
- 3.15. Lakukan packet analysis menggunakan aplikasi Wireshark untuk mengamati pengiriman data menggunakan protokol TCP, gunakan program TCP socket pada praktikum sebelumnya
- 3.16. Lakukan analisa dari pengamatan yang sudah anda lakukan menggunakan protokol TCP dan UDP. Jelaskan apa saja yang dapat anda analisa dari capture paket pada Wireshark, dan apa saja perbedaan pada paket data yang dikirimkan menggunakan TCP dan UDP.

4. Hasil Praktikum

Protokol TCP

4.1. Koneksi dengan Kabel LAN

IP Komp server = "192.168.10.8" --port=12345

IP Komp client = "192.168.10.7" --port=50892

a. Hasil pada console:

```
13 server_address = (host, port)
14
15 print("Starting up echo server on %s port %s" % server_address)
16 # Bind the socket, menentukan port yang digunakan pada host tujuan
17 sock.bind(server_address)
18
19 while True:
20     # Menunggu menerima pesan dari client
21     print("Waiting to receive message from client")
22
23 PROBLEMS 1 OUTPUT TERMINAL PORTS POSTMAN CONSOLE COMMENTS py - praktikum_2
24 PS D:\Serba Serbi Kuliah\Semester 4\Praktikum Jaringan Komputer> cd praktikum_2
25 PS D:\Serba Serbi Kuliah\Semester 4\Praktikum Jaringan Komputer\praktikum_2> py tcp_echo_server.py --port 12345
Starting up echo server on 192.168.10.8 port 12345
Waiting to receive message from client
Data: Hello World!!! This will be echoed
sent Hello World!!! This will be echoed bytes back to ('192.168.10.7', 50892)
Waiting to receive message from client
```

```
22 print("Sending %s" % message)
23 sock.sendall(message.encode('utf-8'))
24

echo_client(port)
File "D:\Mencoba\Dasar Python\cobaecho.py", line 17, in echo_client
sock.connect(server_address)
ConnectionRefusedError: [WinError 10061] No connection could be made because the target machine actively refused it
PS D:\Mencoba\Dasar Python> py cobaecho.py --port 12345
Connecting to 192.168.10.8 port 12345
Sending Hello World!!! This will be echoed
Received: Hello World!!! T
Received: his will be echo
Received: ed
Closing connection to the server
PS D:\Mencoba\Dasar Python>
```

b. Hasil pada Wireshark:

3	0.997713	192.168.10.7	224.0.0.251	MDNS	85	Standard query 0x0000 PTR _microsoft_mcc._tcp.local, "QM" question	
4	0.998066	fe80::26b:6da7:6b18::ff02::fb		MDNS	105	Standard query 0x0000 PTR _microsoft_mcc._tcp.local, "QM" question	
5	1.873480	GigaByteTech_27:f8::	Broadcast	ARP	42	Who has 192.168.10.8? Tell 192.168.10.7	
6	1.874552	00:e0:00:38:cf:10	GigaByteTech_27:f8::	ARP	60	192.168.10.8 is at 00:e0:00:38:cf:10	
7	1.874569	192.168.10.7	192.168.10.8	TCP	66	50892 → 12345 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM	
8	1.875304	192.168.10.8	192.168.10.7	TCP	66	12345 → 50892 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM	
9	1.875362	192.168.10.7	192.168.10.8	TCP	54	50892 → 12345 [ACK] Seq=1 Ack=1 Win=65536 Len=0	
10	1.875434	192.168.10.7	192.168.10.8	TCP	88	50892 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=34	
11	1.883084	192.168.10.8	192.168.10.7	TCP	88	12345 → 50892 [PSH, ACK] Seq=1 Ack=35 Win=1049600 Len=34	
12	1.883310	192.168.10.7	192.168.10.8	TCP	54	50892 → 12345 [FIN, ACK] Seq=35 Ack=35 Win=65536 Len=0	
13	1.883515	192.168.10.8	192.168.10.7	TCP	60	12345 → 50892 [FIN, ACK] Seq=35 Ack=35 Win=1049600 Len=0	
14	1.883533	192.168.10.7	192.168.10.8	TCP	54	50892 → 12345 [ACK] Seq=36 Ack=36 Win=65536 Len=0	
15	1.883888	192.168.10.8	192.168.10.7	TCP	60	12345 → 50892 [ACK] Seq=36 Ack=36 Win=1049600 Len=0	
16	10.176524	192.168.10.7	224.0.0.251	MDNS	386	Standard query response 0x0000 PTR DESKTOP-ED9LSLE._dosvc._tcp.local SRV 0 0 7680 DESKTOP-ED9LSLE.local TXT	
17	10.176760	fe80::26b:6da7:6b18::ff02::fb		MDNS	406	Standard query response 0x0000 PTR DESKTOP-ED9LSLE._dosvc._tcp.local SRV 0 0 7680 DESKTOP-ED9LSLE.local TXT	
18	10.177095	192.168.10.7	224.0.0.251	MDNS	93	Standard query 0x0000 ANY DESKTOP-ED9LSLE._dosvc._tcp.local, "QM" question	
19	10.177224	fe80::26b:6da7:6b18::ff02::fb		MDNS	113	Standard query 0x0000 ANY DESKTOP-ED9LSLE._dosvc._tcp.local, "QM" question	

Frame 12: 54 bytes on wire (432 bits), 54 bytes captured (432 bits)

Ethernet II, Src: GigaByteTech_27:f8:28 (d8:5e:d3:27:f8:28), Dst: 00:e0:00:38:cf:10 (00:e0:00:38:cf:10)

Internet Protocol Version 4, Src: 192.168.10.7, Dst: 192.168.10.8

Transmission Control Protocol, Src Port: 50892, Dst Port: 12345, Seq: 35, Ack: 35, Len: 0

0000 00 e0 00 38 cf 10 d8 5e d3 27 f8 28 00 00 45 00 8 B A (E

0010 00 28 b9 d8 40 00 00 06 00 00 c0 a8 0a 07 c0 a8 (@

0020 0a 08 c6 cc 30 39 5d 0e 49 68 35 f6 32 cd 50 11 ...09] IHS 2 P

0030 01 00 95 7a 00 00 z

No.	Time	Source	Destination	Protocol	Length	Info	
1	0.000000	192.168.10.7	224.0.0.251	MDNS	85	Standard query 0x0000 PTR _microsoft_mcc._tcp.local, "QM" question	
2	0.000224	fe80::26b:6da7:6b18::ff02::fb		MDNS	105	Standard query 0x0000 PTR _microsoft_mcc._tcp.local, "QM" question	
3	0.997713	192.168.10.7	224.0.0.251	MDNS	85	Standard query 0x0000 PTR _microsoft_mcc._tcp.local, "QM" question	
4	0.998066	GigaByteTech_27:f8::	Broadcast	ARP	42	Who has 192.168.10.8? Tell 192.168.10.7	
5	1.874552	00:e0:00:38:cf:10	GigaByteTech_27:f8::	ARP	60	192.168.10.8 is at 00:e0:00:38:cf:10	
7	1.874569	192.168.10.7	192.168.10.8	TCP	66	50892 → 12345 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM	
8	1.875304	192.168.10.8	192.168.10.7	TCP	66	12345 → 50892 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM	
9	1.875362	192.168.10.7	192.168.10.8	TCP	54	50892 → 12345 [ACK] Seq=1 Ack=1 Win=65536 Len=0	
10	1.875434	192.168.10.7	192.168.10.8	TCP	88	50892 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=34	
11	1.883084	192.168.10.8	192.168.10.7	TCP	88	12345 → 50892 [PSH, ACK] Seq=1 Ack=35 Win=1049600 Len=34	
12	1.883310	192.168.10.7	192.168.10.8	TCP	54	50892 → 12345 [FIN, ACK] Seq=35 Ack=35 Win=65536 Len=0	
13	1.883515	192.168.10.8	192.168.10.7	TCP	60	12345 → 50892 [FIN, ACK] Seq=35 Ack=35 Win=1049600 Len=0	
14	1.883533	192.168.10.7	192.168.10.8	TCP	54	50892 → 12345 [ACK] Seq=36 Ack=36 Win=65536 Len=0	
15	1.883888	192.168.10.8	192.168.10.7	TCP	60	12345 → 50892 [ACK] Seq=36 Ack=36 Win=1049600 Len=0	
16	10.176524	192.168.10.7	224.0.0.251	MDNS	386	Standard query response 0x0000 PTR DESKTOP-ED9LSLE._dosvc._tcp.local SRV 0 0 7680 DESKTOP-ED9LSLE.local TXT	
17	10.176760	fe80::26b:6da7:6b18::ff02::fb		MDNS	406	Standard query response 0x0000 PTR DESKTOP-ED9LSLE._dosvc._tcp.local SRV 0 0 7680 DESKTOP-ED9LSLE.local TXT	
18	10.177095	192.168.10.7	224.0.0.251	MDNS	93	Standard query 0x0000 ANY DESKTOP-ED9LSLE._dosvc._tcp.local, "QM" question	
19	10.177224	fe80::26b:6da7:6b18::ff02::fb		MDNS	113	Standard query 0x0000 ANY DESKTOP-ED9LSLE._dosvc._tcp.local, "QM" question	

Frame 11: 88 bytes on wire (704 bits), 88 bytes captured (704 bits)

Ethernet II, Src: 00:e0:00:38:cf:10 (00:e0:00:38:cf:10), Dst: GigaByteTech_27:f8:28 (d8:5e:d3:27:f8:28)

Internet Protocol Version 4, Src: 192.168.10.8, Dst: 192.168.10.7

Transmission Control Protocol, Src Port: 12345, Dst Port: 50892, Seq: 1, Ack: 35, Len: 34

Data (34 bytes)

0000 0d 5c d5 27 f8 28 00 08 00 38 cf 10 08 00 45 00 A ((8 ... E

0010 00 4a e3 d0 40 00 00 06 81 72 c0 a8 0a 00 c0 a8 3 @

0020 0a 07 30 39 c6 cc 35 f6 32 ab 5d 0e 49 68 50 18 ...09 5 2] IHP

0030 10 04 df 40 00 00 48 65 6c 6c 6f 20 57 6f 72 6c @ The llo Worl

0040 64 21 21 21 20 54 68 69 73 20 77 69 6c 20 62 dHl Thi s will b

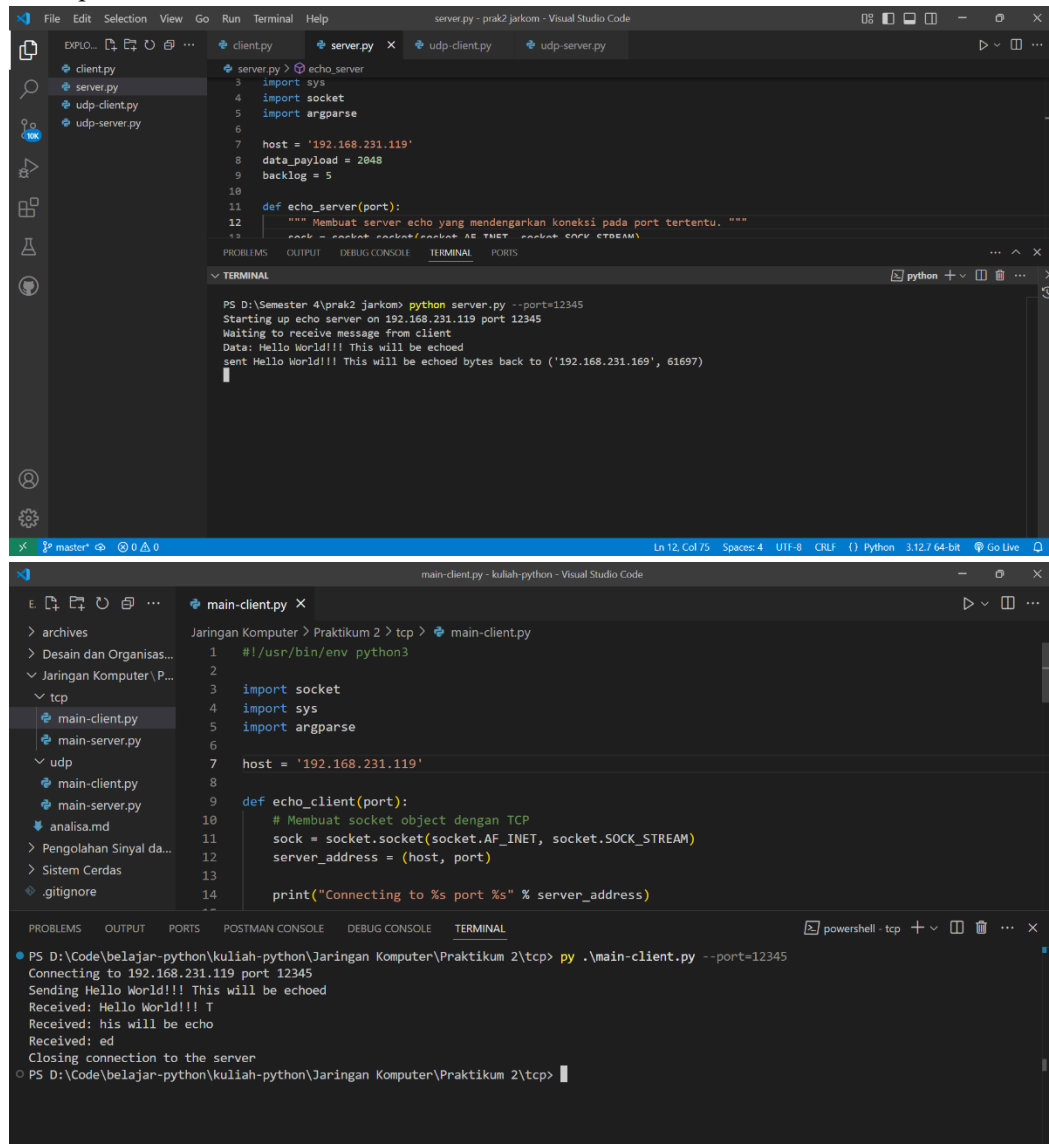
0050 65 20 65 63 68 6f 65 64 e echoed

4.2. Koneksi dengan WIFI

IP Komp server = "192.168.231.119" --port=12345

IP Komp client = "192.168.231.169" --port=61697

a. Hasil pada Console:



The first screenshot shows the Visual Studio Code interface with a file explorer on the left containing files: client.py, server.py, udp-client.py, and udp-server.py. The main editor displays the code for server.py, which is a TCP echo server. The code imports sys, socket, and argparse, sets host to '192.168.231.119', data_payload to 2048, and backlog to 5. It defines a function echo_server(port) that starts a server on the specified port, waits for a connection, receives data, and echoes it back. The terminal shows the command 'python server.py --port=12345' being executed, and the output indicates the server is running and has received a message from the client.

```
server.py - prak2 jarkom - Visual Studio Code
client.py  server.py  udp-client.py  udp-server.py
server.py > echo_server
3 import sys
4 import socket
5 import argparse
6
7 host = '192.168.231.119'
8 data_payload = 2048
9 backlog = 5
10
11 def echo_server(port):
12     """ Membuat server echo yang mendengarkan koneksi pada port tertentu. """
13     sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
14     sock.bind((host, port))
15     sock.listen(backlog)
16     while True:
17         conn, addr = sock.accept()
18         data = conn.recv(data_payload)
19         if data:
20             conn.send(data)
21
22 if __name__ == '__main__':
23     parser = argparse.ArgumentParser()
24     parser.add_argument('--port', type=int, required=True)
25     args = parser.parse_args()
26     echo_server(args.port)
```

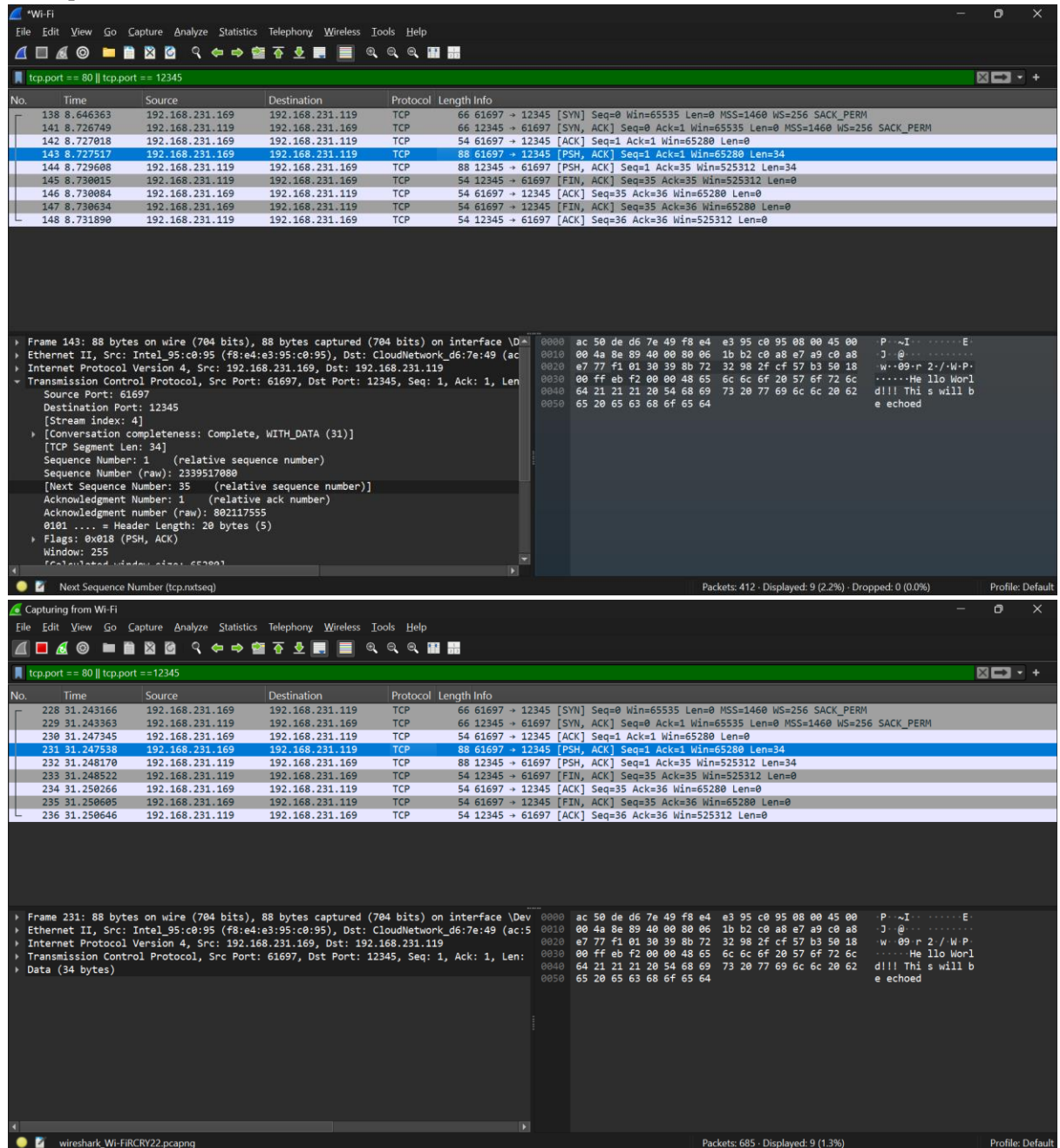
PS D:\Semester 4\prak2 jarkom> python server.py --port=12345
Starting up echo server on 192.168.231.119 port 12345
Waiting to receive message from client
Data: Hello World!!! This will be echoed
sent Hello World!!! This will be echoed bytes back to ('192.168.231.169', 61697)

The second screenshot shows the Visual Studio Code interface with a file explorer on the left containing files: main-client.py, main-server.py, and main-server.py. The main editor displays the code for main-client.py, which is a TCP echo client. The code imports sys, socket, and argparse, sets host to '192.168.231.119', and defines a function echo_client(port) that creates a socket, connects to the server, and sends data. The terminal shows the command 'python main-client.py --port=12345' being executed, and the output indicates the client is connecting to the server and sending a message.

```
main-client.py - kuliah-python - Visual Studio Code
main-client.py  main-server.py  main-server.py
1 #!/usr/bin/env python3
2
3 import socket
4 import sys
5 import argparse
6
7 host = '192.168.231.119'
8
9 def echo_client(port):
10     # Membuat socket object dengan TCP
11     sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
12     server_address = (host, port)
13     sock.connect(server_address)
14     print("Connecting to %s port %s" % server_address)
```

PS D:\Code\belajar-python\kuliah-python\Jaringan Komputer\Praktikum 2\tcp> py .\main-client.py --port=12345
Connecting to 192.168.231.119 port 12345
Sending Hello World!!! This will be echoed
Received: Hello World!!! T
Received: his will be echo
Received: ed
Closing connection to the server
PS D:\Code\belajar-python\kuliah-python\Jaringan Komputer\Praktikum 2\tcp>

b. Hasil pada Wireshark:



4.3. Chat Echo client-server

IP Komp server = "192.168.100.92" --port=12345

IP Komp client = "192.168.100.87" --port=59831

a. Program server:

```
import socket
import argparse
import threading

HOST = 'localhost'
BUFFER_SIZE = 1024

def handle_client(client_socket, client_address):
```

```

"""Handles communication with a single client."""
print(f'Terhubung dengan {client_address}')

while True:
    try:
        data = client_socket.recv(BUFFER_SIZE)
        if not data:
            break

        message = data.decode('utf-8')
        print(f'Pesan dari {client_address}: {message}')

        if message.lower() == "exit":
            print(f'Client {client_address} meminta untuk mengakhiri koneksi.')
            client_socket.sendall("Koneksi ditutup oleh server.".encode('utf-8'))
            break

        client_socket.sendall(data) # Echo the message back to client

    except ConnectionResetError:
        print(f'Client {client_address} terputus secara paksa.')
        break

    client_socket.close()
    print(f'Koneksi dengan {client_address} ditutup.')

def echo_server(port):
    """Main server function to handle multiple clients."""
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    server_address = (HOST, port)
    sock.bind(server_address)
    sock.listen(5)

    print(f'Server berjalan di {HOST} port {port}...')

    while True:
        client_socket, client_address = sock.accept()

        # Create a new thread for each connected client
        client_thread = threading.Thread(target=handle_client, args=(client_socket,
client_address))
        client_thread.start()

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Multi-Client Echo Server")
    parser.add_argument('--port', type=int, required=True, help="Port untuk server")
    args = parser.parse_args()
    echo_server(args.port)

```

b. Program client:

```

import socket
import argparse

```

```

HOST = 'localhost'

def echo_client(port):
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_address = (HOST, port)
    print(f"Menghubungkan ke {HOST} port {port}...")
    sock.connect(server_address)

    try:
        while True:
            message = input("Masukkan pesan: ")
            sock.sendall(message.encode('utf-8'))

            if message.lower() == "exit":
                print("Menutup koneksi ke server.")
                break

            data = sock.recv(1024)
            print(f"Server mengembalikan: {data.decode('utf-8')}")

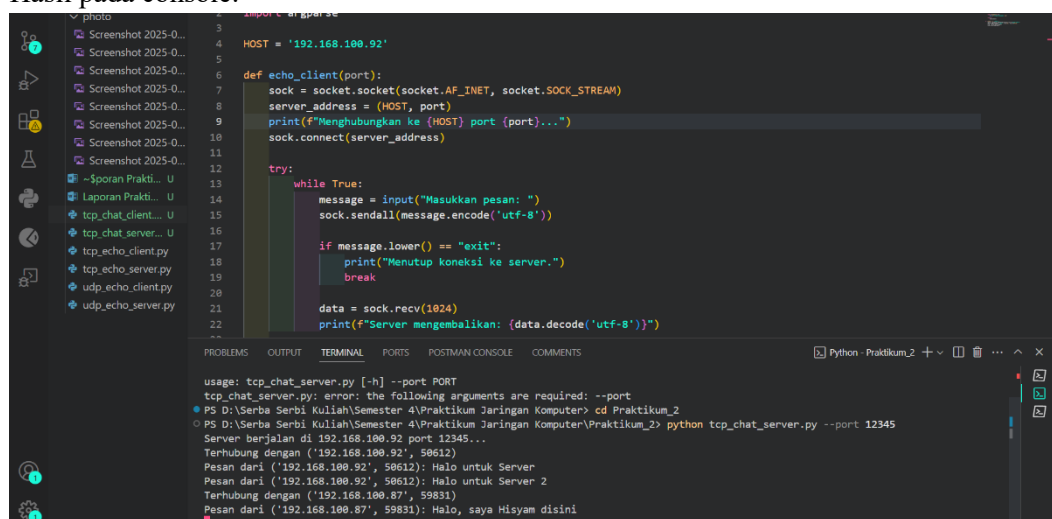
    except Exception as e:
        print(f"Terjadi kesalahan: {e}")

    finally:
        sock.close()

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Echo Client")
    parser.add_argument('--port', type=int, required=True)
    args = parser.parse_args()
    echo_client(args.port)

```

c. Hasil pada console:



```

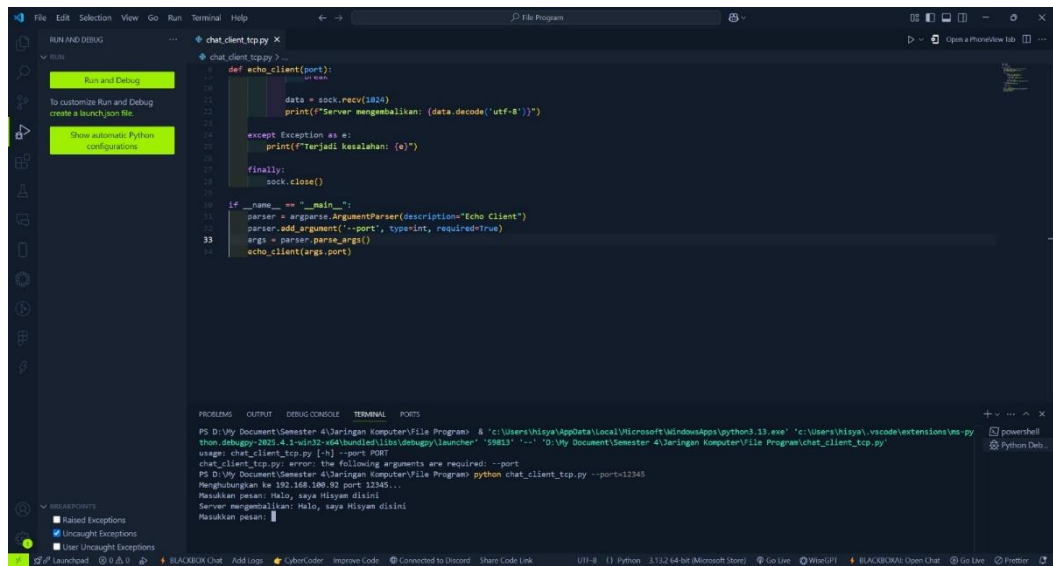
1 import argparse
2
3 HOST = '192.168.100.92'
4
5
6 def echo_client(port):
7     sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
8     server_address = (HOST, port)
9     print(f"Menghubungkan ke {HOST} port {port}...")
10    sock.connect(server_address)
11
12    try:
13        while True:
14            message = input("Masukkan pesan: ")
15            sock.sendall(message.encode('utf-8'))
16
17            if message.lower() == "exit":
18                print("Menutup koneksi ke server.")
19                break
20
21            data = sock.recv(1024)
22            print(f"Server mengembalikan: {data.decode('utf-8')}")
23
24    except Exception as e:
25        print(f"Terjadi kesalahan: {e}")
26
27    finally:
28        sock.close()
29
30 if __name__ == "__main__":
31     parser = argparse.ArgumentParser(description="Echo Client")
32     parser.add_argument('--port', type=int, required=True)
33     args = parser.parse_args()
34     echo_client(args.port)

```

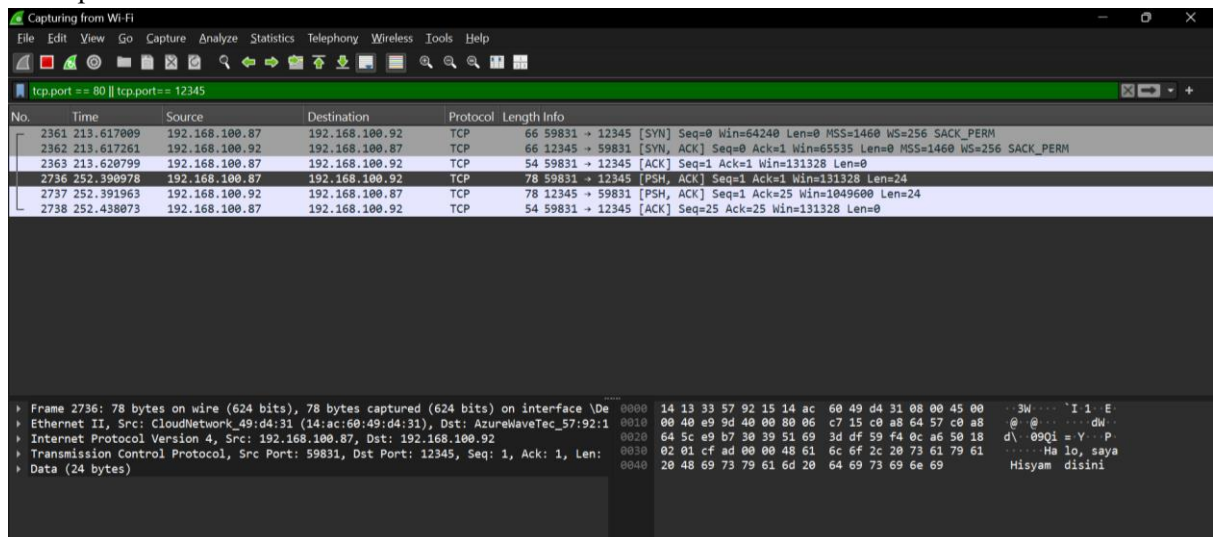
```

usage: tcp_chat_server.py [-h] --port PORT
tcp_chat_server.py: error: the following arguments are required: --port
PS D:\Serba Serbi Kuliah\Semester 4\Praktikum Jaringan Komputer> cd Praktikum_2
PS D:\Serba Serbi Kuliah\Semester 4\Praktikum Jaringan Komputer\Praktikum_2> python tcp_chat_server.py --port 12345
Server berjalan di 192.168.100.92 port 12345...
Terhubung dengan ('192.168.100.92', 58612)
Pesan dari ('192.168.100.92', 58612): Halo untuk Server
Pesan dari ('192.168.100.92', 58612): Halo untuk Server 2
Terhubung dengan ('192.168.100.87', 59831)
Pesan dari ('192.168.100.87', 59831): Halo, saya Hisyam disini

```



d. Hasil pada wireshark:




```

11 0.771857      192.168.10.7      224.0.0.251      NMS     387 Standard query response 0x0000 SRV, cache flush 0 0 7680 DESKTOP-ED9L5LF.local TXT, cache flush A, cache flush 192.168.10.7 AAAA, cac-
12 0.771873      f400:126::da7:b18... ff02::fb         NMS     487 Standard query response 0x0000 SRV, cache flush 0 0 7680 DESKTOP-ED9L5LF.local TXT, cache flush A, cache flush 192.168.10.7 AAAA, cac-
14 46.509051      192.168.10.7      192.168.10.8     UDP     77 60916 + 12345 Len=35
14 46.511316      00:e8:00:38:cfcf:10 Broadcast       ARP     60 Who has 192.168.10.7? Tell 192.168.10.8
15 46.511332      GigaByteTech-27:f8:... 00:e8:00:38:cfcf:10 ARP     42 192.168.10.7 is at d8:5e:d3:27:f8:28
16 46.511922      192.168.10.8      192.168.10.7     UDP     77 12345 + 60916 Len=35
17 46.512575       GigaByteTech-27:f8:... 00:e8:00:38:cfcf:10 ARP     42 Who has 192.168.10.8? Tell 192.168.10.7
18 51.258736      00:e8:00:38:cfcf:10 GigaByteTech-27:f8:... ARP     60 192.168.10.8 is at 00:e8:00:38:cfcf:10

Frame 16: 77 bytes on wire (616 bits), 77 bytes captured (616 bits) on interface 0
Ethernet II, Src: 00:e8:00:38:cfcf:10 (00:e8:00:38:cfcf:10), Dst: GigaByteTech-27:f8:28 (d8:5e:d3:27:f8:28)
Internet Protocol Version 4, Src: 192.168.10.7, Dst: 192.168.10.8
User Datagram Protocol, Src Port: 12345, Dst Port: 60916
Data (35 bytes)
0000  d8 5e d3 27 f8 28 00 e8 00 38 cf 10 08 00 45 00  ... (.S.E
0010  00 3f 95 3c 00 00 11 10 12 c0 a8 0a 08 c0 a8  ... ..Good M
0020  0a 07 10 19 e6 fe 0a 20 85 12 4f 6f 64 20 4d  ... ..
0030  64 72 62 69 6f 67 21 21 20 54 68 69 73 28 77  ... orning!! This w
0040  69 6c 6c 20 62 65 20 65 63 68 6f 65 64      ... ill be choed
```

7	0.519120	192.168.10.7	224.0.0.251	NDNS	93 Standard query 0x0000 ANY DESKTOP-ED915LE._dosvc._tcp.local, "QM" question
8	0.519272	fe80::26b:6da7:6b18::ff02::fb		NDNS	113 Standard query 0x0000 ANY DESKTOP-ED915LE._dosvc._tcp.local, "QM" question
9	0.771081	192.168.10.7	224.0.0.251	NDNS	451 Standard query response 0x0000 PTR, cache flush DESKTOP-ED915LE._dosvc._tcp.local SRV, cache flush 0 0 7680 DESKTOP-ED915LE.local TXT...
10	0.771242	fe80::26b:6da7:6b18::ff02::fb		NDNS	471 Standard query response 0x0000 PTR, cache flush DESKTOP-ED915LE._dosvc._tcp.local SRV, cache flush 0 0 7680 DESKTOP-ED915LE.local TXT...
11	0.771357	192.168.10.7	224.0.0.251	NDNS	387 Standard query response 0x0000 SRV, cache flush 0 0 7680 DESKTOP-ED915LE.local TXT, cache flush A, cache flush 192.168.10.7 AAAA, cac...
12	0.771473	fe80::26b:6da7:6b18::ff02::fb		NDNS	407 Standard query response 0x0000 SRV, cache flush 0 0 7680 DESKTOP-ED915LE.local TXT, cache flush A, cache flush 192.168.10.7 AAAA, cac...
13	46.509051	192.168.10.7	192.168.10.8	UDP	77 60916 → 12345 Len=35
14	46.511316	00:e8:00:38:cf:10	Broadcast	ARP	60 Who has 192.168.10.7? Tell 192.168.10.8
15	46.511332	GigaByteTech_27:f8:28	00:e8:00:38:cf:10	ARP	42 192.168.10.7 is at d8:5e:d3:27:f8:28
16	46.511922	192.168.10.8	192.168.10.7	UDP	77 12345 → 60916 Len=35
17	51.257575	GigaByteTech_27:f8:28	00:e8:00:38:cf:10	ARP	42 Who has 192.168.10.8? Tell 192.168.10.7
18	51.258736	00:e8:00:38:cf:10	GigaByteTech_27:f8:28	ARP	60 192.168.10.8 is at 00:e8:00:38:cf:10

Frame 13: 77 bytes on wire (616 bits), 77 bytes captured (616 bits)	0000 00 e8 00 38 cf 10 d8 5e d3 27 f8 28 08 00 45 00 ... 8 ... (E
Ethernet II, Src: GigaByteTech_27:f8:28 (d8:5e:d3:27:f8:28), Dst: 00:e8:00:38:cf:10 (00:e8:00:38:cf:10)	0010 00 3f b9 da 00 00 00 11 00 00 c0 a8 0a 07 c0 a8 ... ?
Internet Protocol Version 4, Src: 192.168.10.7, Dst: 192.168.10.8	0020 0a 08 ed f4 30 39 00 2b 95 9c 47 6f 6f 64 20 4d ... 09 + ... Good M
User Datagram Protocol, Src Port: 60916, Dst Port: 12345	0030 6f 72 6e 69 6a 67 21 21 21 20 54 68 69 73 20 77 ... oming!! This w
Data (35 bytes)	0040 69 6c 6c 20 62 65 20 65 63 68 6f 65 64 ... ill be e choed

4.5. Koneksi dengan WIFI

IP Komp server = "192.168.231.169" --port=12345

IP Komp client = "192.168.231.88" --port=55598

a. Hasil pada console:

```

main-server.py x
> archives
> Desain dan Organisasi...
> Jaringan Komputer (P...
  > tcp
    > main-client.py
    > main-server.py
  > udp
    > main-client.py
    > main-server.py
  > analisa.md
  > Pengolahan Sinyal da...
  > Sistem Cerdas
  > .gitignore

Jaringan Komputer > Praktikum 2 > udp > main-server.py
1 #!usr/bin/env python3
2
3 import socket
4 import sys
5 import argparse
6
7 host = '192.168.231.169'
8 data_payload = 2048
9
10 def echo_server(port):
11     # Membuat socket object menggunakan UDP
12     sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
13     server_address = (host, port)
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

PROBLEMS OUTPUT PORTS POSTMAN CONSOLE DEBUG CONSOLE TERMINAL
PS D:\Code\belajar-python\kuliah-python\Jaringan Komputer\Praktikum 2\udp> py .\main-server.py --port=12345
Starting up echo server on 192.168.231.169 port 12345
Waiting to receive message from client
received 35 bytes from ('192.168.231.88', 55598)
Data: Good Morning!!! This will be echoed
sent 35 bytes back to ('192.168.231.88', 55598)
Waiting to receive message from client
  
```

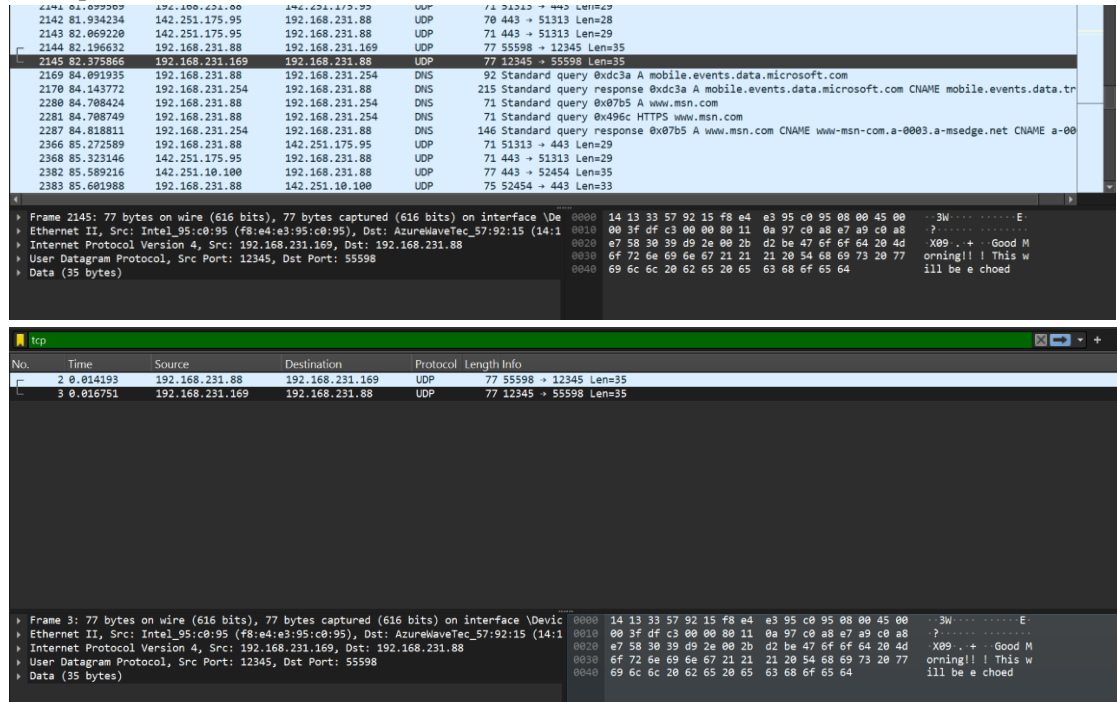


```

Praktikum_2 > udp_echo_client.py > ...
1 #!usr/bin/env python3
2
3 import socket
4 import sys
5 import argparse
6
7 host = '192.168.231.169'
8 data_payload = 2048
9
10 def echo_client(port):
11     # Membuat socket object menggunakan UDP
12     sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
13     server_address = (host, port)
14
15     print("Connecting to %s port %s" % server_address)
16     message = 'This is the message. It will be repeated.'
17
18     try:
19         # Mengirim pesan ke server
20         message = "Good Morning!!! This will be echoed"
21         print("Sending %s" % message)
22         sent = sock.sendto(message.encode('utf-8'), server_address)
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

PROBLEMS OUTPUT TERMINAL PORTS POSTMAN CONSOLE COMMENTS
PS D:\Serba Serbi Kuliah\Semester 4\Praktikum Jaringan Komputer> cd Praktikum_2
PS D:\Serba Serbi Kuliah\Semester 4\Praktikum Jaringan Komputer\Praktikum_2> python udp_echo_client.py --port 12345
Connecting to 192.168.231.169 port 12345
Sending Good Morning!!! This will be echoed
received Good Morning!!! This will be echoed
Closing connection to the server
PS D:\Serba Serbi Kuliah\Semester 4\Praktikum Jaringan Komputer\Praktikum_2>
  
```

b. Hasil pada Wireshark:



5. Analisa

Pada Praktikum Kedua dilakukan percobaan socket programming TCP dan UDP echo client/server. TCP atau Transmission Control Protocol dan UDP atau User Datagram Protocol adalah sebuah protokol dalam koneksi antar satu perangkat dengan perangkat lain di bagian transport layer dari tujuh layer OSI model. Transport layer sendiri adalah layer yang berada di bagian keempat dari atas yang mana berfungsi untuk control aliran dan koreksi data yang dikirim dari server atau sebuah perangkat melalui jaringan dan proses control dan koreksi data tersebut di atur oleh dua protokol yaitu TCP dan UDP tadi.

Protokol TCP adalah protokol yang memiliki sifat connection-oriented yang mana protokol ini akan memastikan setiap paket data yang dikirimkan terverifikasi atau benar-benar diterima oleh perangkat penerima dengan adanya konfirmasi pengiriman yang benar dari penerima, jadi ada koneksi yang terjalin antar dua perangkat untuk memastikan paket data terkirimkan dengan baik. Protokol ini biasanya digunakan untuk pengiriman data-data penting seperti suatu halaman web. Lalu di percobaan echo server/client untuk TCP ini akan dilakukan dengan mengkoneksikan dua perangkat komputer dalam satu jaringan baik lan atau wifi untuk saling berkomunikasi dengan protokol tcp yang mana salah satu perangkat akan bertindak sebagai server dan yang lainnya sebagai client yang mengirimkan sebuah pesan.

Untuk percobaannya akan menggunakan program python dengan library socket, sys, argparse untuk mendukung koneksi jaringan antar komputer dan pengiriman data string anatar keduanya.

```

import socket
import sys
import argparse

host = 'localhost'
data_payload = 2048
backlog = 5

def echo_server(port):
    # Membuat socket object dengan TCP
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    server_address = (host, port)

    print("Starting up echo server on %s port %s" % server_address)

    # Bind the socket, Menentukan port yang digunakan pada host tujuan
    sock.bind(server_address)

    # Listen for incoming connection from clients, Nilai backlog menentukan jumlah maksimum antrian koneksi
    sock.listen(backlog)

    while True:
        print("Waiting to receive message from client")
        # Establish a connection, menunggu koneksi dari client
        client, address = sock.accept()

        # Server menerima data dari client
        data = client.recv(data_payload)

        if data:
            print("Data: %s" % data.decode('utf-8'))

            # Server mengirim data ke client
            client.send(data)
            print("sent %s bytes back to %s" % (data.decode('utf-8'), address))

        # Koneksi berakhir
        client.close()

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Socket Server Example')
    parser.add_argument('--port', action="store", dest="port", type=int, required=True)
    given_args = parser.parse_args()
    port = given_args.port
    echo_server(port)

```

Di program diatas bisa dilihat untuk program komputer server nya, dalam fungsi echo_server aka nada parameter input port yang mana port tersebut sebagai tujuan proses mana yang digunakan pada alamat ip komputer servernya. Lalu terdapat inisialisasi variabel sock untuk inisialisasi protokol IPv4 dan menggunakan protokol TCP. Lalu ada variabel server_address untuk menginputkan ip host dan port yang dituju. Lalu kedua variabel itu masuk dalam sock.bind(server_address) untuk memproses Alamat IP dan port yang di inputkan menggunakan protokol IPv4 dan protokol TCP. Lalu fungsi listen(backlog) untuk membatasi koneksi yang bisa dilakukan oleh server, yang mana disitu backlog di setting di 5, yang mana hanya 5 perangkat saja yang bisa berkoneksi sekaligus dalam satu waktu. Lalu dimulailah loop while true untuk menerima pesan-pesan yang akan dikirimkan oleh client nantinya. Dalam variabel data disitu terdapat data_payload yaitu pembatasan besaran pesan yang bisa dibaca oleh server dari client, yaitu di program tersebut di setting di 2048 byte. If atau saat data diterima maka data atau pesannya akan di print di console dan akan dibalas oleh server lagi kembali ke client. Setelah itu baru koneksi antar dua program echo client/server ditutup oleh server dengan client.close().

Sedangkan untuk program clientnya sendiri adalah

```
import socket
import sys
import argparse

host = 'localhost'

def echo_client(port):
    # Membuat socket object dengan TCP
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_address = (host, port)

    print("Connecting to %s port %s" % server_address)

    # Mengawali koneksi dengan server
    sock.connect(server_address)

    try:
        # Mengirim data ke server
        message = "Hello World!!! This will be echoed"
        print("Sending %s" % message)
        sock.sendall(message.encode('utf-8'))

        # Melihat respon
        amount_received = 0
        amount_expected = len(message)

        while amount_received < amount_expected:
            # Menerima data, balasan dari server
            data = sock.recv(16)
            amount_received += len(data)
            print("Received: %s" % data.decode('utf-8'))

    except socket.error as e:
        print("Socket error: %s" % str(e))

    except Exception as e:
        print("Other exception: %s" % str(e))

    finally:
        print("Closing connection to the server")
        sock.close()

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Socket Client Example')
    parser.add_argument('--port', action="store", dest="port", type=int, required=True)
    given_args = parser.parse_args()
    port = given_args.port
    echo_client(port)
```

Untuk program dari komputer clientnya hampir sama pengaturan variabel yang ada seperti inisialisasi sock port dan ip servernya beserta penggunaan protokol IPv4 dan TCP. Lalu dalam bagian try terdapat variabel message yang berisikan variabel string pesan yang akan dikirimkan ke server nantinya, dengan fungsi sendall() yang memastikan seluruh pesan terkirim nantinya. Lalu setelah data terkirim ke server, client akan menunggu balasan dari server bahwa pesannya telah terbaca. Barulah setelah itu koneksi keduanya di tutup.

Hasilnya server nanti akan menerima pesan yang dikirimkan oleh client, beserta port dan IP yang digunakan oleh client untuk memproses pengirim data tersebut.


```

13     server_address = (host, port)
14
15     print("Starting up echo server on %s port %s" % server_address)
16     # Bind the socket, menentukan port yang digunakan pada host tujuan
17     sock.bind(server_address)
18
19     while True:
20         # Menunggu menerima pesan dari client
21         print("Waiting to receive message from client")

```

PROBLEMS 1 OUTPUT TERMINAL PORTS POSTMAN CONSOLE COMMENTS py - praktikum_2

```

PS D:\Serba Serbi Kuliah\Semester 4\Praktikum Jaringan Komputer> cd praktikum_2
PS D:\Serba Serbi Kuliah\Semester 4\Praktikum Jaringan Komputer> py tcp_echo_server.py --port 12345
Starting up echo server on 192.168.10.8 port 12345
Waiting to receive message from client
Data: Hello World!!! This will be echoed
sent Hello World!!! This will be echoed bytes back to ('192.168.10.7', 50892)
Waiting to receive message from client

```

Lalu di console client pun setelah data terkirim, maka client akan menerima balasan dari server yang mana server akan mengirimkan kembali packet packet data yang diterima oleh server satu persatu, bisa dilihat disitu satu pesan “Hello World!!! This will be echoed” di pecah menjadi tiga packet dan server mengonfirmasi balik tiga kali, baru setelah pesan terkirim secara sempurna koneksinya akan diputus.

```

22     print('Sending %s' % message)
23     sock.sendall(message.encode('utf-8'))

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS powershell + - + ... ^

```

echo_client(port)
File "D:\Mencoba\Dasar Python\cobaecho.py", line 17, in echo_client
    sock.connect(server_address)
ConnectionRefusedError: [WinError 10061] No connection could be made because the target machine actively refused it
PS D:\Mencoba\Dasar Python> py cobaecho.py --port 12345
Connecting to 192.168.10.8 port 12345
Sending Hello World!!! This will be echoed
Received: Hello World!!! T
Received: his will be echo
Received: ed
Closing connection to the server
PS D:\Mencoba\Dasar Python>

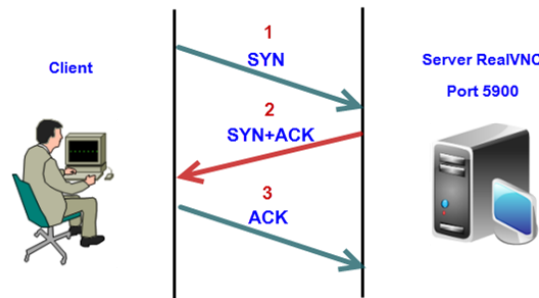
```

Jika dilihat dari hasil analisa wiresharknya

The Wireshark capture shows the following key packets:

- Packet 6: SYN from 192.168.10.8 to 192.168.10.7 on port 12345.
- Packet 7: SYN-ACK from 192.168.10.7 to 192.168.10.8 on port 50892.
- Packet 8: ACK from 192.168.10.8 to 192.168.10.7 on port 12345.
- Packet 9: SYN from 192.168.10.8 to 192.168.10.7 on port 12345.
- Packet 10: SYN-ACK from 192.168.10.7 to 192.168.10.8 on port 50892.
- Packet 11: ACK from 192.168.10.8 to 192.168.10.7 on port 12345.
- Packet 12: SYN from 192.168.10.8 to 192.168.10.7 on port 12345.
- Packet 13: SYN-ACK from 192.168.10.7 to 192.168.10.8 on port 50892.
- Packet 14: ACK from 192.168.10.8 to 192.168.10.7 on port 12345.

Dapat dilihat ada koneksi TCP antar IP 192.168.10.8 dan 192.168.10.7 yang mana masing-masing menggunakan port 12345 dan 50892. Disitu proses koneksi TCP tidak terjadi hanya satu kali tapi beberapa kali dan ada beberapa segment yang terjadi disitu. Ada segment SYN dan ACK. Karena TCP memiliki sifat connection-oriented maka akan ada koneksi antar dua perangkatnya yang mana yang pertama ada segment SYN untuk mengonfirmasi apakah client dan server bisa terhubung satu sama lain, lalu server berbalik mengonfirmasi dengan mengirimkan segment SYN atau synchronize dan ACK ke client. Lalu client balik mengirimkan ACK atau Acknowledge ke server bahwa koneksi berhasil dibuat. Nah tiga proses diawal itu bernama proses three way handshake, Dimana client meminta koneksi, dan dikonfirmasi, lalu koneksi pun terbuat dengan komunikasi silang tadi.



Lalu setelah ada proses three way handshake ada proses transfer data nya oleh client dengan PSH,ACK. PSH sendiri berarti Push Flag untuk segera memproses datanya tanpa menunggu buffer penuh. Lalu ada ACK lagi dari server untuk konfirmasi bahwa data telah diterima. Setelah data telah terkirim semua dan sudah ada konfirmasi dari server, maka proses terakhir adalah termination untuk mematikan koneksi, Dimana client akan mengirimkan FIN (Finish),ACK untuk memberi tau ke server bahwa client ingin menutup koneksi, dan server merespon dengan mengirimkan FIN,ACK juga, barulah koneksi kedua perangkat tertutup. Dari seluruh proses ini bisa diketahui bahwa protokol TCP adalah prokotoel dengan connection-oriented Dimana tiap proses pengiriman akan disertai dengan proses konfirmasi oleh komputer penerima ke komputer pengirim.

Lalu selain ada percobaan TCP ini dengan jalinan satu koneksi yang langsung tertutup, juga dilakukan percobaan TCP untuk mengirimkan data secara langsung dan koneksi antar laptop akan terus berjalan selagi client atau server tidak mematikan koneksinya secara manual.

Berikut program untuk servernya

```

import socket
import argparse
import threading

HOST = '192.168.100.92'
BUFFER_SIZE = 1024

def handle_client(client_socket, client_address):
    """Handles communication with a single client."""
    print(f"Terhubung dengan {client_address}")

    while True:
        try:
            data = client_socket.recv(BUFFER_SIZE)
            if not data:
                break

            message = data.decode('utf-8')
            print(f"Pesan dari {client_address}: {message}")

            if message.lower() == "exit":
                print(f"Client {client_address} meminta untuk mengakhiri koneksi.")
                client_socket.sendall("Koneksi ditutup oleh server.".encode('utf-8'))
                break

            client_socket.sendall(data) # Echo the message back to client

        except ConnectionResetError:
            print(f"Client {client_address} terputus secara paksa.")
            break

    client_socket.close()
    print(f"Koneksi dengan {client_address} ditutup.")

def echo_server(port):
    """Main server function to handle multiple clients."""
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    server_address = (HOST, port)
    sock.bind(server_address)
    sock.listen(5)

    print(f"Server berjalan di {HOST} port {port}...")

    while True:
        client_socket, client_address = sock.accept()

        # Create a new thread for each connected client
        client_thread = threading.Thread(target=handle_client, args=(client_socket, client_address))
        client_thread.start()

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Multi-Client Echo Server")
    parser.add_argument('--port', type=int, required=True, help="Port untuk server")
    args = parser.parse_args()
    echo_server(args.port)
  
```

Untuk program servernya pada pengaturan variabelnya hampir sama seperti sebelumnya tetapi disini terdapat tambahan fungsi `handle_client` untuk mencetak pesan-pesan yang dikirimkan oleh client secara langsung dan disini server akan terus mencetak pesan yang dikirimkan oleh client, selagi client tidak mematikan koneksinya. Di program itu juga tidak ada Batasan perangkat yang terhubung dalam satu waktu.

Lalu untuk program clientnya

```
import socket
import argparse

HOST = '192.168.100.92'

def echo_client(port):
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_address = (HOST, port)
    print(f"Menghubungkan ke {HOST} port {port}...")
    sock.connect(server_address)

    try:
        while True:
            message = input("Masukkan pesan: ")
            sock.sendall(message.encode('utf-8'))

            if message.lower() == "exit":
                print("Menutup koneksi ke server.")
                break

            data = sock.recv(1024)
            print(f"Server mengembalikan: {data.decode('utf-8')}")

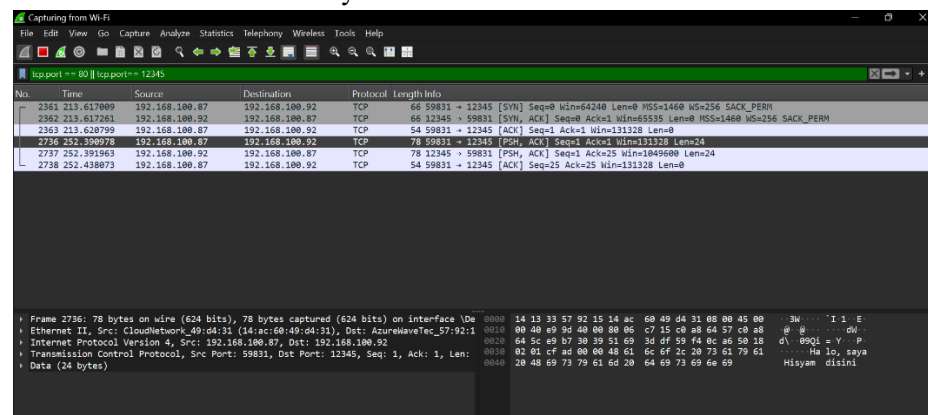
    except Exception as e:
        print(f"Terjadi kesalahan: {e}")

    finally:
        sock.close()

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Echo Client")
    parser.add_argument('--port', type=int, required=True)
    args = parser.parse_args()
    echo_client(args.port)
```

Untuk di program clientnya juga hampir sama seperti sebelumnya tetapi disitu ada tambahan `while true`, dimana client bisa terus memasukkan pesan ke server secara langsung tanpa dari kodingan saja jika koneksinya tidak diputus sendiri oleh clientnya.

Lalu untuk hasil wiresharknya



Hasilnya tentu sama seperti proses protokol TCP sebelumnya, karena protokol yang digunakan disini sama-sama TCP juga.

Selain adanya protokol TCP, terdapat juga protkol di transport layer yaitu UDP. UDP atau User Datagram Protocol adalah protokol yang bersifat connectionless atau tanpa koneksi. Jika TCP tadi tiap adanya pengiriman akan ada verifikasi antar satu komputer ke komputer pengirimnya, tetapi untuk protkol UDP ini tidak ada proses tersebut. Protokol UDP ini memungkinkan komputer pengirim akan mengirimkan sebuah datagram ke komputer penerima lalu langsung melupakannya, tanpa memerlukan adanya verifikasi dari penerima bahwa datagramnya terkirim dengan sempurna.

Jika di program dengan python untuk UDP bagian komputer servernya adalah

```
import socket
import sys
import argparse

host = 'localhost'
data_payload = 2048

def echo_server(port):
    # Membuat socket object menggunakan UDP
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    server_address = (host, port)

    print("Starting up echo server on %s port %s" % server_address)
    # Bind the socket, menentukan port yang digunakan pada host tujuan
    sock.bind(server_address)

    while True:
        # Menunggu menerima pesan dari client
        print("Waiting to receive message from client")
        data, address = sock.recvfrom(data_payload)
        print("received %s bytes from %s" % (len(data), address))
        print("Data: %s" % data.decode('utf-8'))

        if data:
            # Server mengirim pesan ke client
            sent = sock.sendto(data, address)
            print("sent %s bytes back to %s" % (sent, address))

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Socket Server Example')
    parser.add_argument('--port', action="store", dest="port", type=int, required=True)
    given_args = parser.parse_args()
    port = given_args.port
    echo_server(port)
```

Disini variabel-variabel yang digunakan hampir sama dan disini tidak ada batasan untuk berapa komputer yang bisa terkoneksi dalam satu waktu, selain itu dalam loop while true tidak ada proses fungsi accept() dimana server menerima koneksi dari client, dan juga tidak ada fungsi close() untuk menutup koneksi antar dua perangkat. Darisini sudah bisa diketahui bahwa UDP adalah protokol yang mengijinkan sebuah perangkat mengirim sebuah datagram tanpa harus ada proses membuka dan menutup koneksi

Lalu untuk program client nya

```
import socket
import sys
import argparse

host = 'localhost'
data_payload = 2048

def echo_client(port):
    # Membuat socket object menggunakan UDP
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    server_address = (host, port)

    print("Connecting to %s port %s" % server_address)
    message = 'This is the message. It will be repeated.'

    try:
        # Mengirim pesan ke server
        message = "Good Morning!!! This will be echoed"
        print("Sending %s" % message)
        sent = sock.sendto(message.encode('utf-8'), server_address)

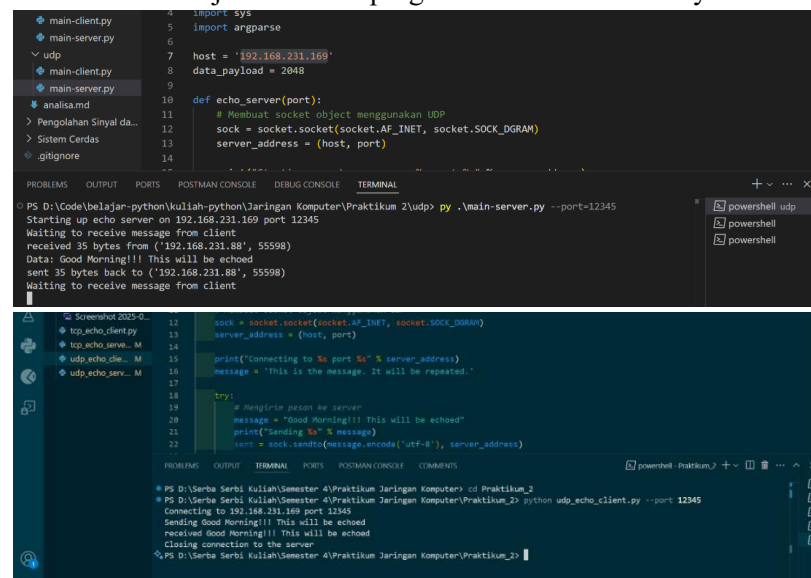
        # Menerima balasan dari server
        data, server = sock.recvfrom(data_payload)
        print("received %s" % data.decode('utf-8'))

    finally:
        print("Closing connection to the server")
        sock.close()

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Socket Client Example')
    parser.add_argument('--port', action="store", dest="port", type=int, required=True)
    given_args = parser.parse_args()
    port = given_args.port
    echo_client(port)
```

Di Program client ini juga sudah tidak ada fungsi connect() untuk server address lagi, karena di program diatas message string yang ada langsung dikirim dengan menggunakan fungsi sendto, sedangkan kalau di TCP tadi menggunakan fungsi sendall. Dan untuk variabel lainnya sama-sama disesuaikan dengan proses protokol UDP nya.

Lalu setelah menjalankan dua program diatas maka hasilnya adalah



```
4 import sys
5 import argparse
6
7 host = '192.168.231.169'
8 data_payload = 2048
9
10 def echo_server(port):
11     # Membuat socket object menggunakan UDP
12     sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
13     server_address = (host, port)
14
15     print("Connecting to %s port %s" % server_address)
16     message = 'This is the message. It will be repeated.'
17
18     try:
19         # Mengirim pesan ke server
20         message = "Good Morning!!! This will be echoed"
21         print("Sending %s" % message)
22         sent = sock.sendto(message.encode('utf-8'), server_address)
23
24         # Menerima balasan dari server
25         data, server = sock.recvfrom(data_payload)
26         print("received %s" % data.decode('utf-8'))
27
28     finally:
29         print("Closing connection to the server")
30         sock.close()
31
32 if __name__ == '__main__':
33     parser = argparse.ArgumentParser(description='Socket Server Example')
34     parser.add_argument('--port', action="store", dest="port", type=int, required=True)
35     given_args = parser.parse_args()
36     port = given_args.port
37     echo_server(port)
```

```
PS D:\Code\belajar-python\Kuliah-python\Jaringan Komputer\Praktikum 2> .\main-server.py --port=12345
Starting up echo server on 192.168.231.169 port 12345
Waiting to receive message from client
received 35 bytes from ('192.168.231.88', 55598)
Data: Good Morning!!! This will be echoed
Sent 35 bytes back to ('192.168.231.88', 55598)
Waiting to receive message from client
```

```
11 sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
12 server_address = (host, port)
13
14 print("Connecting to %s port %s" % server_address)
15 message = 'This is the message. It will be repeated.'
16
17 try:
18     # Mengirim pesan ke server
19     message = "Good Morning!!! This will be echoed"
20     print("Sending %s" % message)
21     sent = sock.sendto(message.encode('utf-8'), server_address)
22
23     # Menerima balasan dari server
24     data, server = sock.recvfrom(data_payload)
25     print("received %s" % data.decode('utf-8'))
26
27 finally:
28     print("Closing connection to the server")
29     sock.close()
30
31 if __name__ == '__main__':
32     parser = argparse.ArgumentParser(description='Socket Client Example')
33     parser.add_argument('--port', action="store", dest="port", type=int, required=True)
34     given_args = parser.parse_args()
35     port = given_args.port
36     echo_client(port)
```

```
PS D:\Serba Serbi Kuliah\Semester 4\Praktikum Jaringan Komputer> cd Praktikum_2
PS D:\Serba Serbi Kuliah\Semester 4\Praktikum Jaringan Komputer\Praktikum_2> python udp_echo_client.py --port 12345
Connecting to 192.168.231.169 port 12345
Sending Good Morning!!! This will be echoed
received Good Morning!!! This will be echoed
Closing connection to the server
PS D:\Serba Serbi Kuliah\Semester 4\Praktikum Jaringan Komputer\Praktikum_2>
```

Untuk hasilnya bisa dilihat di console client dibawah bahwasannya tidak ada konfirmasi 'received' dari server untuk tiap packet data yang dikirim (kalau protokol TCP). Setelah data dikirim, server langsung mengirimkan respon saja (hal ini dikarenakan programnya menginginkan server tetap mengirimkan konfirmasi ke client) dalam rangkaian string message utuh ke client.

Lalu untuk hasil wiresharknya adalah

No.	Time	Source	Destination	Protocol	Length	Info
2142	81.934234	142.251.175.95	192.168.231.88	UDP	70	443 → 51313 Len=28
2143	82.069220	142.251.175.95	192.168.231.88	UDP	71	443 → 51313 Len=29
2144	82.196632	192.168.231.88	192.168.231.169	UDP	77	55598 → 12345 Len=35
2145	82.375836	192.168.231.169	192.168.231.88	UDP	77	55598 → 55598 Len=35
2169	84.091935	192.168.231.88	192.168.231.254	DNS	92	Standard query 0x43a A mobile.events.data.microsoft.com
2170	84.143772	192.168.231.254	192.168.231.88	DNS	215	Standard query response 0x43a A mobile.events.data.microsoft.com CNAME mobile.events.data.tr
2280	84.708424	192.168.231.88	192.168.231.254	DNS	71	Standard query 0x87b5 A www.msn.com
2281	84.708749	192.168.231.88	192.168.231.254	DNS	71	Standard query 0x496c HTTPS www.msn.com
2287	84.818811	192.168.231.254	192.168.231.88	DNS	146	Standard query response 0x87b5 A www.msn.com CNAME www-msn-com.a-0003.a-msedge.net CNAME a-00
2366	85.272589	192.168.231.88	142.251.175.95	UDP	71	51313 → 443 Len=29
2368	85.323146	142.251.175.95	192.168.231.88	UDP	71	443 → 51313 Len=29
2382	85.589216	142.251.10.100	192.168.231.88	UDP	77	443 → 52454 Len=35
2383	85.601988	192.168.231.88	142.251.10.100	UDP	75	52454 → 443 Len=33

Frame 2145: 77 bytes on wire (616 bits), 77 bytes captured (616 bits) on interface De	Ethernet II, Src: Intel_95:c0:95 (f8:e4:e3:95:c0:95), Dst: AzureWaveTec_57:92:15 (14:1
Internet Protocol Version 4, Src: 192.168.231.169, Dst: 192.168.231.88	User Datagram Protocol, Src Port: 12345, Dst Port: 55598
Data (35 bytes)	

Diatas bisa dilihat bahwa proses UDP hanya ada dua saja. Jikalau dibedah yang pertama itu adalah proses pengiriman datagram dari client ke Server. Lalu UDP yang kedua adalah pengiriman datagram balik ke client sebagai tanda konfirmasi bahwa datagram telah diterima oleh server.

Itu perbedaan percobaan protokol TCP dan UDP. Yang mana memiliki perbedaan dari sifat koneksinya. Jikalau TCP yang bersifat connection-oriented cocok untuk mengirimkan data-data penting yang tidak bisa hilang satu pun di perjalanan koneksinya dengan seeptri data halaman web, sedangkan untuk UDP adalah pengiriman datagram tanpa koneksi yang mana hasilnya prosesnya pengirimannya lebih ringkas dan hasilnya data yang dikirim lebih cepat tanpa perlu mempertimbangkan tiap datagramnya dikirim dengan baik atau tidak, dan ini cocok untuk proses layanan streaming video yang tidak perlu selruuh data bisa dikirim dan benar-benar diterima oleh tiap penerimanya, karena penggunaanya yang bersifat kontinu.

6. Tugas

TCP

a. Jelaskan beberapa keunggulan protokol TCP.

Jawab:

Berikut beberapa keunggulan dari protokol TCP:

- TCP yang bersifat connection-oriented lebih bisa menjamin bahwa data sampai ke tujuan dengan benar, karena tiap proses mulai dari koneksi, pengiriman data, dan penutupan koneksi selalu ada verifikasi dari penerimanya
- Data yang dikirimkan juga berurutan karena TCP menggunakan sequence number yang mana memastikan data nanti durutkan agar TCP bisa menyusunnya kembali nanti di penerima sebelum diteruskan di bagian aplikasi
- TCP juga lebih bisa mengontrol pengiriman datanya saat ada jaringan macet, karena walaupun jaringan macet, TCP akan menunggu saat koneksi sudah siap untuk dikirimkan dan penerima sudah memverifikasi data dikirmkan dengan benar

- b. Jelaskan server socket method dan client socket method pada socket programming menggunakan Python.

Jawab:

Dalam program yang sudah dibuat ada beberapa fungsi yang digunakan dari library socket untuk tiap-tiap bagian komputer server ataupun komputer client. Di bagian server tadi metode seperti `socket()` untuk membuat objek socket, `bind(address)` untuk mengikat alamat IP dan port, serta `listen(backlog)` yang memungkinkan server menunggu koneksi dari client. Saat ada client yang mencoba terhubung, server menggunakan `accept()` untuk menerima koneksi, yang kemudian mengembalikan objek socket baru serta alamat client. Setelah terhubung, server dapat menerima data menggunakan `recv(buffer size)` dan mengirim balasan dengan `send(data)` atau `sendto(data)`. Setelah komunikasi selesai, koneksi dapat ditutup menggunakan `close()`.

Lalu di bagian client juga ada metode `socket()` untuk membuat socket baru, lalu untuk menghubungkan diri ke server pada alamat IP menggunakan `connect(address)`, serta `send(data)` atau `sendto(data)` untuk mengirim data. Client kemudian dapat menerima respon dari server menggunakan `recv(buffer size)` dan menutup koneksi dengan `close()`.

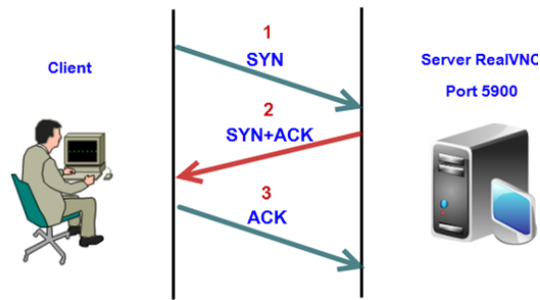
Tentunya di proses tersebut bagian server harus berjalan terlebih dahulu sebelum client agar client bisa menghubungkan dua perangkat itu untuk berkomunikasi dan dapat mengirimkan data ke server yang kemudian di proses dan dibalas oleh server. Lalu dalam prosesnya tentu ada dua protokol yang bisa digunakan yaitu TCP dan UDP. Perbedaan utama antara TCP dan UDP dalam implementasi socket adalah bahwa TCP menggunakan metode yang menjamin keandalan komunikasi, seperti tiga langkah handshake sebelum koneksi terbentuk dan pengiriman ulang data jika terjadi kehilangan, sementara UDP lebih sederhana dan cepat karena tidak memerlukan koneksi sebelum mengirim data serta tidak memiliki mekanisme pengiriman ulang jika paket hilang.

- c. Jelaskan yang anda pahami mengenai mekanisme three-way handshake.

Jawab:

Mekanisme three-way handshake adalah mekanisme koneksi antar dua perangkat server dan client yang menggunakan protokol TCP. Karena namanya three-way, maka dari itu ada tiga proses yang berjalan berurutan untuk mengaktifkan koneksi antar dua perangkat tersebut. Tiga prosesnya adalah

1. Segment SYN atau synchronize yang mana client akan meminta koneksi ke server yang aman disini paket yang dikirim berisi nomor urut awal dari sequence number yang akan digunakan dalam komunikasi
2. Lalu ada segment SYN,ACK atau synchronize-acknowledge yang mana server merespons dari permintaan server tersebut yang mana didalam proses nya mengandung nomor urut nya sendiri dri sequence number
3. Lalu yang terakhir adalah segment ACK atau acknowledge yang mana client mengirimkan ack sebagai konfirmasi bahwa telah diterima SYN,ACK dari server dan koneksi pun terbentuk antara dua perangkat tersebut



Mekanisme ini bertujuan untuk memastikan agar dua perangkat tadi benar-benar siap berkomunikasi, menyinkronkan nomor urut paket data nya dan menghindari kesalahan-kesalahan pengiriman seperti pengiriman ulang paket lama.

UDP

d. Jelaskan beberapa keunggulan protokol UDP.

Jawab:

Berikut beberapa keunggulan UDP:

- Protokol UDP memiliki mekanisme pengiriman yang lebih cepat dan ringkas jika dibandingkan dengan TCP, karena tidak perlu adanya koneksi.
 - Memiliki latensi yang rendah sehingga cocok untuk komunikasi real-time yang mana tidak membutuhkan seluruh data terkumpul atau lebih baik sedikit kehilangan data daripada delay yang lama untuk menunggu data terkirim semuanya, misalnya seperti aplikasi streaming video.
 - Tidak memerlukan koneksi untuk mengirimkan data ke penerima sehingga lebih fleksibel untuk mengirim datanya dan cocok untuk sistem broadcast.
- e. Sebutkan beberapa contoh penerapan protokol UDP pada pengiriman data di jaringan internet, serta jelaskan mengapa aplikasi tersebut menggunakan UDP.

Jawab:

Ada beberapa contoh penerapan protokol UDP yang cocok dengan karakteristik connectionlessnya yaitu:

- Streaming Video dan Audio seperti Youtube dan Netflix. Kedua aplikasi tersebut akan menggunakan protokol UDP untuk mengirimkan data-data video audio dengan cepat dan latensi rendah agar penerima bisa melihat atau mendengarkan dengan secara real-time paket-paket data audio dan videonya. Walaupun mungkin ada paket data hilang, tetapi tidak terlalu berdampak secara signifikan
- Video call seperti Zoom dan Googlemeet, kedua aplikasi ini tentu menggunakan protokol UDP karena cepat untuk mengirmkan data video nya secara realtime. Karena jika menggunakan TCP maka video calnya akan banyak delay karena tiap paket datanya harus dikirim secara benar dan jika hilang harus dikiri ulang lagi, tetapi dengan UDP, data nya bisa diabaikan dengan tentunya batas tolerasni dari pengguna layanan ini agar percakapan tetpa berjalan tanpa adanya delay yang lama.
- DNS atau Domain Name System juga menggunakan UDP untuk mempercepat proses pencarian alamat IP dari nama domain. Karena permintaan DNS hanya membutuhkan pertukaran data kecil antara client dan server, penggunaan TCP akan memperlambat waktu respon. Dengan UDP, query dapat dikirim dan dijawab dalam waktu sangat singkat.

f. Jelaskan perbedaan TCP dan UDP.

Jawab:

Berikut perbedaan antara TCP dan UDP jika dilihat di tiap-tiap aspeknya:

- Koneksi: TCP menggunakan koneksi terjamin (connection-oriented) dengan proses handshaking sebelum data dikirim, sedangkan UDP connectionless, langsung mengirim data tanpa membangun koneksi.
- Keandalan: TCP menjamin data diterima dengan benar melalui acknowledgment dan retransmission, sedangkan UDP tidak menjamin keandalan atau urutan data.
- Kecepatan: TCP lebih lambat karena memiliki mekanisme kontrol, sedangkan UDP lebih cepat karena tidak ada pengendalian koneksi.
- Pengendalian Aliran & Kesalahan: TCP memiliki kontrol aliran dan deteksi kesalahan yang memastikan data sampai dengan benar, sedangkan UDP tidak memiliki mekanisme ini.
- Penggunaan Aplikasi: TCP digunakan untuk aplikasi yang membutuhkan keandalan seperti web browsing, email, dan transfer file, sedangkan UDP digunakan untuk aplikasi yang membutuhkan kecepatan dan latensi rendah seperti streaming, gaming, dan VoIP.
- Urutan Data: TCP memastikan data diterima berurutan, sedangkan UDP tidak menjamin urutan paket yang diterima.
- Overhead & Bandwidth: TCP memiliki overhead lebih besar karena kontrol koneksi dan retransmission, sedangkan UDP lebih efisien dengan overhead lebih kecil.
- Latensi: TCP memiliki latensi lebih tinggi karena proses handshaking dan kontrol aliran, sedangkan UDP memiliki latensi lebih rendah karena tidak ada pengendalian koneksi atau retransmission.

7. Kesimpulan

Secara keseluruhan, TCP (Transmission Control Protocol) dan UDP (User Datagram Protocol) merupakan dua protokol di layer transport yang memiliki karakteristik berbeda dalam hal koneksi, keandalan, dan kecepatan pengiriman data. TCP bersifat connection-oriented, di mana terdapat proses three-way handshake sebelum data dikirim, serta mekanisme acknowledgment dan retransmission yang memastikan data tiba dengan benar dan berurutan. Hal ini membuat TCP sangat cocok untuk aplikasi yang menuntut keandalan tinggi, seperti transfer file, email, dan web browsing.

Di sisi lain, UDP bersifat connectionless, artinya data dapat dikirim langsung tanpa proses pembentukan koneksi. UDP tidak menyediakan mekanisme acknowledgment maupun retransmission, sehingga pengiriman data berlangsung lebih cepat dengan latensi rendah. Meskipun tidak menjamin keutuhan atau urutan data, UDP sangat sesuai untuk aplikasi yang memprioritaskan kecepatan dan real-time, seperti streaming video, VoIP, dan game online. Dengan demikian, pemilihan antara TCP dan UDP bergantung pada kebutuhan aplikasi dan tingkat keandalan yang diinginkan.

8. Lampiran Laporan Sementara

2 Teknik Komputer A

Laporan Sementara Praktikum 2

Mama Anggota kelompok:

- 1). Fransisca Majwa Putri Wibowo
(3223600003)
- 2). Rifqi Raehan Hermawan
(3223600004)
- 3). Muhammad Alfarrel Arya Mahardika
(3223600017)


6/3/21

* Pengujian TCP Echo Server

	PC / Laptop A	PC / Laptop B
Server / Client	Client	Server
IP address	(WiFi) 192.168.231.119 (Ethernet) 192.168.10.7	(WiFi) 192.168.231.119 (Ethernet) 192.168.10.8
Port number	61697 (WiFi) 50892 (Eth)	12345 (WiFi) 12345 (Eth)

* Pengujian pengiriman pesan dengan TCP Echo Server

a). WiFi :

• Client

Py .\main-client.py --port=12345
Connecting to 192.168.231.119 port 12345 ✓
Sending Hello World !!! This will be echoed
Received : Hello World !!! T
Received : ed
Closing connection to the server

• Server

python server.py --port=12345
Starting up echo server on 192.168.231.119 port 12345 ✓
Waiting to receive message from client
Data : Hello World !!! This will be echoed
Sent Hello World !!! This will be echoed bytes back to
(192.168.231.119, 61697) ✓

b). Ethernet :

• Client

Py cbaecho.py --port 12345
Connecting to 192.168.10.8 port 12345 ✓
Sending Hello World !! This will be echoed
Received : Hello World !! T
Received : his will be echo
Received : ed
Closing connection to the server

• Server

Py tcp-echo-server.py --port 12345 ✓
Starting up echo server on 192.168.10.8 port 12345
Waiting to receive message from client
Data : Hello World !!! This will be echoed
Sent Hello World !!! This will be echoed
bytes back to (192.168.10.7, 50892)
waiting to receive message from client i

* Pengujian UDP Echo Server

	PC/Laptop A	PC/Laptop B
Server/Client	Client	Server
IP address	(WIFI) 192.168.231.88	(WIFI) 192.168.231.169
	(Ethernet) 192.168.10.7	(Ethernet) 192.168.10.8
Port number	55598 (WIFI)	12345 (WIFI)
	60916 (Eth)	12345 (Eth)

P
6/3/25

* Pengujian pengiriman pesan dengan UDP Echo server

a). Ethernet :

• Client

Py udp_echo_client.py --port 12345 ;
Connecting to 192.168.10.8 port 12345 ✓
Sending Good Morning!!! This will be echoed
received Good Morning!!! This will be echoed
Closing connection to the server

• Server

Py udp_echo_server.py --port 12345 ✓
Starting up echo server on 192.168.10.8 port 12345
Waiting to receive message from client
received 35 bytes from ('192.168.10.7', 60916) ✓
Data: Good Morning!!! This will be echoed
sent 35 bytes back to ('192.168.10.7', 60916)
Waiting to receive message from client ✓

b). WIFI :

• Client

Python udp_echo_client.py --port 12345
Connecting to 192.168.231.169 port 12345 ✓
Sending Good Morning!!! This will be echoed
received Good Morning!!! This will be echoed
Closing connection to the server

• Server

Py \main-server.py --port=12345
Starting up echo server on 192.168.231.169 port 12345 ✓
Waiting to receive message from client
received 35 bytes from ('192.168.231.88', 55598)
Data: Good Morning!!! This will be echoed
sent 35 bytes back to ('192.168.231.88', 55598) ✓
waiting to receive message from client

