Nama : Alvin Zanua Putra

NRP : 5025231064

Kelas : Pemrograman Jaringan – D

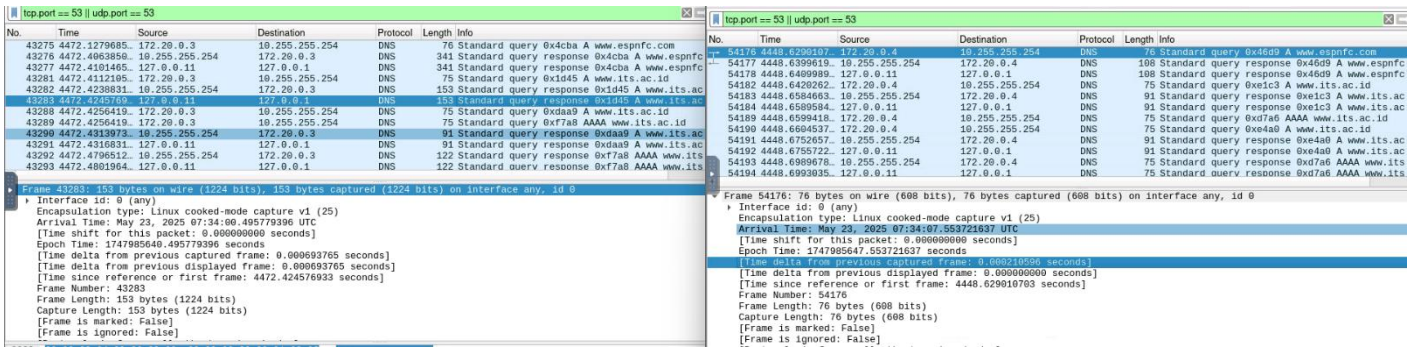
TUGAS 1

NOMOR 1 Langkah Pengerjaan :

a) Akses Mesin 1 dan Mesin 2 melalui IP 60001 dan 60002 dan Akses Juga Wireshark pada masing-masing Mesin 1 dan Mesin 2 melalui IP 50001 dan 50002.
b.) Lalu, hapus komentar pemanggilan fungsi dalam program yaitu pada `get_remote_info()` dan `get_my_socket() ` untuk melihat info remote dan apa nama socketnya, kemudian disusul dengan modifikasi pada domain www.espnfc.com yang typo cosm, dan modofikasi program untuk mengetahui info dari kedua domain saya modif ada dua fungsi get_remote info 1 dan 2.

```
def get_remote_info1():
    remote_host = 'www.espnfc.com'
    try:
        remote_host_ip = socket.gethostbyname(remote_host)
        print(f"ip address dari {remote_host} adalah {remote_host_ip}")
    except Exception as ee:
        print(f"ERROR : {str(ee)}")
def get_remote_info2():
    remote_host = 'www.its.ac.id'
    try:
        remote_host_ip = socket.gethostbyname(remote_host)
        print(f"ip address dari {remote_host} adalah {remote_host_ip}")
    except Exception as ee:
        print(f"ERROR : {str(ee)}")
if __name__=='__main__':
    get_my_info()
    get_remote_info1()
    get_remote_info2()
    get_my_socket()
```

c) Jalankan program socket_info.py di mesin 1 dan mesin 2 dengan command : `python3 socket_info.py`. lalu analisis pada wireshark pilih interface `eth 0` karena ada traffic yang berjalan dan kita hanya deteksi komunikasi dalam 1 mesin saja yaitu per mesin 1 dan mesin 2.



Analisis :

Ketika program dijalankan, ada dua aktivitas utama yang terjadi: **Pertama**, program mencoba terhubung ke situs web **www.espnfc.com**. Saat ini terjadi, program membuat koneksi ke internet menggunakan protokol **TCP**, lalu menerima balasan dari situs tersebut. Artinya, program mengirim permintaan ke server ESPN

dan mendapat respons kembali—ini adalah proses komunikasi normal saat mengakses situs web. **Kedua**, sebelum program bisa terhubung ke situs web tadi, ia harus mencari tahu alamat IP dari nama situs tersebut. Ini dilakukan lewat proses yang disebut **DNS lookup**. Dalam proses ini, program menghubungi server DNS milik kampus **www.its.ac.id** melalui port 53 (port khusus untuk DNS). Program ini mencari dua jenis informasi: alamat IP versi 4 (A record) dan versi 6 (AAAA record). Komunikasi ini terjadi antara alamat IP lokal komputer (misalnya 172.18.0.4 untuk mesin pertama) dan alamat DNS (10.255.255.254). Mesin kedua mlakukan hal yang sama, hanya alamat IP-nya berbeda (172.18.0.3). Untuk melihat proses DNS ini di Wireshark, digunakan filter khusus untuk melihat hanya lalu lintas yang lewat port 53. Ini karena fungsi socket.getaddrinfo() yang digunakan dalam program secara otomatis meminta DNS untuk menerjemahkan nama situs menjadi alamat IP.Kesimpulannya, dari tangkapan di Wireshark terlihat dua proses penting: pertama program terhubung ke internet, dan kedua program meminta bantuan DNS untuk mencari alamat situs.

NOMOR 2. Langkah-langkah pengerjaan :

a) Pertama ubah server_address pada file `client.py` pada mesin2 dari `localhost` menjadi `172.16.16.101`.

 # Connect the socket to the port where the server is listening

server_address = ('172.16.16.101', 10000)

b) Jalankan program dengan command `python3 server.py` pada mesin1 dan `python3 client.py` pada mesin2.

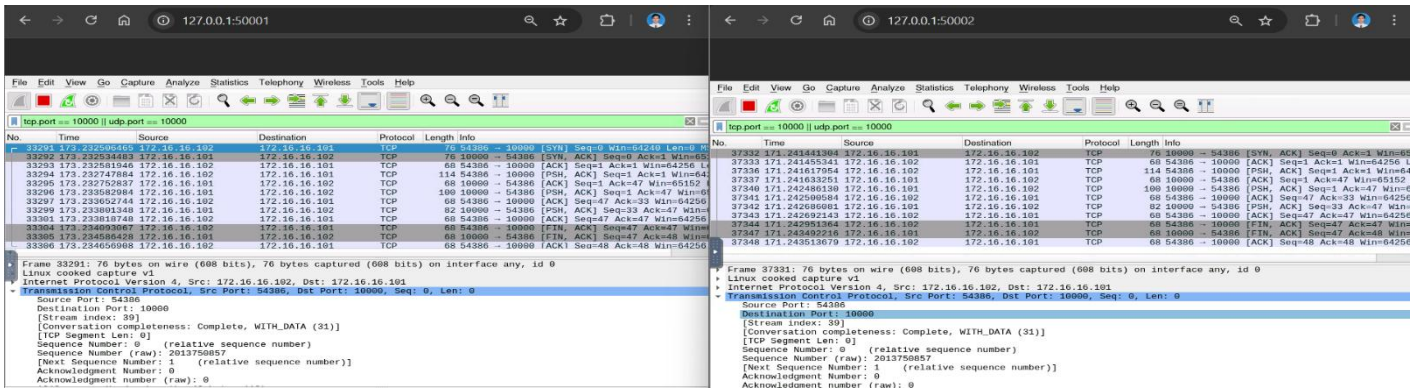
```
(base) jovyan@2117a5c5c2c4:~/work/progjar/progjar1$ python3 server.py
INFO:root:starting up on ('0.0.0.0', 10000)
INFO:root:waiting for a connection
INFO:root:connection from ('172.16.16.102', 34254)
INFO:root:received b'INI ADALAH DATA YANG DIKIRIM ABC'
INFO:root:sending back data
INFO:root:received b'DEFGHIJKLMNOPQ'
INFO:root:sending back data
INFO:root:received b''
INFO:root:waiting for a connection
```

```
(base) jovyan@79666eea79c0:~/work/progjar/progjar1$ python3 client.py
INFO:root:connecting to ('172.16.16.101', 10000)
INFO:root:sending INI ADALAH DATA YANG DIKIRIM ABCDEFGHIJKLMNOPQ
INFO:root:b'INI ADALAH DATA '
INFO:root:b'YANG DIKIRIM ABC'
INFO:root:b'DEFGHIJKLMNOPQ'
INFO:root:closing
(base) jovyan@79666eea79c0:~/work/progjar/progjar1$
```

c) Melakukan analisis dari wireshark yang hasilnya :

Analisis pada wireshark pilih interface `any` karena ada traffic yang berjalan dan kita mendeteksi komunikasi dalam beberapa mesin untuk melakukan komunikasi antar jaringan.

Karena pada mesin 1 dan mesin 2 port nya adalah 10000 maka untuk filter wireshark kita menggunakan filter *tcp.port == 10000 | udp.port = 10000*



Analisis :

Dalam percobaan ini, terdapat dua mesin yang menjalankan peran berbeda: mesin pertama bertindak sebagai server, dan mesin kedua sebagai client. Server dijalankan di mesin 1 dan dikonfigurasi untuk mendengarkan (listen) koneksi pada semua alamat IP (0.0.0.0) melalui port 10000. Ini berarti server siap menerima koneksi dari alamat IP mana pun yang mencoba terhubung ke port tersebut. Sementara itu, client dijalankan di mesin 2 dan ditugaskan untuk mengirimkan data ke server dengan menggunakan protokol TCP. Data yang dikirim adalah dalam bentuk byte dan berisi pesan "INI ADALAH DATA YANG DIKIRIM ABCDEFGHIJKLMNOPQ". Proses pengiriman dilakukan dengan perintah sendall() yang menjamin seluruh data terkirim tanpa terputus.

Dari hasil tangkapan Wireshark di kedua mesin (127.0.0.1:50001 dan 127.0.0.1:50002), terlihat jelas proses tiga langkah handshake TCP yang terjadi sebelum data dikirim, yaitu SYN, SYN-ACK, dan ACK. Setelah koneksi berhasil dibentuk, client mulai mengirim data, dan server membalas dengan mengirimkan kembali data tersebut ke client. Karena TCP merupakan protokol berbasis stream, data yang diterima oleh client tidak selalu langsung utuh, melainkan bisa terpecah

menjadi beberapa bagian kecil tergantung pada kondisi jaringan dan buffer sistem. Inilah mengapa client mencetak bagian data seperti "INI ADALAH DATA" terlebih dahulu, kemudian dilanjutkan dengan "YANG DIKIRIM ABC" dan seterusnya. Proses komunikasi ini berakhir ketika client menutup koneksi, dan hal ini dikenali oleh server saat menerima sinyal berupa empty byte (b'') yang menunjukkan bahwa tidak ada lagi data yang akan dikirim.

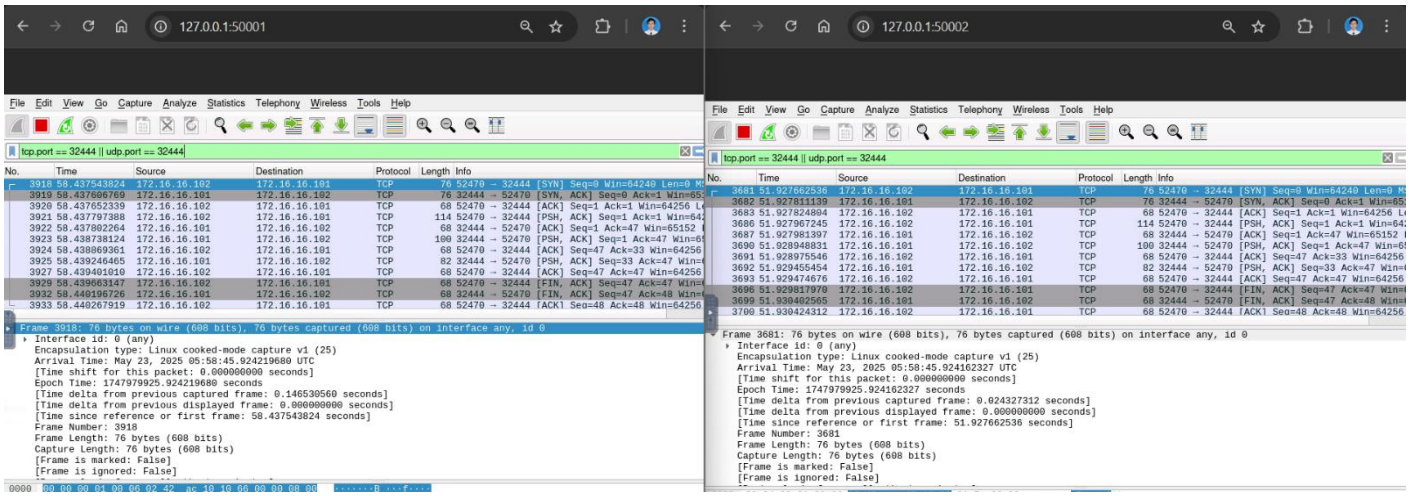
NOMOR 3. Langkah - langkah pengerjaan :

a) Ubah port di file `server.py` menjadi `32444` di mesin-1 dan Ubah port di file `client.py` menjadi `32444` di mesin-2 :
 # Connect the socket to the port where the server is listening
server_address = ('172.16.16.101', 10000)

b) Jalankan program `server.py` di mesin-1 dan `client.py` di mesin 2. Kemudian lakukan analisis di wireshark.

Analisis pada wireshark pilih interface `any` karena ada traffic yang berjalan dan kita mendeteksi komunikasi dalam beberapa mesin untuk melakukan komunikasi antar jaringan.

Karena pada mesin 1 dan mesin 2 port nya adalah 32444 maka untuk filter wireshark kita menggunakan filter *tcp.port == 32444 | udp.port = 32444*



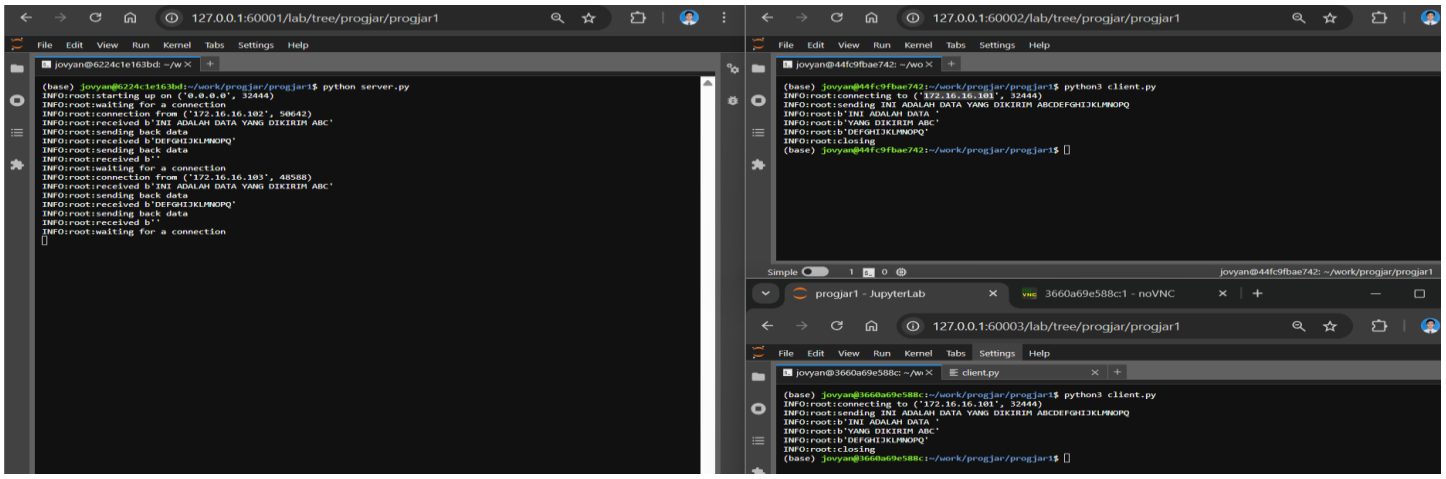
Analisis :

Kali ini keduanya menggunakan port 32444 untuk berkomunikasi. Server dijalankan di mesin dengan IP 172.16.16.101 dan dikonfigurasi untuk mendengarkan koneksi pada port 32444. Client dijalankan di mesin dengan IP 172.16.16.102, yang menginisiasi koneksi TCP ke server dengan tujuan IP 172.16.16.101 port 32444. Proses Komunikasi: Inisialisasi **Koneksi (3-Way Handshake)** Berdasarkan dari Wireshark, komunikasi dimulai dengan client (172.16.16.102) mengirimkan segmen SYN ke server sebagai permintaan untuk membuka koneksi. Server membalas dengan SYN-ACK, dan client menanggapi dengan ACK yaitu tiga langkah awal dalam pembentukan koneksi TCP oleh karena itu kita sebut 3 jalan handshake. Setelah koneksi berhasil dibentuk, client mengirimkan data ke server menggunakan metode sendall() dalam format byte. Paket TCP yang mengandung data ini ditandai dengan flag PSH, ACK. Ini menunjukkan bahwa data didorong ke penerima tanpa menunggu buffer penuh (push function), yang sesuai dengan karakteristik protokol TCP berbasis stream. Respons dikirimkan dalam potongan-potongan (chunks) yang tergantung pada buffer sistem dan kondisi jaringan. Setelah seluruh data berhasil dikirim dan diterima, client mengakhiri koneksi dengan mengirimkan segmen FIN. Server membalas dengan ACK, lalu mengirimkan FIN sebagai balasan untuk menutup koneksi dari sisi server. Client kemudian mengirimkan ACK terakhir untuk menyelesaikan proses terminasi koneksi TCP secara normal. Dari analisis Wireshark di kedua sisi, terlihat bahwa semua tahapan komunikasi TCP—mulai dari handshake, pengiriman data, hingga terminasi koneksi—berjalan dengan benar dan sesuai standar. Port yang digunakan (32444) tidak memengaruhi mekanisme dasar TCP, hanya berfungsi sebagai pengenal koneksi.

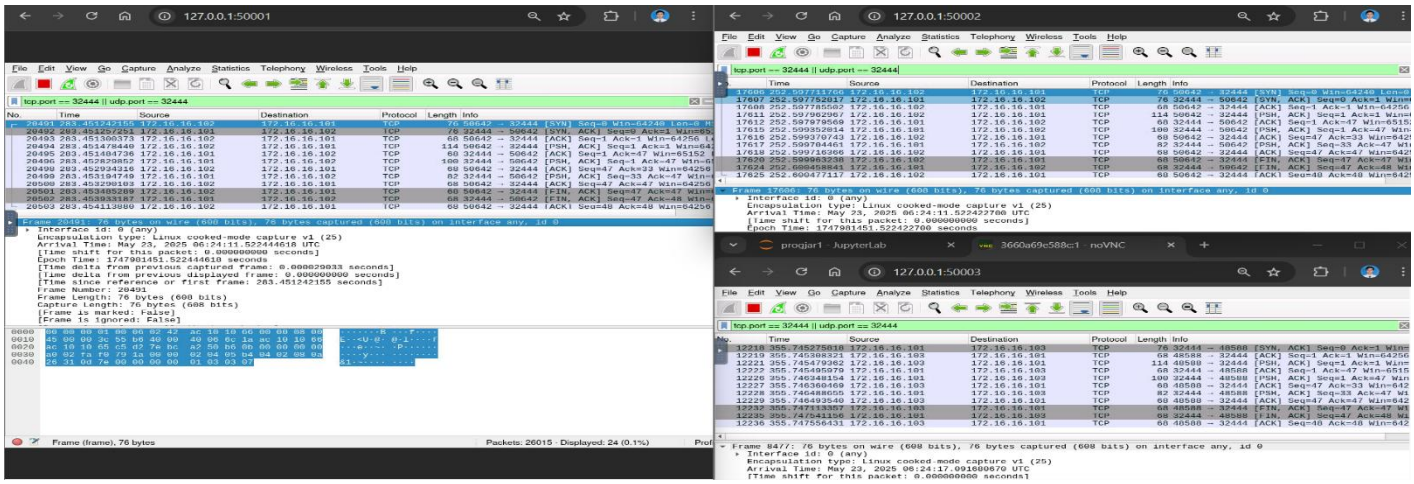
NOMOR 4. Langkah-langkah pengerjaan :

a) Ubah port pada `client.py` di mesin-3 menjadi `32444` dan ubah server address dari `localhost` menjadi `172.16.16.101` :

b) Jalankan program server.py di mesin 1 dan client.py di mesin 2 dan mesin 3 :



c) Lakukan analisis pada wireshark dengan hasil :Pada analisis wireshark diatas jangan lupa pilih interface `any` karena ada traffic yang berjalan dan kita mendeteksi komunikasi dalam beberapa mesin untuk melakukan **ko**munikasi antar jaringan. Karena pada mesin 1 dan mesin 2 serta mesin 3 port nya adalah 32444 maka untuk filter wireshark kita menggunakan filter *tcp.port == 32444 | udp.port = 32444*



Analisis :

Pada kasus ini ketambahan 1 mesin yaitu mesin 3 yang berjalan di port 50003 dengan program client,py di port 32444 juga yaitu terdapat tiga mesin yang terlibat: mesin-1 sebagai server, serta mesin-2 dan mesin-3 sebagai client yang mencoba terhubung ke server melalui port 32444. Meskipun kedua client dijalankan hampir secara bersamaan, hasil tangkapan paket pada Wireshark menunjukkan bahwa server hanya menangani satu koneksi dalam satu waktu secara berurutan. Awalnya, mesin-2 mengirimkan paket SYN ke mesin-1 untuk memulai proses handshake TCP. Server membalas dengan SYN-ACK dan kemudian menerima ACK dari mesin-2, menyelesaikan tiga langkah handshake. Setelah itu, data ditransfer dan sesi ditutup dengan pertukaran FIN dan ACK. Hanya setelah koneksi dari mesin-2 benar-benar selesai dan ditutup, server mulai merespons koneksi dari mesin-3. Mesin-3 kemudian mengulangi proses yang sama: memulai handshake, bertukar data, lalu mengakhiri koneksi. Hal ini menandakan bahwa server hanya menggunakan satu soket yang aktif dalam menerima koneksi secara bergantian menggunakan metode accept(), sehingga koneksi dari mesin-3 harus menunggu hingga koneksi dari mesin-2 selesai. Akibatnya, urutan paket di Wireshark terlihat sekuensial, meskipun secara teknis kedua client mencoba terhubung secara paralel. Fenomena ini adalah ciri dari server yang tidak menggunakan teknik concurrency (seperti threading atau asynchronous I/O), yang menyebabkan pemrosesan koneksi dilakukan satu per satu.