בס"ד

<u>Final project</u>

Matan Adar, ID – 209321553

Lior Davidyan, ID - 213861529

About: This is our final project in the Communication networks course , in this task we were required to build an http app which will operate as a redirect to another server in order to download a file. The app supports two transport layers protocols , the TCP protocol and the Reliable UDP protocol. Moreover, we were also required to simulate a communication between a client, DNS and DHCP servers.

<u>Note: please check the code in UBUNTU</u>

<u>Note: when you want to run code for one part (like DHCP or DNS and etc), which in the main of client, make sure to delete the hash mark before planning execute the function in order to operate the functions well and carefully executed the test.</u>

<u>לשים לב שכאשר רוצים להפעיל פונקציה שנמצאת במיין של הלקוח, צריך למחוק את הסולמית שלפני קריאה הפונקציה עמ"נ שנוכל להפעיל את הפונקציות ולבצע את הבדיקה לאותו נושא (למשל, TCP ואז UDP וכו)</u>

Table of contents:

Code documantaion :

In our code we have used a lot the scapy library in python in DHCP and DNS parts.

Scapy : Scapy is a Python program that enables the user to send, sniff and dissect and forge network packets. This capability allows construction of tools that can probe, scan or attack networks.

In TCP and reliable UDP we used the socket library, and we used requests library to making the photo from the URL.

DHCP

Overview

When a device connects to a network, it doesn't automatically know its IP address, subnet mask, and other network configuration settings. That's where DHCP comes in.

So how does it really works ?

DHCP Discover: When the client joins the network, it broadcasts a DHCP Discover message, asking for an IP address and other network configuration information.

DHCP Offer: DHCP servers on the network receive the DHCP Discover message and respond with a DHCP Offer message. The DHCP Offer message contains an available IP address and other network configuration information, such as subnet mask, default gateway, and DNS server address.

DHCP Request: The client receives the DHCP Offer message and selects one of the offered IP addresses. The client then broadcasts a DHCP Request message, requesting the selected IP address.

DHCP Acknowledge: The DHCP server receives the DHCP Request message and reserves the selected IP address for the client. The server then sends a DHCP Acknowledge message to the client, confirming the allocation of the IP address and other network configuration information.

Configuration: The client receives the DHCP Acknowledge message and configures its network settings based on the information received from the DHCP server. The client then uses the assigned IP address and other network configuration information to communicate on the network.

In the next page we will explain on the code how we implement that DHCP client and server.

בס"ד

Explaining on the code:

DHCP SERVER

The get_unused_ip() function purpose is to return an unused Ip to the client. We are scanning our collection of Ip's and checking whether there is an unused Ip , if there is one we are offering it to client , otherwise we are offering to the client the 0.0.0.0 Ip(that its mean that the DHCP server don’t have more Ip's to give).

```python
# checking if the ip is already in used or if there is no more ips available
MatanAdar
def get_unused_ip():
    """
    Returns an unused IP address for a client to use.
    """
    #End_IP - find what number in the end of the ip can we append to the start of that ip that unused already
    # ips_in_used - array of all the ips that already in used
    global END_IP, ips_in_used

    # Check if there are any more IPs available to use
    # if there is no more ips to use we make the ip to "0.0.0.0"
    if END_IP == 255:
        print("Cant make more ips")
        return "0.0.0.0"

    # Find an unused IP address
    while True:
        ip = f"10.0.2.{END_IP}"
        if ip not in ips_in_used:
            END_IP += 1
            print(f"Found that ip {ip} is not in use")
            print(f"Offering the client this ip: {ip}")
            return ip
        else:
            print("the ip:", ip, "in use already")
            END_IP += 1
```

The got_dhcp_discover() purpose is to handle DHCP Broadcasts messages , we are sniffing requests on port 67 (which is the port for DHCP requests) using the sniff() function from the Scapy library. If the packet we captured is indeed a DHCP discovery message, a new Ip address is assigned to the client by calling the get_unused_ip() function. After we found an Ip address to give to the client we craft a DHCP offer msg and sent to the client by using the sendp() function.

```
def got_dhcp_discover():

    global END_IP, ips_in_used

    pkt = sniff(filter="udp and port 67", count=1, iface="enp0s3")[0]

    if DHCP in pkt and pkt[DHCP].options[0][1] == 1:
        print("DHCP Discover received")

        # we found an unused ip to give to the client in the help func 'get_unused_ip()'
        global offer_client_ip
        offer_client_ip = get_unused_ip()

        # Craft DHCP Offer and offer the client ip to the client that the server find
        dhcp_offer1 = Ether(dst="ff:ff:ff:ff:ff:ff", src=get_if_hwaddr("enp0s3")) / \
                      IP(src="0.0.0.0", dst="255.255.255.255") / \
                      UDP(sport=67, dport=68) / \
                      BOOTP(op=2, yiaddr=offer_client_ip, siaddr="10.0.0.1", giaddr="0.0.0.0", chaddr=pkt[Ether].src, xid=pkt[BOOTP].xid) / \
                      DHCP(options=[("message-type", "offer"),
                                    ("subnet_mask", "255.255.255.0"),
                                    ("router", "10.0.0.1"),
                                    ("name_server", "10.0.0.1"),
                                    "end"])

        time.sleep(1)

        print("sending dhcp offer to client")
        # Send DHCP Offer to the client
        sendp(dhcp_offer1)
```

the dhcp_ack() function puprpose is to handle the forth and last part in the DHCP process which is to send an ack message to the client with the Ip address which the client accepted when he got offered . The function captures the first UDP packet on port 67 (the DHCP server port) using sniff() function. If the packet is a DHCP request message, the server sends a DHCP ACK message back to the client using the information from the request message.

The ACK message contains the assigned IP address and other network configuration options such as the subnet mask, default gateway, and DNS server addresses. The ACK message and server send to the client using the sendp() function. Meanwhile, the code updates the ips_in_used array to track which IP addresses are currently in use.

```python
def dhcp_ack():

    pkt = sniff(filter="udp and port 67", count=1, iface="enp0s3")[0]

    if DHCP in pkt and pkt[DHCP].options[0][1] == 3:
        print("DHCP Request received")

        print("yes. you can lease this ip address:", offer_client_ip)

        print("adding this ip to the list of used ip:")

        # adding to the array the ip address that the client will use
        ips_in_used.append(offer_client_ip)

        # Craft DHCP Ack
        dhcp_ack1 = Ether(dst="ff:ff:ff:ff:ff:ff", src=get_if_hwaddr("enp0s3")) / \
                    IP(src="0.0.0.0", dst="255.255.255.255") / \
                    UDP(sport=67, dport=68) / \
                    BOOTP(op=2, yiaddr=pkt[BOOTP].yiaddr, siaddr="10.0.0.1", giaddr="0.0.0.0",
                          chaddr=pkt[Ether].src, xid=pkt[BOOTP].xid) / \
                    DHCP(options=[("message-type", "ack"),
                                  ("subnet_mask", "255.255.255.0"),
                                  ("router", "10.0.0.1"),
                                  ("name_server", "10.0.0.1"),
                                  "end"])

        time.sleep(1)

        print("sending dhcp ack to the client")
        # Send DHCP Ack to the client
        sendp(dhcp_ack1)
```

Client side

With the dhcp_discover() function we will craft and broadcast a DHCP discovery masssage.

```
def dhcp_discover():

    dhcp_discover1 = Ether(dst="ff:ff:ff:ff:ff") / \
                     IP(src='0.0.0.0', dst='255.255.255.255') / \
                     UDP(sport=68, dport=67) / \
                     BOOTP(op=1, chaddr="4a:e4:66:e8:7a:00", xid=23567342) / \
                     DHCP(options=[("message-type", "discover"), "end"])

    # send DHCP discover to the server
    sendp(dhcp_discover1)
```

The got_dvcp_offer() function puprpose is to the deal with the ip offer from the DHCP server. First, we are sniffing and capturing packets on port 68 (which is the port for DCHP communication) , if a Packet we sniffed contains a DHCP offer , we will put the client_ip_from_server as out ip in the bootp header.Then we are checking to see if the ip we got is 0.0.0.0 we won't send any request to server due to fact that there are no ip's avilable.Otherwise, if we got a valid ip we will craft and send a request message to the server.

```
def got_dhcp_offer():

    global client_ip_from_server

    pkt = sniff(filter="udp and port 68", count=1, iface="enp0s3")[0]

    if DHCP in pkt and pkt[DHCP].options[0][1] == 2:
        print("DHCP Offer received")

        client_ip_from_server = pkt[BOOTP].yiaddr

        if client_ip_from_server == "0.0.0.0":
            print("there is no more available ips from the server")
            return

        print("the client_ip that the server offer is:", client_ip_from_server)

        print("ok, i want this ip address:", client_ip_from_server, "can i lease it?")

        # Craft DHCP Request
        dhcp_request = Ether(dst="ff:ff:ff:ff:ff:ff") / \
                       IP(src="0.0.0.0", dst="255.255.255.255") / \
                       UDP(sport=68, dport=67) / \
                       BOOTP(op=1, chaddr="4a:e4:66:e8:7a:00", xid=pkt[BOOTP].xid) / \
                       DHCP(options=[("message-type", "request"),
                                     ("requested_addr", pkt[BOOTP].yiaddr),
                                     ("server_id", pkt[IP].src),
                                     "end"])

        time.sleep(1)
        print("sending dhcp request to the server")
        # Send DHCP Request to the server
        sendp(dhcp_request)
```

After we have sent a request massage to the server , the server needs to send an ack to finish the communication and to know that the server accepted his request and he got the Ip he asked for, there for we created the got_dhcp_ack() function to deal with it. The function sniff packets on port 68 looking for a DHCP ack packet.

```
def got_dhcp_ack():

    pkt = sniff(filter="udp and port 68", count=1, iface="enp0s3")[0]  #got the pkt in the spot 0

    if DHCP in pkt and pkt[DHCP].options[0][1] == 5:
        print("DHCP ack received")

        print("so my ip address is:", client_ip_from_server)
```

DNS

Overview

The domain name system (DNS) is a naming database in which internet domain names are located and translated into Internet Protocol (IP) addresses. The DNS server taking a string of domain (like "google.com") and give back the Ip address of this domain to know where to send our request.

Lets explain how it happened:

1. The client sends a DNS query to its configured DNS server.

2.The DNS server checks its local cache to see if it has a matching record for the requested domain or host. If it does, it responds to the client with the cached record.

3.If the DNS server does not have a cached record for the requested domain or host, it forwards the query to other DNS servers in the hierarchy until it finds a DNS server that can provide an authoritative answer.

4.When an authoritative DNS server is found, it responds to the DNS server that sent the query with the requested record, and the DNS server responds to the client with the IP address for the requested domain or host.

5.If the DNS server cannot find an authoritative answer for the requested domain or host, it responds with an error code to the client, indicating that the requested name could not be resolved.

6.The client receives the IP address from the DNS server and uses it to communicate with the requested host or domain.

In the next page we will explain on the code.

Explain on the code:

DNS Server:

This is the DNS server , which contains:

dns_cache – which is gonna be our cache collection

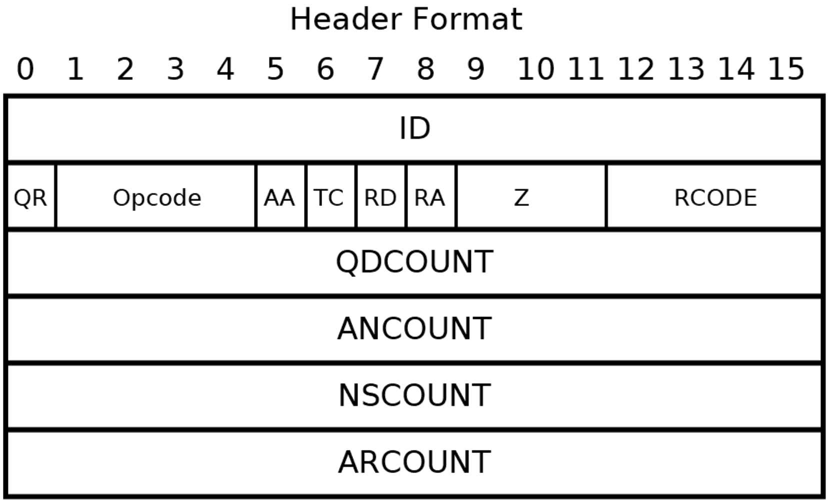
The dns_reply(packet) purpose is to handle the DNS requests from the client, getting a query. the function checking if the same request has been done before by scanning the cache, if it has been done before we will send it straight from the cache to client .Otherwise using a Scapy function sr1() we will perform a DNS lookup for the domain name and add the Ip we extracted from it to the cache.

sr1() - send a specified request and waits for response.

```
# Set the interface to listen and respond on
net_interface = "lo"

# Packet Filter for sniffing specific DNS packet only
packet_filter = "udp port 53"        # Filter UDP port 53

# Function that replies to DNS query
# Create a collection to serve as the cache
dns_cache = {}

def dns_reply(packet):

    # Get the domain name from the DNS query
    domain_name = packet[DNSQR].qname.decode('utf-8')

    print(domain_name)

    # Check if the domain name is in the cache
    if domain_name in dns_cache:
        print('domain is in cache')
        ip_address = dns_cache[domain_name]
    else:
        # Perform a DNS lookup for the domain name
        dns_res = sr1(IP(dst='8.8.8.8')/UDP()/DNS(rd=1, qd=DNSQR(qname=domain_name)), verbose=0)

        # Extract the IP address from the DNS response
        ip_address = dns_res[DNSRR].rdata

        # Add the IP address to the cache
        dns_cache[domain_name] = ip_address
        print('add domain to cache')
```

Then when we got the Ip of the domain, the server will craft a DNS response packet and filling up the DNS header.

then we will use sniff() function to sniff DNS request on port 53 (the ideal port for DNS communications).




```python
    print(str(dns_cache))
    # Construct the DNS response
    dns_response = DNS(
        id=packet[DNS].id,
        qr=1,  # Response
        aa=1,  # Authoritative Answer
        ancount=1,  # One Answer
        qd=packet[DNS].qd,
        an=DNSRR(rrname=domain_name, type='A', rclass='IN', ttl=600, rdata=ip_address)
    )

    # Construct the UDP packet
    udp_packet = UDP(
        sport=packet[UDP].dport,
        dport=packet[UDP].sport
    )

    # Construct the IP packet
    ip_packet = IP(
        dst=packet[IP].src,
        src=packet[IP].dst
    )

    # Construct the Ethernet packet
    ether_packet = Ether(
        dst=packet[Ether].src,
        src=packet[Ether].dst
    )

    # Put the packets together
    response_packet = ether_packet / ip_packet / udp_packet / dns_response

    # Send the DNS response
    sendp(response_packet, iface=net_interface)
    print('sent ip '+str(ip_address))


if __name__ == "__main__":
    while True:
        sniff(filter=packet_filter, prn=dns_reply, store=0, iface=net_interface, count=1)
```

DNS Client

Here we are asking from the user to put a domain name that he want to know the Ip for this domain , then we are crafting a DNS qurey packet and sending it to the server using the sr1() function. then we are printing to the screen the Ip that we got from the DNS server.

```
def dns_socket():
    domain=input('enter domain name : ')
    # Create a DNS query packet
    dns_query_packet = IP(dst="127.0.0.1") / UDP(dport=53) / DNS(rd=1, qd=DNSQR(qname=domain))

    # Send the DNS query packet and get the response
    dns_response_packet = sr1(dns_query_packet)

    # Print the resolved IP address if the DNS query was successful
    if dns_response_packet.haslayer(DNS):
        print(dns_response_packet[DNSRR].rdata)
```

Application

Overview

Our application is an HTTP application which doing a redirect to another server where the file is downloaded from(the client asking from the app but the app doesn't have the answer(response) but he know the server that have, so he connect to the server requesting from him and the server send back the URL to the app and back to the client, and then the client craft the photo) .

We have done two separate implementations for this application transport layer :

1.TCP - The Transmission Control Protocol (TCP) is a transport protocol that is used on top of IP to ensure reliable transmission of packets.

TCP includes mechanisms to solve many of the problems that arise from packet-based messaging, such as lost packets, out of order packets, duplicate packets, and corrupted packets.
lets show what happen in the TPC connection and what TCP does:
When two computers want to send data to each other over TCP, they first need to establish a connection using a three-way handshake.
After that one computer send data to another computer and in tcp the receiver always needs to ack to the sender on what he receive, so that the data will always get to his destination in fully because it need to get ack on each packet that got.
TCP connections can detect out of order packets by using the sequence and acknowledgement numbers.
At the end, they close the connection between them and then the socket closes.

2. Reliable UDP - is a protocol that combines the simplicity and speed of UDP with the reliability of TCP. It provides a lightweight and efficient alternative to TCP for applications that require reliable data transfer over the network. RUDP achieves reliability through the use of selective repeat ARQ (Automatic Repeat Request), where the receiver acknowledges only the successfully received packets, and the sender retransmits only the missing or damaged packets. This reduces the overhead of transmitting and processing acknowledgment packets and allows for faster data transfer. Additionally, RUDP supports congestion control and flow control mechanisms to prevent network congestion and ensure efficient use of network resources. In this project we made all this things manually(Reno CC algorithm, Flow control, acks on each segments) to make the UDP connection to RUDP.

We will show first the TCP part:

TCP

TCP APP

First, we crafted two TCP sockets.

Tcp_app_socket - a reusable socket that can handle at most 5 communication at once ,its purpose is to communicate with the client .

server_socket – a socket that will communicate with the server which holds all the downloadble files.

```
def tcp_app():

    # ******************************************************************************

    # 1
    # creating TCP sockets

    port = 20529
    server_address = "127.0.0.1"

    tcp_app_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    # making that the addr will not say "the addr is already in use"
    tcp_app_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

    tcp_app_socket.bind((server_address, port))

    tcp_app_socket.listen(5)

    # open a connection to the second server that have the object
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.connect(("127.0.0.1", 30553))
```

in this part of the code the app waits for incoming connections from clients,creating a connection socket with the client. after receiving the request from the client the app forwards the request to the server which holds the files(URL). After receiving a response from the server with the URL, the app sends it back to the client.

```
while True:
    connection_socket, addr = tcp_app_socket.accept()
    print(f"connected to server {addr}")

    # ********************************************************************************

    # 2
    # receiving from client the request and sending this request to the server

    request = connection_socket.recv(4096).decode("utf-8")

    print("Received the request from the client")

    print("Got number:", request)

    print("I dont have what you want but i know the server that have this")
    print("i will connect to this server")

    # Sending the request to the second server
    server_socket.send(request.encode("utf-8"))
    print("Sent the request to the second server")

    # ********************************************************************************

    # 3
    # receiving the response from the server and sending this response to the client

    response = server_socket.recv(4096)
    print("Got the response from the second Server")

    connection_socket.send(response)
    print("Sent the response to the Client")

    # ********************************************************************************
```

TCP Client:

we have crafted a TCP socket called tcp_client_socket , that connects to port the http port 20529 (as requested that the ports we are going to use will be in the 20xxx format 529 is the last 3 numbers of my id) , then we are saving the option that the client requested too.

```
def tcp_app_client():

    # ************************************************************************************
    # 1
    # creating TCP socket and asking what request the client want

    tcp_client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    tcp_client_socket.connect(("127.0.0.1", 20529))

    # We are creating a request to the server to give us a link to the oceans song
    request = "I want a picture".encode("utf-8")

    print("The choices of movies are:")
    print("For Avengers-Endgame write 1")
    print("For Avengers-InfinityWar write 2")
    print("For Avengers-FirstAvenger write 3")

    input_choice = input("APP which poster movie do you want? pick 1 to 3! ")

    # *********************************************************************************************
```

after we got the client request the client sending the request to the app and then receiving and receiving the response from the app. The reason we are encoding the messages is because we cant send string or any other type but bytes , so we need to convert the request into bytes inoreder to send it out.

```
# 2
# sending the request to the app and receiving from the app the response

# Sending the request to the app_server
tcp_client_socket.send(input_choice.encode("utf-8"))
print("sent the request to the app_server")

url_response_server = tcp_client_socket.recv(4096).decode("utf-8")
print(url_response_server)
print("Got the response from the app_server")
```

Now that we got the URL from the app we can finally download the wanted image, for this we will use the requests library (Python library for making HTTP requests to web servers).

the client send an HTTP GET request to the received URL and receives a response object.

He then checks if the status code of the response is 302 (which indicates that the requested resource has been temporarily moved to a different URL).

if it is the case(302), the client retrieves the new URL from the 'Location' header in the response object and sends another GET request to the new URL.

If the status code is 200(which indicates that the request was successful), the client writes the contents of the response to a new image file called "EndGame.jpg" and displays the downloaded image.

If the response status code is neither 302 nor 200, the code prints a message indicating that the image download has failed and displays the HTTP status code received from the server.

```
# 3
# create the photo by the url that the app gave us

if input_choice == "1":
    # Send an HTTP request to the URL and get the response object
    response = requests.get(url_response_server, allow_redirects=True)

    print(f"The status code is: {response.status_code}")

    # check if the image have been moved temporality to a diffrent URL
    if response.status_code == 302:
        new_url = response.headers['Location']
        response = requests.get(new_url)

    # Check if the request was successful (HTTP status code 200)
    elif response.status_code == 200:
        # Open a local file with wb (write binary) permission.
        with open("EndGame.jpg", "wb") as file:
            # Write the contents of the response to the file.
            file.write(response.content)
            print("Image downloaded successfully.")
            img = Image.open('EndGame.jpg')
            img.show()
    else:
        print(f"Failed to download image. HTTP status code: {response.status_code}")
```

בס"ד

we are reapiting the same algorithm of choice 1 to choice 2 and 3, for each one we got different URL link from the server and we get different image.

```python
elif input_choice == "2":

    # Send an HTTP request to the URL and get the response object
    response = requests.get(url_response_server, allow_redirects=True)

    print(f"The status code is: {response.status_code}")

    # check if the image have been moved temporality to a diffrent URL
    if response.status_code == 302:
        new_url = response.headers['Location']
        response = requests.get(new_url)

    # Check if the request was successful (HTTP status code 200)
    elif response.status_code == 200:
        # Open a local file with wb (write binary) permission.
        with open("InfinityWar.jpg", "wb") as file:
            # Write the contents of the response to the file.
            file.write(response.content)
            print("Image downloaded successfully.")
            img = Image.open('InfinityWar.jpg')
            img.show()
    else:
        print(f"Failed to download image. HTTP status code: {response.status_code}")

elif input_choice == "3":

    # Send an HTTP request to the URL and get the response object
    response = requests.get(url_response_server, allow_redirects=True)

    print(f"The status code is: {response.status_code}")

    # check if the image have been moved temporality to a diffrent URL
    if response.status_code == 302:
        new_url = response.headers['Location']
        response = requests.get(new_url)

    # Check if the request was successful (HTTP status code 200)
    elif response.status_code == 200:
        # Open a local file with wb (write binary) permission.
        with open("Ultron.jpg", "wb") as file:
            # Write the contents of the response to the file.
            file.write(response.content)
            print("Image downloaded successfully.")
            img = Image.open('Ultron.jpg')
            img.show()
    else:
        print(f"Failed to download image. HTTP status code: {response.status_code}")
```

בס"ד

TCP POSTER SERVER :

we created a reusable TCP socket that will communicate with the application on port 30553.

```python
def img_server_tcp():

    # ***********************************************************************************************

    # 1
    # creating a TCP socket and receiving the request from the app

    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    # making that the addr will not say "the addr is already in use"
    server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

    server_socket.bind(("127.0.0.1", 30553))

    server_socket.listen(3)

    connection_socket, server_addr = server_socket.accept()

    # *******************************************************************************
```

The server now checks what option did the application received from the client so that he can send the URL response to the exact request.

```python
# ****************************************************************************

# 2
# receiving the request from the app and sending the url_response to the app

while True:

    request = connection_socket.recv(4096).decode("utf-8")

    print("Got the Request")

    print(request)

    if request == "1":
        # get the img from the func get_image_EndGame
        url_EndGame = Get_Image_EndGame()
        connection_socket.sendall(url_EndGame.encode("utf-8"))
        print("Sent the file to the app server")

    elif request == "2":
        # get the img from the func get_image_InfinityWar
        url_InfinityWar = Get_Image_InfinityWar()
        connection_socket.sendall(url_InfinityWar.encode("utf-8"))
        print("Sent the file to the app server")

    elif request == "3":
        # get the img from the func get_image_ultron
        url_ultron = Get_Image_Ultron()
        connection_socket.sendall(url_ultron.encode("utf-8"))
        print("Sent the file to the app server")

    else:
        print("the object that you ask for, isn't here")
        exit(1)
```

Here we have a helper functions that contain the URL for each one of the request. we saved the URL so that it will be easier to call them later.

```
    connection_socket.close()
    second_server_socket.close()

# helper function to get the url of the EndGame poster
MatanAdar
def Get_Image_EndGame():

    # URL of the image to download
    url = 'https://lumiere-a.akamaihd.net/v1/images/p_avengersendgame_19751_e14a0104.jpeg'
    return url

# helper function to get the url of the InfinityWar poster
MatanAdar
def Get_Image_InfinityWar():

    # URL of the image to download
    url = 'https://m.media-amazon.com/images/M/MV5BMjMxNjY2MDU1OV5BMl5BanBnXkFtZTgwNzY1MTUwNTM@._V1_.jpg'
    return url

# helper function to get the url of the Ultron poster
MatanAdar
def Get_Image_Ultron():

    # URL of the image to download
    url = 'https://www.vintagemovieposters.co.uk/wp-content/uploads/2021/03/IMG_1741-scaled.jpeg'
    return url
```

בס"ד

Reliable UDP:

Note: when the user write what he want he must use the exact same words that we ask. For example in the first question you can select between Iphone or Android so you must put the exact name like Iphone and not iphone with low letter.

RUDP APP:

first we have created two reusable UDP sockets udp_app_socket a socket that will be used to communicate with the client, we have set the setblocking option to be true so that he wont skip any packets that it needs to receive and send.

The img_server_socket purpose is to communicate with the poster server.

To keep the server on we made an infinity loop, in the loop :

In this couple of lines we will perform a three way handshake between the client and the app, in order to implement the RUDP method.

the app first receives from the client the type of phone he wants to get and the client's maximum window for receiving packets (for implement FLOW CONTROL).

We are checking if the client have typed a valid input that the app can handle and if so we send the client an ack to tell him that we received his request.

(We have done this check for each input(4 in total)).

```
if request_from_client == "Iphone 14":
    print("Got the request")
    got_the_request = "ACK"
    while True:
        try:
            udp_app_socket.sendto(got_the_request.encode("utf-8"), client_addr)
            print("Sent ACK to the client")
            break
        except socket.timeout:
            continue
```

In case we didn't get a valid input , the app will send the client an NACK and the client will try again.

```
else:
    print("Didnt got the request")
    didnt_got_the_reqest = "NACK"
    udp_app_socket.sendto(didnt_got_the_reqest.encode("utf-8"), client_addr)
    print("Sent the client that we didnt get the reuqest")
```

This the three handshake that tells us that the client know the app received the request and then we have completed the handshake.

```
# waiting that the client tell us that he got ACK
while True:
    try:
        udp_app_socket.settimeout(10)  # 10 seconds
        response_to_ack = udp_app_socket.recvfrom(4096)
        response_to_ack = response_to_ack[0].decode("utf-8")
        break
    except socket.timeout:
        continue

while True:
    if response_to_ack == "ACK":
        print("Got ack, great!")
        break
    else:
        continue
```

In this section we are sending the request to the server in TCP connection (Prof. Amit allowed it(we add a photo down below of the answer section 2 of Prof. Amit in the forom)) and receiving back from the server the URL response of the exact request.

```
# 2
# sending the request to the server with TCP socket( like professor amit told us to do) and receiving the url from the server

print("The model phone that the client choice is:", request_from_client)

print("I dont have what you want but i know the server that have this")
print("I will connect to this server")

# Sending the request to the second server
img_server_socket.send(request_from_client.encode("utf-8"))
print("Sent the request to the img server")

response_from_server = img_server_socket.recv(4096)
url_response = response_from_server
print("Got the response from the img Server")

# ****************************************************************************************************
```

ת תגובות באופן מקונן

HTTP REDIRECT
על ידי לייפר עפרי בתאריך 6/03/2023, 15:46

1. האם הכתובות אליהן שרת הרדירקט שלנו מפנה את הלקוח, נדרשות להיות מקומיות שאנחנו יוצרים או שמספיק להפנות לכתובת לא שלנו? לדוגמה: כתובת באינטרנט לא שלנו ממנה אפשר להוריד תמונה

2. במידה והשרתים מקומיים ונוצרו על ידינו והלקוח בסוף פונה אליהן לאחר שדיבר עם שרת הרדירקט, האם מצופה שהחיבור איתם יהיה על RUDP ? מה נדרש והאם ניתן לקבל יותר ניקוד על ידי חיבור RUDP ?

קישור ישיר

תשובה ל: HTTP REDIRECT
על ידי דביר עמית זאב בתאריך 6/03/2023, 17:02

1) הכתובות יכולות להיות מה שתחליטו, ככל שהפיתוח יהיה יותר מורכב כך יהיה הציון
2) החיבור RUDP נדרש בגישה הראשונה לשרת האפליקציה, לאחר מכן אפשר רגיל למרות שעקרונית אם מימשת פעם אחת אז למה לא תמיד

עמית

קישור ישיר | הצגת הודעה מעל

We have set parameters to implement the Reno cc algorithm to make the UDP connection run like an RUDP.

In the middle we can see that we calculate the number of segments that we need to send to the client. After that, we are making two arrays:

1. ack_received – this Boolean array will indicate which seq num got ack (thats mean that the client got this segments[seq num])

2.dup_ack_index- this array will indicate if seq num got three acks, so that we can know if there is a need to implement free transmission.

In the end we can see there is a dictionary segment that will keep all the segments that we are going to create and send in the next part.

```
# 3
# sending the url in segments to the client
# making the UDP socket to work as RUDP socket (using CC reno and ACKS on each segment)

# ************************
# 3.1
# setting parameters and calc how much segments we need to send to the client and making them

# Set up Reno congestion control parameters

cwnd = 1
ssthresh = 16


# Reno congestion control algorithm
url_data = url_response.decode()

# calc the amount of segments app need to send to client to transfer all the data(url) to him
segment_size = 5

url_data_length = len(url_data)

if url_data_length % segment_size > 0:
    remind = 1
else:
    remind = 0

num_segments = int(url_data_length / segment_size) + remind

# boolean array that tell us what seq num of segments got ot the client
ack_received = [False] * num_segments
# array that keep track of the amout of ack each seq num get (for 3 dup ack)
dup_ack_index = [0] * num_segments

# dict that keep all segments
segments = {}
```

Here we are crafting the segments, we can see that every segment contains the letter S in the beginning of the segment and the last segment contains an E in the beginning of the segment to tell that this is the last segment. in addition we are sending the index of the packet in the segments. therefore when we are encountering the last packet the client will know how many packets he needs to receive.

In the end of each segment we are sending the data.

```
# loop that create the segments with letter , seq num and data
i = 0
while i < num_segments:
    index = str(i)
    if i == num_segments - 1:
        segments[i] = "E," + index + "," + url_data[i * segment_size:]
        # index array that keep the segments that going to send but didn't send yet
    else:
        segments[i] = "S," + index + "," + url_data[i * segment_size: (i+1)*segment_size]

    i += 1

# **********************
```

This is an overview of the next couple of parts that we are going to elaborate.

```
# **********************


# 3.2
# checking if all segments have been acknowledged
# 3.2.1 - adding the segments that didn't get ACK yet to the segments_unacked array
# 3.2.2 - if the size of segments_unacked is 0 its mean that all the ack received array is True, and we got ACKs on all the segments we sent
# 3.2.3 - if there still space in chwd(window size) we send to the client the segments until the chwd get full by poping from segments_unacked array and counting the amount segments we're sending
# 3.2.4 - receiving from the client the seq num segment that he got and checking if we got ACK on him already and implementing FLOW control here too.
# if no we're changing ack_received[seq_num] = true and lower the amount of segments we sent and didn't get ack on them yet
# if yes we're adding the seq_num ack to the array of dup ack, to check if we get 3 dup ack to know if we need to make Fast retransmitted
# 3.2.5 - if we got timeout (timeout waiting for ack) we're decreasing ssthresh to be chwd/2 and chwd to be 1
# else we make the ssthresh to be like chwd
# notice that we implement FLow Control in this code because we check that the chwd will not be over the max window size that the client send us,
# and that will make that the app will send more than the client can handle and will not get Situation of "overflow"

# Check if all segments have been acknowledged
```

in this section we append to segment_unacked array the segments that didn't received an ack yet.

```python
# Check if all segments have been acknowledged
while True:
    #  3.2.1
    # put all the segments that we didn't got ack on them(segments that not True in the ack_received array), back in the segment_unacked array
    # index array that keep the segments that didn't get ack yet
    segments_unacked = []
    for i in range(num_segments):
        if ack_received[i] == False:
            segments_unacked.append(i)
```

This section checks if whether we received all the acks for all the packets or not.

If yes it mean we got all the packets so the data transfer from the app to the client is done.

```python
# 3.2.2
if len(segments_unacked) == 0:
    print("received ack on every segments")
    break
```

here we calculating how many packets we are able to send in the current window size and if there is segments to send, and then sending them to the client.

```python
# 3.2.3
# Send new segments up to congestion window size
count_segment_that_sending = 0
while count_segment_that_sending < cwnd and len(segments_unacked) > 0:
    seq_num = segments_unacked.pop(0)
    udp_app_socket.sendto(segments[seq_num].encode(), client_addr)
    print("Sent segment", seq_num)
    count_segment_that_sending += 1
```

in the beginning we are receiving from the client a seq num of the segment that he got and then we are checking if we already got on this seq num an ack (checking if ack_received[seq num]==true).

if it didn't got an ack yet , we change the ack_received[seq num] to be true. And lower the amount of segment that we sent but still didn't got an ack on them (count_segment_that_sending--) and we will increase the amount of acks that we have received in the dup_ack_index array. After this we are checking if there is a need to update the congestion window size. Be aware that we also implement here a flow control because we making sure that that the cwnd isn't bigger than the max window that the client sent.

After this, we are checking if we have three dup acks using the dup_ack_index and if any seq num got three dup ack we perform Fast Retransmit.

```
# 3.2.4
timeout = False
global ack_seq_num
ack_seq_num = 0
while count_segment_that_sending > 0:
    # Receive acknowledgments
    try:
        ack_data, addr = udp_app_socket.recvfrom(1024)
        ack_seq_num = int(ack_data.decode())
        print("Received ACK for segment", ack_seq_num)
        if not ack_received[ack_seq_num]:
            ack_received[ack_seq_num] = True
            count_segment_that_sending -= 1
            dup_ack_index[ack_seq_num] = dup_ack_index[ack_seq_num]+1

            # Update congestion window size using Reno algorithm and implementing FLOW CONTROL
            if cwnd < ssthresh and cwnd < max_window_size_from_client:
                cwnd *= 2
            else:
                if cwnd < max_window_size_from_client:
                    cwnd += 1 / cwnd

            if ack_seq_num == seq_num and dup_ack_index[ack_seq_num] < 3:
                dup_ack_index[ack_seq_num] = dup_ack_index[ack_seq_num]+1
            elif ack_seq_num == seq_num and dup_ack_index[ack_seq_num] == 3:
                print("Fast Retransmit")
                ssthresh = max(int(cwnd/2), 1)
                cwnd = ssthresh + 3

                # reseting the seq_num that we got dup ack on him
                dup_ack_index[ack_seq_num] = 0

                # adding the 3 seq_num to send again
                for i in range(seq_num - 2, seq_num):
                    segments_unacked.append(i)
                break
```

when we received an ack already on a specific seq num ,we going to this else and we are checking again if there is a need for a Fast Retransmit on this seq num.

If we are getting a socket timeout, that means we didn't get an ack for a packet that we have sent therefor we are decreasing the ssthresh to be cwnd / 2 and reseting the cwnd down to 1 , else we increasing window on condition that the cwnd is smaller then the maximum window of the client.

```
                # if we got ack on this seq_num already, so we check if we get 3 dup ack and do Fast retransmit
                else:

                    if dup_ack_index[ack_seq_num] < 3:
                        dup_ack_index[ack_seq_num] = dup_ack_index[ack_seq_num]+1
                    elif dup_ack_index[ack_seq_num] == 3:
                        print("Fast Retransmit")
                        ssthresh = max(int(cwnd/2), 1)
                        cwnd = ssthresh + 3

                        # reseting the seq_num that we got dup ack on him
                        dup_ack_index[ack_seq_num] = 0

                        # adding the 3 seq_num to send again
                        for i in range(seq_num - 2, seq_num):
                            segments_unacked.append(i)
                        break
                    else:
                        dup_ack_count = 0

            except socket.timeout:
                print("Timeout waiting for ACK")
                timeout = True
                break
        # 3.2.5
        if timeout:
            print("Decrease Window")
            ssthresh = cwnd / 2
            cwnd = 1
        else:
            if cwnd < max_window_size_from_client:
                print("Increase Window")
                ssthresh = cwnd

# end while
```

בס"ד

RUDP Client :

creating sockets and getting the request from the user.

```python
def udp_client():

    # ***********************************************************************************************

    # 1
    # creating UDP sockets and asking what request the client want

    # Configure the server address and port number
    app_address = '127.0.0.1'
    app_port = 20529

    # Create a UDP socket
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    # making that the addr will not say "the addr is already in use"
    client_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

    client_socket.bind(("127.0.0.1", 20530))
    client_socket.setblocking(True)
    client_socket.settimeout(10)

    print("hello! which phone do you want to get:")
    input_choice = input("Iphone or Android? ")

    if input_choice == "Iphone":
        mod_choice = input("which model do you like? Iphone 14 or Iphone 13?")

    elif input_choice == "Android":
        mod_choice = input("which model do you like? Galaxy S23 or Galaxy S22? ")

    # ***************************************************************************************
```

we are doing here the three way handshake with the app . We are sending the request with the maximum window size that the client can receive packets. Waiting for response from the app that will be ACK/NACK that indicates if the app got the request or not.

If the client got an ACK he sends the app an ACK for it , and if he didnt the app didn't get the request so the client resending it.

```python
Max_window_size = 65535

request = mod_choice + "," + str(Max_window_size)

# Send
while True:
    try:
        client_socket.sendto(request.encode("utf-8"), (app_address, app_port))
        print("Sent to the Application the request")
        break
    except socket.timeout:
        continue


# getting the response from the app if he got the request
while True:
    try:
        check_ack, app_addr = client_socket.recvfrom(4096)
        check_ack = check_ack.decode("utf-8")
    except socket.timeout:
        continue
    break

# checking what ack we received
while True:
    if check_ack != "ACK":
        print("Application didnt recv the request")
        try:
            client_socket.sendto(mod_choice.encode("utf-8"), (app_address, app_port))
            print("Sent the request again!")
        except socket.timeout:
            continue
    else:
        print("Application got the request!")
        send_ack_to_app = "ACK"
        try:
            client_socket.sendto(send_ack_to_app.encode("utf-8"), (app_address, app_port)
            print("Sent Ack to Application that we recv that ACK from him")
        except socket.timeout:
            continue
        break
```

in this part we are receiving the packets from the app ,we are separating the packets to three parts (letter, seq_num, data) and sending back to the app the seq num of the packets that the client received . If the seq num is not in the segments_dic, its means that this is the first time we received him, and the client will add him to the dic. and when we get the E letter from a specific packet we will know how many packets we need to receive by his seq num(index).

```python
# 3
# receiving the url from the app by the segments

# segemnts is a dictionery because we don't know the order that the segments received in the application
segments_dic = {}

# -1 is like infinty we do it because we don't know how much segments will receive
last_segment = -1

# getting segments from app when we don't know how much segments we need to get (last_segment didn't change)
while last_segment == -1:
    try:
        segment_packet = client_socket.recvfrom(4096)[0]
        segment_packet = segment_packet.decode()
    except socket.error:
        continue

    # spilt the segment we go to 3 things : letter , seq num and data
    letter, seq_num_from_app, segment_data = segment_packet.split(',', 3)
    seq_num = int(seq_num_from_app)

    ack_packet_seq_num = str(seq_num)

    # Sending to the app the seq num of segment we got that the app will know what segments we got (like sending ACK)
    client_socket.sendto(ack_packet_seq_num.encode(), (app_address, app_port))

    # checking if the seq num is in the segments dic
    if seq_num not in segments_dic:
        segments_dic[seq_num] = segment_data
        print("Get segment number " + str(seq_num))

    # Getting to know how much segments the client need to get
    if letter == "E":
        last_segment = seq_num
```

Here we know how much packets we need to get so we can do the same process we did earlier.

In the end we are crafting the data (the URL) by all the data from the packets that we have gathered.

```
# while to get all the packet after we got the last packet, and we know how much packet we need to get
while len(segments_dic) <= last_segment:
    try:
        segment_packet = client_socket.recvfrom(4096)[0]
        segment_packet = segment_packet.decode()
    except socket.error:
        continue

    letter, seq_num_from_app, segment_data = segment_packet.split(',', 3)
    seq_num = int(seq_num_from_app)

    ack_packet_seq_num = str(seq_num)

    client_socket.sendto(ack_packet_seq_num.encode(), (app_address, app_port))

    if seq_num not in segments_dic:
        segments_dic[seq_num] = segment_data
        print("Get segment number " + str(seq_num))

# assemble the full data that we received from the app
data = ""
for i in range(last_segment + 1):
    data += segments_dic[i]

print(data)
```

here we are downloading the picture the same way we did in the TCP(with requests library).

```
# 4
# creating the photo by the url the app gave us

url_response_server = data

# Send an HTTP request to the URL and get the response object
response = requests.get(url_response_server, allow_redirects=True)

print(f"The status code is: {response.status_code}")

# check if the image have been moved temporality to a diffrent URL
if response.status_code == 302:
    new_url = response.headers['Location']
    response = requests.get(new_url)

# Check if the request was successful (HTTP status code 200)
elif response.status_code == 200:
    # Open a local file with wb (write binary) permission.
    with open("Image.jpg", "wb") as file:
        # Write the contents of the response to the file.
        file.write(response.content)
        print("Image downloaded successfully.")
        img = Image.open('Image.jpg')
        img.show()
else:
    print(f"Failed to download image. HTTP status code: {response.status_code}")

# ****************************************************************************
```

RUDP POSTER-SERVER :

Creating TCP sockets and making a connection socket between the server and the app(like Prof. Amit told we can do, make a TCP connection between the app and the server).

```
# 1
# creating a socket and receiving the request from the app

img_server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# making that the addr will not say "the addr is already in use"
img_server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

img_server_socket.bind(("127.0.0.1", 30553))
img_server_socket.listen(5)
img_server_socket.setblocking(True)
img_server_socket.settimeout(10)


connection_socket, app_addr = img_server_socket.accept()
print(f"connected to server {app_addr}")
```

Receiving from the app the request and sending this to the handle_request function.

```
# 2
# receiving the request from the app and sending the url_response to th

while True:
    try:
        request = connection_socket.recv(4096)
    except socket.error:
        continue

    handle_request(connection_socket, request, app_addr)
```

we are giving the URL of the image according to the app request , we can see that in every block we are using a help function thats holds the specific URL that we need.

we are giving the URL of the image according to the app request)that came from the client) , we can see that in every block we are using a help function that's holds the specific URL that fit for the response.

```python
# helper function to handle the request and send exactly the url to the app
MatanAdar
def handle_request(connection_socket, request, app_addr):

    request = request.decode("utf-8")
    print("Got the Request")
    print(request)

    if request == "Iphone 14":
        # get the img from the func get_image_EndGame
        url_iphone14 = Get_Image_Iphone14()
        connection_socket.sendto(url_iphone14.encode("utf-8"), app_addr)
        print("Sent the file to the app server")

    elif request == "Iphone 13":
        # get the img from the func get_image_InfinityWar
        url_iphone13 = Get_Image_Iphone13()
        connection_socket.sendto(url_iphone13.encode("utf-8"), app_addr)
        print("Sent the file to the app server")

    elif request == "Galaxy S23":
        # get the img from the func get_image_ultron
        url_GalaxyS23 = Get_Image_GalaxyS23()
        connection_socket.sendto(url_GalaxyS23.encode("utf-8"), app_addr)
        print("Sent the file to the app server")

    elif request == "Galaxy S22":
        # get the img from the func get_image_ultron
        url_GalaxyS22 = Get_Image_GalaxyS22()
        connection_socket.sendto(url_GalaxyS22.encode("utf-8"), app_addr)
        print("Sent the file to the app server")

    else:
        print("the object that you ask for, isn't here")
```

These are the help functions for each request.

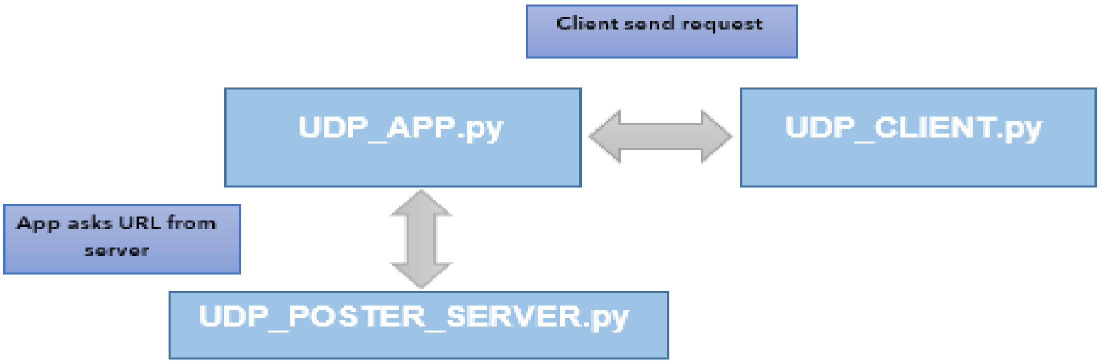
```python
# helper function to get the url of the Iphone 14
MatanAdar
def Get_Image_Iphone14():

    # URL of the image to download
    url = 'https://9to5mac.com/wp-content/uploads/sites/6/2022/01/iphone-14-news-design.jpg?quality=82&strip=all'
    return url

# helper function to get the url of the Iphone 13
MatanAdar
def Get_Image_Iphone13():

    # URL of the image to download
    url = 'https://i.ytimg.com/vi/l0EvriCfmrE/maxresdefault.jpg'
    return url

# helper function to get the url of the Galaxy S23
MatanAdar
def Get_Image_GalaxyS23():

    # URL of the image to download
    url = 'http://johnlewis.scene7.com/is/image/JohnLewis/109920785'
    return url

# helper function to get the url of the Galaxy S22
MatanAdar
def Get_Image_GalaxyS22():

    # URL of the image to download
    url = 'https://tecnocell.co.il/wp-content/uploads/2022/11/%D7%A1%D7%9E%D7%A1%D7%95%D7%A0%D7%92-SAMSUNG-S22-ULTRA-256GB-12GB-RAM-%D7%A9%D7%97%D7%95%D7%A8.jpg'
    return url
```

DIAGRAM

The general process

EXPLANATION:

The client sends a request to the application, and the application forwards the request to the server. The server return an answer for the exact request to the application, and it passes the answer back to the client.

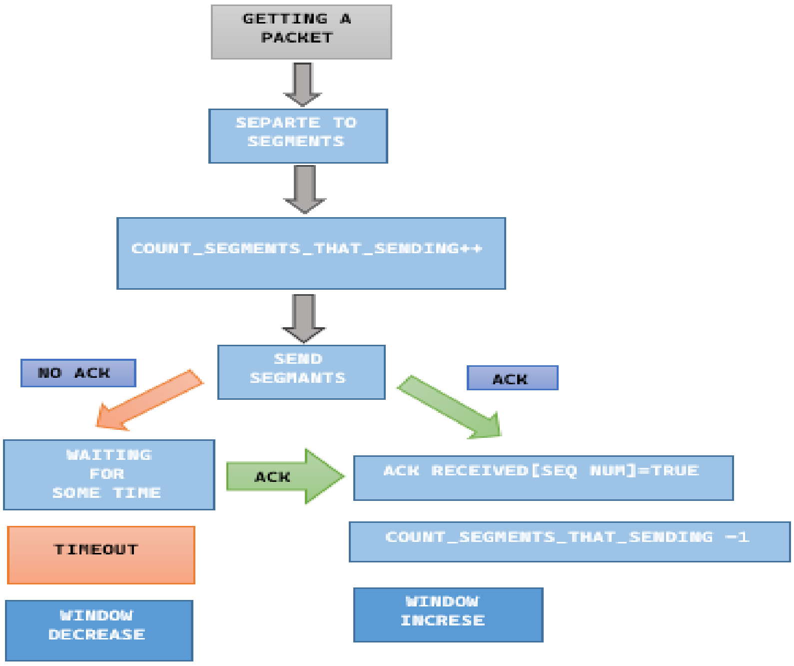


DIGARAM OF SEGMENTS

EXPLANATION:

When we want to send a packet we first separate it into small segments , first we are counting how many segments we need to make that the amount of each packet will not go over 64kb.
After this we count how much segments we can send in the current window size and sending them(count_segments_that_sending). When we sending them we count them to know how much we send and still didn't got ack yet on them from the client.

if a segment got an ack we will update the amount of segment we sent and didn't got ack yet(count_segments_that_sending)_ if a segment didn't got an ack after an estimated time there will be a timeout and the segments that didn't get ack will go back to the array that keep the segments that didn't got ack and send them again.



HOW WE HANDLE A PACKET LOST:

To deal with lost packets we have given each packet a time limit so that if the segment didn't receive an ack at that estimated time the system will declare a time out and the window will decrease, and the sender will resend the lost packet .
More then that, the client send in the first time with the request the max window size that he can get, so that will not be overflow of segments that the app send and the client cant receive because his window is full, so we let the app know that more then the max size window he cant send us and there we prevent a packet lost.

HOW WE HANDLE A LATENCY:

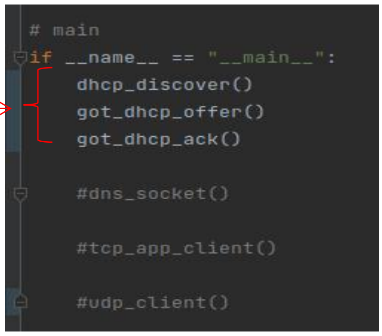
Latency is referring to the amount of time that take to the UDP packet to get from client to the server. We are handle it when setting the socket to have a timeout that when he gets there we will understand that there is a problem probably because the client didn't receive the segment, and send the segment again until we get all the segments. Now we try to set the timeout kinda low but not to much low that even if we send, it take time to the segment to get to his destination. When we making the timeout its make that it will not take a lot of time to get all the data that need to transfer and will be less latency because the time to get all the data will be less.

בס"ד

Wireshark and Terminal

DHCP

Let's see the results in terminal after we run the **dhcp_server.py** and **client.py(make sure that the functions dhcp_discover, got_dhcp_offer and got_dhcp_ack in main in client are available to run)**:



```
# main
if __name__ == "__main__":
    dhcp_discover()
    got_dhcp_offer()
    got_dhcp_ack()

    #dns_socket()

    #tcp_app_client()

    #udp_client()
```

client (first request) size

we can see the communication between the client and the server. We can see that we send a packet at first(DHCP discover), and then we received a DHCP offer from the server. We can see that the Ip that the server offer is 10.0.2.0, so we then asking if we can lease this Ip by sending a DHCP request. After that we got DHCP ack to tell that we can lease this Ip and use it.

```
.
Sent 1 packets.
DHCP Offer received
the client_ip that the server offer is: 10.0.2.0
ok, i want this ip address: 10.0.2.0 can i lease it?
sending dhcp request to the server
.
Sent 1 packets.
DHCP ack received
so my ip address is: 10.0.2.0
root@King:/home/lior/PycharmProjects/pythonProject/RN_FinalProject#
```

server size

we can see that we got from the client a DHCP discover. We found what Ip is not in use already and offering this IP to the client. Then we got the DHCP request from the client and we send him back a DHCP ack to tell him that he can use this Ip that we offered.

```
_server.py
DHCP Discover received
Found that ip 10.0.2.0 is not in use
Offering the client this ip: 10.0.2.0
sending dhcp offer to client
.
Sent 1 packets.
DHCP Request received
yes. you can lease this ip address: 10.0.2.0
adding this ip to the list of used ip:
sending dhcp ack to the client
.
Sent 1 packets.
```

client(second request) size

we can see like the first request that the process is the same , but the different we can see that we offered a different Ip now because the Ip from the first request is already in use , so the DHCP server offer a different Ip to the client.

```
.
Sent 1 packets.
DHCP Offer received
the client_ip that the server offer is: 10.0.2.1
ok, i want this ip address: 10.0.2.1 can i lease
sending dhcp request to the server
.
Sent 1 packets.
DHCP ack received
so my ip address is: 10.0.2.1
```

server (second response) size

we can see that like the first request we do all the process again, but this time the different is we see that he offer a different Ip because the Ip from the first request is already in use.

```
DHCP Discover received
Found that ip 10.0.2.1 is not in use
Offering the client this ip: 10.0.2.1
sending dhcp offer to client
.
Sent 1 packets.
DHCP Request received
yes. you can lease this ip address: 10.0.2.1
adding this ip to the list of used ip:
sending dhcp ack to the client
.
Sent 1 packets.
```

wireshark : (wireshark file: dhcp.pcapng)

we capture in this: enp0s3

and we filter in wireshark with DHCP like this : dhcp

first request :

14	4.031088908	0.0.0.0	255.255.255.255	DHCP	286 DHCP Discover	- Transaction ID 0x1679bee
15	4.031578195	10.0.2.2	10.0.2.16	DHCP	590 DHCP Offer	- Transaction ID 0x1679bee
16	5.084319040	0.0.0.0	255.255.255.255	DHCP	304 DHCP Offer	- Transaction ID 0x1679bee
17	6.146117282	0.0.0.0	255.255.255.255	DHCP	298 DHCP Request	- Transaction ID 0x1679bee
18	6.146604944	10.0.2.2	10.0.2.16	DHCP	590 DHCP ACK	- Transaction ID 0x1679bee
19	7.176084581	0.0.0.0	255.255.255.255	DHCP	304 DHCP ACK	- Transaction ID 0x1679bee

blue arrow – discover packet: the client sent a broadcast message.

red arrow - offer packet: offering the client an Ip, beacuse its the first request from the server, the Ip that the server offer is 10.0.2.0.

Your (client) IP address: 10.0.2.0

green arrow – Request packet: the client asks for the Ip the server offered.

Option: (50) Requested IP Address (10.0.2.0)

brown arrow - ACK packet: the server acknowledged that the client wants the offered Ip.

Note : The other packets in the picture are the communication between my home router DHCP server , due to the fact that we broadcasted a general DHCP message.

Second request :

10.773208612	0.0.0.0	255.255.255.255	DHCP	286 DHCP Discover	- Transaction ID 0x1679bee
10.773510694	10.0.2.2	10.0.2.16	DHCP	590 DHCP Offer	- Transaction ID 0x1679bee
11.821984641	0.0.0.0	255.255.255.255	DHCP	304 DHCP Offer	- Transaction ID 0x1679bee
12.888250619	0.0.0.0	255.255.255.255	DHCP	298 DHCP Request	- Transaction ID 0x1679bee
12.888983505	10.0.2.2	10.0.2.16	DHCP	590 DHCP ACK	- Transaction ID 0x1679bee
13.948139888	0.0.0.0	255.255.255.255	DHCP	304 DHCP ACK	- Transaction ID 0x1679bee

The second request from the server , everything in the communication is the same as the first request. However now the client is offered a different Ip:

blue arrow :

Your (client) IP address: 10.0.2.1

red arrow :

```
Option: (50) Requested IP Address (10.0.2.1)
    Length: 4
    Requested IP Address: 10.0.2.1
```

בס"ד

DNS

Lets see the results in the terminal by running the **dns_server.py** and the **client.py(make sure that the function dns_socket in main in client are available to run)**:

```
# main
if __name__ == "__main__":
    #dhcp_discover()
    #got_dhcp_offer()
    #got_dhcp_ack()

    dns_socket()

    #tcp_app_client()

    #udp_client()
```

Client size:

we can see that we enter a domain name(google.com), and send it to the DNS server and received a Ip address of this domain (172.217.16.206).

blue block – first request

red block – second request

```
please enter a domain name : google.com
Begin emission:
Finished sending 1 packets.
............*
Received 14 packets, got 1 answers, remaining 0 packets
172.217.16.206
root@King:/home/lior/PycharmProjects/pythonProject/RN_FinalProject# python3 clie
nt.py
please enter a domain name : google.com
Begin emission:
Finished sending 1 packets.
.*
Received 2 packets, got 1 answers, remaining 0 packets
172.217.16.206
```

Server size:

We can see that in the first request the client asked for Ip address for the domain google.com but the server didn’t have it in his cache(didn’t got this request before), so he send DNS query and get the Ip from there , adding the Ip to the cache and send it back to the client.

After that we can see that the second request is the same as the first one(google.com), but this time the server don’t need to ask DNS query because he have the Ip address of this domain in his cache, so he just pull it from there and send it to the client.

blue block – first request: the domain was never asked before therefore adding it to the cache.
red block – second request: domain already in cache.

```
server.py
add domain to cache
{'google.com.': '172.217.16.206'}
.
Sent 1 packets.
sent ip 172.217.16.206
domain is in cache
{'google.com.': '172.217.16.206'}
.
Sent 1 packets.
sent ip 172.217.16.206
```

wireshark : (wireshark file: dns.pcapng)

we capture in loopback: Loopback: lo

and we filter in wireshark with DNS like this: dns

1	0.000000000	10.0.2.15	10.0.2.15	DNS	72 Standard query 0x0000 A google.com
2	0.061897851	10.0.2.15	213.57.22.5	DNS	72 Standard query 0x0000 A google.com
3	0.070032287	213.57.22.5	10.0.2.15	DNS	88 Standard query response 0x0000 A google.com A 172.217.16.206
4	0.098517523	10.0.2.15	10.0.2.15	DNS	98 Standard query response 0x0000 A google.com A 172.217.16.206
5	6.827694714	10.0.2.15	10.0.2.15	DNS	72 Standard query 0x0000 A google.com
6	6.850856656	10.0.2.15	10.0.2.15	DNS	98 Standard query response 0x0000 A google.com A 172.217.16.206

blue arrow – a qurey from the client to our DNS server

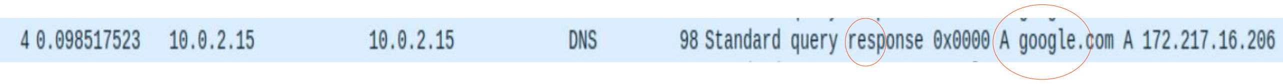
```
Domain Name System (query)
    Transaction ID: 0x0000
  Flags: 0x0100 Standard query
      0... .... .... .... = Response: Message is a query
      .000 0... .... .... = Opcode: Standard query (0)
      .... ..0. .... .... = Truncated: Message is not truncated
      .... ...1 .... .... = Recursion desired: Do query recursively
      .... .... .0.. .... = Z: reserved (0)
      .... .... ...0 .... = Non-authenticated data: Unacceptable
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  Queries
    google.com: type A, class IN
```

orange arrow – our DNS server asks from the DNS server of our router for the Ip of the domain the client asked for.

שרתי DNS של IPv4 213.57.22.5
213.57.2.5

dark arrow – router DNS server returns our own server the requested domain Ip.

red Arrow – our DNS server returns the client the requested domain ip.



Purple arrow – second request – nothing different from the first request

Green arrow – second answer – this time since its the same request as the first , as we can see in the picture above the server didn't need to ask from the router server for the domain Ip , he just sent it straight from the cash.

APPLICATION

Notes: in this part of the Application, for both TCP and RUDP, we used port 20529 (529 is the 3 last digit of the ID of lior) to connect with the client and the app. And we used port 30553 (553 is the last 3 digit of the ID of matan) to connect app with the server.

Lets start first with TCP:

TCP

Let's see the results in the terminal by running the **TCP_poster_server.py** and the **TCP_app.py** and the **client.py(make sure that the function tcp_app_client in main in client is available to run)**:

```
# main
if __name__ == "__main__":
    #dhcp_discover()
    #got_dhcp_offer()
    #got_dhcp_ack()

    #dns_socket()

    tcp_app_client()

    #udp_client()
```

Client side:

We can see that we have a selection of choices to choose from and the user can select each one of the 3 options. After he select(for example like in this photo the user choose 3) so the request move on to the app.

After this we can see that we got the URL of the poster we asked for and then the photo will apper on the screen (the client craft the photo with the URL link with the requests library).

```
matan@matan-VirtualBox:~/PycharmProjects/RN_FinalProject$ python3 client.py
The choices of movies are:
For Avengers-Endgame write 1
For Avengers-InfinityWar write 2
For Avengers-FirstAvenger write 3
APP which poster movie do you want? pick 1 to 3! 3
sent the request to the app_server
https://www.vintagemovieposters.co.uk/wp-content/uploads/2021/03/IMG_1741-scaled.jpeg
Got the response from the app_server
The status code is: 200
Image downloaded successfully.
matan@matan-VirtualBox:~/PycharmProjects/RN_FinalProject$
```

App size:

We can see that we connected to the client (port 20529) and we received the request from him. And then the server tell us that he don't have what we asked for, but he know the server that have the answer to the request.

So the app connect to the server on port 30553 and send the request to the app.

The app wait for a response from the server and send the response to the client.

```
matan@matan-VirtualBox:~/PycharmProjects/RN_FinalProject$ sudo python3 TCP_app.py
connected to server ('127.0.0.1', 20529)
Received the request from the client
Got number: 3
I dont have what you want but i know the server that have this
i will connect to this server
Sent the request to the second server
Got the response from the second Server
Sent the response to the Client
```

Server size:

Server received from the app the request and he find the URL for the request and then send it to the app.

```
matan@matan-VirtualBox:~/PycharmProjects/RN_FinalProject$ python3 TCP_poster_server.py
connected to server ('127.0.0.1', 30555)
Got the Request
3
Sent the file to the app server
```

Wireshark to the tcp: (TCP wireshark: tcp.pcapng)

we capture in this: Loopback: lo

and we filter in wireshark with TCP like this: tcp.port ==20529 || tcp.port == 30553

We can see that we make the 3 hand shake at the beginning and then sending in port 20529 to the app and then sending from app to the server in port 30553, receiving from the server the response to the app (prt30553) and then to the client (port 20529) all in the tcp connection.

Apply a display filter ... <Ctrl-/> + new

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	TCP	76	30555 → 30553 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=1918496334 TSecr=0 WS=128
2	0.000018286	127.0.0.1	127.0.0.1	TCP	76	30553 → 30555 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=1918496334 TSecr=1918496334 WS=128
3	0.000030078	127.0.0.1	127.0.0.1	TCP	68	30555 → 30553 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1918496334 TSecr=1918496334
4	7.422763815	127.0.0.1	127.0.0.1	TCP	76	20529 → 20530 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=1918503757 TSecr=0 WS=128
5	7.422783712	127.0.0.1	127.0.0.1	TCP	76	20530 → 20529 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=1918503757 TSecr=1918503757 WS=128
6	7.422796712	127.0.0.1	127.0.0.1	TCP	68	20529 → 20530 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1918503757 TSecr=1918503757
7	8.782131366	127.0.0.1	127.0.0.1	TCP	69	20529 → 20530 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=1 TSval=1918505116 TSecr=1918503757
8	8.782250645	127.0.0.1	127.0.0.1	TCP	68	20530 → 20529 [ACK] Seq=1 Ack=2 Win=65536 Len=0 TSval=1918505116 TSecr=1918505116
9	8.782618937	127.0.0.1	127.0.0.1	TCP	69	30555 → 30553 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=1 TSval=1918505117 TSecr=1918496334
10	8.782749637	127.0.0.1	127.0.0.1	TCP	68	30553 → 30555 [ACK] Seq=1 Ack=2 Win=65536 Len=0 TSval=1918505117 TSecr=1918505117
11	8.783335343	127.0.0.1	127.0.0.1	TCP	153	30553 → 30555 [PSH, ACK] Seq=1 Ack=2 Win=65536 Len=85 TSval=1918505118 TSecr=1918505117
12	8.783465224	127.0.0.1	127.0.0.1	TCP	68	30555 → 30553 [ACK] Seq=2 Ack=86 Win=65536 Len=0 TSval=1918505118 TSecr=1918505118
13	8.784145792	127.0.0.1	127.0.0.1	TCP	153	20530 → 20529 [PSH, ACK] Seq=1 Ack=2 Win=65536 Len=85 TSval=1918505118 TSecr=1918505116
14	8.784288928	127.0.0.1	127.0.0.1	TCP	68	20529 → 20530 [ACK] Seq=2 Ack=86 Win=65536 Len=0 TSval=1918505118 TSecr=1918505118
15	11.514824114	127.0.0.1	127.0.0.1	TCP	68	20529 → 20530 [FIN, ACK] Seq=2 Ack=86 Win=65536 Len=0 TSval=1918507849 TSecr=1918505118
16	11.555183472	127.0.0.1	127.0.0.1	TCP	68	20530 → 20529 [ACK] Seq=86 Ack=3 Win=65536 Len=0 TSval=1918507889 TSecr=1918507849

בס"ד

RUDP:

Let's see the results in the terminal by running the **UDP_poster_server.py** and the **UDP_app.py** and the **client.py(make sure that the function udp_client in main in client is available to run)**:

```
# main
if __name__ == "__main__":
    #dhcp_discover()
    #got_dhcp_offer()
    #got_dhcp_ack()

    #dns_socket()

    #tcp_app_client()

    udp_client()
```

Client size:

We can see that we have a selection of choices to choose from and the user can select each one of the 2 options at first and then select between 2 options again . After he selected (for example like in this photo the user choose Iphone and then Iphone 14) so the request move on to the app.

After this we can see that we got the URL of the poster we asked for, that send in couple of segments that not go over 64kb and then we connect all the data of the segments to one data that is our URL. then the photo will appear on the screen (the client craft the photo with the URL link with the requests library).

```
matan@matan-VirtualBox:~/PycharmProjects/RN_FinalProject$ python3 client.py
hello! which phone do you want to get:
Iphone or Android? Iphone
which model do you like? Iphone 14 or Iphone 13?Iphone 14
Sent to the Application the request
Application got the request!
Sent Ack to Application that we recv that ACK from him
Get segment number 0
Get segment number 1
Get segment number 2
Get segment number 3
Get segment number 4
Get segment number 5
Get segment number 6
Get segment number 7
Get segment number 8
Get segment number 9
Get segment number 10
Get segment number 11
Get segment number 12
Get segment number 13
Get segment number 14
Get segment number 15
Get segment number 16
Get segment number 17
Get segment number 18
Get segment number 19
Get segment number 20
https://9to5mac.com/wp-content/uploads/sites/6/2022/01/iphone-14-news-design.jpg?quality=82&strip=all
The status code is: 200
Image downloaded successfully.
matan@matan-VirtualBox:~/PycharmProjects/RN_FinalProject$
```

App size:

We can see here that we got the request from the client, after this in the third line we can see the we got the number 16384, its representing the max window size that the client can get that not cause a packet lost and overflow (for FLOW CONTROL).

We can see that we got acks after that , that its represent the 3 handshake between the client and the app that the app got the request.

After this we send the request to the server in TCP connection and getting back the response from the server.

After this we split the URL response from the server to couple of segments to not get over the 64kb and then sending the segments for the client. If we got ack we move on and if we didn't got ack we get timeout and send again the segment that didn't got ack(we can see it in the packet lost terminal couple of pages under this).

We do it until we got ack on all the segments that contain the URL response data.

(the rest of the photo is in the next page)

```
matan@matan-VirtualBox:~/PycharmProjects/RN_FinalProject$ sudo python3 UDP_app.py
Iphone 14
16384
Got the request
Sent ACK to the client
Got ack, great!
Got ack, great!
The model phone that the client choice is: Iphone 14
I dont have what you want but i know the server that have this
I will connect to this server
Sent the request to the img server
Got the response from the img Server
Sent segment 0
Received ACK for segment 0
Increase Window
Sent segment 1
Sent segment 2
Received ACK for segment 1
Received ACK for segment 2
Increase Window
Sent segment 3
Sent segment 4
Sent segment 5
Received ACK for segment 3
Received ACK for segment 4
Received ACK for segment 5
Increase Window
Sent segment 6
Sent segment 7
Sent segment 8
Sent segment 9
Received ACK for segment 6
Received ACK for segment 7
Received ACK for segment 8
Received ACK for segment 9
Increase Window
```

```
Sent segment 6
Sent segment 7
Sent segment 8
Sent segment 9
Received ACK for segment 6
Received ACK for segment 7
Received ACK for segment 8
Received ACK for segment 9
Increase Window
Sent segment 10
Sent segment 11
Sent segment 12
Sent segment 13
Sent segment 14
Received ACK for segment 10
Received ACK for segment 11
Received ACK for segment 12
Received ACK for segment 13
Received ACK for segment 14
Increase Window
Sent segment 15
Sent segment 16
Sent segment 17
Sent segment 18
Sent segment 19
Sent segment 20
Received ACK for segment 15
Received ACK for segment 16
Received ACK for segment 17
Received ACK for segment 18
Received ACK for segment 19
Received ACK for segment 20
Increase Window
received ack on every segments
```

Server size:

We got the request from the app and send him back the URL response to the exact request he asked for.

```
matan@matan-VirtualBox:~/PycharmProjects/RN_FinalProject$ python3 UDP_poster_server.py
connected to server ('127.0.0.1', 47022)
Got the Request
Iphone 14
Sent the file to the app server
```

<u>Wireshark of the RUDP</u>: (RUDP wireshark: RUDP.pcapng)

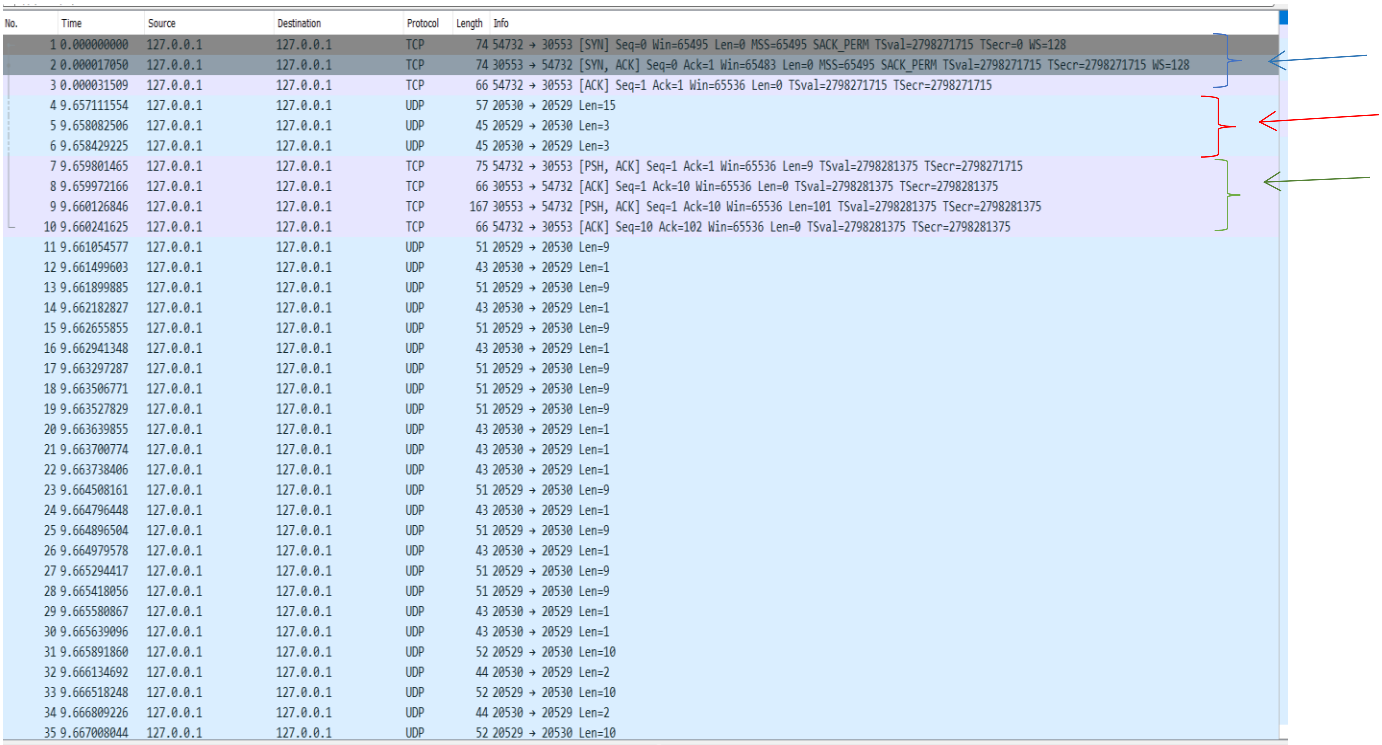
we capture in this: Loopback: lo

and we filter in wireshark with TCP and UDP like this: udp.port ==20529 || tcp.port == 30553

first we can see the connection between the server and the app in TCP connection and port 30553(blue arrow).

We can see that we send in a UDP connection (port 20529) doing the 3 handshake between the app and the client to be RUDP (red arrow) ,and then the request moving on to the server in TCP connection in port 30553 and the response come back to the app (green arrows) and then we see a lot of UDP packets, the segments that the app send to the client with getting ack on all of them that the client got them, that's why we see every segment the app (port 20529 to port 20530) the client send ack(port 20530 to port 20529)(with len 1), and the segment with the data we can notice the lines with the len 9 or 10 we can see this segments had the data in them.

(the rest of the photo is in the next page)



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	TCP	74	54732 → 30553 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=2798271715 TSecr=0 WS=128
2	0.000017050	127.0.0.1	127.0.0.1	TCP	74	30553 → 54732 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=2798271715 TSecr=2798271715 WS=128
3	0.000031509	127.0.0.1	127.0.0.1	TCP	66	54732 → 30553 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=2798271715 TSecr=2798271715
4	9.657111554	127.0.0.1	127.0.0.1	UDP	57	20530 → 20529 Len=15
5	9.658082506	127.0.0.1	127.0.0.1	UDP	45	20529 → 20530 Len=3
6	9.658429225	127.0.0.1	127.0.0.1	UDP	45	20530 → 20529 Len=3
7	9.659801465	127.0.0.1	127.0.0.1	TCP	75	54732 → 30553 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=9 TSval=2798281375 TSecr=2798271715
8	9.659972166	127.0.0.1	127.0.0.1	TCP	66	30553 → 54732 [ACK] Seq=1 Ack=10 Win=65536 Len=0 TSval=2798281375 TSecr=2798281375
9	9.660126846	127.0.0.1	127.0.0.1	TCP	167	30553 → 54732 [PSH, ACK] Seq=1 Ack=10 Win=65536 Len=101 TSval=2798281375 TSecr=2798281375
10	9.660241625	127.0.0.1	127.0.0.1	TCP	66	54732 → 30553 [ACK] Seq=10 Ack=102 Win=65536 Len=0 TSval=2798281375 TSecr=2798281375
11	9.661054577	127.0.0.1	127.0.0.1	UDP	51	20529 → 20530 Len=9
12	9.661499603	127.0.0.1	127.0.0.1	UDP	43	20530 → 20529 Len=1
13	9.661899885	127.0.0.1	127.0.0.1	UDP	51	20529 → 20530 Len=9
14	9.662182827	127.0.0.1	127.0.0.1	UDP	43	20530 → 20529 Len=1
15	9.662655855	127.0.0.1	127.0.0.1	UDP	51	20529 → 20530 Len=9
16	9.662941348	127.0.0.1	127.0.0.1	UDP	43	20530 → 20529 Len=1
17	9.663297287	127.0.0.1	127.0.0.1	UDP	51	20529 → 20530 Len=9
18	9.663506771	127.0.0.1	127.0.0.1	UDP	51	20529 → 20530 Len=9
19	9.663527829	127.0.0.1	127.0.0.1	UDP	51	20529 → 20530 Len=9
20	9.663639855	127.0.0.1	127.0.0.1	UDP	43	20530 → 20529 Len=1
21	9.663700774	127.0.0.1	127.0.0.1	UDP	43	20530 → 20529 Len=1
22	9.663738406	127.0.0.1	127.0.0.1	UDP	43	20530 → 20529 Len=1
23	9.664508161	127.0.0.1	127.0.0.1	UDP	51	20529 → 20530 Len=9
24	9.664796448	127.0.0.1	127.0.0.1	UDP	43	20530 → 20529 Len=1
25	9.664896504	127.0.0.1	127.0.0.1	UDP	51	20529 → 20530 Len=9
26	9.664979578	127.0.0.1	127.0.0.1	UDP	43	20530 → 20529 Len=1
27	9.665294417	127.0.0.1	127.0.0.1	UDP	51	20529 → 20530 Len=9
28	9.665418056	127.0.0.1	127.0.0.1	UDP	51	20529 → 20530 Len=9
29	9.665580867	127.0.0.1	127.0.0.1	UDP	43	20530 → 20529 Len=1
30	9.665639096	127.0.0.1	127.0.0.1	UDP	43	20530 → 20529 Len=1
31	9.665891860	127.0.0.1	127.0.0.1	UDP	52	20529 → 20530 Len=10
32	9.666134692	127.0.0.1	127.0.0.1	UDP	44	20530 → 20529 Len=2
33	9.666518248	127.0.0.1	127.0.0.1	UDP	52	20529 → 20530 Len=10
34	9.666809226	127.0.0.1	127.0.0.1	UDP	44	20530 → 20529 Len=2
35	9.667008044	127.0.0.1	127.0.0.1	UDP	52	20529 → 20530 Len=10

בס"ד

RUDP Packet lost wireshark and terminal:

Client size:

we can see in the first line of the photo that we run the command that lost 10% packets.

We can see that we got segment 8 and 9 but didn't got 7 yet, it because the packet been lost . but after this we received the segment 7 because app notice that he didn't got ack(had timeout) so he send again the segment.

```
matan@matan-VirtualBox:~/PycharmProjects/RN_FinalProject$ sudo tc qdisc add dev lo root netem loss 10%
[sudo] password for matan:
matan@matan-VirtualBox:~/PycharmProjects/RN_FinalProject$ python3 client.py
hello! which phone do you want to get:
Iphone or Android? Iphone
which model do you like? Iphone 14 or Iphone 13?Iphone 13
Sent to the Application the request
Application got the request!
Sent Ack to Application that we recv that ACK from him
Get segment number 0
Get segment number 1
Get segment number 2
Get segment number 3
Get segment number 4
Get segment number 5
Get segment number 6
Get segment number 8
Get segment number 9
Get segment number 7
Get segment number 10
https://i.ytimg.com/vi/l0EvriCfmrE/maxresdefault.jpg
The status code is: 200
Image downloaded successfully.
matan@matan-VirtualBox:~/PycharmProjects/RN_FinalProject$
```

App size:

We can see here that we got the request from the client, after this in the third line we can see the we got the number 16384, its representing the max window size that the client can get that not cause a packet lost and overflow (for FLOW CONTROL).

We can see that we got acks after that , that its represent the 3 handshake between the client and the app that the app got the request.

After this we send the request to the server in TCP connection and getting back the response from the server.

After this we split the URL response from the server to couple of segments to not get over the 64kb and then sending the segments for the client. If we got ack we move on and if we didn't got ack we get timeout and send again the segment that didn't got ack.

We can see that we got after segment 3, we send segment 4 and didn't got him so we had timeout and send him again and then got ack on him that he arrived to the client.

(the photos of the terminal of the app is in the next page)

```
matan@matan-VirtualBox:~/PycharmProjects/RN_FinalProject$ sudo python3 UDP_app.py
[sudo] password for matan:
Iphone 13
16384
Got the request
Sent ACK to the client
Got ack, great!
Got ack, great!
The model phone that the client choice is: Iphone 13
I dont have what you want but i know the server that have this
I will connect to this server
Sent the request to the img server
Got the response from the img Server
Sent segment 0
Received ACK for segment 0
Increase Window
Sent segment 1
Sent segment 2
Received ACK for segment 1
Timeout waiting for ACK
Decrease Window
Sent segment 2
Received ACK for segment 2
Increase Window
Sent segment 3
Sent segment 4
Received ACK for segment 3
Timeout waiting for ACK
Decrease Window
Sent segment 4
Timeout waiting for ACK
Decrease Window
Sent segment 4
Received ACK for segment 4
Increase Window
```

```
I will connect to this server
Sent the request to the img server
Got the response from the img Server
Sent segment 0
Received ACK for segment 0
Increase Window
Sent segment 1
Sent segment 2
Received ACK for segment 1
Timeout waiting for ACK
Decrease Window
Sent segment 2
Received ACK for segment 2
Increase Window
Sent segment 3
Sent segment 4
Received ACK for segment 3
Timeout waiting for ACK
Decrease Window
Sent segment 4
Timeout waiting for ACK
Decrease Window
Sent segment 4
Received ACK for segment 4
Increase Window
Sent segment 5
Sent segment 6
Received ACK for segment 5
Received ACK for segment 6
Increase Window
Sent segment 7
Sent segment 8
Sent segment 9
Received ACK for segment 8
Timeout waiting for ACK
Decrease Window
Sent segment 7
Received ACK for segment 7
Increase Window
Sent segment 9
Sent segment 10
Received ACK for segment 9
Received ACK for segment 10
Increase Window
received ack on every segments
```

Server size:

We got the request from the app and send him back the URL response to the exact request he asked for.

```
matan@matan-VirtualBox:~/PycharmProjects/RN_FinalProject$ python3 UDP_poster_server.py
connected to server ('127.0.0.1', 53040)
Got the Request
Iphone 13
Sent the file to the app server
```

Wireshark of the packet lost: (packet lost wireshark: RUDP with pack lost.pcapng)

We can see in the couple of first lines that we have connection between the server and app (on port 30553) and after that we can see that we do 3 handshake in udp connection , after this send the request to the app in udp connection(blue arrow) and then sending that from app to server in tcp connection, getting the response from the server to the app(green arrow) , and after this sending the segments and receiving acks on them.

When we can see that the packet been lost?

If we see that the time between segment is 10 sec more it mean that the packet lost and after the time out app send again to the client and got ack on this segment that he sent again. We can see in example in line 18 to line 19 then is different of 10 and we can see that is the packet that been lost and sent again.(red arrow)

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	TCP	66	30553 → 57270 [FIN, ACK] Seq=1 Ack=1 Win=512 Len=0 TSval=1920768215 TSecr=1919193877
2	0.048371320	127.0.0.1	127.0.0.1	TCP	66	57270 → 30553 [ACK] Seq=1 Ack=2 Win=512 Len=0 TSval=1920768263 TSecr=1920768215
3	1.977015066	127.0.0.1	127.0.0.1	TCP	66	57270 → 30553 [FIN, ACK] Seq=1 Ack=2 Win=512 Len=0 TSval=1920770192 TSecr=1920768215
4	1.977044195	127.0.0.1	127.0.0.1	TCP	66	30553 → 57270 [ACK] Seq=2 Ack=2 Win=512 Len=0 TSval=1920770192 TSecr=1920770192
5	13.987604461	127.0.0.1	127.0.0.1	TCP	74	53040 → 30553 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=1920782202 TSecr=0 WS=128
6	15.055349003	127.0.0.1	127.0.0.1	TCP	74	30553 → 53040 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=1920783270 TSecr=1920782202 WS=128
7	15.055393553	127.0.0.1	127.0.0.1	TCP	66	53040 → 30553 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1920783270 TSecr=1920783270
8	20.924189674	127.0.0.1	127.0.0.1	UDP	57	20530 → 20529 Len=15
9	20.924394167	127.0.0.1	127.0.0.1	UDP	45	20529 → 20530 Len=3
10	20.925042082	127.0.0.1	127.0.0.1	UDP	45	20530 → 20529 Len=3
11	20.925564884	127.0.0.1	127.0.0.1	TCP	75	53040 → 30553 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=9 TSval=1920789140 TSecr=1920783270
12	20.925587094	127.0.0.1	127.0.0.1	TCP	66	30553 → 53040 [ACK] Seq=1 Ack=10 Win=65536 Len=0 TSval=1920789140 TSecr=1920789140
13	20.926032646	127.0.0.1	127.0.0.1	TCP	118	30553 → 53040 [PSH, ACK] Seq=1 Ack=10 Win=65536 Len=52 TSval=1920789141 TSecr=1920789140
14	20.926158165	127.0.0.1	127.0.0.1	TCP	66	53040 → 30553 [ACK] Seq=10 Ack=53 Win=65536 Len=0 TSval=1920789141 TSecr=1920789141
15	20.926462992	127.0.0.1	127.0.0.1	UDP	51	20529 → 20530 Len=9
16	20.926584103	127.0.0.1	127.0.0.1	UDP	43	20530 → 20529 Len=1
17	20.926844730	127.0.0.1	127.0.0.1	UDP	51	20529 → 20530 Len=9
18	20.927324094	127.0.0.1	127.0.0.1	UDP	43	20530 → 20529 Len=1
19	30.938139236	127.0.0.1	127.0.0.1	UDP	51	20529 → 20530 Len=9
20	30.938272253	127.0.0.1	127.0.0.1	UDP	43	20530 → 20529 Len=1
21	30.939822593	127.0.0.1	127.0.0.1	UDP	51	20529 → 20530 Len=9
22	30.940295501	127.0.0.1	127.0.0.1	UDP	51	20529 → 20530 Len=9
23	30.940275930	127.0.0.1	127.0.0.1	UDP	43	20530 → 20529 Len=1
24	50.955321722	127.0.0.1	127.0.0.1	UDP	51	20529 → 20530 Len=9
25	50.956017617	127.0.0.1	127.0.0.1	UDP	43	20530 → 20529 Len=1
26	50.956801669	127.0.0.1	127.0.0.1	UDP	51	20529 → 20530 Len=9
27	50.956984768	127.0.0.1	127.0.0.1	UDP	51	20529 → 20530 Len=9
28	50.957297219	127.0.0.1	127.0.0.1	UDP	43	20530 → 20529 Len=1
29	50.957712028	127.0.0.1	127.0.0.1	UDP	43	20530 → 20529 Len=1
30	50.957967561	127.0.0.1	127.0.0.1	UDP	51	20529 → 20530 Len=9
31	50.958196046	127.0.0.1	127.0.0.1	UDP	51	20529 → 20530 Len=9
[illegible]	[illegible]	[illegible]	[illegible]	UDP	43	20530 → 20529 Len=1
[illegible]	[illegible]	[illegible]	[illegible]	UDP	51	20529 → 20530 Len=9
[illegible]	[illegible]	[illegible]	[illegible]	UDP	43	20530 → 20529 Len=1
[illegible]	[illegible]	[illegible]	[illegible]	UDP	51	20529 → 20530 Len=9

Google Chrome

218 התראות חדשות

Additional Questions

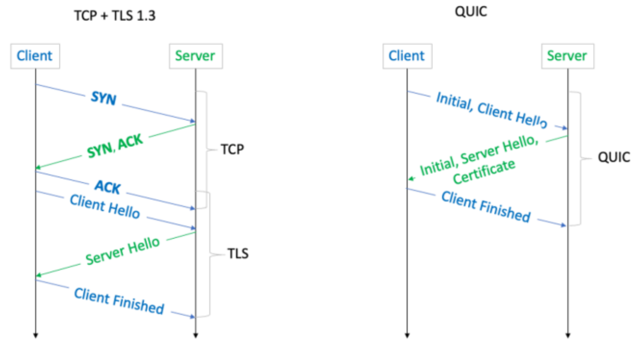
Q1: List at least four major differences between TCP and QUIC protocol

Answer:

1.The QUIC protocol has a heavier security then the TCP due to the fact that he has encryption and Ssl protocol for much more secured communication

2.TCP uses fast retransmit protocol for packet lost while QUIC uses forward error connection for packet lost

3. TCP has the first connection using the three way handshake and then transmiting the data and in the end closing connection , however the QUIC protocol transmits the data together with the first handshake and finishing the communication in the same time.



4.Support for multiplexion - Quic is faster then TCP due to the fact that his support for multiplexing is more integrated into the protocol, allowing multiple streams of data to be transmitted over a single connection more efficiently in contrast to TCP where TCP's congestion control algorithm is designed to treat each connection as a separate entity, which can lead to inefficient use of network resources when multiple connections are used.

Q2: list 2 main diffrences between VEGAS and Cubic

Answer:

a) VEGAS reacts very fast for congstion due to the fact that this mehod decreasing the congestion window the moment it recognize that there is a congestion contrast to CUIBIC which is reacting very slowly when congestion accurs by decreasing the window slowly and gradually.

b) VEGAS Calculates its bandwidth according to the RTT in contrast to CUBIC which adjust its sending rate based on the bandwidth.

Q3: explain the BPG protocol and what is the diffrence between this protocol and the OSPF protocol does BPG operate by short routes .

Answer: Border Gateway Protocol (BGP) is a standardized exterior gateway protocol designed to exchange routing and reachability information among autonomous systems (AS) on the Internet. In BGP, the autonomous system boundary routers (ASBR) send path-vector messages to advertise the reachability of networks. Each router that receives a path vector message must verify the advertised path according to its policy. If the message complies with its policy, the router modifies its routing table and the message before sending the message to the next neighbor. It modifies the routing table to maintain the autonomous systems that are traversed in order to reach the destination system. It modifies the message to add its AS number and to replace the next router entry with its identification. OSPF is used for routing within a single autonomous system (AS) and calculates the shortest path between nodes using a detailed map of the network topology. On the other hand, BGP is used for routing between different ASes and determines the best path by using a list of ASes. While OSPF is designed for faster convergence and more detailed knowledge of the network topology, BGP is designed for scalability in larger networks with many ASes.

In light of what i said above BPG does not operate by the shortest route.

4.

5. DNS is protoco which finds ip based on a domain name or adress compare to ARP sends a broadcast message to all the devices on the network to find the MAC address corresponding to a particular IP address. The device with the matching IP address then replies with its MAC address.

Bibliography

1. https://d1wqtxts1xzle7.cloudfront.net/78185400/cc-27-libre.pdf?1641547994=&response-content-
   disposition=inline%3B+filename%3DPerformance_Analysis_of_TCP_Congestion_C.pdf&Ex-
   pires=1678564827&Signature=Fymxh~k6Kupb4EBtzt1c6YVMSHrfF3kMZzAGsRukK6O21JkkmJ5hVzX-
   LJCjGAVKcnStHuhIP4xLQ-iYaBEJbDWDLxI~cL0HX0nOSC9KHdqM36FdDG5xeOWHOeWphS41IH-
   VvykZHCbj4CqYVIJSuTUTed1LW6zO-fkS8oquYrDdQXCSmM-
   sHsqy2Y~JdsJDVaXRUJ3mVWqqFJM2hvW6BjQrn9DFoyvclYDOqC-aMivnsFoXErOxVr21oOdU36BY-
   PaMLtcdI6wk70X7avo-2YdyZu6amu314~it3Fk1w4pqv2RNlngrUeegtprzd0Swmp-
   dEGFUXbPva2dhuoqTllxxPA__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA -Reno CC algo
2. https://www.ccexpert.us/network-design-2/windowing-and-flow-control.html - window change and flow control
3. https://www.geeksforgeeks.org/flow-control-in-data-link-layer/ - we using option 2 "sliding window flow control"
4. https://www.geeksforgeeks.org/tcp-tahoe-and-tcp-reno/ - explain on reno CC algorithm and photo to see the change in window and ssthresh
5. https://pythontic.com/modules/socket/udp-client-server-example - udp connection between client and server and diagram to show the transfer data between them and code example
6. https://developer.mozilla.org/en-US/docs/Web/HTTP/Status - HTTP status codes list that we need to know when we using requests lib
7. https://docs.python.org/3/library/socket.html - explaining on the socket lib and what each thing do there
8. https://bunny.net/academy/dns/what-is-a-dns-and-recursive-query/ - explain on dns query
9. https://www.studytonight.com/network-programming-in-python/working-with-udp-sockets - udp sockets connection with examples of code and diagrama of the transfer data between them
10. https://www.geeksforgeeks.org/reliable-user-datagram-protocol-rudp/ - explainten on RUDP and all the thing we need to do to make it RUDP (think its very important to use this website and things in here to let me understand that we understand)
11. https://thepacketgeek.com/scapy/building-network-tools/part-09/ - DNS using scapy (you told me you used scapy in dns so if you used this take this)
12. https://www.programcreek.com/python/example/125966/scapy.all.DHCP - DHCP using scapy with code examples
13. https://www.howtouselinux.com/post/understanding-tcp-sequence-number - explain on seq-number
14. https://www.khanacademy.org/computing/computers-and-internet/xcae6f4a7ff015e7d:the-internet/xcae6f4a7ff015e7d:transporting-packets/a/transmission-control-protocol--tcp - tcp explanation and showing the process that happening in the tcp connection
15. https://scapy.readthedocs.io/en/latest/usage.html#dns-requests – dns with scapy
16. https://thepacketgeek.com/scapy/building-network-tools/part-09/ -dns with scapy
17. https://scapy.readthedocs.io/en/latest/api/scapy.layers.dns.html - dns with scapy
18. מודל הקורס
19. חוברת הקורס
20.