Networks Final Projects

Saeed ElSayed Abokhatwa 6829

Hosssam ElDin Ahmed 6721

Ehab Sherif Ahmed 6738

Task: Design and implement a system that simulates TCP packets using a UDP connection; points achieved:

- Build HTTP.
- Used stop and wait.
- Calculated Checksum of packets.
- Implemented packet loss and packet corruption.
- Took special consideration to the following: retransmission, duplicate packet, sequence number, handshake, flags like (SYN, SYNACK, ACK, FIN), and timeout.

Stages of code:

1. Use python UDP socket in server side and client side to start sending client requests

```python
import socket
from utilities import *

serverAddressPort = ("127.0.0.1", 20001)
window_size = 1024

# Create a datagram socket
UDPServerSocket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
UDPServerSocket.bind(serverAddressPort)
tcp_recv(window_size,UDPServerSocket)
```

2. Implementing reliable data transfer principles, starting with generating the TCP header for client to send it

```
def Tcp_send(data,Sequence_number,Acknowledgment_number,Flags,Window,Urgent_pointer,UDPClientSocket,dest_addr,src_addr=('127.0.0.1', 14)):
    #Generate TCP pseudo header
    src_ip, dest_ip = ip2int(src_addr[0]), ip2int(dest_addr[0])
    src_ip = struct.pack('!4B', *src_ip)
    dest_ip = struct.pack('!4B', *dest_ip)

    Reserved = 0

    protocol = socket.IPPROTO_TCP  #is 6 for TCP

    #Check the type of data
    try:
        data = data.encode()
    except AttributeError:
        pass

    src_port = src_addr[1]
    dest_port = dest_addr[1]

    data_len = len(data)

    tcp_length = 20 + data_len   #tcp header length is 20

    checksum = 0
    pseudo_header = struct.pack('!BBH', Reserved, protocol, tcp_length)

    pseudo_header = src_ip + dest_ip + pseudo_header


    TCP_header=struct.pack(f"2H 2I 4H",src_port,dest_port
                ,Sequence_number,Acknowledgment_number,
                Flags,Window,checksum,Urgent_pointer)


    #cpy=pseudo_header + TCP_header + data
```

Psuodo header = source ip address + destination ip address and header we created

3. Extracting / unpacking the data from the header at the server + Stop and wait (client won't send another packet until last packet is received)

```
def tcp_recv(window_size,UDPServerSocket):
    Reserved = 0
    protocol = socket.IPPROTO_TCP

    print("UDP server up and listening")
    # Listen for incoming datagrams
    serverSeqnumber = 0
    expected_seq_num = 1
    three_way_flag=0
    print("Waiting for connection.....")
    flag_get=0
    msg=[]
    while True:
        data, src_addr = UDPServerSocket.recvfrom(window_size)

        size_of_payload=len(data)-20
        unpacked_data=struct.unpack(f"2H 2I 4H {size_of_payload}s",data)

        source_port = unpacked_data[0]
        destination_port = unpacked_data[1]
        sequence_number = unpacked_data[2]
        acknowledgment_number = unpacked_data[3]
        flags = unpacked_data[4]
        window = unpacked_data[5]
        check_sum = unpacked_data[6]
        urgent_pointer = unpacked_data[7]
        payload = unpacked_data[8]
        if three_way_flag==0:
            if flags == 20482:
                print("SYN received, Sending a SYN-ACK")
                serverFlags = 20498
                Tcp_send("",serverSeqnumber,sequence_number+1,serverFlags
                ,window,urgent_pointer,UDPServerSocket,src_addr,src_addr=UDPServerSocket.getsockname())
                data, src_addr = UDPServerSocket.recvfrom(window_size)
                size_of_payload=len(data)-20
                unpacked_data=struct.unpack(f"2H 2I 4H {size_of_payload}s",data)
                flags = unpacked_data[4]
```

4. Implement packet loss and packet corruption

```
            ack = sequence_number+size_of_payload
            ack = str(ack)
            clientAddressPort=src_addr
            UDPServerSocket.sendto(ack.encode(), clientAddressPort)
            print('Received packet:', payload)
            expected_seq_num = ack
        else:
            print('Received duplicate packet:', sequence_number)
    else:
        print("Message corrupted,waiting for a retransmission")
    time.sleep(1.2)
    if random.random() < 0.3:
        time.sleep(1.2)
```

```
if random.random() > 0.5:
      UDPClientSocket.sendto(TCP_header+data, dest_addr)
      #print("Packet sent")
    else:
        print("Packet lost")
```

5. To solve packet corruption problem, we used the checksum

```
def checksum_func(data):
    checksum = 0
    data_len = len(data)
    if (data_len % 2):
        data_len += 1
        data += struct.pack('!B', 0)

    for i in range(0, data_len, 2):
        w = (data[i] << 8) + (data[i + 1])
        checksum += w

    checksum = (checksum >> 16) + (checksum & 0xFFFF)
    checksum = ~checksum & 0xFFFF
    return checksum


def verify_checksum(data, checksum):
    data_len = len(data)
    if (data_len % 2) == 1:
        data_len += 1
        data += struct.pack('!B', 0)

    for i in range(0, data_len, 2):
        w = (data[i] << 8) + (data[i + 1])
        checksum += w
        checksum = (checksum >> 16) + (checksum & 0xFFFF)

    return checksum
```

6. We first used the flags to implement 3 way handshake (SYN, ACK, SYNACK)

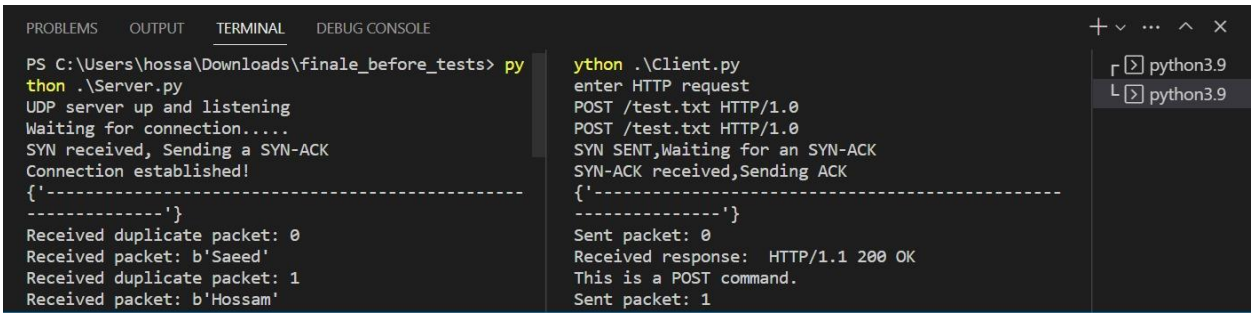
We assumed sequence number and acknowledgment number the client sends are initially: 0

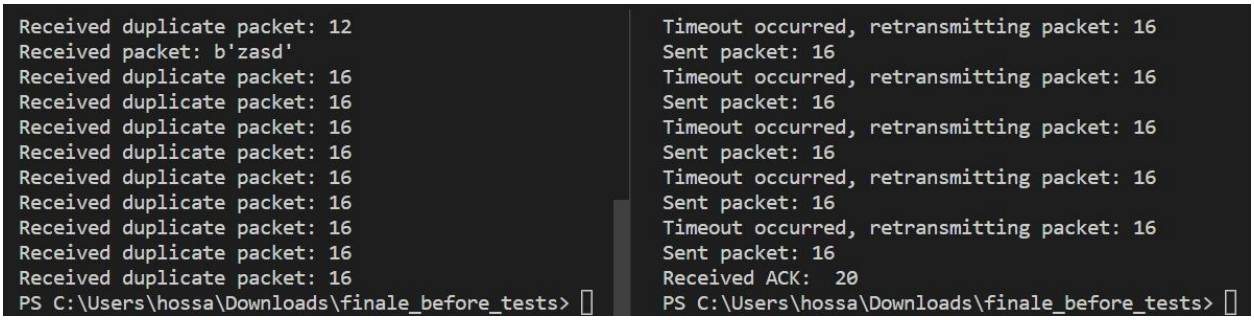
```
while three_way_flag==0:
    UDPClientSocket.settimeout(2.0)
    flags = 20482
    Source_port=clientAddressPort[1]
    Destination_port=serverAddressPort[1]
    Sequence_number=datapointer
    data_t=""
    Acknowledgment_number=0
    Window=1024
    Urgent_pointer=0
    Tcp_send(data_t,Sequence_number,Acknowledgment_number,flags
            ,Window,Urgent_pointer,UDPClientSocket,serverAddressPort,src_addr=clientAddressPort)
    print("SYN SENT,Waiting for an SYN-ACK")
    time.sleep(1.2)
    try:
        synack,addr = UDPClientSocket.recvfrom(Window)
    except socket.timeout:
        print("Couldn't connect to the server, Retrying to connect!")
        continue
    size_of_payload=len(synack)-20
    unpacked_data=struct.unpack(f"2H 2I 4H {size_of_payload}s",synack)
    flags = unpacked_data[4]
    sequence_number = unpacked_data[2]
    if flags==20498:
        flags = 20496
        print("SYN-ACK received,Sending ACK")
        Tcp_send(data_t,datapointer,sequence_number+1,flags
            ,Window,Urgent_pointer,UDPClientSocket,serverAddressPort,src_addr=clientAddressPort)
        three_way_flag=1
    print({"------------------------------------------------------------"})
    time.sleep(1.5)
```

7. To handle packet loss, packet retransmission, duplicate packet, we used sequence number and acknowledgment number (implemented in header) We assumed the sequence number and the acknowledgment number of the first packet sent are initially: 1

```
def Tcp_send(data,Sequence_number,Acknowledgment_number,Flags,Window,Urgent_pointer,UDPClientSocket,dest_addr,src_addr=('127.0.0.1', 14)):
    #Generate TCP pseudo header
    src_ip, dest_ip = ip2int(src_addr[0]), ip2int(dest_addr[0])
    src_ip = struct.pack('!4B', *src_ip)
    dest_ip = struct.pack('!4B', *dest_ip)

    Reserved = 0

    protocol = socket.IPPROTO_TCP  #is 6 for TCP

    #Check the type of data
    try:
        data = data.encode()
    except AttributeError:
        pass

    src_port = src_addr[1]
    dest_port = dest_addr[1]

    data_len = len(data)

    tcp_length = 20 + data_len   #tcp header length is 20

    checksum = 0
    pseudo_header = struct.pack('!BBH', Reserved, protocol, tcp_length)

    pseudo_header = src_ip + dest_ip + pseudo_header


    TCP_header=struct.pack(f"2H 2I 4H",src_port,dest_port
                ,Sequence_number,Acknowledgment_number,
                Flags,Window,checksum,Urgent_pointer)


    #cpy=pseudo_header + TCP_header + data
```

8. Implemented HTTP Request and Responses





9. Use FIN flag to close the connection

```
flags = 20481
Sequence_number=0
data_t=""
Acknowledgment_number=0
Window=1024
Urgent_pointer=0
Tcp_send(data_t,Sequence_number,Acknowledgment_number,flags
            ,Window,Urgent_pointer,UDPServerSocket,src_addr,src_addr=UDPServerSocket.getsockname())
fin,addr = UDPServerSocket.recvfrom(Window)
size_of_payload=len(fin)-20
unpacked_data=struct.unpack(f"2H 2I 4H {size_of_payload}s",fin)
flags = unpacked_data[4]
if flags == 29496:
    UDPServerSocket.close()
    break
```