Project 2 report

I. Implementation and Code Overview

To run the script just type in:

```
$python3 Multimedia_Content_Retrieval.py
```

As requested, using the same method as in the last programming task a UDP socket was created for sending and receiving packets.

```
#creating the socket -> Task 1
sock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
address = "129.187.223.200"
port = 3000
```

For retrieving the text, a packet was sent containing the TEXT string encoded. (A timeout of 10 seconds was always checked)

While the socket containing the end of the file wasn't received, the received text was written to a file named TextFile.txt (precautions were taken to take the end of the received text as not to leave white spaces ('\0')).

In the end, the file was read as to display the contents of such.

```
#Retrieving the text -> Task 2
def task2():
    s = ''
    data = ""
    packet = "TEXT"
    sock.settimeout(10);
    read1 = True
    sock.sendto(packet.encode(),(address,port));
    #while True:
    #    data, addr = sock.recvfrom(1024) # buffer size is 1024 bytes
    #    print ("received message:", data)
    file = open('TextFile.txt','w')
    while (s != 'END\x00'):
        time_out = 0;
        try:
            #print("entrei")
            data = sock.recv(900);
            s = data.decode()
            if (s != 'END\x00'):
                file.write(s.rstrip('\0'))
        except socket.timeout:
            #print("sai")
            read1 = False
            time_out = time_out + 1;
        ts1 = int(time.time_ns());
        if(not read1):
            continue
    file.close()
    file = open('TextFile.txt','r')
    print(file.read())
    file.close()
```

Then, a packet containing "MUSIC STU_ID" was sent in order to retrieve a music file.

To do so (and again with checked timeouts) while the end of the file was not reached, a socket was received and its data stored and parsed.

After this, the data was ordered by their sequence number and written into filemp3.

```
#read RTP traffic of an audio file -> Task 3
# MUSIC STU_ID
def task3():
    global payload
    payload=[]
    packet = "MUSIC 03714404"

    read1 = True
    sock.sendto(packet.encode(),(address,port));
    s = ""
    while (True):
        time_out = 0;
        try:
            #print("entrei")
            data = sock.recv(900);
            #s = data.decode()
            try:
                if( data.decode() == 'END\x00'):
                    print("Last Packet")
                    break
            except:
                parse_packet(data)
                continue
        except socket.timeout:
            #print("sai")
            read1 = False
            time_out = time_out + 1;
        ts1 = int(time.time_ns());
        if(not read1):
            continue

    with open("sound.mp3", "ab") as filemp3:
        print("writing")
        payload.sort(key=operator.itemgetter('seq'))
        realtotalpackets = len(payload)
        expectedpackets =  payload[len(payload)-1]["seq"] - payload[0]["seq"] + 1
        print("total/expected -> " + str(realtotalpackets) + ":" + str(expectedpackets))
        if(realtotalpackets < expectedpackets):
            print("Packets missing")
        for x in payload:
            filemp3.write(x["data"].tobytes())
        md5func("sound.mp3")
        filemp3.close()
```

To do the parsing a function which receives a packet as parameter and returns the data and its sequence number was created.

```
def parse_packet(packet):
    bit_array = bitstring.BitArray(bytes = packet)
    version = bit_array[0:2].uint
    padding = bit_array[3]
    extension = bit_array[4]
    csrc_count = bit_array[4:8].uint
    marker = bit_array[9]
    payload_type = bit_array[9:16].uint
    sequence_number = bit_array[16:32].uint
    time_stamp = bit_array[32:64].uint
    ssrc = bit_array[64:96].uint
    #cids = []
    #temp_count = 96
    #for x in range(csrc_count):
    #    cids.append(bit_array[temp_count:temp_count+32].uint)
    #    temp_count += 32
    data = bit_array[96:]
    payload.append({"seq":sequence_number,"data":data})
```

After this, a function for task 4 was created.

The goal of this function is to request a compressed video clip from the server.

The logical sequence is the same as in task 3 for the retrieval.

The video without retransmission is then store.

Afterwards lost packets were considered. To do so, if the number of expected (calculated through the sequence numbers of the first and the last packets) vs obtained packets is different than a recursive trial function is called. Then the video with retransmissions was stored.

```
def task4():
    global payload
    payload=[]
    packet = "VIDEO 03714404"
    realtotalpackets = 0
    expectedpackets = 0
    read1 = True
    sock.sendto(packet.encode(),(address,port));
    s = ""
    while (True):
        time_out = 0;
        try:
            #print("entrei")
            data = sock.recv(900);
            #s = data.decode()
            try:
                if( data.decode() == 'END\x00'):
                    print("Last Packet")
                    break
            except:
                parse_packet(data)
                continue
        except socket.timeout:
            #print("sai")
            read1 = False
            time_out = time_out + 1;
        ts1 = int(time.time_ns());
        if(not read1):
            continue
    payload.sort(key=operator.itemgetter('seq'))
    realtotalpackets = len(payload)
    expectedpackets =  payload[len(payload)-1]["seq"] - payload[0]["seq"] + 1

    with open("video_no_retransmission.bin", "ab") as movie:
        print("writing")

        print("total/expected -> " + str(realtotalpackets) + ":" + str(expectedpackets))
        for x in payload:
            movie.write(x["data"].tobytes())
        movie.close()

    if(realtotalpackets < expectedpackets):
        recursivetrial()

    realtotalpackets = len(payload)

    with open("video_with_retransmission.bin", "ab") as movie:
        print("writing")

        print("total/expected -> " + str(realtotalpackets) + ":" + str(expectedpackets))
        for x in payload:
            movie.write(x["data"].tobytes())
        movie.close()
```

The function for recursive trial checks for missing elements by calling a function that does so.

For all the missing elements it asks for retransmission and stores them in the payload array. (orderly as explained in task 3).

```
def recursivetrial():
    numbers = []
    for v in payload:
        numbers.append(v["seq"])
    for missing in missing_elements(numbers):
        packet = "R " + str(missing)
        #print(packet)
        sock.sendto(packet.encode(),(address,port));
        try:
            #print("entrei")
            data = sock.recv(900);
            #s = data.decode()
            try:
                if( data.decode() == 'END\x00'):
                    print("Last Packet")
                    break
            except:
                parse_packet(data)
                continue
        except socket.timeout:
            #print("sai")
            continue
    payload.sort(key=operator.itemgetter('seq'))
    realtotalpackets = len(payload)
    expectedpackets =  payload[len(payload)-1]["seq"] - payload[0]["seq"] + 1
    if(realtotalpackets < expectedpackets):
        numbers = []
        recursivetrial()
    else:
        return
```

The missing elements function calculates the missing elements by receiving a vector of numbers (sequence numbers in our case) and calculating (and) sorting the missing ones through differences.

```
def missing_elements(L):
    start, end = L[0], L[-1]
    return sorted(set(range(start, end + 1)).difference(L))
```

At the end the task2, 3 and 4 are called.

```
task2()
task3()
task4()
```

II. Results

After the described coding for task 3 and 4 were completed as described above, it some extra tasks were requested.

For task 3 we were expected to play out the music file using any music player. If the file played out properly, which it did, then we had successfully achieved the task.

Afterwards we were expected to calculate the md5sum of the downloaded music file. Despite attempting to do this directly through python (the attempt is available in the source code) the provided tool was used as the result was as follows:

```
sound.mp3                         87877a37a54346e237794de8a52a824a
```

For task 4 we were supposed to download the video decoder from the web page and decode the two compressed video files. Then we were requested to play the decoded .yuv files to understand the influence of packet loss on video applications.

At the end we were supposed to calculate the PSNR of the two video files which we could get at the end of the decoding process. The results were as follows:

- Video without retransmissions:

```
-------------------- Average SNR all frames ------------------------------
 SNR Y(dB)           : 33.21
 SNR U(dB)           : 39.79
 SNR V(dB)           : 40.75
 Total decoding time : 19.546 sec
--------------------------------------------------------------------------
 Exit JM 10 (FRExt) decoder, ver 10.2
```

- Video with retransmissions:

```

-------------------- Average SNR all frames ------------------------------
 SNR Y(dB)           : 37.88
 SNR U(dB)           : 40.92
 SNR V(dB)           : 41.30
 Total decoding time : 20.353 sec
--------------------------------------------------------------------------
 Exit JM 10 (FRExt) decoder, ver 10.2
```

#####