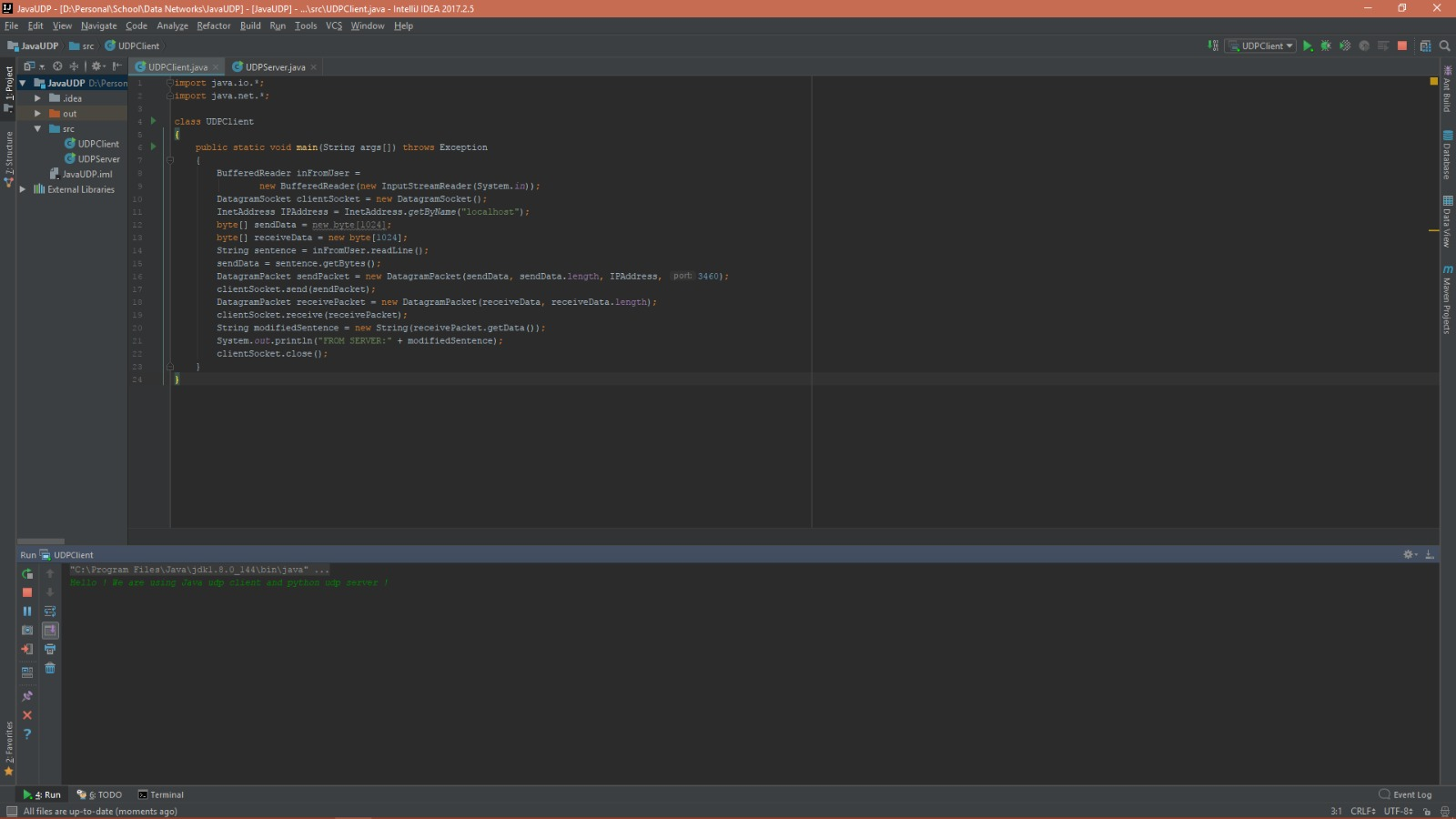
Datanetworks Assignment 1: UPD client/server

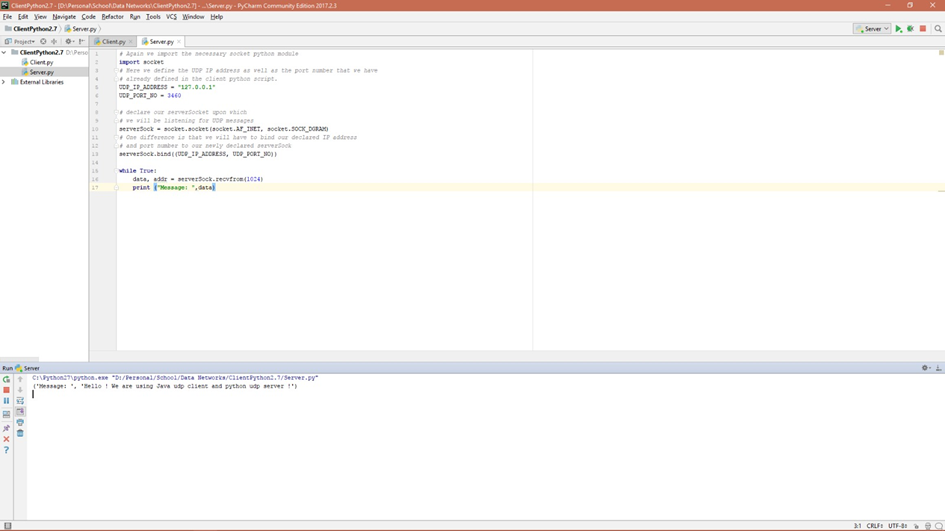
Names: Miroslav , Gechev 429556  
 Danny, Drogt 408374

Choice of language: java and python   
Java and python has been chosen for the assignment as both students understood atleast one of these languages completely. Given the fact that the assignment has to be written in two languages we believed it was a good choice to take atleast the language we believe we understand the best.

As we never created a server and client ever before, a lot of time has been following tutorials and guides on how to set up the client and server. A top down approach has been followed. Meaning that first the variables were defined (ip adress, port adress etc). followed by setting them up in a connection and at last trying to send the data. We decided to use java as client and used its readline function to send strings to the python server. We felt it was a bit easier to use java as client and python as server than the other way around. For that reason, we decided to do it this way. The server and client are rather small in terms of lines of code. With both not costing more than 30 lines of code. In the screenshot more can be found.

A small problem was found in the beginning when trying to create a multilanguale connection. At first, we decided it would be best to understand a single language connection. However, we forgot that because of that a port was already being used. Which resulted in us being faced with a couple of errors which turned out to be trying to use a port which was already in use. Easiest solution was used resulting in us just using a different port.





**Java Client**

import java.io.\*;

import java.net.\*;

class UDPClient

{

public static void main(String args[]) throws Exception

{

BufferedReader inFromUser =

new BufferedReader(new InputStreamReader(System.in));

DatagramSocket clientSocket = new DatagramSocket();

InetAddress IPAddress = InetAddress.getByName("localhost");

byte[] sendData = new byte[1024];

byte[] receiveData = new byte[1024];

String sentence = inFromUser.readLine();

sendData = sentence.getBytes();

DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, IPAddress, 3460);

clientSocket.send(sendPacket);

DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);

clientSocket.receive(receivePacket);

String modifiedSentence = new String(receivePacket.getData());

System.out.println("FROM SERVER:" + modifiedSentence);

clientSocket.close();

}

}

**Server in python**

# Again we import the necessary socket python module

import socket

# Here we define the UDP IP address as well as the port number that we have

# already defined in the client python script.

UDP\_IP\_ADDRESS = "127.0.0.1"

UDP\_PORT\_NO = 3460

# declare our serverSocket upon which

# we will be listening for UDP messages

serverSock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

# One difference is that we will have to bind our declared IP address

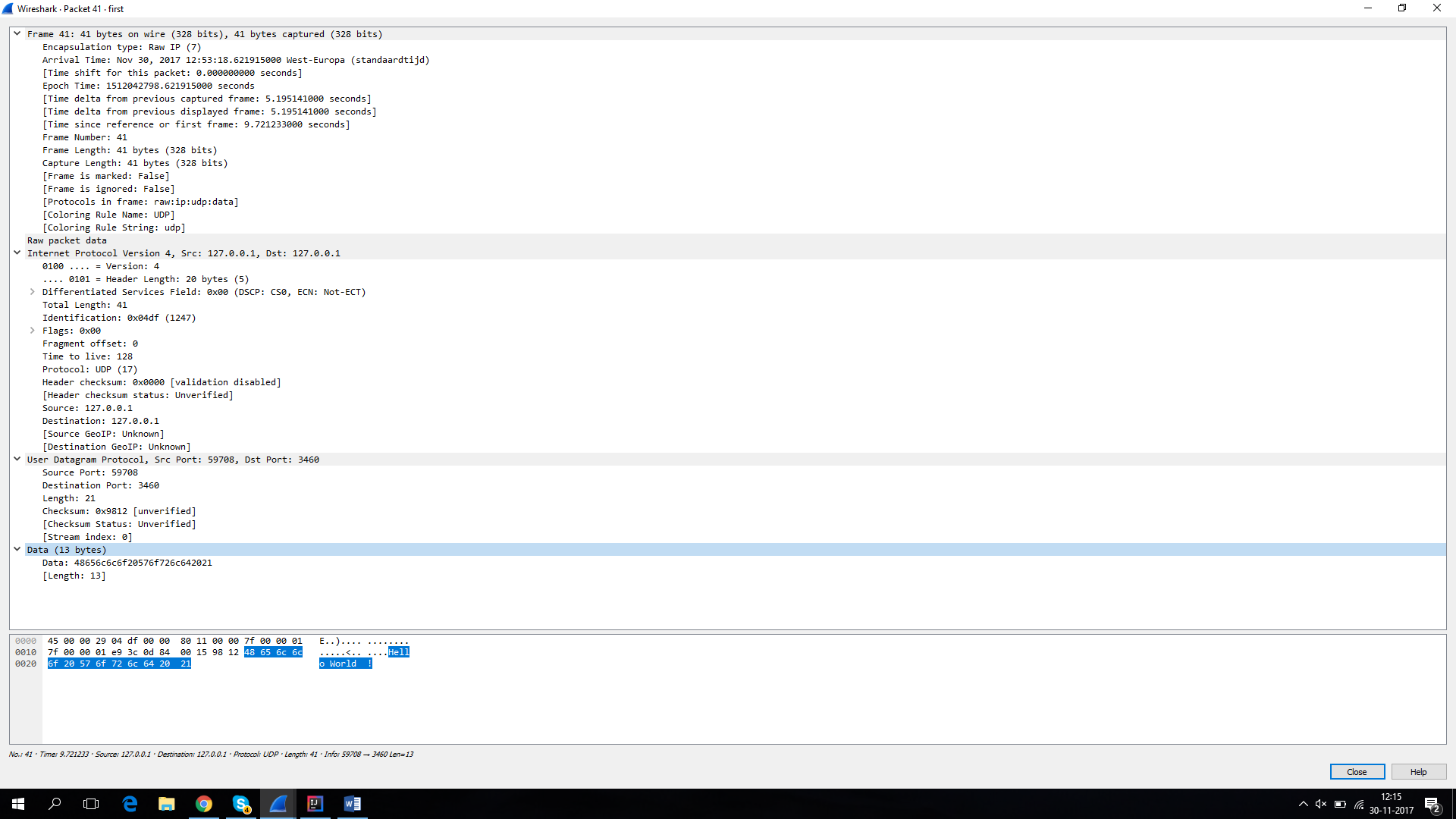
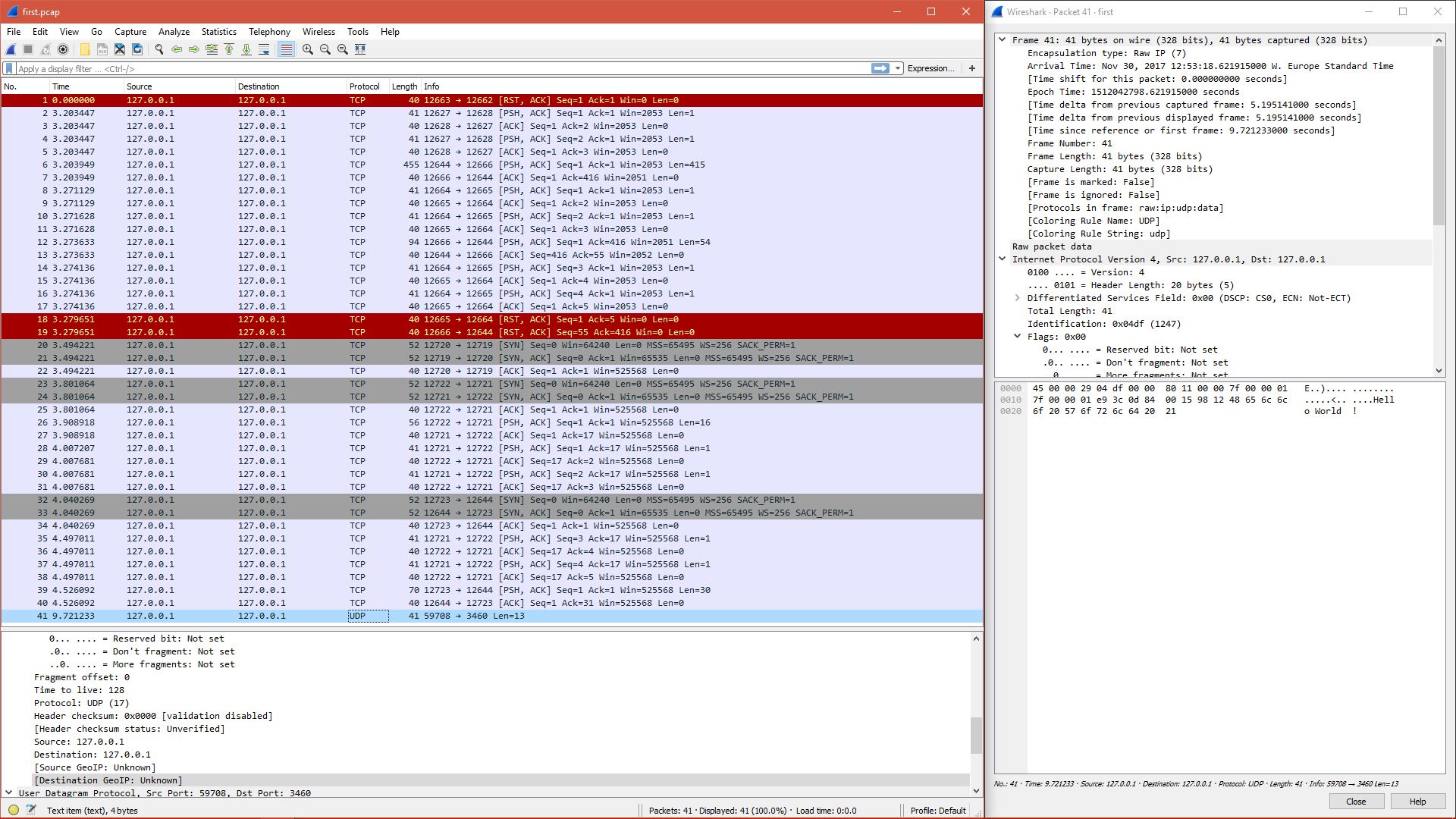
# and port number to our newly declared serverSock

serverSock.bind((UDP\_IP\_ADDRESS, UDP\_PORT\_NO))

while True:

data, addr = serverSock.recvfrom(1024)

print ("Message: ",data)



We get alot of information in the whireshark overview. First of all we see that the encapsulation type is RaW IP. Meaning that the header of the sended package consists either of ipv4 or ipv6 (in this case ipv4 as can been seen under internet protocol tab. Protocols in frame show that the way the data package is build is in the form: raw ip, udp, data. Which makes sense. So at first the destination is set (the raw ip) followed by the way the machine talks (following protocol udp) and at last what the machine has to say (the data itself). At the internet protocol tab we see src: 127.0.0.1 and dst:127.0.0.1 which would imply that the data has been send and arrived at the same adress. Meaning it was done locally. By clicking on the different tabs one can see which parts of the package belong to which layers. Clicking on internet protocol version 4 shows the first row (together with a part of the second) completely belongs to ipv4. Under udp tab we see that we send data to our predetermined port which is 3460 but we have send that same data from a random port which was 59708. Each tap also keeps the length of it’s data in the package (presuming it is in bytes). Ipv4 also got in its data string an identification code to determine it is reading the ipv4.