Practical Work 01

UDP/TCP messages with Python

Objectives : exchange information over standard computer network with UDP and TCP IP sockets using python throughs examples like a simple chat application and a micro HTTP server.

Knowledge: sockets, OSI layer 3 (IP-Network), layer 4 (Transport UDP/TCP) and layer 7 (Protocol) and python programing language.

1 Setting up

If you know how to write and launch multiple python scripts simultaneously you can go to section 2.

1.1 Setting python development environment

If you don't have python 3 on your computer the standard python development kit is sufficient and can be downloaded with this link (python 3.x) : https://www.python.org/downloads/ . (you can also download more advanced distributions with additional tools like anaconda https://www.continuum.io/downloads etc.).

Install python distribution, launch python script editor idle (it has been installed by default with python), then create a new files (File – new file). Now write on that file :

```
print("Network courses are perfect !")
```

Save the file using the extension .py at the end of your file. Click on Run-Run Module… the script should be executed. Otherwise find help.

1.2 Multiple IDLE instances

In our lab we wanna execute the client script and also the server script at the same time (i.e. in parallel). We need two instance of python where one will execute the client and the other one will execute the server.

Lots of IDE like Spyder and Pycharm allows it easaly, but if you are using Idle you'll need to execute it twice. Here are solutions to launch two instances of Idle to be able to run two python scripts in parallel.

1.2.1 On Mac/Linux :

To lunch several Idle instance you can use the terminal application :

Launcher → other → Terminal

or use: keyboard shortcut cmd+space and then write Terminal then enter.

On the terminal type:

```
idle3 &
```

as many time as you want to have Idle instance.

1.2.2 On Windows :

Launch IDLE : press the windows key, then write i d l e , then press enter
On the taskbar « barre des tâches », right click on the idle icon, then click on the top line containing the text “idle”.

2 Exercises

First with UDP then with TCP.

2.1 UDP

Here are two python scripts implementing a simple udp message exchange.

The first script is a udp receiver. It binds its udp socket to a local IP/PORT, witch means that it asks to the OS if it could associate its socket to a given PORT on one IP of the machine. Then it waits for incoming messages (here utf8 text) on the socket and displays it infinitely.

```
""" Simple UDP receiver
"""
import socket

UDP_IP = "127.0.0.1"
UDP_PORT = 50005

sock = socket.socket(socket.AF_INET, # Internet
                     socket.SOCK_DGRAM) # UDP
sock.bind((UDP_IP, UDP_PORT))

while True:
    data, addr = sock.recvfrom(1024) # buffer size is 1024 bytes
    print("received message:", data.decode('utf8'))
    print("from:",addr)
```

The second script sends “hello world” text encoded using utf-8 to a given IP/PORT.

```
""" Simple UDP sender
"""

import socket

UDP_IP = "127.0.0.1"
UDP_PORT = 50005
MESSAGE = "Hello, World!"

print("UDP target IP:", UDP_IP)
print("UDP target port:", UDP_PORT)

sock = socket.socket(socket.AF_INET, # Internet
                     socket.SOCK_DGRAM) # UDP
sock.sendto(MESSAGE.encode('utf8'), (UDP_IP, UDP_PORT))
```

2.1.1 Test those scripts until you see “hello world” on a console.

NB : using the bind function may trigger a warning from your Firewall. You should allow incoming messages/ connections with your firewall or disable it for this practical work.

We now want to listen incoming messages on the network. For us, the network is the IONIS LAN witch is reachable through IONIS or IONIS Portal Wifi SSID.

2.1.2 Make an UDP listener called “remote_sensors.py” that listen on port 52001 on IONIS LAN and displays received messages. The messages are send encoded with ascii code (so you should decode it accordingly).

NB: the messages are sent to the LAN broadcast address (ie. 10.9.255.255)

on MacOS you should bind your socket to the broadcast address and not your ip address, while

on windows you should bind your socket to your IP address

NB2 : Firewall -

Your firewall may bloc incomming traffic . You may need to add appropriate rules to allow incomming messages or connection. On Windows type win+r then wf.msc then add a rule to allow incomming UDP traffic on parts 52001 and 52002 . If it is still not working, verify if there is no rule bloking network for python application.

2.1.3 Drone data frame

Now we are going to recieve and decode data from sensords of a drone.Change the script to display incoming messages from port 52002.

The data from the drone are transmited as raw bytes, i.e. not in text encoding and follow this description :

date – pitch – roll – yaw – battery – barometer – agx- agy – agz

the date is in a double precision float (8bytes)

the pitch,roll,yaw and battery are stored in an int (4 bytes)

barometer, agx,agy,agz are stored as single precision floats (4bytes)

The easyest way to exctract the fields of this kind frame in python is to use the struct module. In the following example the frame is composed of one double (d), followed by an int (i), then a float (f)

```
import struct
format = 'dif'   # byte array format : d for double , i for int, f for float
decoded = struct.unpack(format,recieved_bytes)
# or : decoded = struct.unpack(format,recieved_bytes[0:40])
```

Now adapt your code to read, unpack and display drone data broadcasted to port 52002.

For addictionnal information see:
Documentation on struct module https://docs.python.org/3/library/struct.html
Format characters: https://docs.python.org/3/library/struct.html#format-characters

To diplay date you can follow this example :

```
import datetime
print(datetime.datetime.fromtimestamp(decoded[0]))
```

2.1.4 Now you need to use two sockets with the select function to be able to read the two streams for the samer script

The problem here is that the recvfrom() function is a blocking function. If there is nothing to read the function blocks the script. There is mainly two solution to manage this issue: using threads (we'll see that on the next Practical Work) or using the select function (https://docs.python.org/3.7/library/select.html) or combine the two solution (i.e. threads + select)

The select function take as arguments 3 lists of sockets (or file etc.) and waits until those socket are "readable" "writable" or "triggering exception".

The select function returns a tuple containing 3 lists of socket when one socket is ready. To use this function to know witch socket is ready to be red we can do like this:

```
import select
#(...)
sock1 = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock2 = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
#(...)
```

```
socketList = [sock1,sock2]

while True:
        ls = select.select(socketList,[],[],1)
        for s in ls[0]:
            data, addr = s.recvfrom(1024) # buffer size is 1024 bytes
            print("#message from:",addr)
            print("#message to  :",s.getsockname())
            print("#message     :", data)   # depending on getsockname you should adapt
                                            # your decoding method
```

2.1.5 change your script to use the select function and avoid missing messages.

2.1.6 Messages are incoming with different speed rate. Tell us what is the port that receive messages with highest speed and the one used to receive messages with lowest speed :

3 client server TCP / IP using python

3.1 Preamble:

3.1.1 What is the 127.0.0.1 IP address? What is-it used for ?

3.1.2 The source code of TCP client and server are available on the annexes.

According to your knowledge, which code corresponds to the client and which one to the server? Why ?

3.1.3 On server side two sockets are used. Explain why :

3.1.4 Add the following lines one the client or server script to the correct place:

```
print("binding to "+ TCP_IP +":"+str(TCP_PORT)," ...")
print("Waiting client ...")
print('Client connected with address:', addr)
print('connecting to '+ TCP_IP + ':' + str(TCP_PORT) +'...')
print('sending data...')
```

3.1.5 Modify the IP addresses and the port number as follow: 127.0.0.1 and 55000. Then test the scripts.

PS : You need to have two instances, one for the client and the other one for the sever. You will need to use the terminal to have the two instances.

3.2 Mini chat

3.2.1 On your machine (using 127.0.0.1):

Modify the TCP client and server to enable users to write the text that will be send (with the input() function).

Then use a while loop to make client and server send and receive messages infinitely.

In our implementation the client and the server will not be able to send en receive message in parallel . To enable this we need to use thread wich is not covered in this pratical work.

It means that after sending a message, you need to wait for a response before sending a new message.

3.2.2 Choose a colleague :

Choose a colleague, connect to the same LAN, identify your IP addresses and test your mini chat with your colleague. On the server side, you may need to authorize the entering/incoming connections in your firewall.

4 Micro HTTP server

Copy your implemented version of the TCP server and save it in a new file. You can name it as microHTTPsrv.py.

1 – Modify this server script to make it displays the received client messages.

2 – You will now use a web browser as the TCP client. Try to connect to the server using a web browser : simply write the ip address and prot number of your server in the address field of your web browsser (for example:

127.0.0.1:55000/

Now observe the message received by the server. .

3 – In the server side , send the following reply to the client

```
import time
http_head =  "HTTP/1.1 200 OK\r\n"
http_head += "Date:"+ time.asctime() +"GMT\r\n"
http_head += "Expires: -1\r\n"
http_head += "Cache-Control: private, max-age=0\r\n"
http_head += "Content-Type: text/html;"
http_head += "charset=utf-8\r\n"
http_head += "\r\n"
data =  "<html><head><meta charset='utf-8'/></head>"
data += "<body><h1>In43 is the best course ! ÉÇ </h1>"
data += "</body></html>\r\n"
data += "\r\n"
http_response = http_head.encode("ascii") + data.encode("utf-8")
```

5 ANNEXES

5.1 Client/Server TCP (python 3)

```
""" IPSA IN43 TC 2016 - TP01 - Client/Server
    ????? Mini Client/Server using TCP ?????
    from https://wiki.python.org/moin/TcpCommunication
"""
import socket

TCP_IP = '127.123.234.1'
TCP_PORT = 50005
BUFFER_SIZE = 1024

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

sconn.bind((TCP_IP, TCP_PORT))
sconn.listen(1)

s, addr = sconn.accept()
rawdata = s.recv(BUFFER_SIZE)
print("received data:", rawdata.decode('ascii'))
s.send(rawdata)  # echo

s.close()
sconn.close()
```

```
""" IPSA IN43 TC 2016 - TP01 - Client Serveur
```

```
    ????? Mini Client/Serveur en TCP ?????
    from https://wiki.python.org/moin/TcpCommunication
"""
import socket

TCP_IP = '127.123.234.1'
TCP_PORT = 50005
BUFFER_SIZE = 1024                      # read size
msg = "Hello, Everyone!"

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((TCP_IP, TCP_PORT))

print('sending data...')
s.send(msg.encode('ascii'))

rawdata = s.recv(BUFFER_SIZE)
print("received data:\n", rawdata.decode('ascii'))

s.close()
```

5.2 Notes on IPv4 adresses with python :

127.0.0.1 → localhost (the machine itself)

255.255.255.255 or '<broadcast>' are used to broadcast UDP msg on the LAN

on a 10.9.0.0/16 network, the broadcast address is 10.9.255.255

0.0.0.0 or '' are used to desigante all local IP.

5.3 Some instructions with python

5.3.1 Infinite loop that display users inputs until « end » is written:

```
while data != 'end':
    data = input()
    print(data)
```

5.3.2 Obtain the machine name :

```
import socket
name=socket.gethostname()
```

full machine name :

```
name=socket.gethostbyaddr(socket.gethostname())[0]
```

5.3.3 Obtain the user name :

```
import getpass
```

```
user=getpass.getuser()
```

5.3.4 Obtain the date or the current time :

cf : https://docs.python.org/3.4/library/time.html

```
import time
time.localtime()
time.gmtime()
time.strftime("%H:%M:%S")
time.asctime()
```

5.3.5 closing properly socket used in a while…

Most network app use a while True : structure (an infinit loop). So it is hard to get out of this loop to call the socket close() fonction.

One way to manage this problem is to use the try/except structure like this :

```
try :
    YOU PROGRAM CODE HERE USING SOCKET s

except KeyboardInterrupt:
    print("Stopping due to signint interupt (i.e. ctrl-c)")
    s.close()
```