# Sensing and Actuation Networks and Systems [2021-2022]

## *Assignment 01 – Socket programming in Python*

### Introduction

This assignment aims to get familiarised with the TCP/IP protocol stack and basic networking concepts. The basic communication using the client/server model, for both TCP and UDP, is exemplified using the Python sockets API. Finally, a simple network traffic analysis is performed to corroborate the concepts discussed on this assignment.

### Objectives

Students successfully concluding this work should be able to:

- Comprehend basic concepts of networking and the TCP/IP stack
- Distinguish the differences between TCP and UDP
- Understand the basics of socket programming using Python
- Perform fundamental network traffic analysis using `netstat` and `wireshark`

### Support Material

- Slides explaining the TCP/IP protocol stack
- A video describing the differences in communications using TCP and UDP
- Source code using the Python sockets API for client/server examples using TCP and UDP

## Socket API

- socket()	# creates a new socket
- bind()	# bind a socket to a specific IP address and port
- listen()	# makes a socket ready for accepting connections
- accept()	# accepts connections in a specific socket
- connect()	# connect to a specific address using a socket
- send()	# send data to socket
- recv()	# read data from socket
- close()	# close socket
- recvfrom()	# reads a number of bytes from and UDP socket
- sendto()	# send datagrams to an UDP socket
- gethostbyname()	# returns IP address of host
- gethostbyaddr()	# given an IP address returns hostname, alias list, IP address

Further details on sockets in Python can be found at:
https://docs.python.org/3/library/socket.html

## UDP sockets overview

Figure 1 depicts the timeline of a typical scenario between a UDP client and server. UDP only adds another layer of addressing (ports) on top of the logical (IP) address, and detect and discard datagrams with errors. The server initiates first and then any UDP client can connect, sending requests/messages to the server, which may respond, until the client closes the connection. When an application wants to connect with another, it requests the operating system to create an instance of socket by invoking the function `socket()`, indicating the communication protocol and the socket type to use.

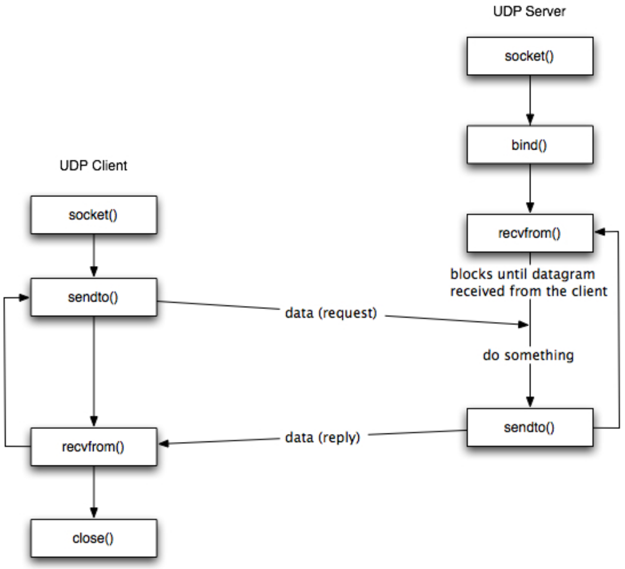


Figure 1. UDP sockets overview[1]

[1] https://www.cs.dartmouth.edu/~campbell/cs60/socketprogramming.html

The bind() function assigns the protocol address to the socket (IP address and port). An UDP socket can be used as soon as it is created (unlike TCP sockets). Finally, the close() function is invoked when the application finishes using the socket.

## TCP sockets overview

Figure **2** shows the timeline of a typical TCP connection. TCP also adds an additional addressing layer on top of IP, but also offers a *stateful service* and has a different operational mode than UDP, thus, the implementation of TCP sockets differs from UDP sockets.

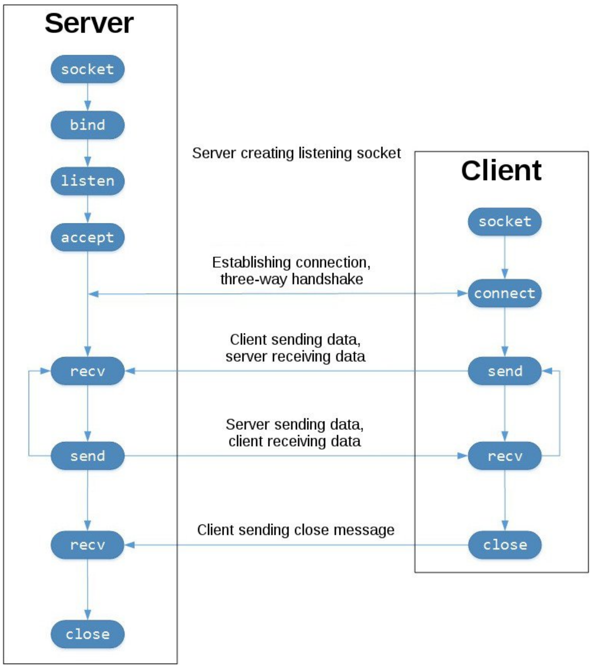

Server
socket
bind
listen
accept
recv
send
recv
close
Server creating listening socket
Establishing connection, three-way handshake
Client sending data, server receiving data
Server sending data, client receiving data
Client sending close message
Client
socket
connect
send
recv
close

Figure 2. TCP sockets overview[2]

The socket() and bind() functions work exactly the same as with UDP, except they receive different parameters to indicate this is a *statefull connection* (TCP). The function listen() indicates to the operating system that this socket is ready to accept incoming connections.

TCP is a *connection-oriented* protocol, thus it handles the concept of connections that each endpoint must accept. The accept() function returns the following connection in the queue assigned to a given socket. The connect() function is invoked by the client to establish a connection with the server. The close() function closes the socket once the application finishes using it.

## Python installation

The examples in this assignment use Python. If Python is not installed in your machine, download it from https://www.python.org/downloads/ .

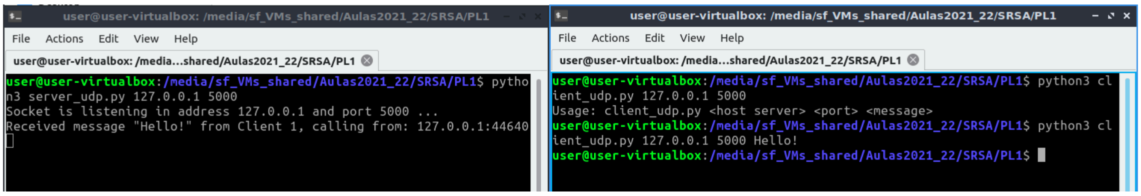
[2] https://realpython.com/python-sockets/

In some operating systems `python` is executed as `python3`. Refer to your version documentation for details.

## UDP example

**Goal:** Understand how the UDP protocol works using the Python sockets API.

**Activities:** Download the example code from UC Student (`server_udp.py`, `client_udp.py`) and review it. Determine your local IP address and run the server program using the following command: `python server_udp.py <local_address> <port>`. From another shell or computer, run the client program using the following command: `python client_udp.py <server_address> <port> <message>`.


user@user-virtualbox: /media/sf_VMs_shared/Aulas2021_22/SRSA/PL1
File Actions Edit View Help
user@user-virtualbox: /media...shared/Aulas2021_22/SRSA/PL1
user@user-virtualbox: /media/sf_VMs_shared/Aulas2021_22/SRSA/PL1
File Actions Edit View Help
user@user-virtualbox: /media...shared/Aulas2021_22/SRSA/PL1

```
user@user-virtualbox:/media/sf_VMs_shared/Aulas2021_22/SRSA/PL1$ pytho
n3 server_udp.py 127.0.0.1 5000
Socket is listening in address 127.0.0.1 and port 5000 ...
Received message "Hello!" from Client 1, calling from: 127.0.0.1:44640
```

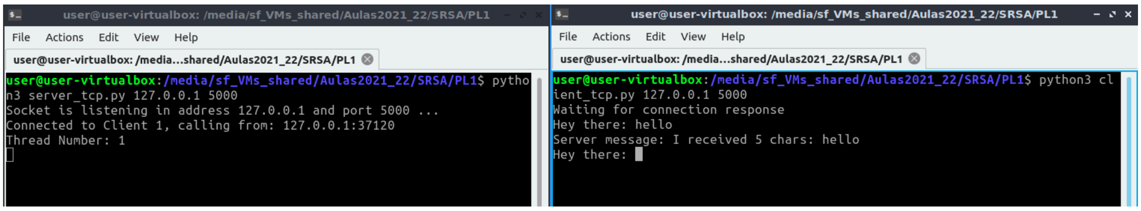
```
user@user-virtualbox:/media/sf_VMs_shared/Aulas2021_22/SRSA/PL1$ python3 cl
ient_udp.py 127.0.0.1 5000
Usage: client_udp.py <host server> <port> <message>
user@user-virtualbox:/media/sf_VMs_shared/Aulas2021_22/SRSA/PL1$ python3 cl
ient_udp.py 127.0.0.1 5000 Hello!
user@user-virtualbox:/media/sf_VMs_shared/Aulas2021_22/SRSA/PL1$
```

Figure 3 – Example of UDP code running in Linux

## TCP example

**Goal:** Understand how the TCP protocol works using the Python sockets API.

**Activities:** Download the example code from UC Student (`server_tcp.py`, `client_tcp.py`) and review it. Identify the differences in the code regarding the previous exercise. Determine your local IP address and run the server program using the following command: `python server_tcp.py <local_address> <port>`. From another shell or computer, run the client program using the following command: `python client_tcp.py <server_address> <port>`.


user@user-virtualbox: /media/sf_VMs_shared/Aulas2021_22/SRSA/PL1
File Actions Edit View Help
user@user-virtualbox: /media...shared/Aulas2021_22/SRSA/PL1
user@user-virtualbox: /media/sf_VMs_shared/Aulas2021_22/SRSA/PL1
File Actions Edit View Help
user@user-virtualbox: /media...shared/Aulas2021_22/SRSA/PL1

```
user@user-virtualbox:/media/sf_VMs_shared/Aulas2021_22/SRSA/PL1$ pytho
n3 server_tcp.py 127.0.0.1 5000
Socket is listening in address 127.0.0.1 and port 5000 ...
Connected to Client 1, calling from: 127.0.0.1:37120
Thread Number: 1
```

```
user@user-virtualbox:/media/sf_VMs_shared/Aulas2021_22/SRSA/PL1$ python3 cl
ient_tcp.py 127.0.0.1 5000
Waiting for connection response
Hey there: hello
Server message: I received 5 chars: hello
Hey there:
```

Figure 4 – Example of TCP code running in Linux

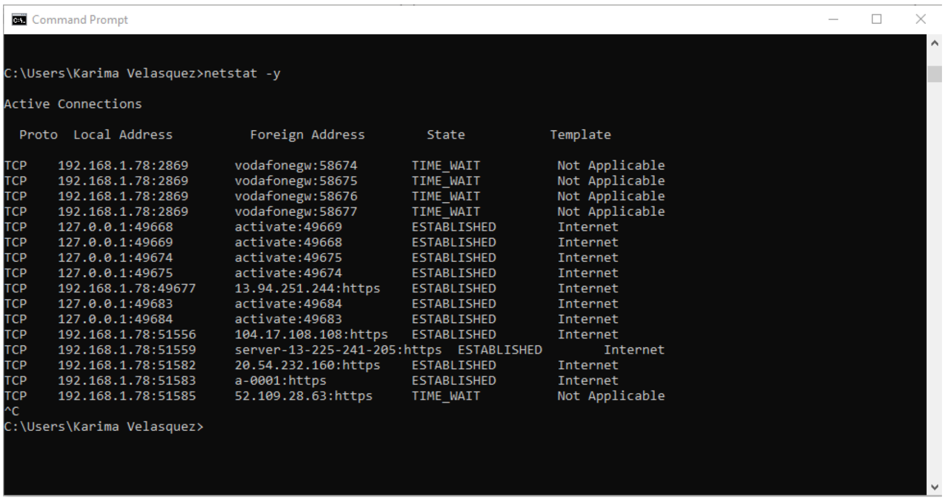
## Using netstat

**Goal:** Use netstat to display information about the connections created by the sockets example.

Netstat[3] (network statistics) is a command-line network utility that helps analyse active TCP/IP connections. The information displayed by netstat includes active connections, ports on which the computer is listening, routing table, and IP statistics.

[3] https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/netstat

**Activities:** Use the command `netstat --help` to explore the options available for netstat. Using netstat, identify the connections created by the previous exercises (UDP example, TCP example).



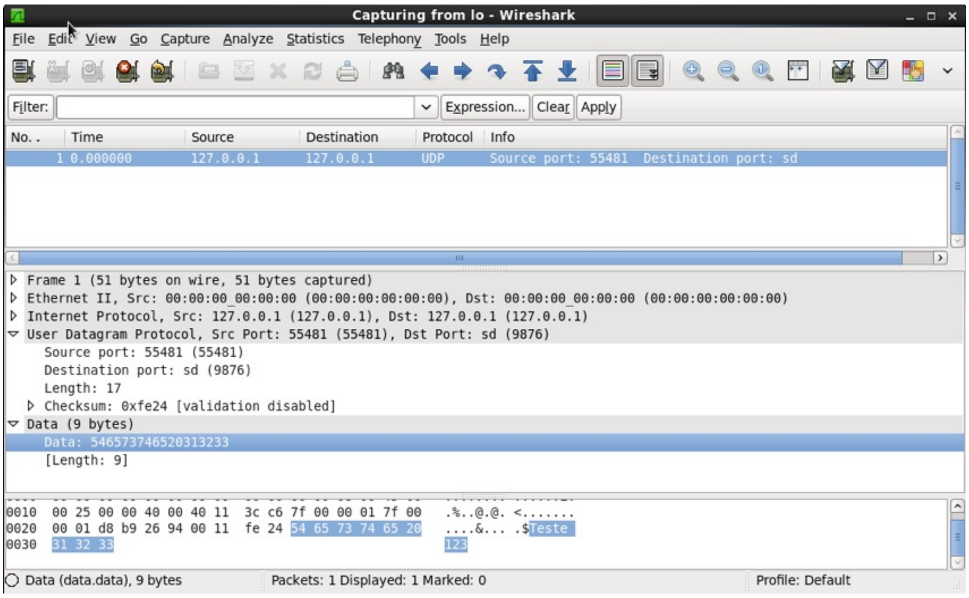
```
C:\Users\Karima Velasquez>netstat -y

Active Connections

  Proto  Local Address          Foreign Address        State           Template

TCP    192.168.1.78:2869      vodafonegw:58674       TIME_WAIT        Not Applicable
TCP    192.168.1.78:2869      vodafonegw:58675       TIME_WAIT        Not Applicable
TCP    192.168.1.78:2869      vodafonegw:58676       TIME_WAIT        Not Applicable
TCP    192.168.1.78:2869      vodafonegw:58677       TIME_WAIT        Not Applicable
TCP    127.0.0.1:49668        activate:49669         ESTABLISHED      Internet
TCP    127.0.0.1:49669        activate:49668         ESTABLISHED      Internet
TCP    127.0.0.1:49674        activate:49675         ESTABLISHED      Internet
TCP    127.0.0.1:49675        activate:49674         ESTABLISHED      Internet
TCP    192.168.1.78:49677     13.94.251.244:https    ESTABLISHED      Internet
TCP    127.0.0.1:49683        activate:49684         ESTABLISHED      Internet
TCP    127.0.0.1:49684        activate:49683         ESTABLISHED      Internet
TCP    192.168.1.78:51556     104.17.108.108:https   ESTABLISHED      Internet
TCP    192.168.1.78:51559     server-13-225-241-205:https  ESTABLISHED        Internet
TCP    192.168.1.78:51582     20.54.232.160:https    ESTABLISHED      Internet
TCP    192.168.1.78:51583     a-0001:https           ESTABLISHED      Internet
TCP    192.168.1.78:51585     52.109.28.63:https     TIME_WAIT        Not Applicable
^C
C:\Users\Karima Velasquez>
```

Figure 5. Example of use of the netstat command in Windows

## Using Wireshark

**Goal:** Use Wireshark to display information about the connections and data generated by the sockets example.

Wireshark[4] is a network sniffer and packet analyser used for network troubleshooting. By using a network sniffer such as Wireshark, the messages between client and server can be seen. The information available includes the packet header and the data.



```
No. .   Time        Source       Destination   Protocol  Info
      1 0.000000    127.0.0.1    127.0.0.1     UDP       Source port: 55481  Destination port: sd
```

```
▷ Frame 1 (51 bytes on wire, 51 bytes captured)
▷ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▷ Internet Protocol, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
▽ User Datagram Protocol, Src Port: 55481 (55481), Dst Port: sd (9876)
    Source port: 55481 (55481)
    Destination port: sd (9876)
    Length: 17
  ▷ Checksum: 0xfe24 [validation disabled]
▽ Data (9 bytes)
    Data: 546573746520313233
    [Length: 9]
```

```
0010  00 25 00 00 40 00 40 11  3c c6 7f 00 00 01 7f 00   .%..@.@. <.......
0020  00 01 d8 b9 26 94 00 11  fe 24 54 65 73 74 65 20   ....&... .$Teste 
0030  31 32 33                                            123
```

○ Data (data.data), 9 bytes    Packets: 1 Displayed: 1 Marked: 0    Profile: Default

Figure 6. Example of capture from Wireshark

[4] http://www.wireshark.org

**Activities:** Using Wireshark, identify the communication flow from the previous exercises (UDP example, TCP example). Use the filters `ip.addr==<address>`, `tcp.port==<port>`, and `udp.port==<port>` to facilitate the analysis. Identify the differences between the TCP and UDP flows.