Lab title - Distribution transparency
Learning outcomes - An ability to program network sockets using Python. A greater understanding of the challenges to achieving distribution transparency.

Sockets are a fundamental method for performing network communication. Sockets are a communication method offered by the operating system; they are not a type of middleware. However many middleware communication methods are build using sockets. In this lab you will be programming sockets using the Python programming language.

1. Creating a socket in Python
First we create a socket using the socket function in the socket Python module/library (see socket_1.py below). To run this code open a terminal and enter the command “python socket_1.py”.

```
import socket # import the socket module
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

socket_1.py

Here a socket instance is created and passed two parameters AF_INET and SOCK_STREAM. The first parameter AF_INET refers to the address family IP version 4 or IPv4. In this context, an address to which a message is sent consists of an (IP address, port number)-pair. Each computer has a unique IP address. Messages are sent to an end point called a *port* specified by a *port number*. Some ports are reserved for standard services (for example, port number 21 is reserved for FTP).

The second parameter SOCK_STREAM means a *connection-oriented* communication protocol called TCP is used. In a connection-oriented communication protocol, before exchanging data the sender and receiver first explicitly establish a connection. Furthermore, when they are done, they release (terminate) the connection.

2. Error handling
If any of the socket functions fail then Python throws an exception called socket.error which must be caught. An example of such error handling is provided below (socket_2.py).

```
import socket   # for sockets
import sys  # for exit

try:
    #create an AF_INET, STREAM socket (TCP)
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
except (socket.error, msg):
    print('Failed to create socket. Error code: ' + str(msg[0]) + ' , Error message : ' +msg[1])
    sys.exit();

print('Socket Created')
```

socket_2.py

3. Connecting to a Server
Next we shall try to connect to a remote server using a socket. To connect to a remote server we require an IP address and port number to connect to. Here we use the IP address of google.com as an example. In Python, the function socket.gethostbyname can

be used to obtain this IP address. Once we have the IP address, we can connect to a port using the connect function.

```
import socket   #for sockets
import sys  #for exit

try:
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
except(socket.error, msg):
    print('Failed to create socket. Error code: ' + str(msg[0]) + ' , Error message : '+msg[1])
    sys.exit();

print('Socket Created')

host = 'www.google.co.uk'
port = 80

try:
    remote_ip = socket.gethostbyname( host )

except socket.gaierror:
    print('Hostname could not be resolved. Exiting')
    sys.exit()

print('Ip address of ' + host + ' is ' + remote_ip)

s.connect((remote_ip , port)) #Connect to remote server
print('Socket Connected to ' + host + ' on ip ' + remote_ip)
s.close()
```

socket_3.py

4. Sending and Receiving Data

In the following example we first connect to an IP address and use the sendall function to send a HTTP GET request. The message is actually a http command to fetch the main page of a website. We then use the recv function to receive a reply from the server.

```
import socket   #for sockets
import sys  #for exit

try:
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
except socket.error:
    print('Failed to create socket')
    sys.exit()
print('Socket Created')

host = 'www.google.co.uk'
port = 80

try:
    remote_ip = socket.gethostbyname( host )
```

```
except socket.gaierror:
    print('Hostname could not be resolved. Exiting')
    sys.exit()

s.connect((remote_ip , port)) # Connect to remote server
print('Socket Connected to ' + host + ' on ip ' + remote_ip)

message = "GET / HTTP/1.1\nHost: " + host + "\n\n"
try :
    s.sendall(message.encode())
except socket.error:
    print('Send failed')
    sys.exit()

print('Message send successfully')
reply = s.recv(4096) # Now receive data;  4096 is the number of bytes received.
print(reply)
s.close()
```

socket_4.py

5. Creating a server

We now create a simple server client program. A server has a bind() method which binds a socket to a specific IP and port where it will listen for incoming requests. Next a server has a listen() method which puts the server into listen mode. This allows the server to listen to incoming connections. Lastly, a server has accept() and close() methods. The accept method initiates a connection with the client and the close method closes the connection with the client.

```
import socket

# next create a socket object
s = socket.socket()
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
print("Socket successfully created")

# reserve a port on your computer in our case it is 12345 but it can be anything
port = 12345

# Next bind to the port; we have not typed any ip in the ip field. Instead we have input an
# empty string which makes the server listen to requests coming from other computers
# on the network
s.bind(('', port))
print("socket binded to %s" %(port))

# put the socket into listening mode
s.listen(5)
print("socket is listening")

# a forever loop until we interrupt it or an error occurs
while True:
        # Establish connection with client.
```

```
        c, addr = s.accept()
        print('Got connection from', addr)

        # send a thank you message to the client.
        message = "Thank you for connecting"
        c.send(message.encode('utf-8'))

        # Close the connection with the client
        c.close()

s.close()
```

socket_5.py

In the above code, a socket object is created and a port is reserved on the computer. Next the server is binded to the specified port. Passing an empty string means the server listens to incoming connections from other computers as well. If you pass 127.0.0.1 or the string localhost it will only listen to those calls made within the local computer. Next the server is put into listen mode. The parameter 5 passed to listen means that 5 connections are kept waiting if the server is busy and if a 6th socket tries to connect then the connection is refused. Lastly the code within the while loop accepts all incoming connections and closes those connections after a thank you message has been sent.

6. Creating a client

We now create a client to connect to the above server. To connect to the server using telnet first run the server program in one terminal and enter the following command in a second terminal. Telnet or TELecommunication NETwork is a network protocol which connects to remote machines over a local area network or the Internet.

telnet localhost 12345

The following is a python client which connects to the above server (socket_5.py).

```
import socket

# Create a socket object
s = socket.socket()

# Define the port on which you want to connect
port = 12345

# connect to the server on local computer
s.connect(('127.0.0.1', port))

# receive data from the server
print(s.recv(1024))
# close the connection
s.close()
```

socket_6.py

To do – run the server and client on different computers. Determine the IP address for the computer upon which you wish to run the server (use the ifconfig tool and see inet). Now change localhost ('127.0.0.1') in the client script to this IP address.

7. Questions

Close the server and attempt connecting using the client. What happens and why?

What is distribution transparency? Using WebSockets for communication does not offer distribution transparency. Justify this statement (hint: there are a number of reasons including naming in the context of *location transparency.* Also consider the previous question).

Design a middleware for communication build on WebSockets which offers greater distribution transparency than WebSockets.

Write a port scanner and scan the ports on your machine.