Mawlana Bhashani Science and Technology University

Santosh, Tangail-1902.

Lab Report

Department of Information and Communication Technology

Report No: 02
Report Name: Socket Programming
Course Title: Network Planning and Design Lab.
Course Code: ICT-3208

Submitted By	Submitted To
Name: **S.M. Hazrat Ali** ID: **IT-17016** Session: 2016-17 3rd Year 2nd Semester Dept. of Information & Communication Technology, MBSTU.	**Nazrul Islam** **Assistant Professor** Dept. of Information & Communication Technology, MBSTU.

Objective: Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while other socket reaches out to the other to form a connection. Server forms the listener socket while client reaches out to the server.
They are the real backbones behind web browsing. In simpler terms there is a server and a client.

Socket programming is started by importing the socket library and making a simple socket.

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Here we made a socket instance and passed it two parameters. The first parameter is AF_INET and the second one is SOCK_STREAM. AF_INET refers to the address family ipv4. The SOCK_STREAM means connection oriented TCP protocol.

Now we can connect to a server using this socket.

Server side: Server-side network programming involves designing and implementing programs to be run on a server. Server-side applications run as processes on a dedicated physical machine, virtual machine, or cloud infrastructure. Server-side applications receive requests from the clients and perform tasks as requested by the clients.

Server side code:

```
import socket # for socket

import sys

  try:

      s = socket.socket(socket.AF_INET,

      socket.SOCK_STREAM) print "Socket successfully created"


except socket.error as err:

     print "socket creation failed with error %s" %(err)



port = 80
```

try:

```
host_ip = socket.gethostbyname('www.google.com')
except socket.gaierror:
        print "there was an error resolving the host"


   sys.exit()


s.connect((host_ip, port))
print "the socket has successfully connected to google \on port == %s" %(host_ip)
```

Client side: In a client environment, each computer still holds (or can still hold) its (or some) resources and files. Other computers can also access the resources stored in a computer, as in a peer-to-peer scenario. One of the particularities of a client/server network is that the files and resources are centralized. This means that a computer, the server, can hold them and other computers can access them. Since the server is always ON, the client machines can access the files and resources without caring whether a certain computer is ON.

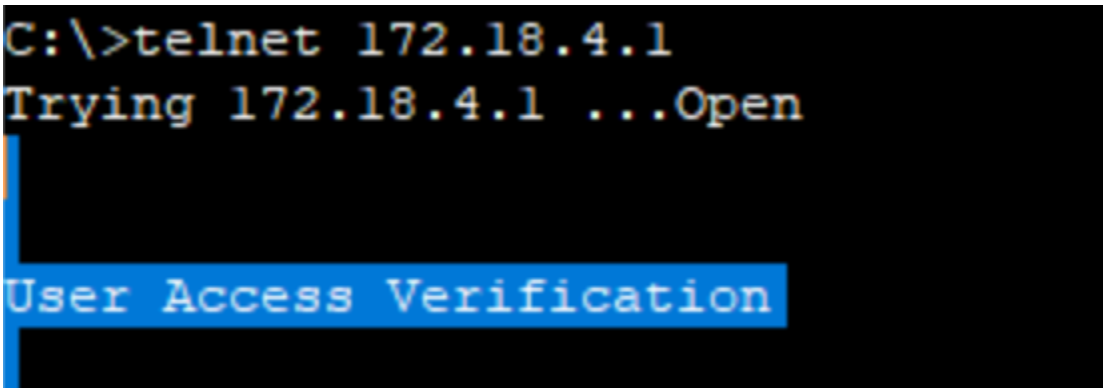
Client side code:

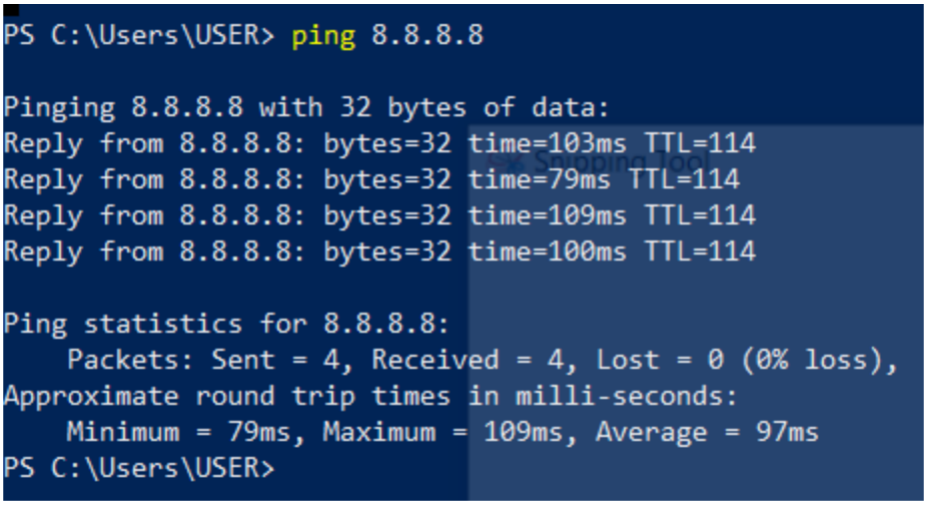
```
# standard Python
sio = socketio.Client()


# asyncio
sio = socketio.AsyncClient()
sio.connect('http://localhost:127.0.0.1')
await sio.connect('http://localhost:127.0.0.1')
sio.event(namespace='/chat')
def my_custom_event(sid, data):
   pass


@sio.on('connect', namespace='/chat')
```

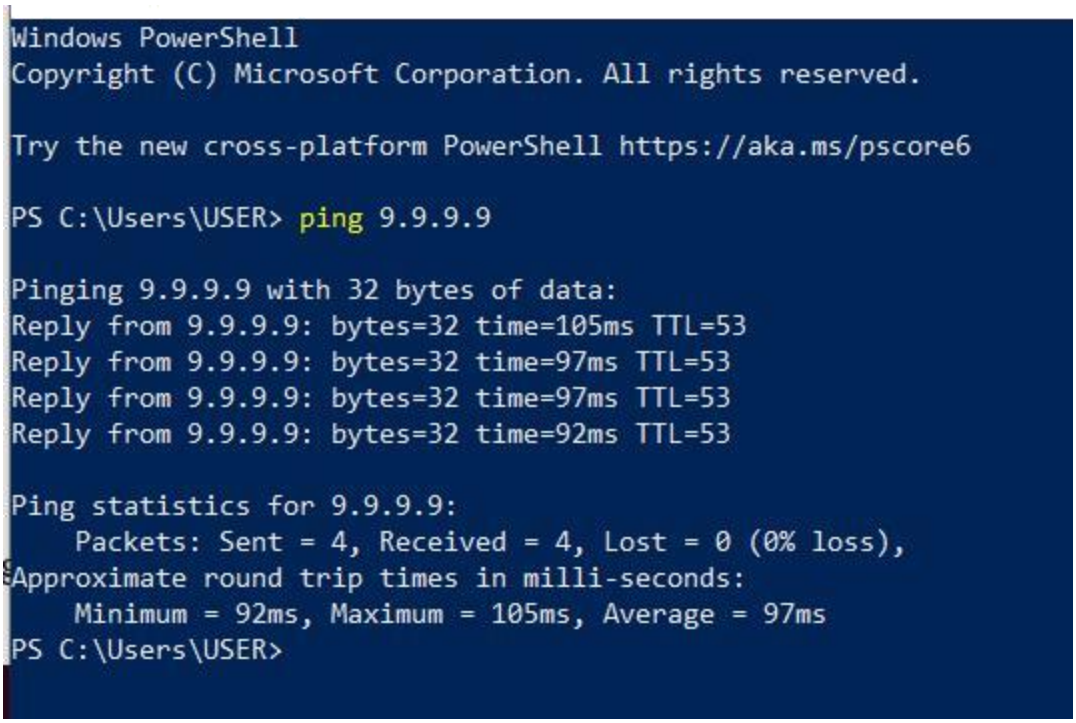
def on_connect():

tracert(172.18.4.1)

Output : Socket successfully created the socket has successfully connected to google on port == 80 to IP 172.18.4.1







Conclusion: Sockets are the endpoints of a bidirectional communications channel. Sockets may communicate within a process, between processes on the same machine, or between processes on different continents.

Sockets may be implemented over a number of different channel types: Unix domain sockets, TCP, UDP, and so on. The socket library provides specific classes for handling the common transports as well as a generic interface for handling the rest.