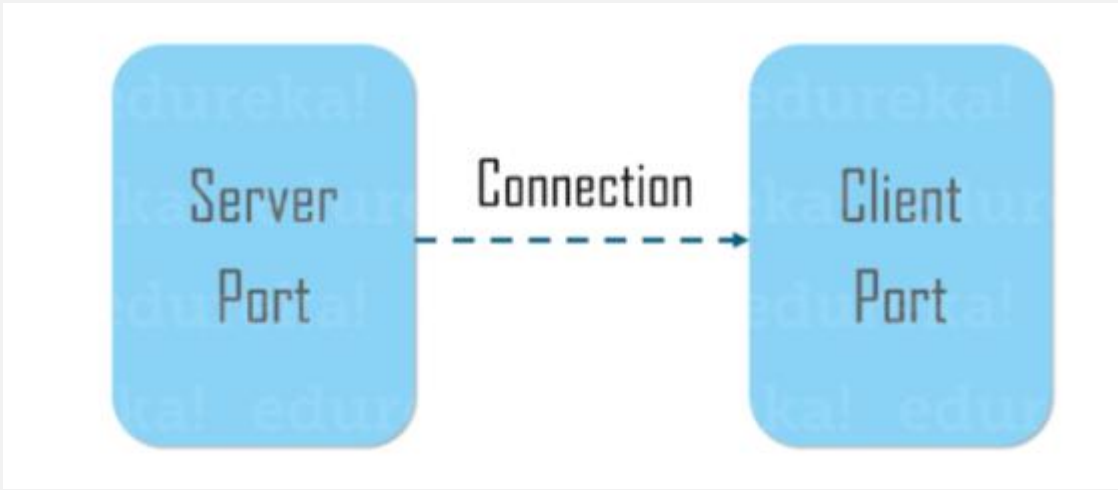
Socket Programming using UDP



Networking is an interesting concept where we come across a lot of terms. Today we shall know about UDP and see how we can write a socket program for the same.

Binding

This joining of IP and port number is considered as binding. We bind the IP of a system to a port number of a program. It helps us to uniquely identify a particular program on the system.

for eg: *192.168.0.129:1802/sk.py*

This will take you to the sk.py file running on port number 1802 of IP 192.168.0.129

By default, httpd is binded to port number 80 and ssh to port number 22. As port number tells the process, it has to be unique for each program.

Socket

The combination of IP and port number is considered as Socket. It gives us a clear idea about the destination where we want to send and receive data.

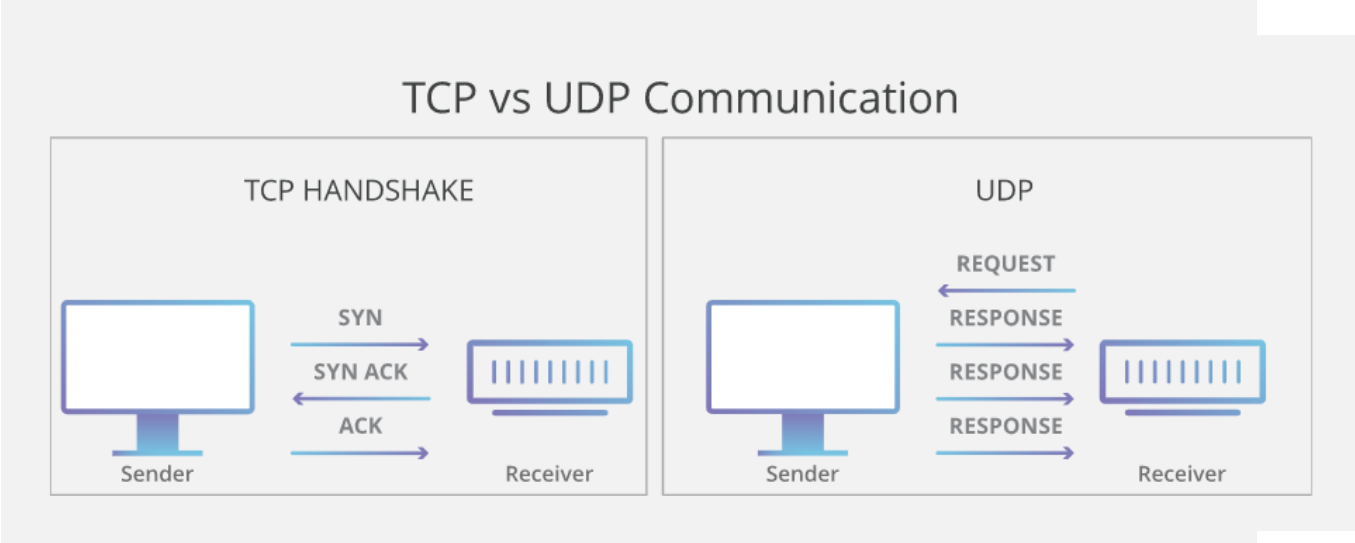
ip=192.168.0.129

port=1802

bind(ip,port)

To understand socket programming in depth, we come across some other terms. Let us understand them in simple words and then move towards the socket programming using UDP.

Protocol



TCP

When system A sends data to system B, system B will acknowledge that it has received the data. Only then, further transfer will take place. It is also known as handshaking. Due to this whole handshaking and acknowledgement process, this protocol is pretty slow.

TCP can be used when we want to send some critical data. The cases where speed doesn't matter but we want to make sure that data is being transferred successfully.

UDP

After receiving data from system A, B won't give any acknowledgement for the same. The transfer will take place

without any cross-checking. Due to this, UDP is faster than TCP. However, there are chances of data loss in this process.

This protocol is best suited where speed matters a lot. For eg. conferences, or webinars.

Socket Program

We shall write the **receiver program on Linux**, and then understand the meaning of each line.

```
import socket

myprotocol=socket.SOCK_DGRAM

addfamilyname=socket.AF_INET

s=socket.socket(afn,myp)

ip="192.168.0.129"

port=1802

s.bind((ip,port))

x=s.recvfrom(1024)
```

We import socket. UDP uses SOCK_DGRAM for transferring data, so we store it in variable myprotocol. For UDP, the address family name is AF_INET. Both the protocol and address family name is different for TCP, which we shall see some other day.

After defining them, we shall now take them in a socket module in variable name s.

Now, we define the IP of the receiver, port number where our code is running and bind them.

x=s.recvfrom(1024) says that it will receive maximum size of packet 1024 and it is the buffer size.

We will now write the **sender's program on windows.**

```
import socket

mypp=socket.SOCK_DGRAM

afnn=socket.AF_INET

s=socket.socket(afnn,mypp)

s.sendto ("hello".encode()("192.168.0.129",1802))
```

All the lines of code is similar to the receiver side program except the last line. Last line says that we need to send this data, i.e., hello to the following IP and port number.

We have user “hello”.encode() because socket program does not support transfer to string data.

encode() converts string datatype to byte

decode() converts bytes datatype to string

Linux will receive data from windows as shown in the last line of the sender code. Instead of the IP and port number of linux, that of windows will appear. However, we won’t get acknowledge if linux is receiving the sent data or not. But the transfer will happen rather quickly.

This is UDP, User DataGram Protocol.