Networks 3 assignment
Report

18398783
jamie.clarke49@mail.dcu.ie

Link to youtube video - https://www.youtube.com/watch?v=gCDxQpZFCno

Sockets

What are the three types of network socket and what are their differences?

The three types of network sockets are :

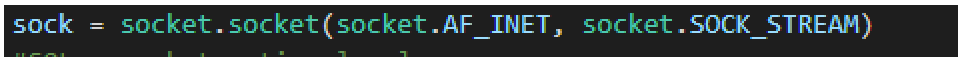
- Socket streams that allow communication through TCP (SOCK_STREAM).
- Datagram sockets allow communication through UDP (SOCK_DGRAM).
- Raw sockets that provide access to ICMP (SOCK_RAW).

The first difference between socket streams and Datagram sockets are how they allow communication and what they use for this. As stated above socket streams use TCP whereas datagram sockets use UDP. UDP is the faster of the two as it is a connectionless protocol[1], retransmission of lost data is only possible using socket streams. The next difference between the three types of sockets is their way of dataflow. For example in socket stream we can use bidirectional, reliable, sequenced, and unduplicated flow of data with no record boundaries whereas in datagram sockets only bidirectional messages are supported. If we then compare this to how raw sockets manage dataflow, it is completely different. It is not as "traditional" at transferring data like the other two, instead it is best known for carrying everything including redirect instructions, timestamps etc. What it is best known for is its echo requests. One device sends a packet to another and asks for confirmation of receiving this package. This is a great way of ensuring packets are not lost as you won't receive a confirmation and can resend it after.[2] For datagram sockets, the main use of them is their great speed in comparison to socket streams which are more used for their reliability and ordered nature.

What is the process of creating a network socket?

To begin with we must first call the socket constructor and choose what type of network socket we will be using, as you can see we have chosen socket stream for this example.

"socket.socket(socket.AF_INET, socket.SOCK_STREAM)".



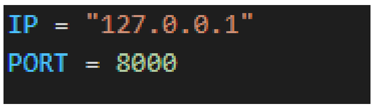
```
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

[1] https://www.lifesize.com/en/blog/tcp-vs-udp/#:~:text=TCP%20is%20a%20connection%2Doriented,is%20only%20possible%20with%20TCP.

[2] https://www.pingplotter.com/wisdom/article/packet-type-differences#:~:text=The%20Internet%20Control%20Message%20Protocol,a%20traditional%20data%20packet%20protocol.&text=One%20device%20sends%20out%20an,confirming%20it%20received%20the%20request.

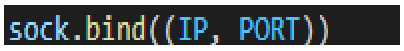
We also wanted to use an IPV4 address so chose the family AF_INET for our address family.(AF stands for address family). We also assigned this a variable as we can see above, we called our variable “sock”. We have now told this operating system that we are looking to open a TCP socket using the IPV4 addressing protocol.

The next thing we must do is bind the address, basically telling the socket what address we will be listening on and what port.

.

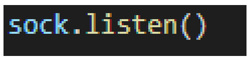
```
IP = "127.0.0.1"
PORT = 8000
```

To bind these addresses we simply type “sock.bind(IP, PORT)” . This binds the two together.



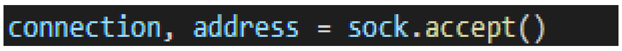
```
sock.bind((IP, PORT))
```

Next we call “sock.listen” to tell the socket to start waiting for connections.



```
sock.listen()
```

Finally we call “sock.connect” to accept a connection. This will return the address of the mahine making the connection.



```
connection, address = sock.accept()
```

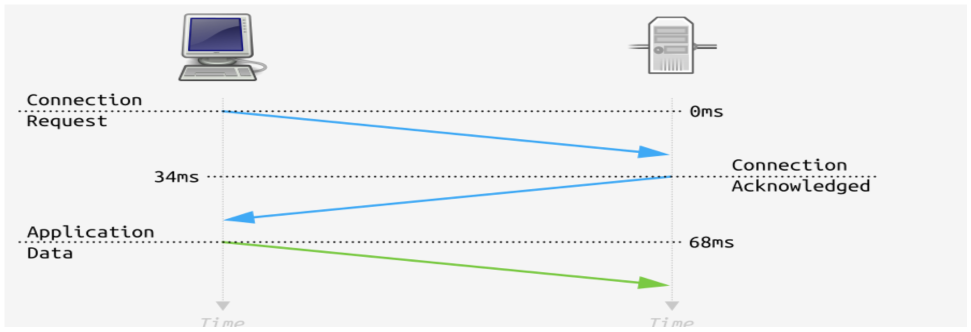
That is all we need in order to create a network socket, with a few additional steps if we wish to send data.

<u>The desktop computer with the IP address 192.168.0.5 is running multiple applications What are the steps involved when the user opens their browser and visits the website discord.com , make sure to take account of the protocols involved in each step.</u>

1. To begin with the user enters discord.com’s URL into their search bar. Next the browser uses DNS to find the website's IP address, which in this case is 162.159.128.233 making it a class B address. The browser will now use the IP address returned by DNS to communicate with the web server that hosts the website. It will connect to port 80 using the TCP protocol.

2. DNS translates www.discord.com into something the computer can understand, an IP address. When we use DNS it can use both a TCP protocol or a UDP protocol depending on the scenario. For translating a website URLs to IP addresses we use the UDP protocol.
3. Next step we use a 3 way handshake to ask the website for a connection. We begin by asking discord’s server if they are open for new connections. If the website agrees then it sends back a confirmation that they are open to new connections. Finally your desktop sends back an acknowledgement that it has received its confirmation to allow new connections. This

step is shown well in the next diagram.



[3]

4. Next your desktop downloads the website's data, including the HTML of the website, so it appears the same for everyone who accesses the website.. It does this by sending a request to the website asking for its data. Once the website receives this request it returns a response in a particular format, containing the requested data .
5. If you choose to close this tab, the connection is ended and you must go back through these steps in order to reconnect to this website.

Websockets

What are the differences between a websocket and a normal network socket?

Although they achieve a very similar result, they do it through very different methods. To begin with websockets usually run from a browser connection using HTTP protocol in comparison to how sockets use either IP/TCP protocols. Sockets are more powerful than websockets as they run through TCP/IP and are not restricted by HTTP protocols. Normal sockets also have more freedom then websocket as you can choose the protocol you wish to use (TCP or UDP) whereas you have to use the HTTP protocol in websockets. For websockets, they are event driven, meaning they wait for something to happen, e.g open, message. Normal sockets using TCP are quicker than websockets averaging around 0.002 seconds compared to 0.02 seconds. Regular sockets also have a lower memory footprint mainly due to their play framework. Websockets also keep the connection open to the server the entire time you are interacting with the page, meaning it wastes a lot of the server's resources.

Websockets provide full-duplex communication , what does this mean, and how did we achieve similar functionality before websockets were released?

A full duplex deceive is capable of simultaneously transmitting data and receiving data over one channel.[4] A full duplex device would be a modem, which transmits data to a server while simultaneously receiving it. So in websockets we can send and receive data at the same time, we often see this as websockets being “bi-directional”. Before websockets were released, we could achieve something similar to full-duplex communication by using polling. As the server can't send any data to the client in polling without a request, the client periodically sends a request asking for data.

[3] https://www.houkconsulting.com/2018/02/what-happens-when-visit-website/
[4] https://www.comms-express.com/infozone/article/half-full-duplex/

The main disadvantage of polling is that it is a lot more resource intensive than websockets. If we look at long polling, it uses the HTTP protocol like websockets meaning there is little to no delay between each event. Long polling is more efficient though as it only sends data when it is needed, in comparison to websocket which constantly sends data.

Detail the steps, requests and protocols involved when you:

Start your app server .

The steps involved in starting my app server are as follows:

Begin by running server.py file in a terminal. We will also need to have another 2 terminals open for the clients to try to connect to the chat room. On the other two terminals we run client.py and are asked for a username. Once the clients enter their username they are connected to the server. The server receives confirmation that a new user has connected to the server. This is the TCP protocol working as a 2 way handshake, the client requests the data from the server, the server returns this and gets a confirmation message to say so.

Visit it in a browser

To visit my chatroom in a browser, you must go to the webpage provided in the folder. There you will see two chat boxes. The top one is meant for the user on that page with the second supposed to show the other user's response. Using HTML and socket.io we attempt to pull the user's response and show it on the webpage.

Send a message

In order to send a message the user must first put in a username, once they receive confirmation that they are connected(which is done by showing them their IP address and Port number) they can type a message, which will be sent to the server. Then the other user using the same chat room can type a response after user 1’s message comes through to them. This is done through a single TCP connection, using full-duplex communication.