**NETWORK BASICS**

1. **TCP CLIENT**

**In this we will set up a TCP client that can be used for multiple purposes like**

import socket

target\_host = "www.google.com"

target\_port = 80

# create a socket object

client = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

# connect the client

client.connect((target\_host,target\_port))

# send some data

client.send("GET / HTTP/1.1\r\nHost: google.com\r\n\r\n")

# receive some data

response = client.recv(4096)

print response

Explanation:

This line creates a socket object. socket.AF\_INET specifies that the socket will use the IPv4 address family, and socket.SOCK\_STREAM specifies that it will be a TCP socket (which is connection-oriented).

Connect method establishs a connection between server and the client.

client.send(): This part refers to a function likely from a networking library (like socket in Python) that allows sending data over a network connection.

GET / HTTP/1.1\r\n: This is the actual HTTP request. Here's what it means:

GET: This specifies the HTTP method, which is GET in this case. GET requests are used to retrieve information from a server.

/: This indicates that the request is for the root path (homepage) of the website.

HTTP/1.1: This specifies the HTTP protocol version being used (in this case, HTTP 1.1).

\r\n: This is a newline character (carriage return and line feed). It separates the request line from the headers.

Host: google.com\r\n: This is a header line specifying the hostname of the server being requested, which is google.com in this case.

\r\n: Another newline character, indicating the end of the headers.

1. UDP Client

import socket

target\_host = "127.0.0.1"

target\_port = 80

# create a socket object

client = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

# send some data

client.sendto("AAABBBCCC",(target\_host,target\_port))

# receive some data

data, addr = client.recvfrom(4096)

print data

This creates a UDP socket. socket.AF\_INET specifies the address family (IPv4), and socket.SOCK\_DGRAM specifies the socket type (UDP).

This sends the string "AAABBBCCC" to the target host and port. Note: In Python 3, you should send bytes, so you would typically use b"AAABBBCCC" instead of "AAABBBCCC".

This waits to receive data from the socket. It can receive up to 4096 bytes. The recvfrom method returns a tuple containing the received data and the address of the sender.

Because UDP is connectionless protocol there is no call to connect() beforehand.

1. TCP Server

import socket

import threading

bind\_ip = "0.0.0.0"

bind\_port = 9999

server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server.bind((bind\_ip,bind\_port))

server.listen(5)

print "[\*] Listening on %s:%d" % (bind\_ip,bind\_port)

# this is our client-handling thread

def handle\_client(client\_socket):

# print out what the client sends

request = client\_socket.recv(1024)

print "[\*] Received: %s" % request

# send back a packet

client\_socket.send("ACK!")

client\_socket.close()

while True:

client,addr = server.accept()

print "[\*] Accepted connection from: %s:%d" % (addr[0],addr[1])

# spin up our client thread to handle incoming data

client\_handler = threading.Thread(target=handle\_client,args=(client,))

y client\_handler.start()

This Python code sets up a simple multi-threaded TCP server that listens for incoming connections on a specified IP address and port. Here’s a step-by-step explanation:

Socket Setup:

bind\_ip = "0.0.0.0"

bind\_port = 9999

server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server.bind((bind\_ip, bind\_port))

server.listen(5)

print "[\*] Listening on %s:%d" % (bind\_ip, bind\_port)

* bind\_ip and bind\_port define the IP address and port on which the server will listen. 0.0.0.0 means the server will listen on all available IP addresses.
* A socket is created using socket.AF\_INET (IPv4) and socket.SOCK\_STREAM (TCP).
* server.bind((bind\_ip, bind\_port)) binds the socket to the specified IP address and port.
* server.listen(5) puts the server into listening mode and allows it to accept incoming connections. The 5 is the maximum number of queued connections.

The server prints a message indicating it is listening on the specified IP and port.

Client Handling:

def handle\_client(client\_socket):

request = client\_socket.recv(1024)

print "[\*] Received: %s" % request

client\_socket.send("ACK!")

client\_socket.close()

* handle\_client is a function that handles communication with a connected client.
* client\_socket.recv(1024) receives data from the client. The 1024 specifies the maximum amount of data to be received at once (in bytes).
* The received data is printed to the console.

client\_socket.send("ACK!") sends a response ("ACK!") back to the client.

client\_socket.close() closes the connection to the client.

Main Loop:

while True:

client, addr = server.accept()

print "[\*] Accepted connection from: %s:%d" % (addr[0], addr[1])

client\_handler = threading.Thread(target=handle\_client, args=(client,))

client\_handler.start()

The server enters an infinite loop, continuously waiting for incoming connections.

* client, addr = server.accept() accepts a new connection and returns a new socket object (client) and the address of the client (addr).

A message is printed to indicate a new connection has been accepted.

A new thread is created to handle the client connection using the handle\_client function.

client\_handler.start() starts the new thread, allowing the server to handle multiple clients concurrently.