ChainSpoT

Supervised by:

Assoc. Prof: Sahar Abd El-Rahman

Dr: Heba-Allah Adly

Team Members:

- Ahmed Hesham Salah
- Aya Mohamed Amin
- Doaa Abusriaa Abd-Elatty
- Sara Ashraf Elsayed

- Mohamed Yasser Mohamed

Acknowledgment

We thank Allah for completing the project and we hope that it will be useful.

we would like to express our deep gratitude to Assoc Prof Dr. Sahar Abd El-Rahman and Dr.Heba-Allah Adly, our project supervisors, for their patient guidance, enthusiastic encouragement, and useful critiques of this research work. We have been extremely lucky to have supervisors who cared so much about our work, and who responded to our questions and queries so promptly.

we would like to express our very great appreciation to Dr. Abdallah Saad for his valuable and constructive suggestions during the planning and development of this project. His willingness to give his time so generously has been very much appreciated.

Also, we would like to thank the college for providing us with laboratories that are equipped with the required tools such as computers and the Internet.

We would like to thank the Discussion committee for their time and efforts.

Finally, we would like to thank our families for their unconditional support.

Table Of Content

List Of Figures

Abstract

We aim to make people's lives better. If you do not have network infrastructure hardware, do not worry. You can start your new network without it and communicate with your friends or colleagues. You can chat or share files without any constraints. Imagine a building without network infrastructure (no cables, no switches, no hubs, no access points, and no routers). It is more common in developing countries. A Windows Hotspot will be a good solution for this issue. We will develop the hotspot option using wireless network technology in our application ChainSpoT. However, Windows Hotspot has limitations, as it can only communicate with 8 devices. We aim to develop software that manages Wi-Fi and hotspots on your device to make a chain of hotspots that allows you to chat, share files, voice streams, video streams, and even play video games with each other. All these are locally, so there is no need to have a portable access point or switch in your pocket every time you go. No need for the internet, no device limitation, and no coverage area limitation. Just open the ChainSpoT software and click "start". ChainSpoT will do everything. connects you to the best node, opens your hotspot to make another tree branch, opens the server (if you are the maser), connects to the server (if you are the client), and reconnects you to the chain if your connection drops.

Keywords: ChainSpoT, Network, Infrastructure networks, Chat application, Wi-Fi, Loopback Interface, Hotspot.

Chapter One: Introduction

1.1Problem Statement

The main problem is that we did not find an easy method for local communication without an infrastructure network, or something that does not cost me and has no limitations so, we are targeting solutions to different problems for different sectors of usage:

The most critical sector is the military sector:
Military departments are mostly concerned with upgrading their security system to provide ultimate security for their data and information. But in the military, there are a lot of wide areas. No access point in the entire world can cover all this area alone. Our project increases coverage area based on clients of the network. Military soldiers move a lot to different sites and military tents. They carry a lot of equipment already. Current Network solutions are permanent and require a lot of equipment so it will be better for moving soldiers to eliminate network equipment.

But some other sectors, like small companies and small offices, are looking for reducing the cost of intermediary devices used in current networks.

For users, it is much effort to share big-size data. It is very costly and takes much time with the internet, but if these users are in the same wireless coverage area (Room, House, Café, School, College, …etc.) it is more convenient to share this data locally, but they need network equipment to make this happen fast and free. For isolated sites without internet coverage, there is no way to communicate.

1.2 Objectives

- Aiming to connect the laptops and computers without an intermediary device to facilitate communication, sharing, and provision of intermediate devices for members of the same group.
- Creating a chain of hotspots and desktop applications to manage them.
- Making a Program for chatting and sharing files as a proof of concept.

- Monitoring performance and measuring its parameters empirically by testing it in the Lab, and enhancing their parameters to achieve a good performance, much more user capacity, and making more coverage area for the network.
- Securing data delivery.

1.3 Stakeholders

- Residents of the building who do not have network infrastructure.
- Soldiers in the camps and places without network coverage.
- Small companies and offices to save network infrastructure costs.
- People who need to send large files locally but have limited internet access.
- People who need to chat locally and secured.

1.4 Project Scope

We will create an algorithm to reimplement the Ad-Hoc concept using infrastructure and create a chatting and sharing files program as proof of concept.

Our focus will be on Windows 10 laptop devices and desktop computers with wireless cards only. Since we do not have too many laptops in the early stage, we will use desktop computers in our faculty lab and buy external wireless adapters beside our laptops.

1.5 constraints

- ChainSpoT tree will grow until we hit a level where connection became slow and cannot grow further.
- Network setup is required every time you use the network.
- Network setups take a long time.
- Connection quality depends on the quality of wireless network cards in the network.

Because of all those reasons, ChainSpoT networks are a temporary solution and cannot be a permanent solution.

1.6 Cost

- The biggest advantage of our project is it is almost free and costless if users have a laptop only.
- For desktop users, it costs about 10$ for every device to buy and install an external network adapter.

1.7 Project Outcomes

At the end of the project, any user can start a free wide range of networks without being constrained by the number of clients, then use this network for chatting, sharing files …etc. without any infrastructure just by their laptops or desktop computers (with wireless adapter) just by ChainSpoT software at any time. No need for user experience in the network. The software will be very

user-friendly. Just by a few steps the network starts, and the client connects to it, we will make a tutorial for the user to teach him how to use our program. And for the superuser, we will make him a statistics section for the network statistics to help him in troubleshooting the network.

Chapter Two: Background and Related Work

2.1 Related Work

The current solutions that we have found in our research:
If you want to make two or more devices communicate with each other, you need to Buy switches and cables, Connect the switch with your devices and configure it.
If you want the devices to go online, you need to buy a router and configure it.
If you want to drop cables, you can buy an access point too. This is coastal, complex, and hard but efficient and Practical in most cases, but these are permanent solutions. What if you want a temporary or portable solution? This can happen by using the mobile hotspot feature in most smartphones or laptops today, but it has its limitations.

SHARE it (an example of a hotspot local sharing application) [6].

SHARE is an extremely popular file-sharing application designed to freely transfer several types of data between Android, iOS, macOS, and Windows. If you plan to transfer important data from one device to another, then SHARE is a good choice. However, this application also has some limitations.
SHARE is mainly used to transfer data and files between different devices.

How does it work?

It tries to start the hotspot from the sender or receiver and the other one connects to this hotspot. If one of them can't start or connect to a hotspot, it uses Bluetooth instead of a hotspot (connection will be slower for sure), Then the sender sends the data (files/photos) to the receiver.

Disadvantages

- It is a peer-to-peer application, that allows only transferring files between two hosts only.
- The supported data types are relatively small.

- The transfer process can get stuck sometimes.

We will try to overcome these disadvantages

- ChainSpoT is a broadcast application, that allows transferring files and chatting between all hosts in the network.
- The supported data types are relatively small.
- The transfer process can get stuck sometimes.

2.2 Wireless Network Modes

PCs are connected to your network with physical cabling. Now it is time to cut the cord and look at one of the most exciting developments in network technology, wireless networking [5].

Instead of a physical set of wires running among networked PCs, servers, printers, or what-have-you, a wireless network uses radio waves to enable these devices to communicate with each other. This offers great promise to the people who've spent time pulling cable up through ceiling spaces and down behind walls, but wireless networking is more than just a convenient solution sometimes, it could be the only networking solution that works in some cases, for example, imagine having clients whose offices are housed in a building designated as a historic landmark which You can't go punching holes in it to make a room for network cable runs, so Wireless networking will be the only valid solution.

Wireless networks operate at the same OSI layers and use the same protocols as wired networks. The thing that differs is the type of media-radio waves instead of cables-and the methods for accessing the media.

A hotspot is a physical location where users can connect to the Internet, using Wi-Fi, via a wireless local area network (WLAN) using a router connected to an Internet service provider. Most people call these locations "Wi-Fi hotspots" or "Wi-Fi connections".

A hotspot can be found in either a private or public location, such as a coffee shop, hotel, airport, or even an airline. While many public hotspots offer free wireless access to an open network, others require payment.

2.2.1 Infrastructure Mode

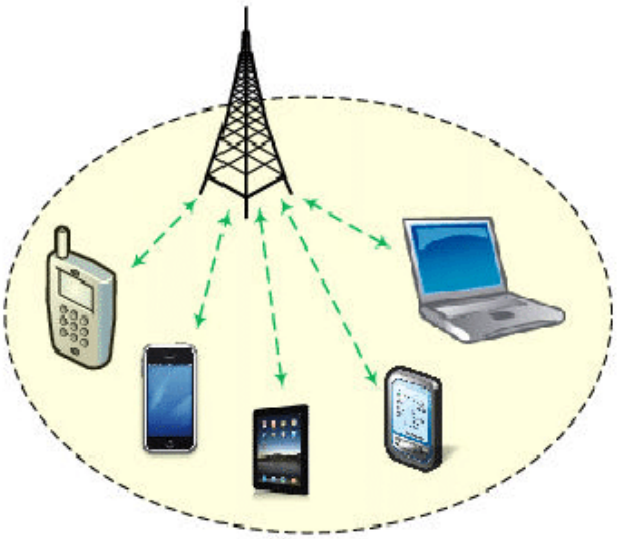


Figure 1 Infrastructure Network[8]

As shown in **Figure1** above, Infrastructure mode is a wireless network framework that has a central WLAN access point/router at the heart of the network. Wireless devices communicate with each other in infrastructure mode via a WLAN access point/router.

It is the most popular mode to connect devices like laptops or smartphones to another network on the internet or intranet. By associating every client with an access point and this access point is connected in turn to another network so the client can send and receive packets of data through the access point. This mode is Server and clients (Master and slave) concept, which needs a server to serve the whole group.

Wireless networks running in infrastructure mode use one or more WAPs to connect the wireless network nodes centrally. This configuration is like the star

topology of a wired network. Infrastructure mode is also used to connect wireless and wired network segments. If you plan to set up a wireless network for many PCs, or you need to have centralized control over the wireless network, then infrastructure mode is what you need.

A Basic Service Set is a single WAP that serves a certain area (BSS). This service area can be extended by adding more access points. This is called, appropriately, an Extended Service Set (ESS).

Infrastructure mode wireless networks require a little more planning than ad hoc mode networks, such as where to locate the WAPs to offer appropriate coverage, and they provide a stable setting for permanent wireless network deployments. The infrastructure mode is better suited to business networks, or networks that need to share dedicated resources such as Internet connections and centralized databases.

2.2.2 Ad-Hoc Mode

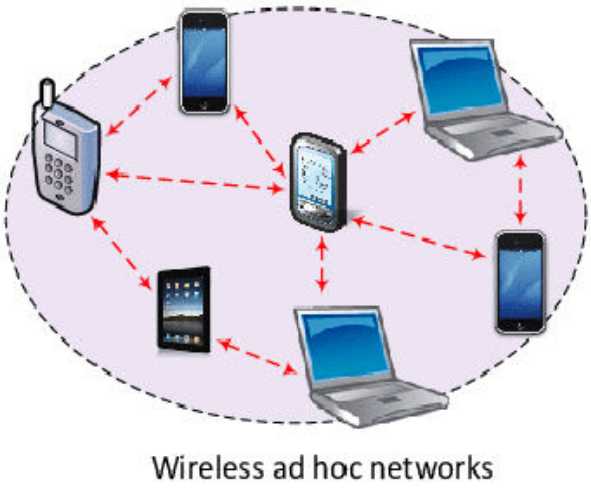


Figure 2 Ad-hoc Network[8]

As shown in **Figure2** above, The Ad-hoc mode refers to a wireless network structure where devices can communicate directly with each other. It is a built network without the use of existing infrastructure (no cables, no switches, no hubs, no access point, and no routers).

It is not popular as the infrastructure mode. It's developed by associating a group of computers and can send frames of data to each other directly. No access point is used. No need for a server to serve the whole group. Everyone in the group acts like a server and a client at the same time.

It is used for setting up the on-the-fly network, it is not important to have expensive equipment. The ad hoc network does not have a single failure point.

When there is a need for direct sharing of data or other files with another computer then ad hoc networks serve useful even when Wi-Fi network access is not there.

Deployment of wireless ad hoc networks is fast, and results are also produced similarly in case of an emergency where the suitability of wireless networks is there. Ad Hoc wireless network could be used for sharing the internet connection of the computer with another computer.

2.2.2.1 setting up an Ad-hoc network

automatically selects a Class B IP address for any node not connected to a DHCP server or hard coded to an IP address. Finally, ensure that the File and Printer Sharing service is running on all nodes.

2.2.3 Infrastructure Mode Vs Ad-Hoc Mode.

Configuring NICs for ad hoc mode networking requires you to address four things: SSID, IP addresses, channel, and sharing. You must set the NICs to function in ad hoc mode. Each wireless node must use the same network name (SSID). Also, no two nodes can use the same IP address-although this is unlikely with modern versions of Windows and the Automatic Private IP Addressing (APIPA) feature that.

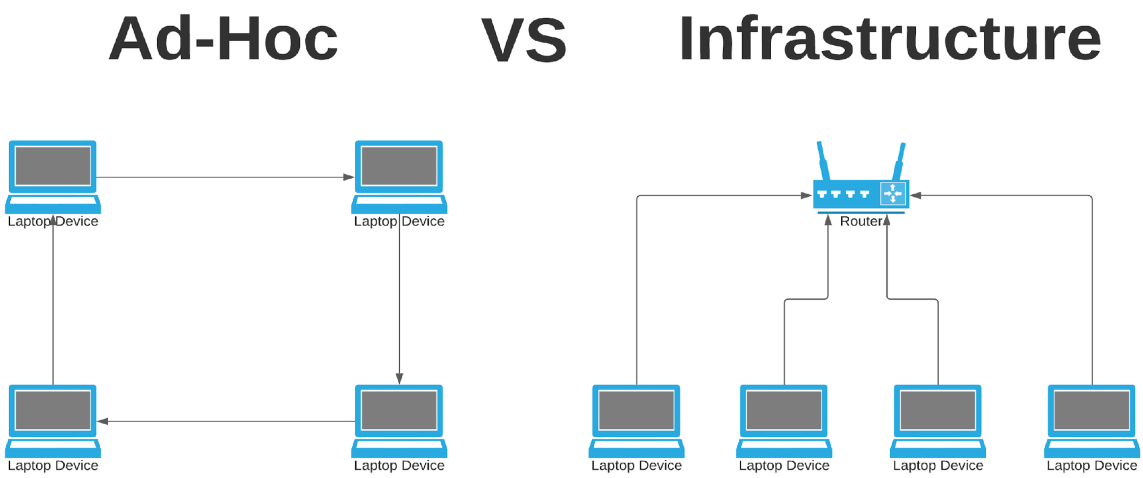


Figure 3 comparison between Ad-Hoc and Infrastructure connection *[9]*

The foremost common way our wireless devices are connected is through the infrastructure method of wireless networking. This implies our multiple devices are all connected to one access point, usually a wireless router to speak with the network. An ad-hoc wireless network is different in the sense that there is no universal access point needed and the nodes can freely communicate with one another directly. A node is often described as any single device that may transfer and receive data to and from other devices nearby.
This method of networking can often be called 'peer to peer' since we are transferring local files between nodes without an infrastructure connection. But the accidental is employed in embedded systems only, rarely employed in Linux but not in windows 10 OS, which makes our project one amongst its kind.

2.3 Chain Start

Our project is used to build a network of laptops but without any infrastructure devices (access point, switch, router, ...) Or using wires. Our project aims to implement one ad-hoc network using several infrastructure networks as shown in **Figure 4**.

This is developed when a master (user) begins the network, opens a hotspot, and can connect with clients, the user joins an existing network, up to eight and each client, in turn, can connect with another eight clients and so on as a tree as shown in the figure.

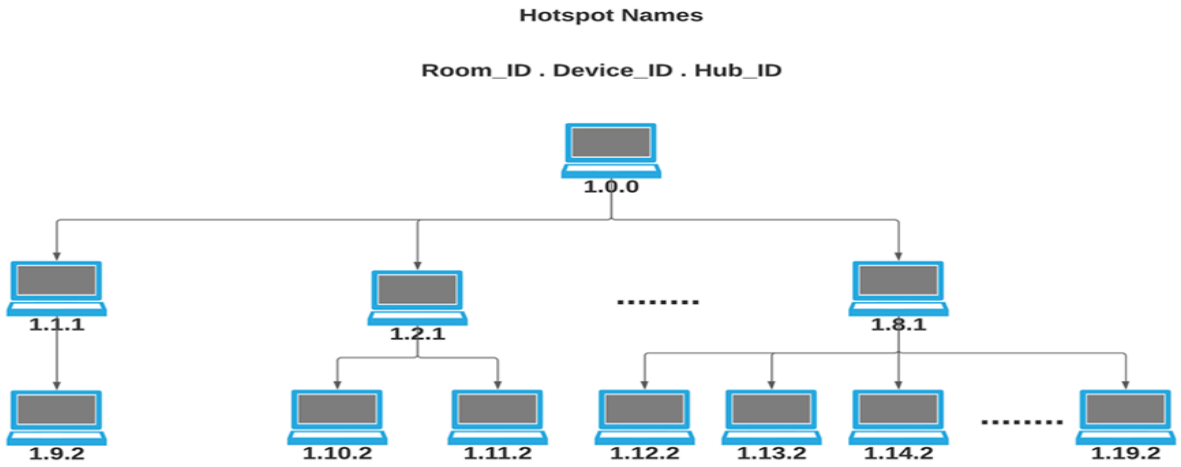


Figure 4 Example of ChainSpoT Network [9]

Master can start hotspot network, no infrastructure devices needed, just by his laptop or desktop computer (with wireless adapter).

The client can join the network which the master has created. The wide range network continues to grow (more users mean more coverage range) without being constrained by the number of clients.

Users (client or master) can send and receive text messages and files.

2.4 Static algorithm to determine the best node to connect

We use a power shell to control the windows hotspot. The names of hotpots (nodes) are used to carry information for newcomers and to organize the

connections among the network nodes. Lack of information for the users before entering the network. the newcomer user who wants to join the network won't have any information about the network or know the number of the connected branches or leaves he just sees the power and SSID: To solve this problem the program will share some information through the name of the node (SSID). And from this information, it will choose the best node to connect.

Room ID: If we have multiple ChainSpoT networks in the same place. Room ID will distinguish between them.

Device ID: Unique number for every device.

Hub ID: Describe the depth (number of nodes) between host and master.

Our priority is **Hub ID,** less hub depth means better connection also considering **Signal Power** do not be less than a certain limit.

If **Hub ID** is equal for 2 nodes Our second priority is **Signal Power.**

For the same **Hub ID,** bigger power means a better connection.

We apply the static algorithm to reduce the number of nodes which we must change in the Dynamic algorithm.

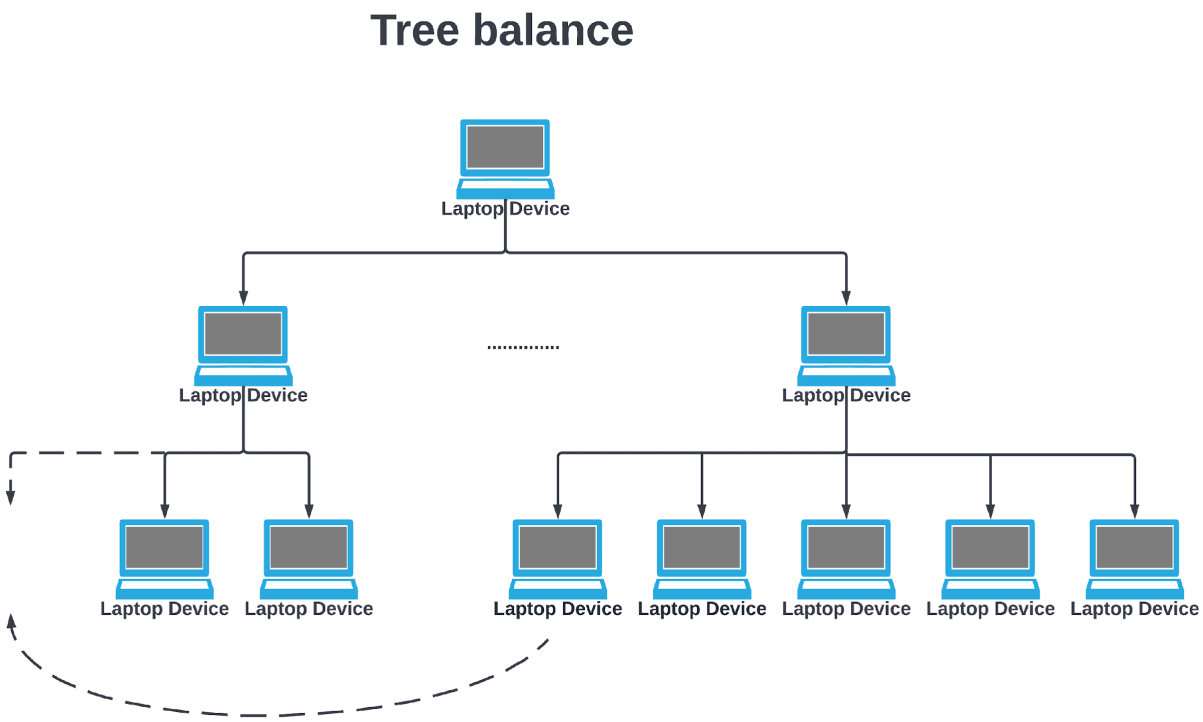
2.5 Dynamic Algorithm

The dynamic algorithm does whatever the static algorithm cannot do Because newcomer in the static algorithm just see the power of surrounding networks and their SSID (we put some information in SSID for the helping static algorithm) but this is not enough.

After the **ChainSpoT** network is built by the static algorithm. Nodes communicate with each other to make the balance of the **ChainSpoT** tree. After making a static algorithm as shown in the previous figure.

Every device collects and sends its information to the master. Then master restructures the tree to make the tree balance.

For example, a laptop with a hotspot name “1.19.2” if its power according to “1.1.1” is good enough. It will disconnect from its parent “1.8.1” and connect to the new parent “1.1.1” device to make tree balance. We will not make a 100% tree balance because this will cause a high waiting time for all the users. So, we trade between tree balance and waiting time.



2.6 Operating system dependency

2.6.1 Wi-Fi Network Profile

To connect to any Wi-Fi network, you must have a profile for this network. As shown in **Figure 5**, this profile contains Wi-Fi SSID, password, connection mode, connection type, protection type, and other properties.

```
syntax                                                                    Copy

<?xml version="1.0" encoding="US-ASCII"?>
<WLANProfile xmlns="https://www.microsoft.com/networking/WLAN/profile/v1">
    <name>SampleWPA2PSK</name>
    <SSIDConfig>
        <SSID>
            <name>SampleWPA2PSK</name>
        </SSID>
    </SSIDConfig>
    <connectionType>ESS</connectionType>
    <connectionMode>auto</connectionMode>
    <autoSwitch>false</autoSwitch>
    <MSM>
        <security>
            <authEncryption>
                <authentication>WPA2PSK</authentication>
                <encryption>AES</encryption>
                <useOneX>false</useOneX>
            </authEncryption>
        </security>
    </MSM>
</WLANProfile>
```

Figure 5 WPA2-Personal Profile Sample

2.6.2 “Netsh” Command Using CMD

Netsh is a command-line scripting utility (which can run from CMD or PowerShell) that allows you to display or modify the network configuration of a computer that is currently running. Netsh commands can be run by typing commands at the netsh prompt and they can be used in batch files or scripts. Remote computers and the local computer can be configured by using netsh commands.

Netsh also provides a scripting feature that allows you to run a group of commands in batch mode against a specified computer. With netsh, you can save a configuration script in a text file for archival purposes or to help you configure other computers.

You can also use the netsh command-line tool in the command prompt to view, export, import, delete wireless network profiles, recover the network security key, delete wireless network profiles, connect to the Wi-Fi network, and open and manage hosted network and other network management utility. But netsh starts

to become a legacy. New hardware network cards **no longer support** open hotspots with a net. **Figure 6** shows some netsh wlan commands.

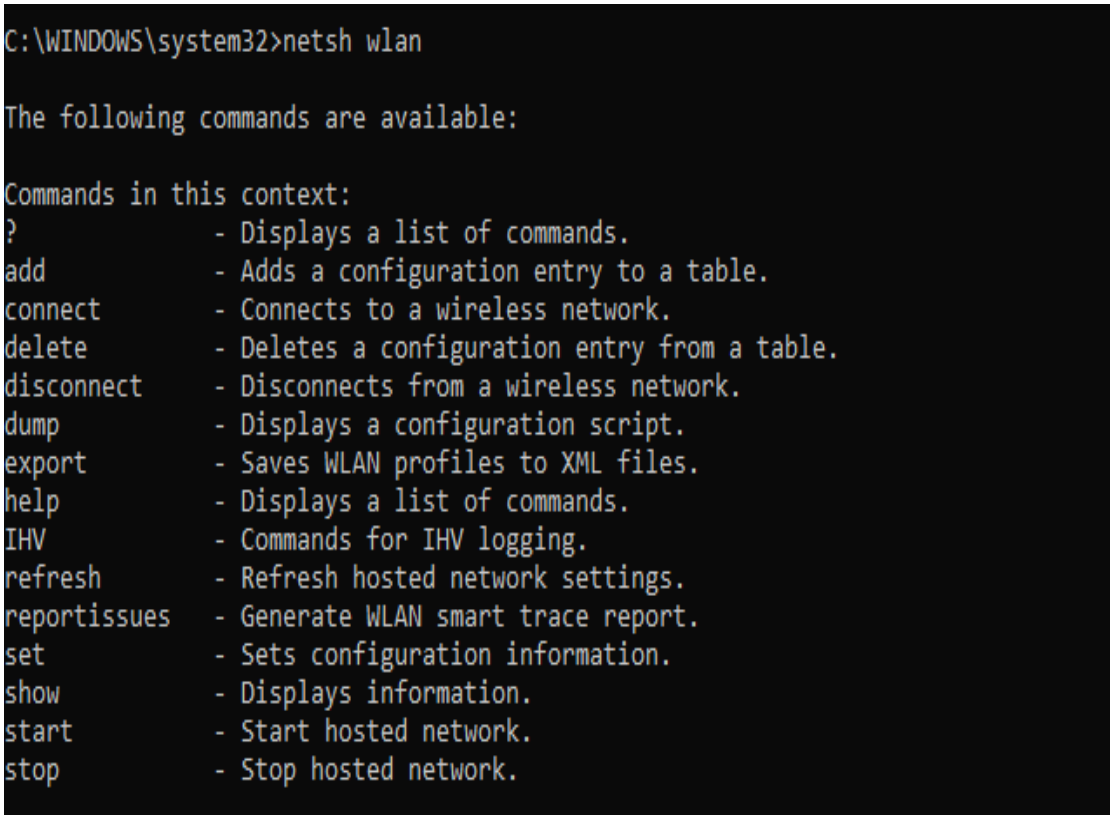
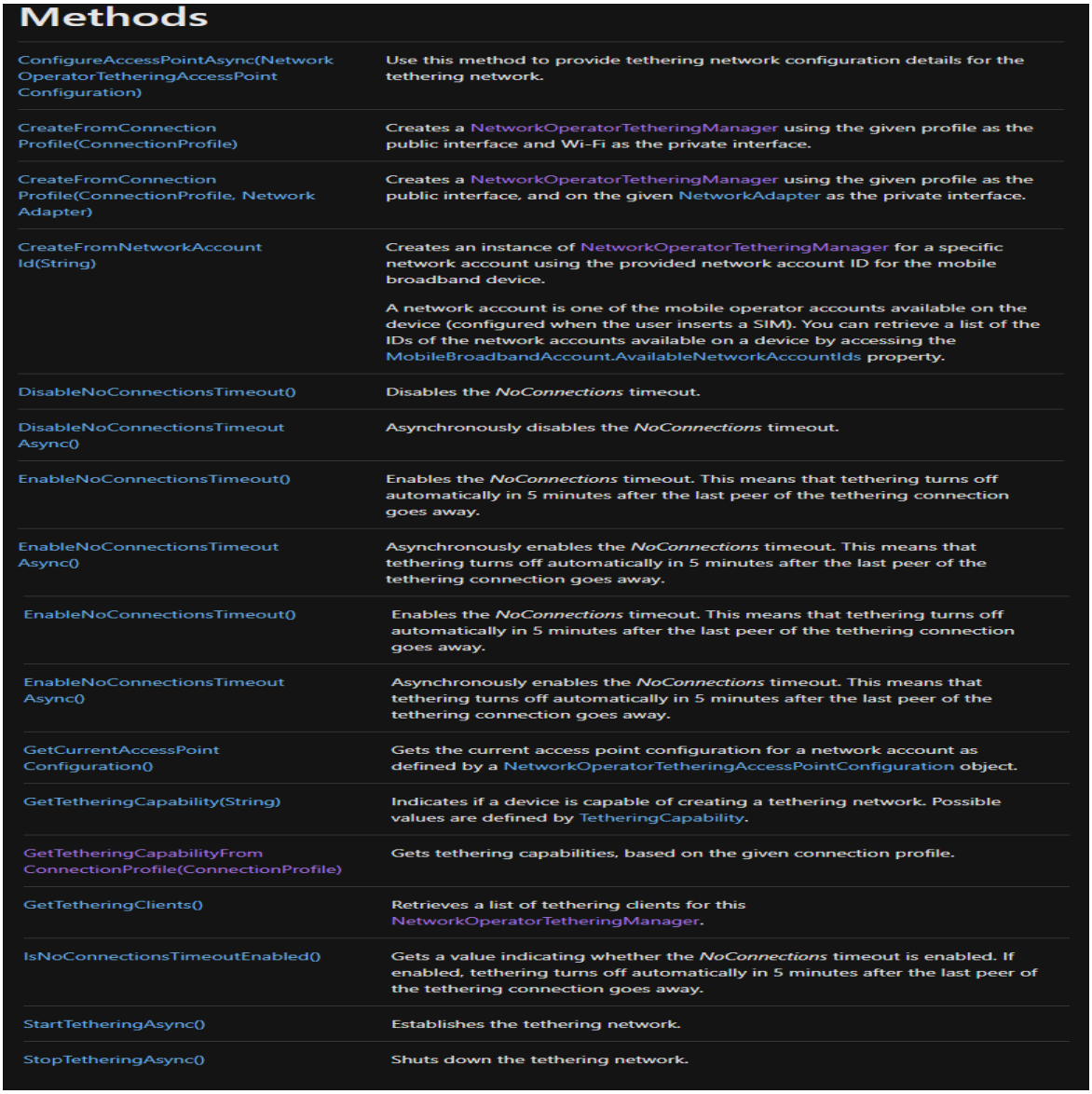


Figure 6 netsh wlan Commands

2.6.3 Windows API using PowerShell

New laptops and PCs have new network cards which do not support open hotspots with nets. So, we need to use windows API to open hotspots using PowerShell**.**

NetworkOperatorTetheringManager Class: This interface exposes the methods and properties used to control and configure tethering capabilities; these methods are shown in **Figure 7**.



Methods

ConfigureAccessPointAsync(NetworkOperatorTetheringAccessPointConfiguration)	Use this method to provide tethering network configuration details for the tethering network.
CreateFromConnectionProfile(ConnectionProfile)	Creates a NetworkOperatorTetheringManager using the given profile as the public interface and Wi-Fi as the private interface.
CreateFromConnectionProfile(ConnectionProfile, NetworkAdapter)	Creates a NetworkOperatorTetheringManager using the given profile as the public interface, and on the given NetworkAdapter as the private interface.
CreateFromNetworkAccountId(String)	Creates an instance of NetworkOperatorTetheringManager for a specific network account using the provided network account ID for the mobile broadband device. A network account is one of the mobile operator accounts available on the device (configured when the user inserts a SIM). You can retrieve a list of the IDs of the network accounts available on a device by accessing the MobileBroadbandAccount.AvailableNetworkAccountIds property.
DisableNoConnectionsTimeout()	Disables the *NoConnections* timeout.
DisableNoConnectionsTimeoutAsync()	Asynchronously disables the *NoConnections* timeout.
EnableNoConnectionsTimeout()	Enables the *NoConnections* timeout. This means that tethering turns off automatically in 5 minutes after the last peer of the tethering connection goes away.
EnableNoConnectionsTimeoutAsync()	Asynchronously enables the *NoConnections* timeout. This means that tethering turns off automatically in 5 minutes after the last peer of the tethering connection goes away.
EnableNoConnectionsTimeout()	Enables the *NoConnections* timeout. This means that tethering turns off automatically in 5 minutes after the last peer of the tethering connection goes away.
EnableNoConnectionsTimeoutAsync()	Asynchronously enables the *NoConnections* timeout. This means that tethering turns off automatically in 5 minutes after the last peer of the tethering connection goes away.
GetCurrentAccessPointConfiguration()	Gets the current access point configuration for a network account as defined by a NetworkOperatorTetheringAccessPointConfiguration object.
GetTetheringCapability(String)	Indicates if a device is capable of creating a tethering network. Possible values are defined by TetheringCapability.
GetTetheringCapabilityFromConnectionProfile(ConnectionProfile)	Gets tethering capabilities, based on the given connection profile.
GetTetheringClients()	Retrieves a list of tethering clients for this NetworkOperatorTetheringManager.
IsNoConnectionsTimeoutEnabled()	Gets a value indicating whether the *NoConnections* timeout is enabled. If enabled, tethering turns off automatically in 5 minutes after the last peer of the tethering connection goes away.
StartTetheringAsync()	Establishes the tethering network.
StopTetheringAsync()	Shuts down the tethering network.

Figure 7 NetworkOperatorTetheringManger Methods

2.6.4 Subprocess Python Library

The subprocess module provides a consistent interface for creating and working with additional processes. It defines one class, Popen, and a few wrapper functions that use that class. The constructor for Popen takes arguments to set up the new process so the parent can communicate with it via pipes. It provides all the functionality of the other modules and functions it replaces, and more. The API is consistent for all uses, and many of the extra steps of overhead needed

(such as closing extra file descriptors and ensuring the pipes are closed) are "built-in" instead of being handled by the application code separately.

2.6.5 PyQt Python Library

PyQt is a Python binding for Qt, which is a set of C++ libraries and development tools that include platform-independent abstractions for Graphical User Interfaces (GUI), as well as networking, threads, and regular expressions, SQL databases, SVG, OpenGL, XML, and many other powerful features.

We will design our program layout using Qt Designer. Then we will convert it to PyQt code.

2.7 Performance of the network

2.7.1 Network performance monitoring:

It is end-to-end network monitoring of the end-user experience. It differs from traditional monitoring because performance is monitored from the end-user perspective, and is measured between two points in the network, for example:

1) The performance of a user who works in the office, and the application they use in the company's data center.
2) The performance between two offices in a network.
3) The performance between the head office and the Internet.
4) The performance between your users and the cloud.
5) The performance between the master and the clients and the clients with each other in the tree (the case in our project).

2.7.2 Network Performance Concept

It is the analysis and review of collective network statistics, to define the quality of services offered by the underlying computer network that is primarily measured from an end-user perspective.

More simply, network performance refers to the analysis and review of network performance as seen by end-users [7].

There are three important concepts:

- Analysis and Review: Before you can analyze and compare network performance measurement data over time, you must first measure key network metrics associated with network performance and collect a history of the data you have measured.
- Measuring Performance: Network Performance refers to the quality of the network. The quality will be different depending on where in the network the measurements are taken (Distance between Master and Clients), The quality will also vary depending on when the measurements are taken. For example, your network may be performing well early during the workday with fewer users online and begin to degrade later in the day when more users connect to the network.
- The End User: The end-user experience is the most important factor when measuring performance. But just hearing about the user experience is not enough! With the right monitoring tools, we can turn the user's experience into measurable metrics, and translate those measurements into areas for improvement.

2.7.3 Network Performance Parameters

The most important parameters that Performance is Depending on are:

2.7.3.1 Latency

In any network, latency refers to the time it takes for data to reach its destination across the network. You usually measure network latency as a round trip delay, in milliseconds, considering the time it

takes for the data to get to its destination and then back again to its source.

Measuring the round-trip delay for network latency is important because a computer that uses a TCP/IP network sends a limited amount of data to its destination and then waits for an acknowledgment that the data has reached its destination before sending any more. Therefore, this round-trip delay has a significant impact on network performance.

2.7.3.2 Throughput

Throughput refers to the amount of data passing through the network from point A to point B in a determined amount of time. When referring to communication networks, throughput is the rate of data that was successfully delivered over a communication channel.

Measuring network throughput is usually done in bits per second (bit/s or bps).

"Internet Connection Speed" or "Internet Connection Bandwidth" is a general term used by internet companies to sell you high-speed internet, but is used by default to mean throughput, which is the actual rate of packet delivery over a specific medium.

That is why the best way to learn how to measure network throughput is to use Speed Tests.

How does Latency Affect Throughput?

When learning how to monitor latency, it is important to note that latency also affects the maximum throughput of a data transmission, which is how much data can be transmitted from point A to point B at each time.

But the reason that latency affects throughput is because of TCP (Transmission Control Protocol). TCP makes sure all data packets reach their destination successfully and in the right order. It also requires

that only a certain amount of data is transmitted before waiting for an acknowledgment.

2.7.3.3 Packet Loss

Packet loss refers to the number of data packets that were successfully sent out from one point in a network but were dropped during data transmission and never reached their destination.
Packet loss needs to be measured to know how many packets are being dropped across the network to be able to take steps to ensure that data can be transmitted as it should be. Knowing how to measure packet loss provides a metric for determining good or poor network performance.

2.7.3.4 User Quality of Experience

All the metrics we mentioned, in addition to user requirements and user perceptions, play a role in determining the perceived performance of your network.

Each metric on its own gives you an idea of how your infrastructure is performing, but you need to look at all these factors to give a true measurement of network performance.

The best way to measure and quantify user experience is by measuring User Quality of Experience (QoE). QoE allows you to measure performance from the end-user perspective and is the perception of the user of the effectiveness and quality of the system or service. Users base their opinions about the network on their perception of QoE.

Finally, there are many parameters we can take into consideration for measuring performance and we mentioned some of these parameters, but we found that it is so difficult to measure these parameters theoretically by Equations, so we are forced to measure these parameters **empirically** by testing it in the Lab.

Chapter Three: Analysis and Methods

In this chapter, we are going to discuss the main methods we will use to implement our project (ChainSpoT) and present the test cases on which our work is based.

3.1 Windows CMD

We will use CMD to control the Wi-Fi interface and Windows 10 Netsh Wlan Commands for Wi-Fi Management:

We can view, troubleshoot, and configure virtually every network adapter on a local or remote computer using these commands.

We can use the Netsh WLAN command also to generate reports, import, export, and delete wireless profiles from our devices Laptops, Pcs).

Commands we will use and their usage:

1- netsh WLAN show profiles

Views wireless network profiles.

2-netsh WLAN export profile name="profile name" key=clear folder=c:\

Exports wireless profile.

3-netsh wlan add profile filename="path_and_filename.xml" interface="interface_name"

Imports the wireless profile from the XML file.

4-netsh wlan set profileorder name=[profile name]interface=[interface_name] priority=1

Sets a wireless network's priority.

5-netsh WLAN set profile parameters name=[profilename] connectionmode=manual

Automatically stops the connection to a wireless network.

6-netsh WLAN set hostednetwork mode=allow ssid=[your_virtual_network_name] key=[your_network_password]

Configures the Wireless Hosted Network.

7-`netsh WLAN start hostednetwork`

Enables the Wireless Hosted Network.

8-`netsh WLAN stop hostednetwork`

Disables the Wireless Hosted Network.

9-`netsh WLAN show hostednetwork`

Retrieves the Wireless Hosted Network details.

10- `netsh Wlan`

With netsh WLAN help you can list all available options as you see in figure 6.

3.2 Windows API Using PowerShell

New laptops and PCs have new network cards which do not support open hotspots with nets so, we need to use windows API to open hotspots using PowerShell [11].

To manage hotspot using PowerShell, at first the following function is created.

```
Add-Type -AssemblyName System.Runtime.WindowsRuntime

$asTaskGeneric = ([System.WindowsRuntimeSystemExtensions].GetMethods() | ?
{ $_.Name -eq 'AsTask' -and $_.GetParameters().Count -eq 1 -and
$_.GetParameters()[0].ParameterType.Name -eq 'IAsyncOperation`1' })[0]

Function Await($WinRtTask, $ResultType) {

    $asTask = $asTaskGeneric.MakeGenericMethod($ResultType)

    $netTask = $asTask.Invoke($null, @($WinRtTask))

    $netTask.Wait(-1) | Out-Null

    $netTask.Result }

Function AwaitAction($WinRtAction) {
```

```
    $asTask = ([System.WindowsRuntimeSystemExtensions].GetMethods() | ? {
$_.Name -eq 'AsTask' -and $_.GetParameters().Count -eq 1 -and
!$_.IsGenericMethod })[0]

    $netTask = $asTask.Invoke($null, @($WinRtAction))

    $netTask.Wait(-1) | Out-Null }

$connectionProfile =
[Windows.Networking.Connectivity.NetworkInformation,Windows.Networking.C
onnectivity,ContentType=WindowsRuntime]::GetInternetConnectionProfile()

$tetheringManager =
[Windows.Networking.NetworkOperators.NetworkOperatorTetheringManager,Win
dows.Networking.NetworkOperators,ContentType=WindowsRuntime]::CreateFro
mConnectionProfile($connectionProfile)
```

Then the function can be called to accomplish any purpose to manage the hotspot

To open hotspot:

```
Await($tetheringManager.StartTetheringAsync())
([Windows.Networking.NetworkOperators.NetworkOperatorTetheringOperationRe
sult])
```

To close hotspot:

```
AwaitAction($tetheringManager.ConfigureAccessPointAsync($accessPointConfig
uration))
([Windows.Networking.NetworkOperators.NetworkOperatorTetheringOperationRe
sult])
```

To change name and password

```
$accessPointConfiguration =
$tetheringManager.GetCurrentAccessPointConfiguration()

$accessPointConfiguration.Ssid = <Your SSID>

$accessPointConfiguration.Passphrase = <8 to 32 characters>
```

During work, we have noticed that we are not able to open a hotspot when there is no connection with the internet and the above PowerShell function is only valid to manage a hotspot when there the s internet.

After a lot of research and not reaching any solution on any source or website.

We returned to the used PowerShell function we got from reference (9) to manage the hotspot.

By Checking the code accurately, we have noticed the function

GetInternetConnectionProfile()

that returns the currently connected network profile with the internet and returns null if there is no internet so due to this function, we cannot open the hotspot or manage it on a device not connected to the internet.

we changed this function to another function

GetConnectionProfiles()

Which returns all the saved network profiles regardless of the existence of the internet and by looping around them with a for loop in the PowerShell we can get an object with only one saved network profile that enables us to manage the hotspot.

So, the function changed to the following:

```
[Windows.System.UserProfile.LockScreen,Windows.System.UserProfile,ContentType=WindowsRuntime] | Out-Null

Add-Type -AssemblyName System.Runtime.WindowsRuntime

$asTaskGeneric = ([System.WindowsRuntimeSystemExtensions].GetMethods()|?{$.Name-eq'AsTask'-and $_.GetParameters().Count-eq 1 -and $_.GetParameters()[0].ParameterType.Name -eq 'IAsyncOperation`1' })[0]

Function Await($WinRtTask, $ResultType) {

    $asTask = $asTaskGeneric.MakeGenericMethod($ResultType)
```

```
    $netTask = $asTask.Invoke($null, @($WinRtTask))
    $netTask.Wait(-1) | Out-Null
    $netTask.Result }
Function AwaitAction($WinRtAction) {
    $asTask=
([System.WindowsRuntimeSystemExtensions].GetMethods()|?{$.Name-eq
'AsTask' -and $_.GetParameters().Count -eq 1 -and !$_.IsGenericMethod })[0]
    $netTask = $asTask.Invoke($null, @($WinRtAction))
    $netTask.Wait(-1) | Out-Null }
$connectionProfile=
[Windows.Networking.Connectivity.NetworkInformation,Windows.Networking.C
onnectivity,ContentType=WindowsRuntime]::GetConnectionProfiles()
foreach($i in $connectionProfile){
$randomProfile =$i  }
$tetheringManager=
[Windows.Networking.NetworkOperators.NetworkOperatorTetheringManager,Win
dows.Networking.NetworkOperators,ContentType=WindowsRuntime]::CreateFro
mConnectionProfile($randomProfile)
```

3.3 Network Operator Tethering Manager Methods

CreateFromConnectionProfile($connectionProfile):

Creates a NetworkOperatorTetheringManager [10] using the given profile as the public interface and Wi-Fi as the private interface and returns it.

Parameters of the function: Connection profile to be used.

GetConnectionProfiles():

Gets all the saved connection profiles even if they are not associated with the internet connection currently.

It has no parameters.

GetCurrentAccessPointConfiguration():

Indicates the network account id and specifies the passphrase used for authentication when establishing a connection over the tethering network to Get the current access point configuration for the network account.

It has no parameters.

GetTetheringClients():

Retrieves a list of tethering clients for this NetworkOperatorTetheringManager.

And returns a list of clients.

StartTetheringAsync():

Establishes the tethering network and returns the result of the tethering network operation.

StopTetheringAsync():

Shuts down the tethering network and returns the result of the tethering network operation.

Python 3: We will use Python to control PowerShell and the workflow of the ChainSpoT program.

3.4 Subprocess Python Library

The Subprocess module allows you to spawn new processes, connect to their input/output/error pipes, and simply obtain their return code: it is used for running external programs and reading their outputs in your python code.

Examples of external programs that we use subprocess to obtain their outputs:

1) command prompt (CMD)

2) Windows PowerShell

Subprocess functions:

the most useful function of the subprocess is the subprocess. the run() function which is used to run an external program from your Python code as shown in Figure 8 (from our ChainSpoT program).

```
def run(self,cmd):
    self.completed = subprocess.run(["powershell", "-Command", cmd], capture_output=True)
    return self.completed
```

Figure 8 Subprocess example

Subprocess can be used in two ways:

1) using convenience functions: call, check_call, and check_output

2) Popen interface class, which allows great customization, being able to replicate the behavior of any of the convenience functions.

Convenience functions:

Subprocess. call: runs command with arguments, wait for it to complete, then returns the return code

Check_output: run the command with arguments and return its output as a string as shown in fig below

Popen: the popen class offers the flexibility to handle fewer common cases not covered by the convenience functions.

3.5 PyQt Python Library

It is one of the most common and powerful platform GUI libraries. PyQt API is a set of modules containing many classes and functions.

PyQt is compatible with all the common operating systems including Linux, Windows, and Mac OS. It is dual-licensed, available under GPL as well as a commercial license.

PyQt is a Python binding for Qt, which is a set of C++ libraries and development tools that include platform-independent abstractions for Graphical User Interfaces (GUI), as well as networking, threads, and regular expressions, SQL databases, SVG, OpenGL, XML, and many other powerful features [15].

We will design our program layout using Qt Designer by using its simple and easy drag-and-drop interface so we can make the design we want or imagine without having to write a code. It starts by choosing the needed widgets or then dropping the required components on them. We can also change the form or each component's properties easily. Then we will convert it to PyQt code by the following command:

```
pyuic5 -x filename.ui -o filename.py
```

3.6 The connection of the devices forward and reverse

To make the chatting and sharing files program, the devices of the network should:

- connect each other
- have a route to pass through to reach all devices.

In the intermediary device-connected network the router or switch handles the routing and connection between the connected devices. For ChainSpoT the

network has no intermediary device to connect the devices, The network is connected via a hotspot which will connect only one hub.

To determine how to implement it we went in three different ways:

1. Windows Routing table.
2. Sockets & Threads.
3. Gns3 virtual Router.

As we were not certain about any solution, so we went for all of them in parallel.

3.6.1 Windows Routing Table.

In networking, Routing is the process of deciding the path to the destination and directing the packets through this path [12].

With the routing process, routers decide where to send the packets based on their routing table which contains different routing destinations' IP addresses with their subnet masks gateways addresses and other information.

Windows allows us to get the IPv4 routing table by **route PRINT -4** command on the command prompt.

For more information about the routing table and its parameters see Appendix A.

Due to the lack of window resources for networks built by hotspots, our results depend on testing.

From testing, we will try to understand the routing tables we get from the test cases and make the best use of them.

3.6.1.1 Testing

First test case

Making a network of 4 devices connected with no internet in a serial branch.

The second device is connected to the first device by hotspot, the third device is also connected to the second device by hotspot, and so on making the shown network in **Figure 9**.

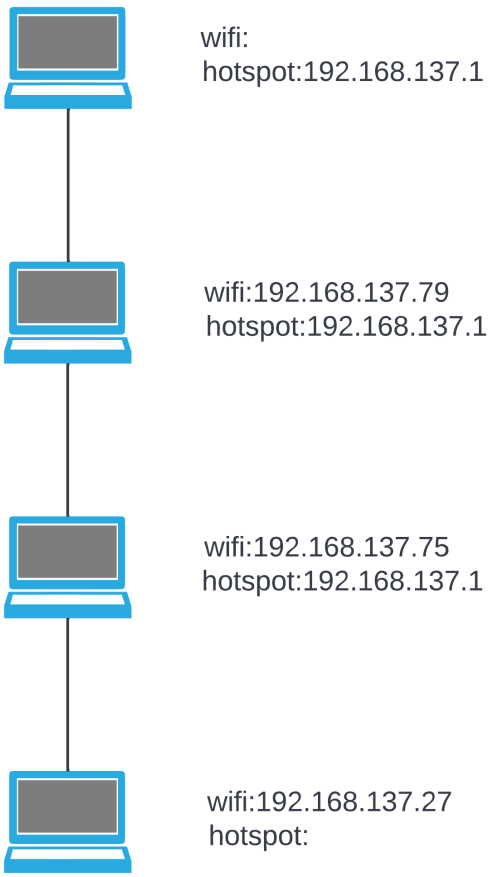


Figure 9 First Test Case *[9]*

All devices in the network have the same hotspot interface IP of 192.168.137.1

The routing table of the first device shown in figure 10:

```
IPv4 Route Table
===========================================================================
Active Routes:
Network Destination        Netmask          Gateway       Interface  Metric
        127.0.0.0        255.0.0.0         On-link         127.0.0.1    331
        127.0.0.1  255.255.255.255         On-link         127.0.0.1    331
  127.255.255.255  255.255.255.255         On-link         127.0.0.1    331
    192.168.137.0    255.255.255.0         On-link     192.168.137.1    306
    192.168.137.1  255.255.255.255         On-link     192.168.137.1    306
  192.168.137.255  255.255.255.255         On-link     192.168.137.1    306
        224.0.0.0        240.0.0.0         On-link         127.0.0.1    331
        224.0.0.0        240.0.0.0         On-link     192.168.137.1    306
  255.255.255.255  255.255.255.255         On-link         127.0.0.1    331
  255.255.255.255  255.255.255.255         On-link     192.168.137.1    306
===========================================================================
Persistent Routes:
  None
```

Figure 10 Routing Table for Device 1

It has no Wi-Fi interface in its routing table but has a Hotspot interface because it starts the network with no internet.

The network ID 127.0.0.0 is a reserved network for loopback addresses to the localhost. The loopback address is also known as localhost which means this computer. It is used to access the network services that are running on the host by its loopback network interface with a range from 127.0.0.0 to 127.0.0.255 *[13].*

127.255.255.255 is also a broadcast loopback address in a network with ID 127.255.255.0

As said this device has only the hotspot interface in the routing table with IP 192.168.137.1 and as shown in the above table this interface has three routes with three destinations:

Route to 192.168.137.0 network of the current device from hotspot IP 192.168.137.1 and other parameters as shown in the above routing table.

Route to 192.168.137.1 which is the hotspot interface.

Route to 192.168.137.255 is a broadcast address in the network 192.168.137.0

224.0.0.0 which is a network providing multicasting addresses with a range of addresses from 224.0.0.0 to 224.0.0.255

255.255.255.255 which is the global broadcast address, which means that the IP stack is supposed to send the packet to all network interfaces, and routers that are configured to forward broadcasts are supposed to send them on.

The routing table of the second device shown in Figure11.

```
IPv4 Route Table
===========================================================================
Active Routes:
Network Destination        Netmask          Gateway       Interface  Metric
          0.0.0.0          0.0.0.0    192.168.137.1   192.168.137.79     55
        127.0.0.0        255.0.0.0         On-link         127.0.0.1    331
        127.0.0.1  255.255.255.255         On-link         127.0.0.1    331
  127.255.255.255  255.255.255.255         On-link         127.0.0.1    331
    192.168.137.0    255.255.255.0         On-link    192.168.137.79    311
    192.168.137.0    255.255.255.0         On-link     192.168.137.1    311
    192.168.137.1  255.255.255.255         On-link     192.168.137.1    311
   192.168.137.79  255.255.255.255         On-link    192.168.137.79    311
  192.168.137.255  255.255.255.255         On-link    192.168.137.79    311
  192.168.137.255  255.255.255.255         On-link     192.168.137.1    311
        224.0.0.0        240.0.0.0         On-link         127.0.0.1    331
        224.0.0.0        240.0.0.0         On-link    192.168.137.79    311
        224.0.0.0        240.0.0.0         On-link     192.168.137.1    311
  255.255.255.255  255.255.255.255         On-link         127.0.0.1    331
  255.255.255.255  255.255.255.255         On-link    192.168.137.79    311
  255.255.255.255  255.255.255.255         On-link     192.168.137.1    311
===========================================================================
Persistent Routes:
  None
```

Figure 11 Routing Table for Device 2

It is different from the first one as it has Wi-Fi and Hotspot interfaces in the routing table, so it has additional routes from the Wi-Fi interface.

From the routing table, we can find that the Wi-Fi interface has an IP address of 192.168.137.79

0.0.0.0 is the default route used when the desired destination is a different IP not listed in the routing table to where packets are directed.

The routing table of the third device shown in Figure 12.

```
IPv4 Route Table
===========================================================================
Active Routes:
Network Destination        Netmask          Gateway       Interface  Metric
          0.0.0.0          0.0.0.0    192.168.137.1   192.168.137.75     55
        127.0.0.0        255.0.0.0         On-link         127.0.0.1    331
        127.0.0.1  255.255.255.255         On-link         127.0.0.1    331
  127.255.255.255  255.255.255.255         On-link         127.0.0.1    331
    192.168.137.0    255.255.255.0         On-link    192.168.137.75    311
    192.168.137.0    255.255.255.0         On-link     192.168.137.1    311
    192.168.137.1  255.255.255.255         On-link     192.168.137.1    311
   192.168.137.75  255.255.255.255         On-link    192.168.137.75    311
  192.168.137.255  255.255.255.255         On-link    192.168.137.75    311
  192.168.137.255  255.255.255.255         On-link     192.168.137.1    311
        224.0.0.0        240.0.0.0         On-link         127.0.0.1    331
        224.0.0.0        240.0.0.0         On-link    192.168.137.75    311
        224.0.0.0        240.0.0.0         On-link     192.168.137.1    311
  255.255.255.255  255.255.255.255         On-link         127.0.0.1    331
  255.255.255.255  255.255.255.255         On-link    192.168.137.75    311
  255.255.255.255  255.255.255.255         On-link     192.168.137.1    311
===========================================================================
Persistent Routes:
  None
```

Figure 12 Routing Table for Device 3

It also has both Wi-Fi and hotspot interfaces as it is centered between two devices.

From the routing table, we can find that the Wi-Fi interface has an IP address of 192.168.137.75

The routing table of the fourth device

It has no hotspot interface in the routing table as it is the end of the network.

The routes which are shown in **Figure 13**. are the same as those discussed above.

```
IPv4 Route Table
===========================================================================
Active Routes:
Network Destination        Netmask          Gateway       Interface  Metric
          0.0.0.0          0.0.0.0    192.168.137.1   192.168.137.27     55
        127.0.0.0        255.0.0.0         On-link         127.0.0.1    331
        127.0.0.1  255.255.255.255         On-link         127.0.0.1    331
  127.255.255.255  255.255.255.255         On-link         127.0.0.1    331
    192.168.137.0    255.255.255.0         On-link   192.168.137.27    311
   192.168.137.27  255.255.255.255         On-link   192.168.137.27    311
  192.168.137.255  255.255.255.255         On-link   192.168.137.27    311
        224.0.0.0        240.0.0.0         On-link         127.0.0.1    331
        224.0.0.0        240.0.0.0         On-link   192.168.137.27    311
  255.255.255.255  255.255.255.255         On-link         127.0.0.1    331
  255.255.255.255  255.255.255.255         On-link   192.168.137.27    311
===========================================================================
Persistent Routes:
  None
```

Figure 13 Routing Table for Device 4

From the four above routing tables, we noticed that each device is considered a network and acts as a router with a routing table so 192.168.137.79,192.168.137.75 and 192.168.137.27 are not in the same network.

The Wi-Fi interface and the hotspot interface are separated from each other.

By testing the connection by the ping IP command, we got the following results:

Upward connection

After testing and trying to understand the tables, we think that each device can ping or connect to its upper direct connected devices by route 1 in the routing table as each upper device has a different IP address from those listed in the routing table of the current device.

```
Network Destination        Netmask          Gateway       Interface  Metric
          0.0.0.0          0.0.0.0    192.168.137.1   192.168.137.79     55
```

```
Network Destination        Netmask          Gateway       Interface  Metric
          0.0.0.0          0.0.0.0    192.168.137.1   192.168.137.75     55
```

```
Network Destination        Netmask          Gateway       Interface  Metric
          0.0.0.0          0.0.0.0    192.168.137.1   192.168.137.27     55
```

So, it is possible to send packets by sending them to the next hop and then the coming hop until reaching the destination.

When we added a router to the network, with different IP from IPs listed in the routing tables of the devices in the network, connected to the first device. other devices in the network could see and ping it by the above routes.

For example, the last connected device in the network can ping the router by using route one in its routing table which directs the packets to the upper device through the hotspot and the upper device to the next device until reaching the router.

Downward connection

We think that each device uses the following routes in its routing table:

When needing packets to be directed to a certain location it uses the route with a subnet mask 255.255.255.255 to force packets to go only to this destination.

Second test case

Making a network of 4 devices in a way that makes one the parent(master) and the other three devices the children to this parent as shown in Figure 14.

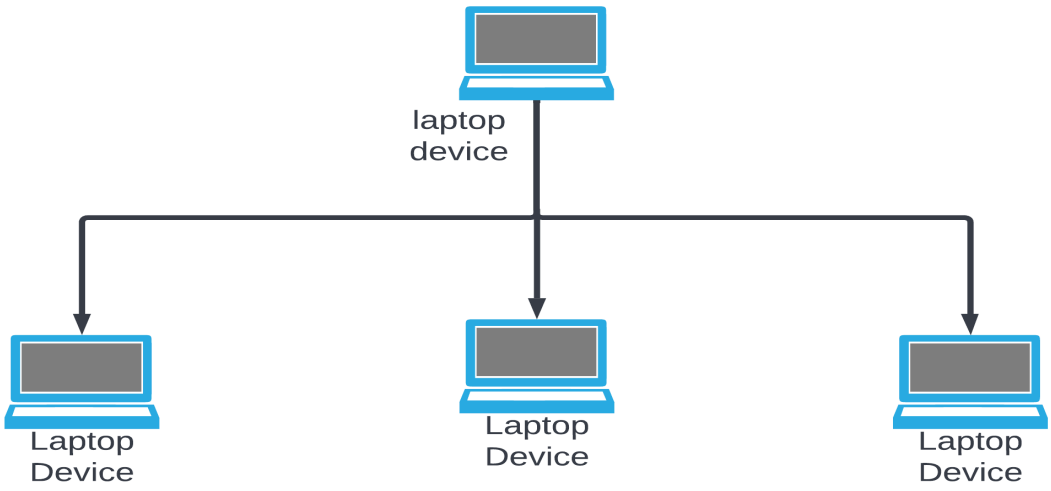


Figure 14 Second Use Case *[9]*

Each child connects to the parent by hotspot.

We noticed that:

Each child can send packets to its parent.

The children can also send packets to themselves.

When we expanded the network by making grandchildren as shown in Figure 15, these grandchildren could not send packets to themselves as they are considered different networks, they could only send packets to their parents (children of the master).

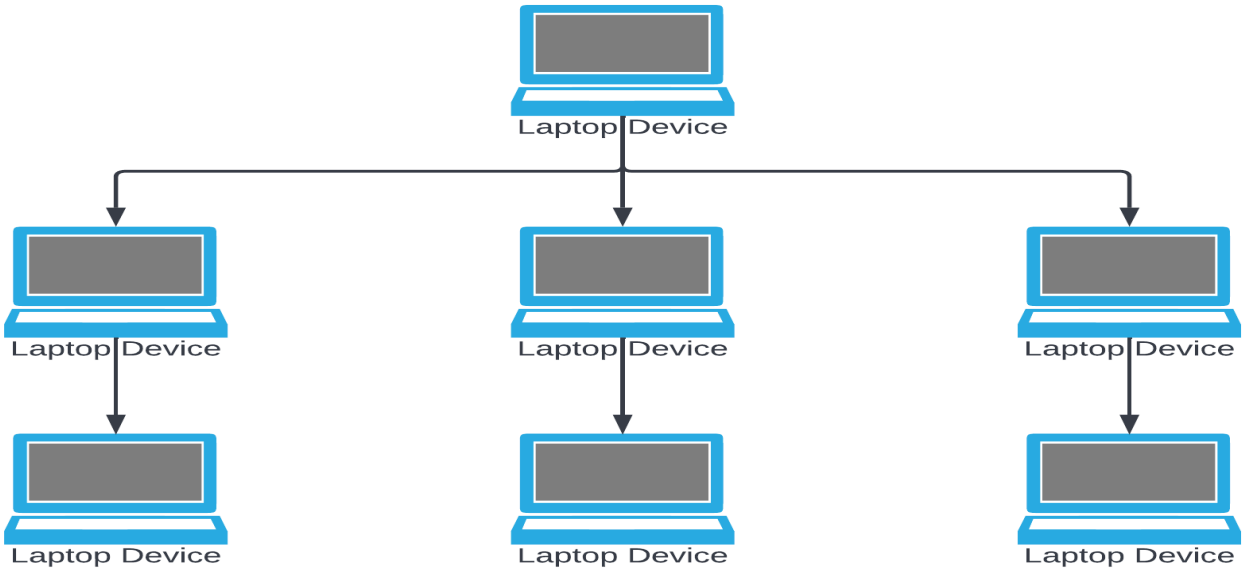
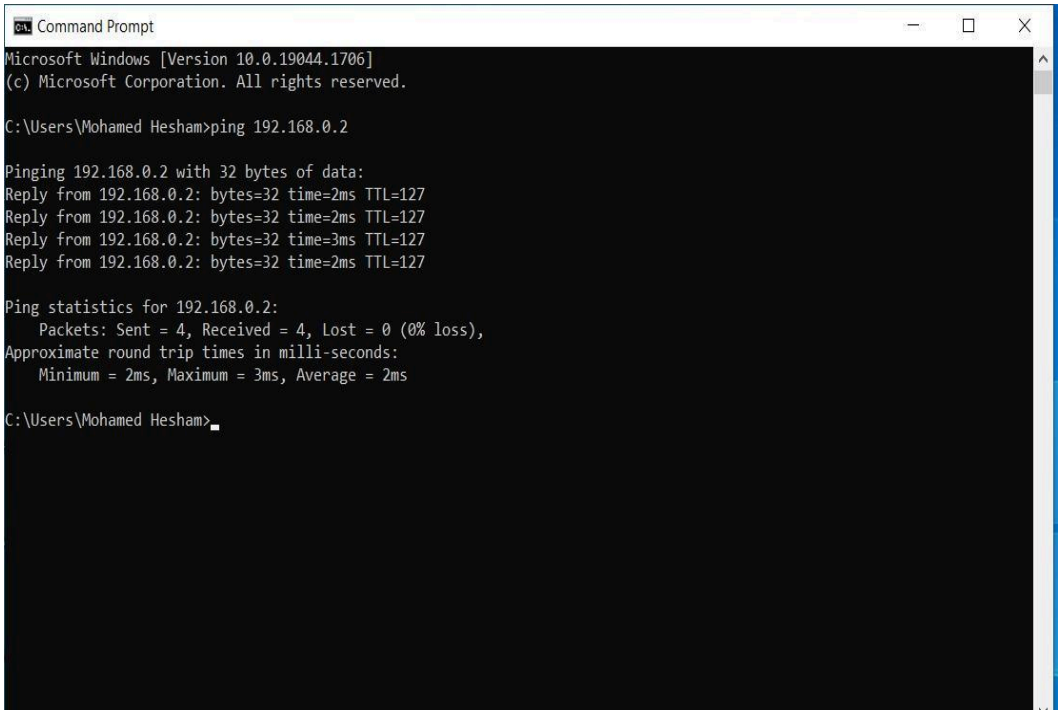


Figure 15 Grandchildren use case [9]

From the two test cases, Our based tree in the project can be accomplished by a centralized network by setting a loopback interface, with a different IP from the destination IPs listed in the routing tables of the whole devices in our network, in the device which starts the network as explained above when adding a router with different IP so, also other devices in the network can connect to this loopback interface via route one in the routing table of each device through the hotspot.

When setting the IP of the loopback interface 192.168.0.2 with subnet mask 255.255.255.0 and testing the ping command from any device on the network we receive successful replays.



For more information about the loopback interface see Appendix B

3.6.2 The Sockets & Threading

Sockets and the sockets API [11] are used to send messages across a network. They provide inter-process communication (IPC) for more information about sockets see Appendix C

3.6.2.1 The Decentralized Approach

Each node is considered as a client and a server at the same time so all nodes will have their listeners on until other nodes try to connect to them.

In a simple case of three connected devices, the connection works well, and the simple messages are sent to the target device. As shown in **Figure 16**. we attempted to send a message “Ahmed” from the client device connected via

hotspot to another device which also connected via hotspot to the master. The message is successfully sent to all devices [2].

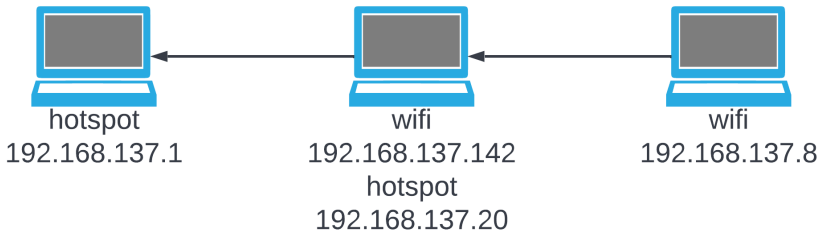


Figure 16 Simple Example [9]

In **Figure 17**. for the sender here the program is sending a simple message "Ahmed."

Figure 17 Code of The Simple Example 1

Next, the 2nd device will receive the message on the hotspot interface, print it, and forward it to the last device through the Wi-Fi interface.

```
import socket
host_hot= "192.168.137.20"
port= 5000
host_wifi= "192.168.137.1"


s_wifi= socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s_wifi.connect((host_wifi,port))


s_hot= socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s_hot.bind((host_hot,port))
s_hot.listen(1)
con , add = s_hot.accept()
while True:
    mess = con.recv(1024)
    mess = mess.decode()
    print(mess)
    s_wifi.send(mess.encode())
```

```
*Python 3.7.9 Shell*
File Edit Shell Debug Options Window Help
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 18:58:18) [MSC v.1900 64 bit (AMD64)] o
win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: C:/Users/Mohamed Hesham/Desktop/repeater.py =============
ahmed
```

Figure 18 Code of The Simple Example 2

At the last device, the message is received on the hotspot interface and printed.

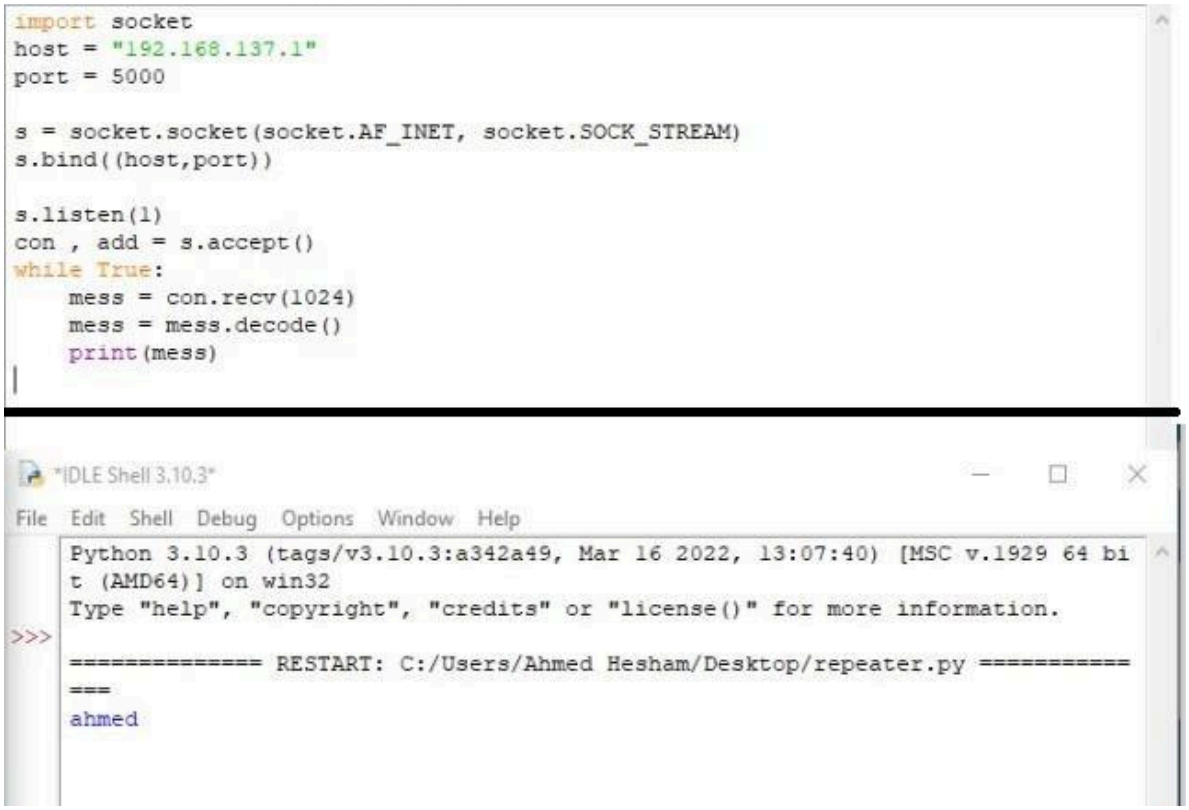


Figure 19 Code of The Simple Example 3

As shown in the figures 18,19 above the code is as simple as the program.

It is just one way; the message is sent from beginning to end.

It is not a reliable chatting form. Reliable chatting has two ways of communication each user sends and receives. So, each user should have a listener and sender working at the same time which leads us to the usage of threading.

Threading in a chat program enables us to run the sending and receiving codes at the same time, but threading makes the code more complicated.

What about tree of devices chatting with each other? the more devices the more complicated the code is.

For the tree network:

The structure of the network tree will be sent to all nodes and each node will be able to know its position in the tree and will decide which path to take to reach the message to its destination.

Since we can have multiple clients to each node(server), each of them is put in a separate thread and start listening for his commands.

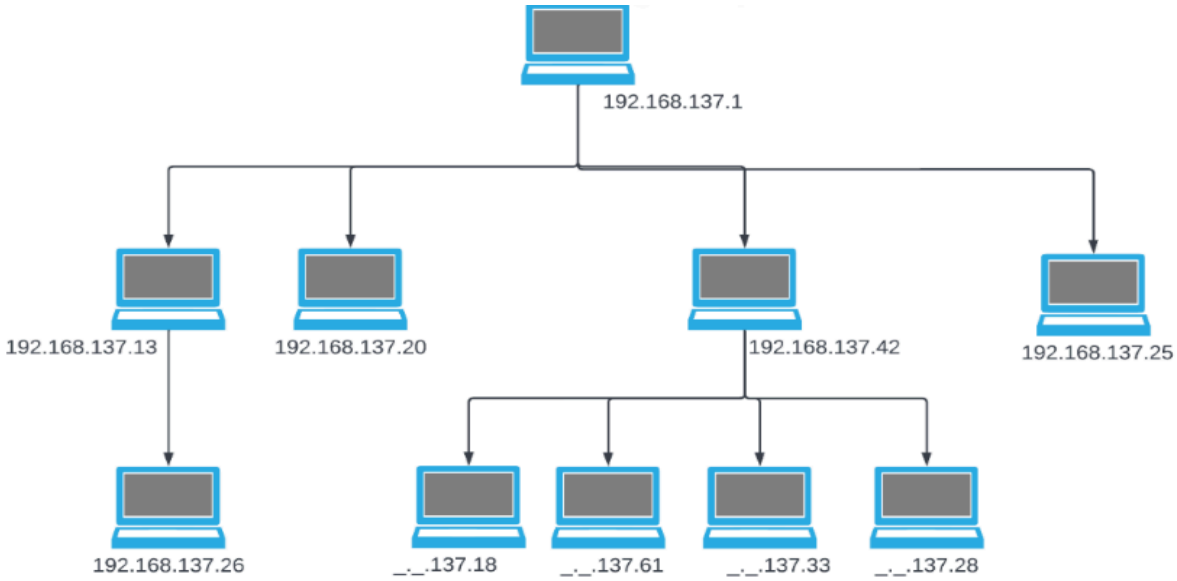


Figure 20 Decentralized Use Case[9]

As shown in the Figure 20, the ChainSpoT chat program using sockets needs a code for the listener and sender (two linkers up & down) between two different devices. This will be done by threading which makes the code even more complicated.

For explanation device (192.168.137.42) 'll be connected to nine devices, it will have a listener of (8) for its Children (hotspot-connected devices) and a listener of (1) for its parent (Wi-Fi-connected device). It also has 2 linkers to transfer message from Children to parent and reverse vice. For the linker, it will need to look for the address and take a decision to know where to send the message depending on the structure of the tree (network). All these processes must be up together all the time which increases the need for threading, makes the code even more

complicated, and increases the consumption of hardware resources. And the same for all other devices in the tree.

3.6.2.2 The Centralized Approach

As mentioned in 3.6.1.1, The base of the centralized approach is the loopback solution which depends on a centralized device “master.”

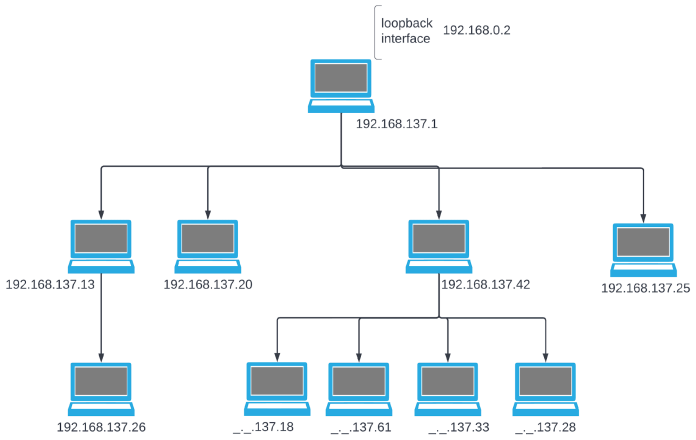


Figure 21 Centralized Use Case [9]

The tree consists of a master device with a loopback interface acting like a server and connected device (clients).

Clients first connect to the server then send their messages to the server and the server forwards them to its connected devices. The master forwards the message to all devices "broadcast”, or the destination device “unicast”.

3.6.3 GNS3

It is open-source software that emulates Cisco router and switches hardware to simulate complex networks. We can use GNS3 on any computer to test with various router configurations.

It allows us to build our network and create many projects by using Cisco and non-Cisco technology, and we can access these projects anytime without needing of internet connection [14].

To see a small introduction about Gns3 and the configuration of the network you can see Appendix D

CONFIGURATION AND BUILDING OUR NETWORK:

After installing GNS3, we can configure our network and we use the bridged from loopback interface in some steps shown in Appendix D.

After these steps, the connection is made as shown in the **Figure 22**.

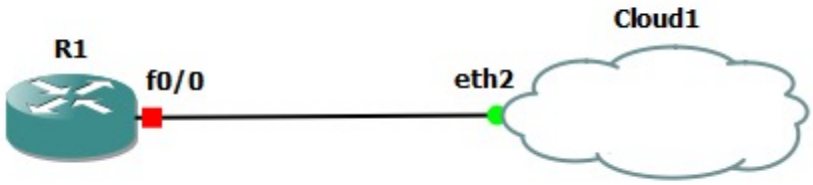


Figure 22 GNS3 Network

Now we start the router. Then, open the console to configure the Ip address of the router.

IP Configuration Commands:

i. conf t
ii. interface fastEthernet 0/0
iii. ip add 192.168.0.1 255.255.255.0
iv. no shut

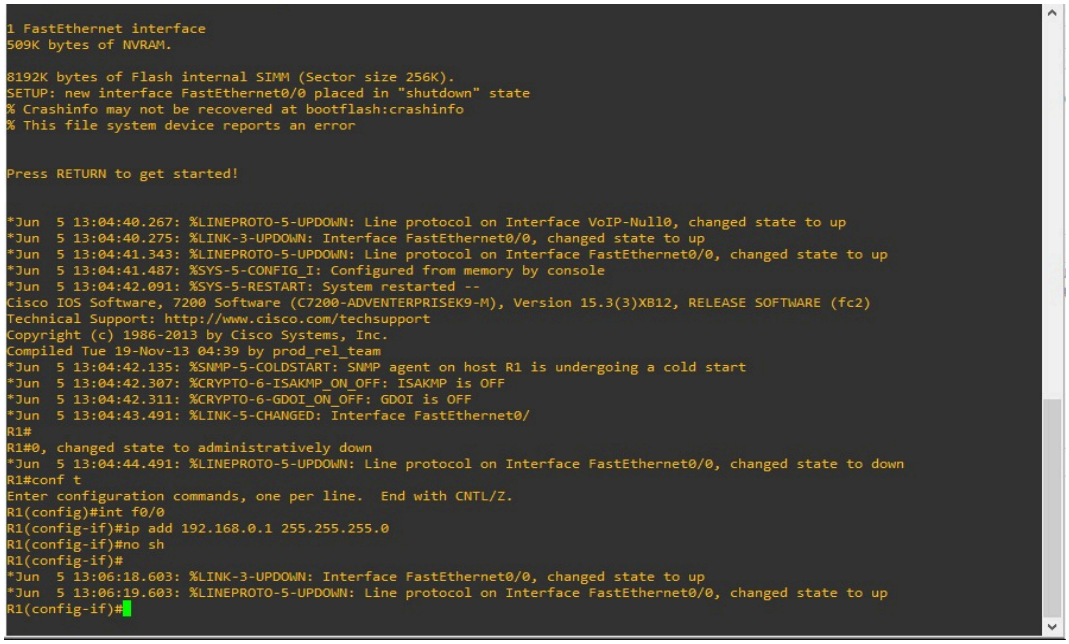
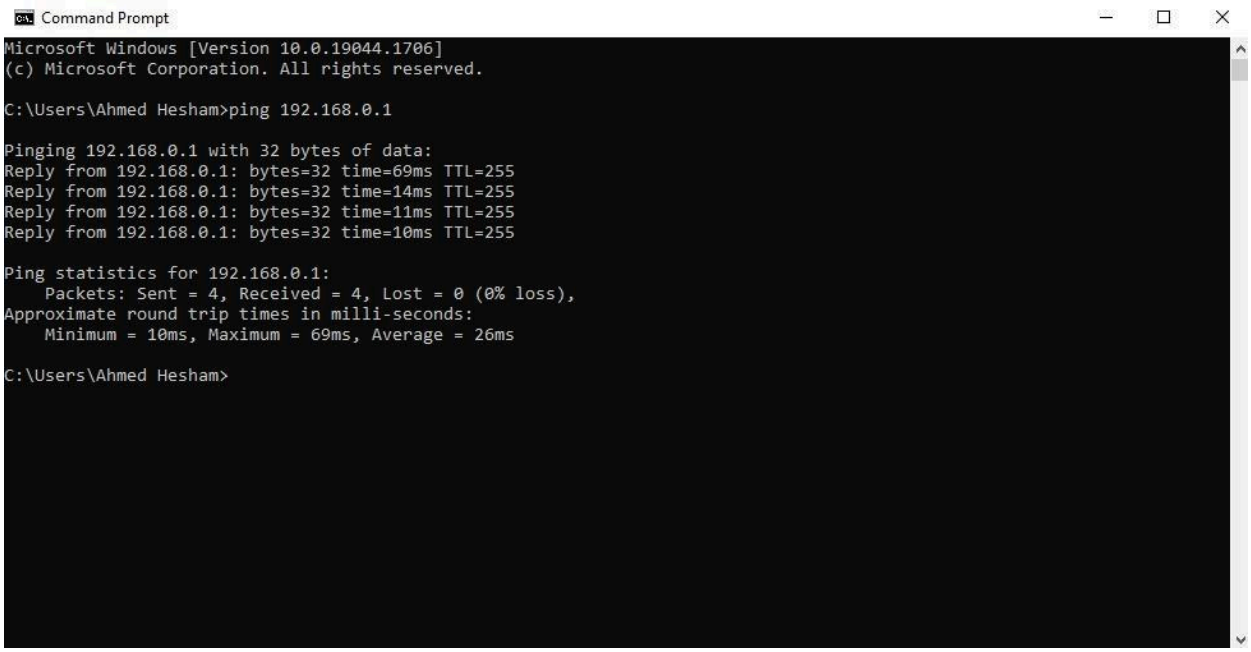


Figure 23 Router Configuration

Now we can connect the cloud (the physical PC) to the router and ping successfully.



Chapter Four: Implementation

In this chapter, we will discuss how our project is implemented using python programming code to build the ChainSpoT application.

4.1 Tutorial

The initial interface of the program is the tutorial.

At first, our code of the program must be run as an administrator by the following function:

```
def run_as_admin(self,argv=None, debug=False):
    shell32 = ctypes.windll.shell32
    if argv is None and shell32.IsUserAnAdmin():
        return True

    if argv is None:
        argv = sys.argv
    if hasattr(sys, '_MEIPASS'):
        # Support pyinstaller wrapped program.
        arguments = map(str, argv[1:])
    else:
        arguments = map(str, argv)
    argument_line = u' '.join(arguments)
    executable = str(sys.executable)
    if debug:
        print ('Command line: ', executable, argument_line)
    ret = shell32.ShellExecuteW(None, u"runas", executable, argument_line, None, 1)
    if int(ret) <= 32:
        return False
    return None
```

The function is set and called in the tutorial code form to be run once opening the program because managing Wi-Fi in the incoming forms, essentially needs windows CMD and PowerShell to be run as an administrator.

4.2 Master or client

After the tutorial, there are two options for the user to choose from them as shown in **Figure 24**.

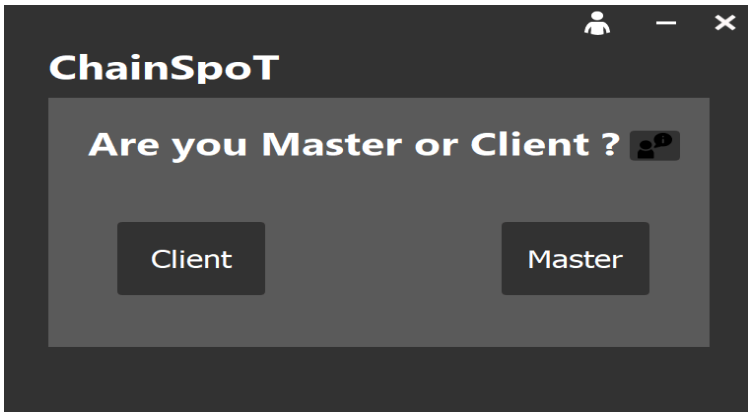


Figure 24 Maser or Client Form

4.3Master

If the user chooses to be a master to start a network, the following window will be shown. **(Figure 25).**

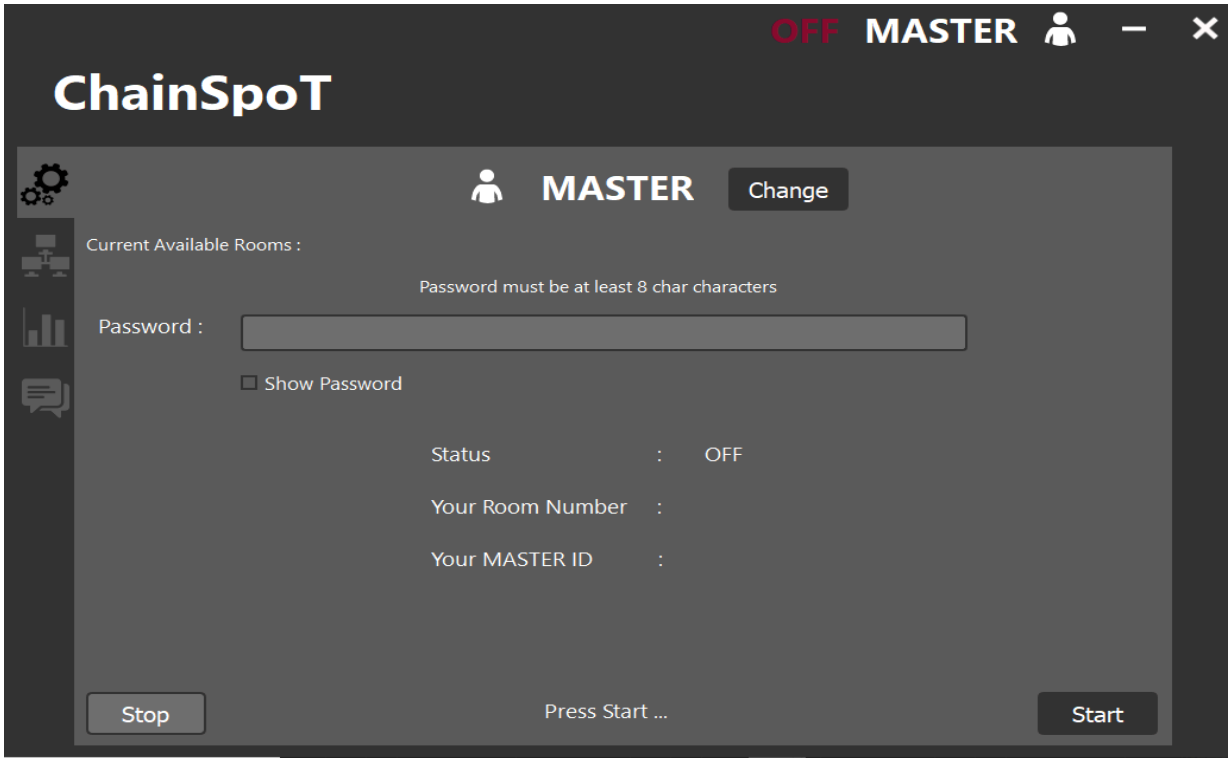


Figure 25 Master Form

The master is allowed to set a password to its network, and it is given a unique room number and master ID to begin the network.

- The power_wifi function shown in the image below is used to get the around available wi-fi interfaces with their SSID and power returned in a dictionary by extracting only SSIDs and corresponding powers and for ensuring getting all available interfaces at a time, the function is repeated twice.

```
def power_wifi(self,power = {},prev_check=0):
    if prev_check==0:
            power = {}
    cmd_output = subprocess.check_output("netsh wlan SHOW NETWORKS MODE=BSSID", shell=True)
    cmd_output = cmd_output.decode("ascii")
    cmd_output = cmd_output.replace("\r","")
    list_output = cmd_output.split("\n")
    list_output = list_output[4:]
    count = 0
    while count < len(list_output):
       if "SSID" in list_output[count] and list_output[count][9:].strip() != "" :
            try:
               if "Signal" in list_output[count+5]:
                    power[list_output[count][9:].strip()]=int(''.join(list(filter(str.isdigit, list_output[count+5]))))
            except:
               pass
       count += 1
    if prev_check == 0 :
            QtTest.QTest.qWait(5000)
            return self.power_wifi(power,1)
    else:
            return power
```

- The parse_wifi function shown in the image below is used to return only the power of the meaning interfaces in our project named in a certain way of three numbers, a list of spots IDs for all devices, a sorted list of the available rooms, and a sorted list of computer IDs using the output of the above power function.

```
def parse_wifi(self):
    power=self.power_wifi()
    networks= list(power.keys())
    r = re.compile("\d+\.\d+\.\d+")
    id_list = list(filter(r.match, networks))
    room_list = []
    comp_id=[]
    if id_list:
       for i in id_list:
          room_list.append(int(i.split(".")[0]))
          comp_id.append(int(i.split(".")[1]))
    return power , id_list, sorted(room_list) , sorted(comp_id)
```

- The interface_scan function (called by a created thread) shown in the image below is implemented to scan the Wi-Fi interface using the pywifi library.

Creates variable wifi from pywifi library.

Determines which interface to be scanned.

If any event interrupted the thread, it breaks and doesn't continue scanning.

Else scan the selected interface and sleep every five seconds.

This helps in showing continually updated information about the available rooms.

```
def interface_scan(self):
    wifi = pywifi.PyWiFi()
    iface = wifi.interfaces()[0]
    while True:
            if (self.stop_threads):
                    break
            print("i1")
            iface.scan()
            time.sleep(5)
```

- The next thread (room_timer_check function) shown in the image below is implemented using the function **parse_wifi()** to show the available rooms of networks.

```
def room_timer_check(self):
    while True:
        if (self.stop_threads):
            break
        power,id_list,room_list,comp_id =self.parse_wifi()
        self.room_show_signal.emit(str( list(dict.fromkeys(room_list)) ).replace(","," , ").replace("[","").replace("]",""))
        time.sleep(2)
```

- The run() function which is used to run windows PowerShell in our code is shown in image below.

```
def run(self,cmd):
    self.completed = subprocess.run(["powershell", "-Command", cmd], capture_output=True)
    return self.completed
```

- For the master to open the hotspot it uses the following function using some PowerShell functions and commands.

def start_hotspot(self,hotspot_name,password)

It must start the hotspot via the loopback interface which is called Ethernet 2 .

- During the operation of the network the master checks the hotspot continually by the next function to guarantee a stable connection with the other devices.

def check_hotspot(self,prev_check=0):

It just checks hotspot if it turns on or off.

4.4Chat application

Server-side:

In the function which starts the hotspot in master.py code we start two threads one for the server (master) and the other for the client as shown in image below.

```
self.t_server = Thread(target = self.server_run )
self.t_server.start()
self.t_client = Thread(target = self.client_run )
self.t_client.start()
```

- **def server_run():** this function (shown in the image below) initializes a list for the clients (who will connect to the server), an empty variable for the last receiving message which will store the message until it is completely sent to the target and finally a server socket, then calls create_listening_server() function.

```
def server_run(self):
    self.clients_list = []
    self.last_received_message = ""
    self.server_socket = None
    self.create_listening_server()
```

- **def create_listening_server():** this function (shown in the image below) creates a socket for the server giving it the loopback interface IP (192.168.0.2) of the master(server) device and port number (for ex: port=10319) as parameters using bind() method then make the socket

listens for incoming connections with listen() method and finally the function receive_messages_in_a_new_thread() will be called.

```
#listen for incoming connection
def create_listening_server(self):

    self.server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM) #create a socket using TCP port and ipv4
    local_ip = '192.168.0.2'
    local_port = 10319
    # this will allow you to immediately restart a TCP server
    self.server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    # this makes the server listen to requests coming from other computers on the network
    self.server_socket.bind((local_ip, local_port))
    print("Listening for incoming messages..")
    self.server_socket.listen(100) #listen for incomming connections / max 100 clients
    self.receive_messages_in_a_new_thread()
```

- **def receive_messages_in_a_new_thread():** this function (shown in the image below) contains a while loop which includes the whole code which is responsible for establishing a connection between the server and client by accept() method.

```
client = so, (ip, port) = self.server_socket.accept()
```

The second part of the while loop contains the part of code for Adding the client to the clients and list then starting a thread for this client to receive messages.

```
self.add_to_clients_list(client)
print('Connected to ', ip, ':', str(port))
t =Thread(target=self.receive_messages, args=(client,))
t.start()
```

- **def receive_messages:** this function (shown in the image below) takes the client object as an argument initializing a variable to contains this object socket then initializing an empty buffer variable which will receive the message using recv() method then decoding this message storing it in the last_received_message global variable and calling the broadcast_to_all_clients() giving it the client's socket object as an argument.

```
#fun to receive new msgs
def receive_messages(self, client):
    so, (ip, port) = client
    incoming_buffer=""
    while True:

        incoming_buffer = so.recv(1024) #initialize the buffer


        if not incoming_buffer :
            break

        self.last_received_message = incoming_buffer.decode('utf-8')
        self.broadcast_to_all_clients(so)  # send to all clients
    so.close()
```

- **def broadcast_to_all_clients():** this function (shown in the image below) takes the sender socket object and then sends the message to all clients except the sender client.

```
def broadcast_to_all_clients(self, senders_socket):
    for client in self.clients_list:
        socket, (ip, port) = client
        if socket is not senders_socket :
            socket.sendall(self.last_received_message.encode('utf-8'))
```

- **def add_to_clients_list():** this function (shown in the image below) is used for adding a client to the client_list.

```
def add_to_clients_list(self, client):
    if client not in self.clients_list:
        self.clients_list.append(client)
```

- **def delete_from_clients_list():** this function (shown in the image below) is used for deleting a client from the client list.

```
def delete_from_clients_list(self, client):
    for i in self.clients_list:
        if i == client:
            self.clients_list.remove(i)
```

Client side:

- **Def client_run():** this function (shown in the image below) just calls three functions initialize_socket() and listen_for_incomming_messages_in_a_thread() and on_join().

```
def client_run(self):
    self.initialize_socket()
    self.listen_for_incoming_messages_in_a_thread()
    self.on_join()
```

- **Def initialize_socket():** this function (shown in the image below) initializes a client socket connecting it to the loopback interface-id (192.168.0.2) of the master (server) with port 10319 by all done by connect() method of the sockets.

```
def initialize_socket(self):
    self.client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # initialazing socket with TCP and IPv4
    remote_ip = '192.168.0.2' # IP address
    remote_port = 10319 #TCP port
    self.client_socket.connect((remote_ip, remote_port)) #connect to the remote server
```

- **Def listen_for_incomming_messages_in_a_thread():** this function (shown in the image below) just creates a thread for receiving the incoming messages safe of the client from the server.

```
def listen_for_incoming_messages_in_a_thread(self):
    thread = Thread(target=self.receive_message_from_server, args=(self.client_socket,))
    thread.start()
```

- **Def receive_message_from_server():** this function (shown in the image below) receives the message using recv() method then checks if word 'parent' in the message that means that the json tree structure is in the message so it calls add_to_tree() function then continues another iteration of while loop of receiving and if the word 'joined' in the message it emits a signal to the GUI thread which shows a message "user has joined" in the chat plaintext.

```
#function to recieve msg
def receive_message_from_server(self, so):
    buffer=""
    while True:
        try:
            buffer = so.recv(1024)

        if not buffer:
            break
        message = buffer.decode('utf-8')
        if '{"parent": 'in message:
            message = json.loads(message)
            self.add_to_tree(message)
            continue

        if "joined" in message:
            user = message.split(":")[1]
            message = user + " has joined"

        self.chat_show_signal.emit(message)
    so.close()
```

- **Def add_to_tree():** if the hotspot IP of the node is already in the tree structure, we replace the newest one with the old one to keep the tree structure updated during the run time of the program and if not, then we add the new one in its position of the tree structure. The add_to_tree() function is shown in the image below.

```
def add_to_tree(self, leaf):
    for i in self.tree:
        if i["hotspot_name"] == leaf["hotspot_name"]:
            self.tree.remove(i)
    self.tree.append(leaf)
```

4.5Tree Structure

This is one of the trickiest parts of the ChainSpoT code which controls the visualization of the network tree structure during the runtime of the application.

- on_join function (shown in the image below):

```
def on_join(self):
    #put hotspot name
    self.client_socket.send(("joined:" + self.master_ID).encode('utf-8'))
    self.tree_thread_stop = 0
    tree_thread = Thread(target=self.tree_send, args=()) # Create a thread for the send and receive in same time
    tree_thread.start()
```

Each time a client joins the server room (automatically after connecting to the network) a thread will be created for this client calling his tree_send function.

- tree_send function:

We can describe this function in three parts:

I. Getting the average of pings to the loopback interface Ip of the Master (Server).

```
def tree_send(self):
    while True:
            if self.tree_thread_stop == 1:
                    break
            time_now = int(time.time())
            for i in self.tree:
                if time_now - int(i["time"]) > 100:
                    self.tree.remove(i)
            cmd_output = subprocess.check_output("ping 192.168.0.2", shell=True).decode().split("\n")[-2]
            if "Average" in cmd_output:
                    cmd_output = cmd_output.replace("\r","")
                    cmd_output = cmd_output.replace(" ","")
                    cmd_output = cmd_output.split("=")
                    ping=  cmd_output[-1]
            else:
                    ping=""
```

II. Creating a dictionary that will be transformed into JSON structure for each client containing parent, wi-fi IP, hotspot name, ping, and time. This JSON will be sent to the server which will send it back to all clients in the network (including the client in the server device).

```
myleaf ={
"parent": "",
"wifi_ip": "",
"hotspot_name": self.master_ID,
"ping": ping,
"time":str(int(time.time()))
}
```

III. Transforming the dictionary to Json structure sending it on client socket and then calling the add_to_tree function which is described before in the clients' side functions section.

```
data = json.dumps(myleaf)
self.client_socket.sendall(bytes(data,encoding="utf-8"))
self.add_to_tree(myleaf)
```

IV. Drawing the tree using Graphviz library which made in three steps:

- Instantiating step (shown in the image below): a double Dot objects (dot-tree and dot_statistics) which are graph objects will be created from Digraph class which have methods to add nodes and edges to the graph. The for loop is used for creating nodes for each graph giving each node an index (for connecting edges between nodes) and its hotspot IP (for tree graph) or hotspot IP, node IP, Wi-Fi IP and ping (for statistics graph) all that for visualizing the node associated with all its dependencies.

```
# instantiating object
dot_tree = Digraph(comment='A Round Graph')
dot_statistics = Digraph(comment='A Round Graph')
for key in self.tree:
    dot_tree.node(str(self.tree.index(key) ),key["hotspot_name"])
    dot_statistics.node(str(self.tree.index(key) ),"Node name: "+key["hotspot_name"]+"\nNode IP: "+key["wifi_ip"]+"\nPing: "+key["ping"])
```

- Adding edges (shown in the image below): an empty array is created, then a double for loop for checking each device and its parent and concatenating between their indices in each index of this array for ex: a = ['12', '13', '25', '26', '34', '47', '46'] (will make edges between 1 and 2, 1 and 3, 2 and 5, etc.) that happens using dot. edges and passing the array to it (done for dot_tree

graph and dot_statistics graph).

```
# Adding edges
a = []
for key in self.tree:
    for i in self.tree:
        if key["parent"] == i["hotspot_name"]:
            a.append(f'{self.tree.index(i)}{self.tree.index(key)}')
print(self.tree)
self.tree = []
dot_tree.edges(a)
dot_statistics.edges(a)
```

- Saving images (shown in the image below): we set the extension of Png to the graph objects and then render the image of each graph. mov_bin() and delete_bin() are functions used to copy Graphviz files to the working directory in the processing time of rendering the image and then delete them after the image is rendered.

```
# saving images
dot_tree.format = 'png'
dot_statistics.format = 'png'
allfiles = self.mov_bin()
dot_statistics.render("statistics", view = False)
dot_tree.render("tree", view = False)
self.tree_show_signal.emit()
self.delete_bin(allfiles)
time.sleep(50)
```

4.6 Client

If the user chooses to be a client to join a room, the following window will be shown (**Figure 26**)

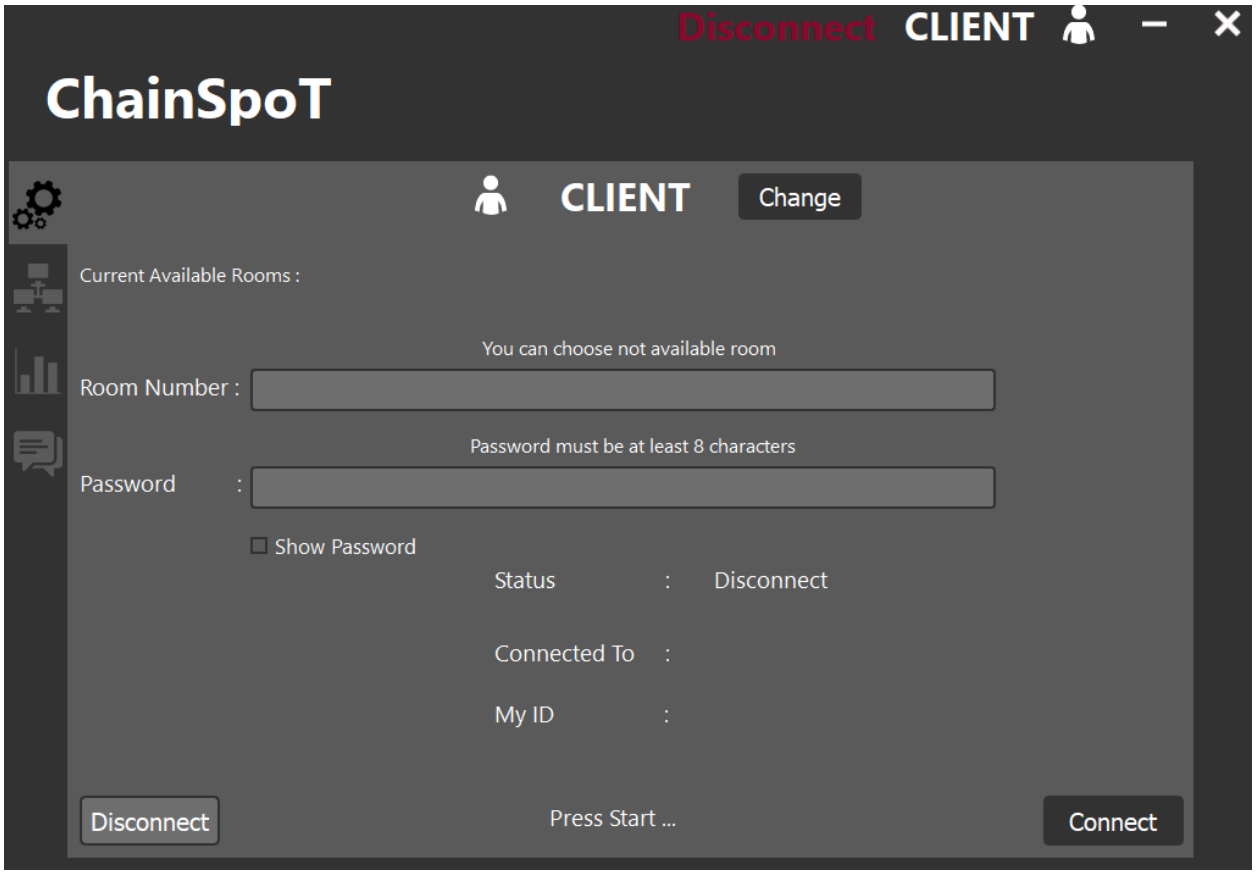


Figure 26 Client Form

Here the client is allowed to enter the room number which wants to join with its password.

- Check function:

After connecting to either master or nodes break down can happen the hotspot may go off or the Wi-Fi may turn off.

The check function shown in the image below is always running while the application is turned on to check if the hotspot and Wi-Fi are working. If the hotspot is off, it reopens. If the Wi-Fi is down due to the breakdown of connected node the app reconnects the device to the same node if it is back again or to the next best node available.

```
def check(self):                                                                                    ⚠ 16  ⚠ 376
    while True:
        power,id_list,room_list,comp_id =self.parse_wifi()
        if (self.stop_threads):
            break
        self.room_show_signal.emit(str( list(dict.fromkeys(room_list)) ).replace(",","," , ").replace("[","").replace("]",""))

        if (self.wifi_connect == "" and (self.state =="" or self.state =="disconnect")):
            self.lab_user_2.setStyleSheet("color:rgb(138, 11, 49);")
            self.lab_user_2.setText("Disconnect")
            self.label_22.setText("Disconnect")

        elif(self.wifi_connect != "" and self.state =="connect" and self.check_connect(self.wifi_connect)):
            self.lab_user_2.setStyleSheet("color:rgb(50,205,50);")
            self.lab_user_2.setText("Connect")
            self.label_22.setText("Connect")
            self.label_17.setText(self.wifi_connect)
            self.pushButton_3.show()
            self.pushButton_7.show()
            self.pushButton.setEnabled(False)
            self.pushButton_2.setEnabled(True)
            self.Master.setTabEnabled(2,True)
            self.Master.setTabEnabled(3,True)

            if self.check_hotspot() == True:
                self.label_15.setText("                              ChainSpoT Hotspot Still Up")

            else:
                if self.wifi_connect != "":
                    self.label_15.setText("                          Trying To Start ChainSpoT Hotspot ...")
                    power,id_list,room_list,comp_id_just_in_case =self.parse_wifi()
                    comp_id = comp_id + comp_id_just_in_case
                    while True:
                        my_comp_id = random.randint(1, 999)
                        if my_comp_id not in comp_id:
                            break
                    self.hotspot_name = self.room_ID +"."+str(my_comp_id)+"."+str(int(self.wifi_connect.split(".")[2])+1)

                    check=self.start_hotspot(self.hotspot_name,self.password)


                    if check and self.check_hotspot() == True:
                        self.label_19.setText(self.hotspot_name)
                        self.label_15.setText("                        ChainSpoT Hotspot Start Successfully")
                    else:
                        self.label_15.setText("                         Failed To Start ChainSpoT Hotspot")

        elif(self.wifi_connect != "" and self.state =="connect"):
            self.lab_user_2.setStyleSheet("color:rgb(138, 11, 49);")
            self.lab_user_2.setText("Disconnect")
            self.label_22.setText("Disconnect")
            self.wifi_connect =""
            self.forward_signal.emit()
            self.stop_threads= True
            break

        time.sleep(5)
```

4.7Client connect

After the client chose the room, the next window will be shown to determine the best wi-fi to connect based on certain programmed criteria as shown in the **Figure 27**.

Figure 27 Client Connect Form

4.7.1 Sorting function

As mentioned before in 2.3 the choosing of the best node to connect to have three priorities taken from its hotspot name.

I. Hub number
II. Power of hotspot signal

III. ID number

The best_wifi_list function shown in the image below shows the code for sorting the hotspot networks. They are sorted from the best to the worst and the device choose the best.

```
def best_wifi_list(self,power,id_list):
    sorted_ip=[]
    s = re.compile(str(self.room_ID)+".\d+\.\d+")
    sorted_ip = list(filter(s.match, id_list))
    sorted_ip=sorted(sorted_ip,key=lambda t:(int(t.split('.')[2]),-power[t],int(t.split('.')[1])))
    return sorted_ip
```

4.7.2 Connecting to the best node

After sorting the nodes in the Sorting function, the best node to connect to is the master or the first node in the list. You are connected to the shortest path to the whole tree after the node itself. However, the node signal can be weak or there are 8 devices connected to it. If the 8 devices are connected to the node the new device cannot connect to it the problem is there is no way for unconnected device to know the number of connected devices of the node hotspot, and if it tried to connect to the device would not give any errors or stop; it will continue to try to connect. So, there are limitation to connect to the first.

- The first node in the list (or the Master if seen) its single is at least 50%
- If the time taken to connect it exceeded a certain duration (15 seconds) the device will stop trying to connect the master and connect to client with best choice.

If the first node or master limitations are not satisfied, the device will try to connect to other clients. As for the clients also have limitations.

- The power at least 60 %
- Exceeding certain duration of 15 seconds

If the client node does not satisfy the limitations, the device tries to connect to the next node and then it is given a unique id in the network. The image below shows the try_connect function.

```
def try_connect(self):
    while True:
            if (self.stop_threads):
                    break
            power,id_list,room_list,comp_id =self.parse_wifi()
            sorted_ip = self.best_wifi_list(power,id_list)
            self.chain_show_signal.emit(sorted_ip,power)
            for wifi_try_connect in sorted_ip:
                    if (self.stop_threads):
                            break
                    self.trying_show_signal.emit(sorted_ip.index(wifi_try_connect))
                    if sorted_ip.index(wifi_try_connect) == 0 and power[wifi_try_connect] >= 50:
                            status = self.connect_wifi(wifi_try_connect)
                            if status:
                                    self.wifi_connect=wifi_try_connect
                                    self.connected_show_signal.emit(sorted_ip.index(wifi_try_connect))
                                    break
                            else:
                                    self.failed_show_signal.emit(sorted_ip.index(wifi_try_connect))
                    elif power[wifi_try_connect] >= 60 :
                            status = self.connect_wifi(wifi_try_connect)
                            if status:
                                    self.wifi_connect=wifi_try_connect
                                    self.connected_show_signal.emit(sorted_ip.index(wifi_try_connect))
                                    break
                            else:
                                    self.failed_show_signal.emit(sorted_ip.index(wifi_try_connect))
                    else:
                            self.failed_show_signal.emit(sorted_ip.index(wifi_try_connect))
            QtTest.QTest.qWait(2000)
```

Chapter Five: Testing

The next figures will show the result of several forms of the application.

5.1 Tutorial

At first when you open the application the tutorial screen will appear, this screen contains basic information about the application as shown in Figure 28.

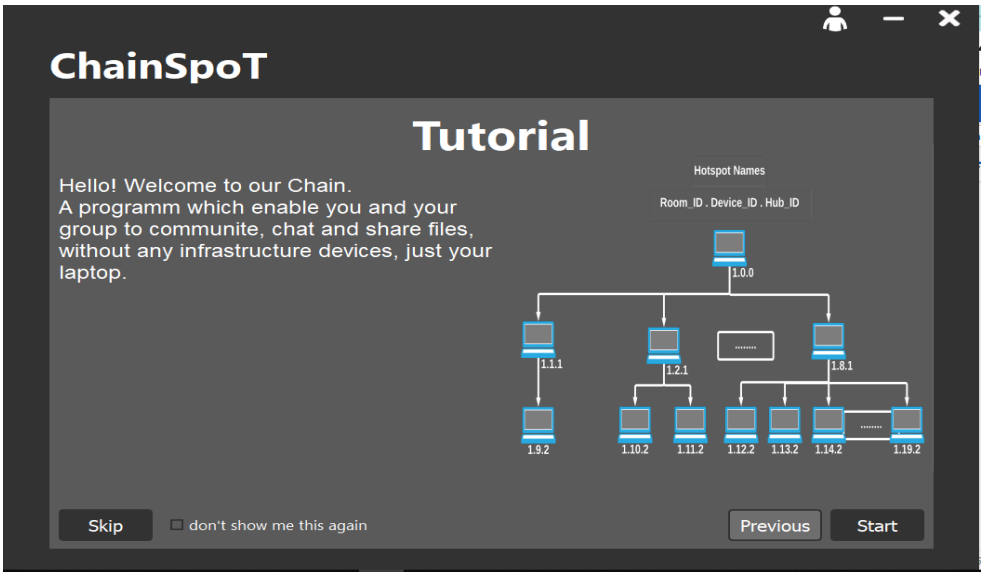
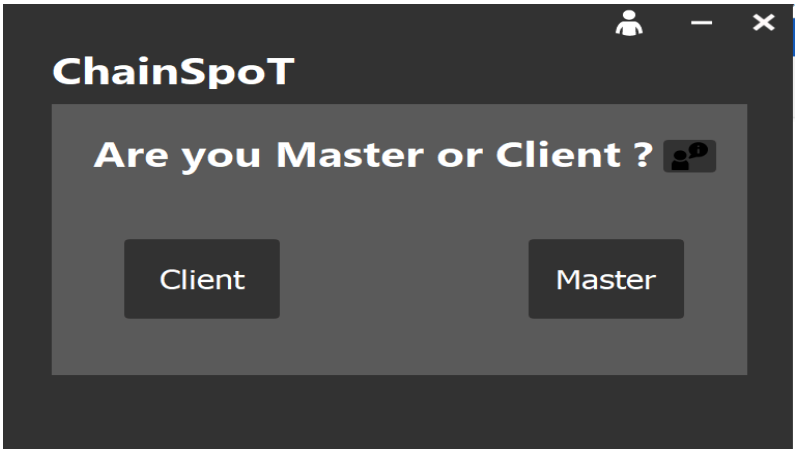


Figure 28 Tutorial Form

You can continue to read or skip if you are familiar with the application.

5.2 Master Or Client

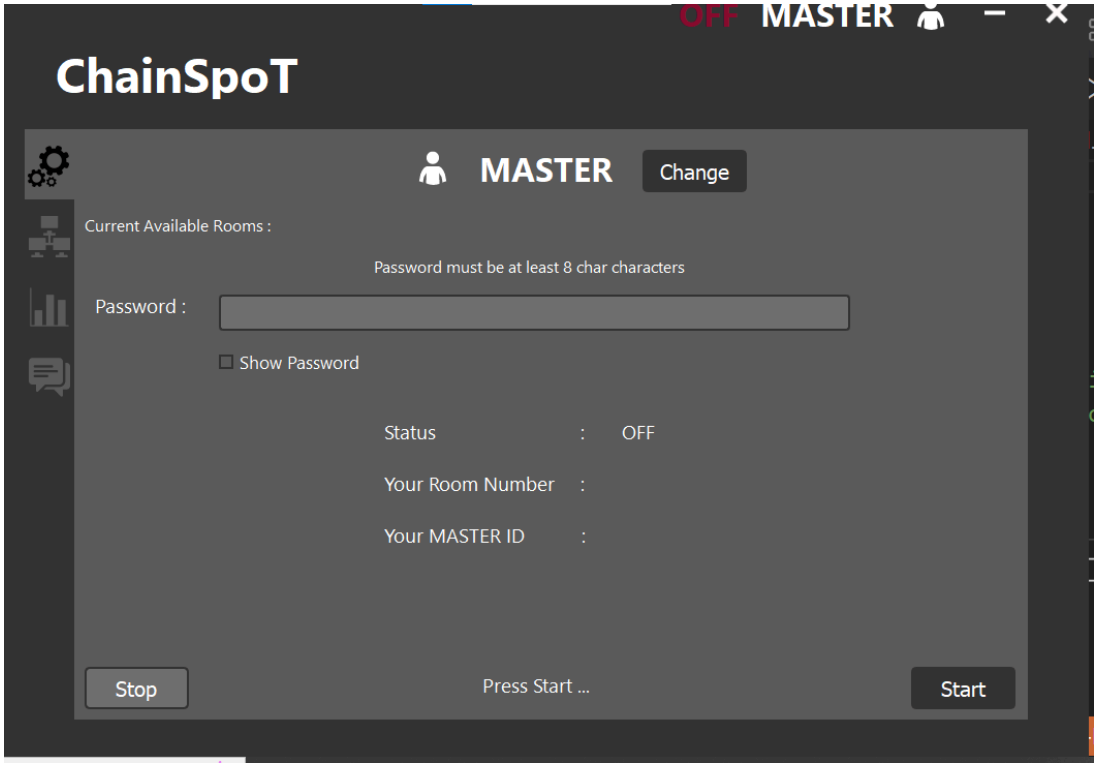
When you skip the tutorial, the master or client screen appears

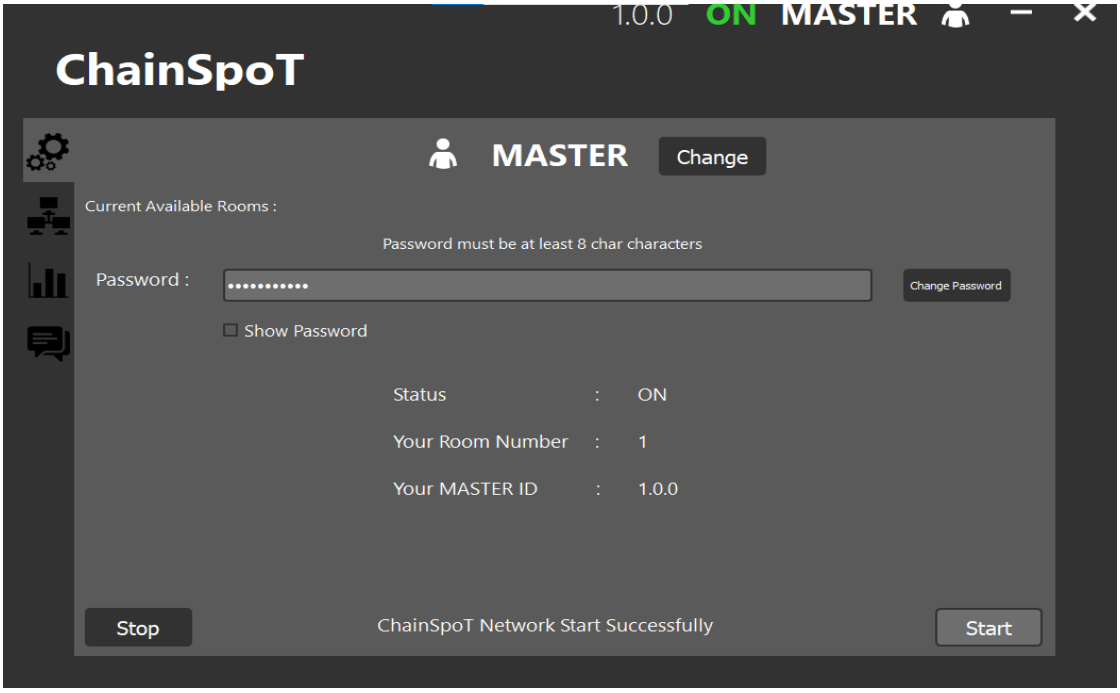


This screen makes you choose which option do you need the application for.

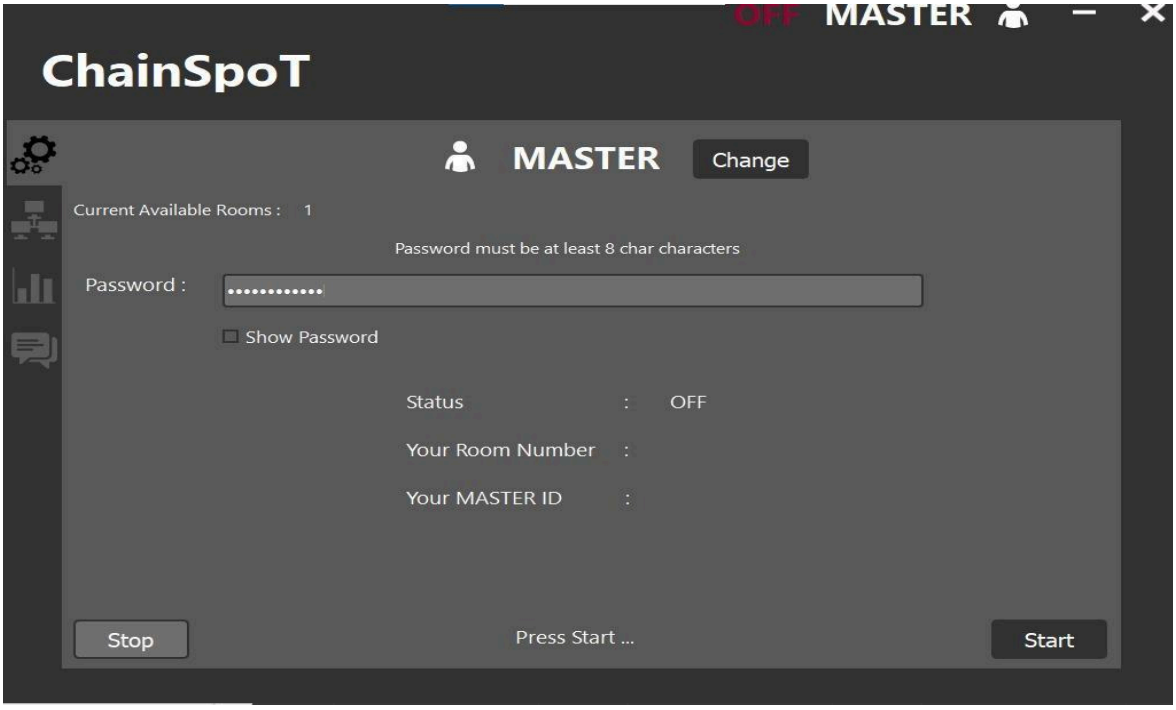
5.3 Master Test

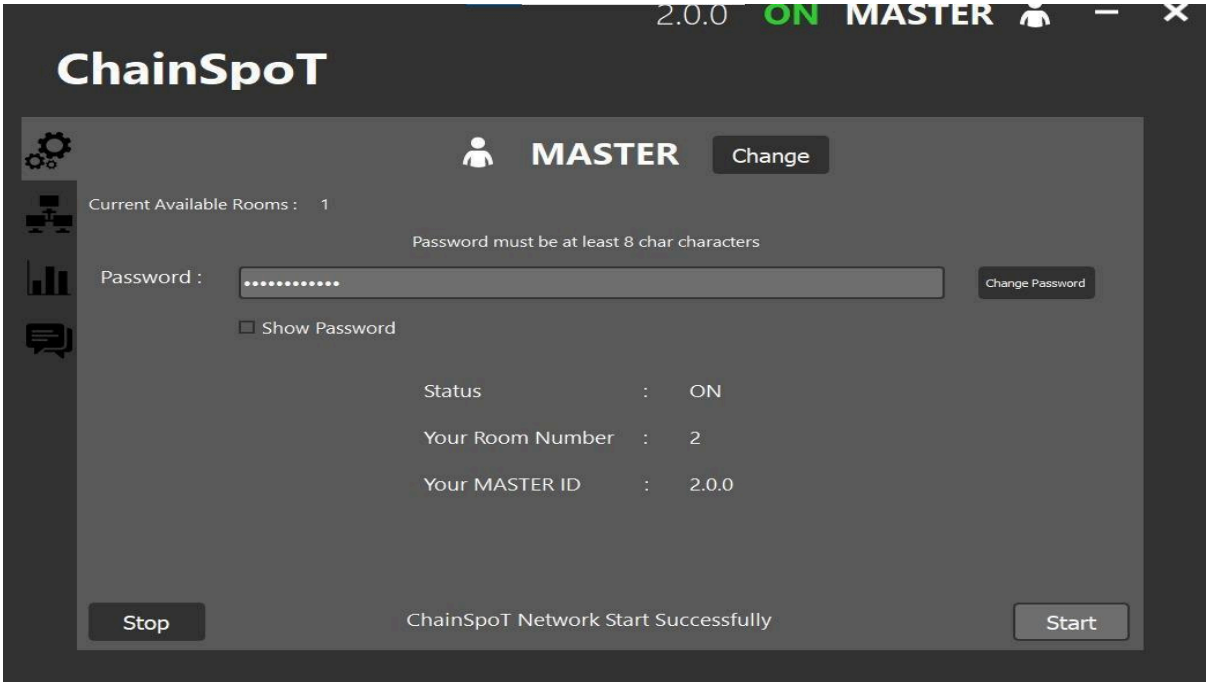
If you open the application and you are not surrounded by any other rooms, the application will give you the number "1" for you room ID and your hotspot will be "1.0.0".





If you open the application and you are surrounded by other rooms, the application will give you the next number of the last room number for you room ID. As you see here in the following example, The Master sees room 1. So, when you press start, it will open Room 2.



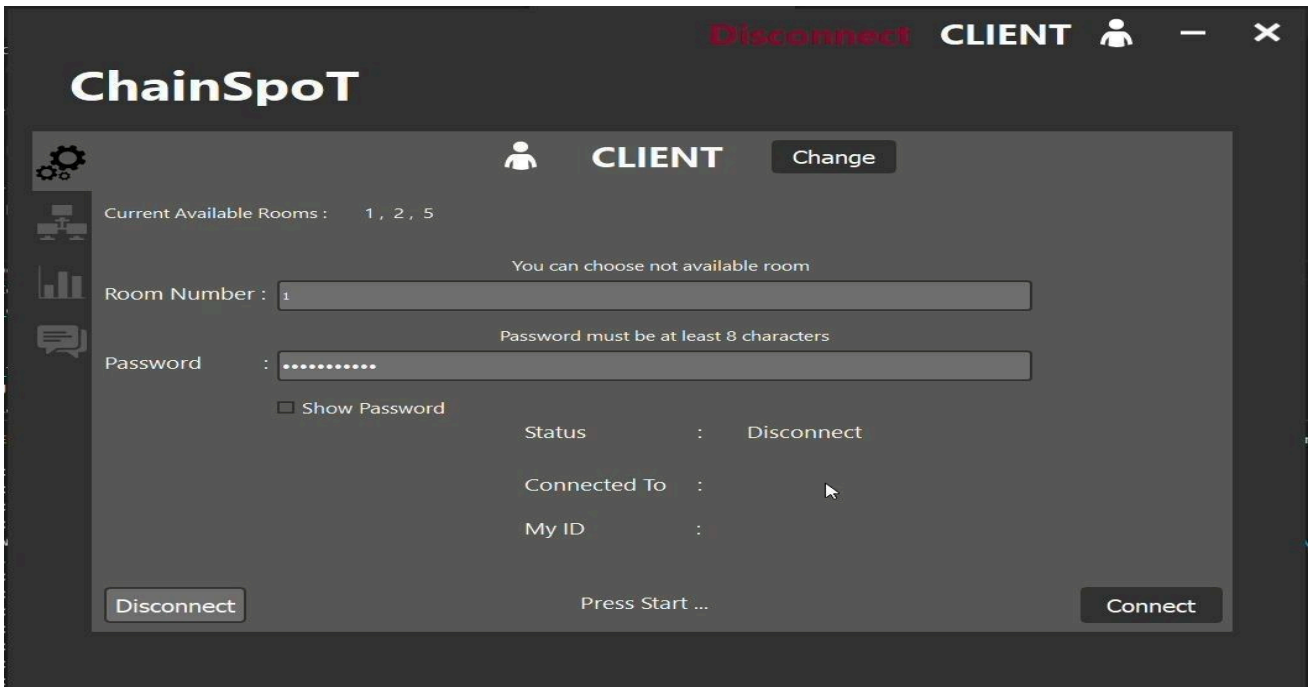


After the Master opens a hotspot with a unique name, it will set up TCP server at loopback interface (which all the clients can see it).

5.4 Client Test

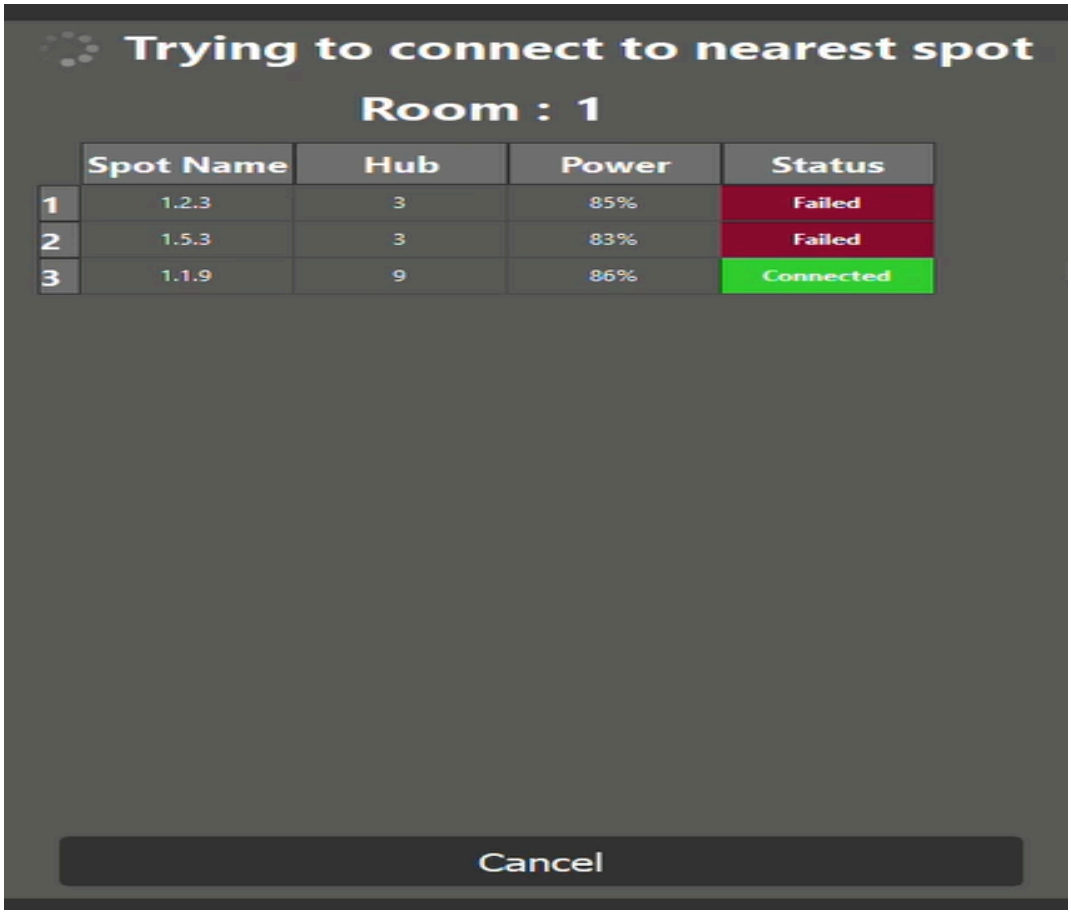
Now if someone is interested in joining your room, he would just need to enter the room number and the password. After entering the room number, the

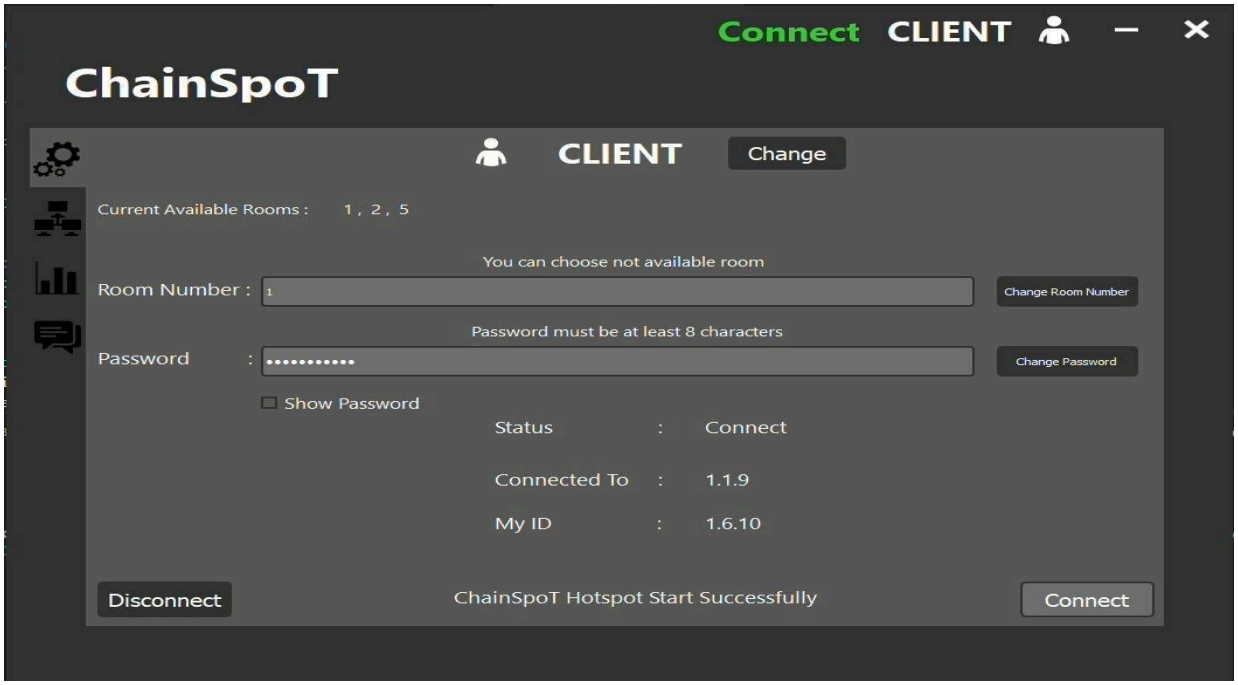
application will try to connect to the best node.



5.5 Client Connect Test

When client fills the room number and password and presses “Connect” it will lead to an open Client Connect Form Which lists all seen nodes sorted by Hub Count, power then Device ID. The application tries to connect them one by one. Until connection is established then the application goes to the Client form again with Connected state.


Trying to connect to nearest spot
Room : 1
Spot Name
Hub
Power
Status
1
1.2.3
3
85%
Failed
2
1.5.3
3
83%
Failed
3
1.1.9
9
86%
Connected
Cancel


Connect
CLIENT
ChainSpoT
CLIENT
Change
Current Available Rooms : 1 , 2 , 5
You can choose not available room
Room Number :
1
Change Room Number
Password must be at least 8 characters
Password :
Change Password
Show Password
Status : Connect
Connected To : 1.1.9
My ID : 1.6.10
Disconnect
ChainSpoT Hotspot Start Successfully
Connect

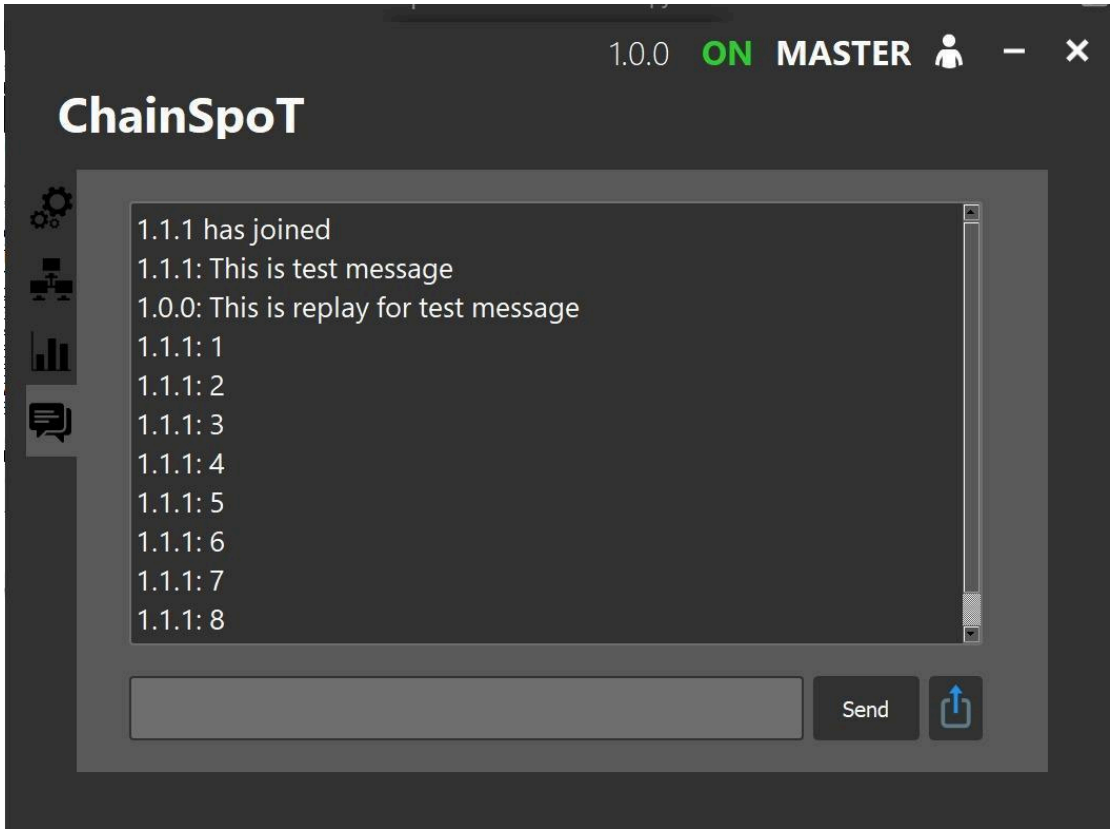
If anything interrupts the connection the client will return to the client connect window to connect to any hotspot.

5.6 The Service of The Application

After entering to a room as a master or client, the service of the application will start to work.

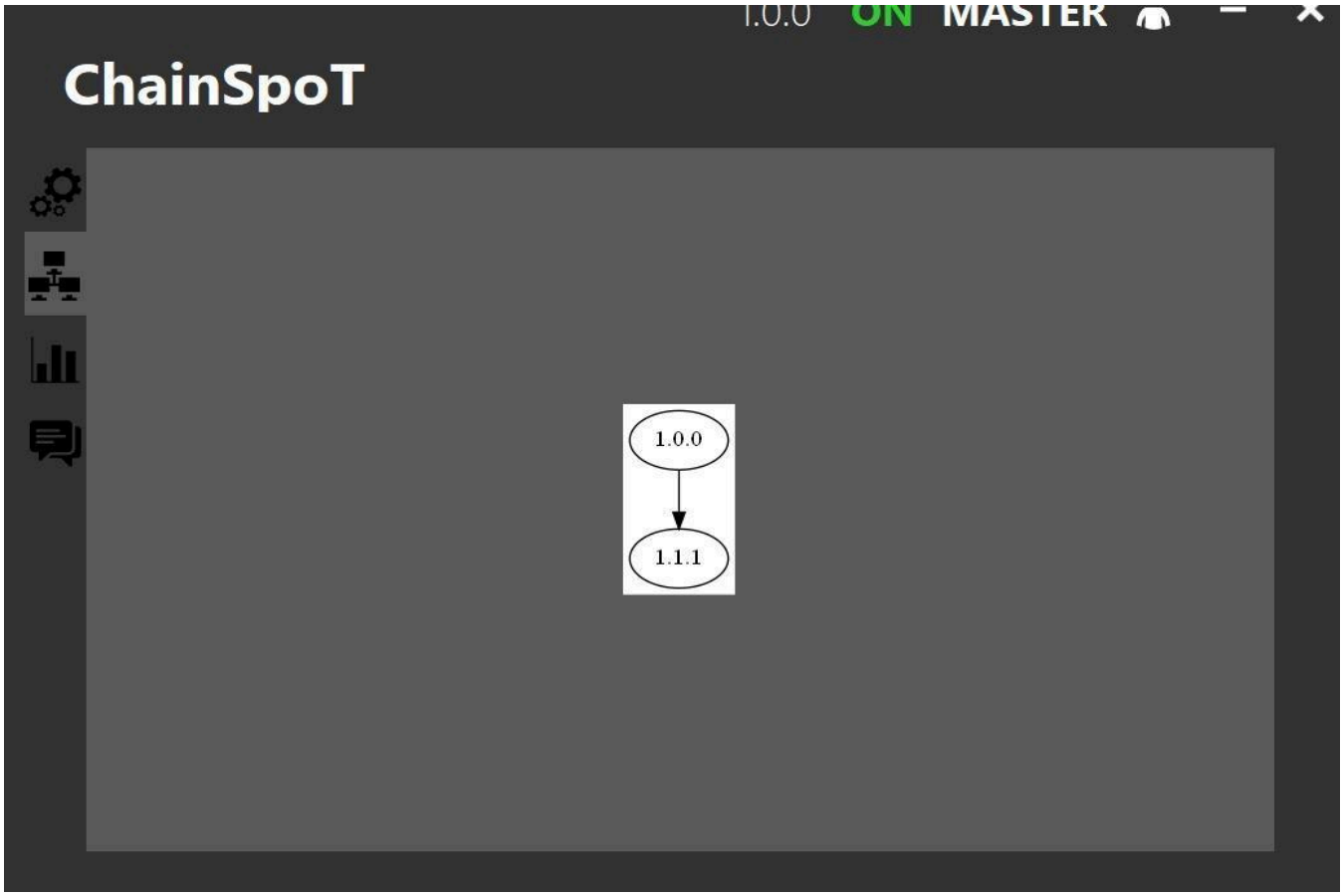
5.6.1 chat application

When a client connects to the ChainSpoT network. It tries to connect to the server (which located in the loopback interface of the Master). The client sends messages to the server. The server broadcasts all received messages to all clients excepting the one who sent these messages.



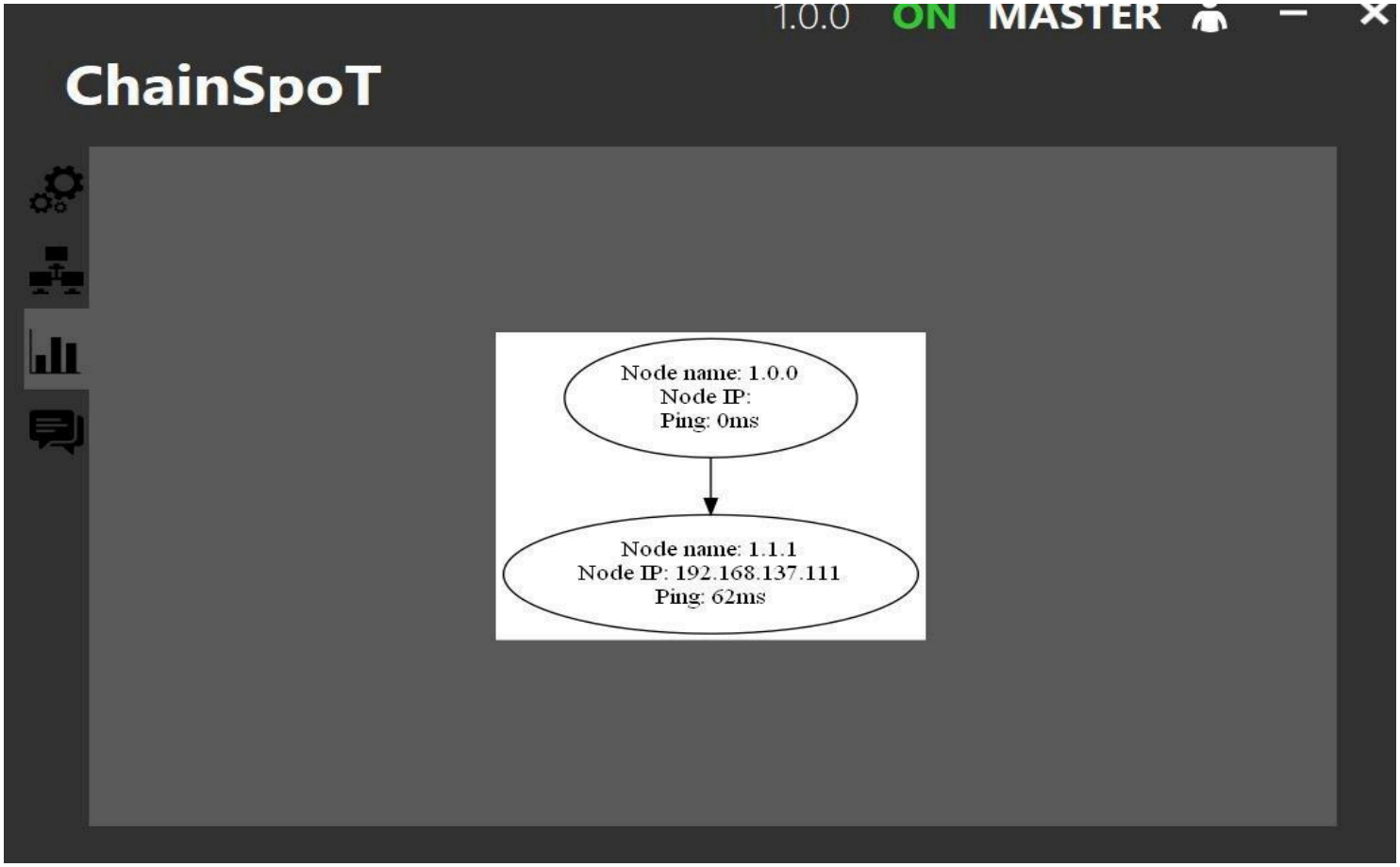
5.6.2 tree draw

As we showed before in 4.5 Tree Structure in tree_send() function, Each device sends its parent hotspot name, Wi-Fi IP, his hotspot name, average ping, and current time to the server. The server receives these information and broadcasts it to all clients. The client receives this information and build the tree with it to show the hierarchical structure of ChainSpoT network.



5.6.3 Statistics

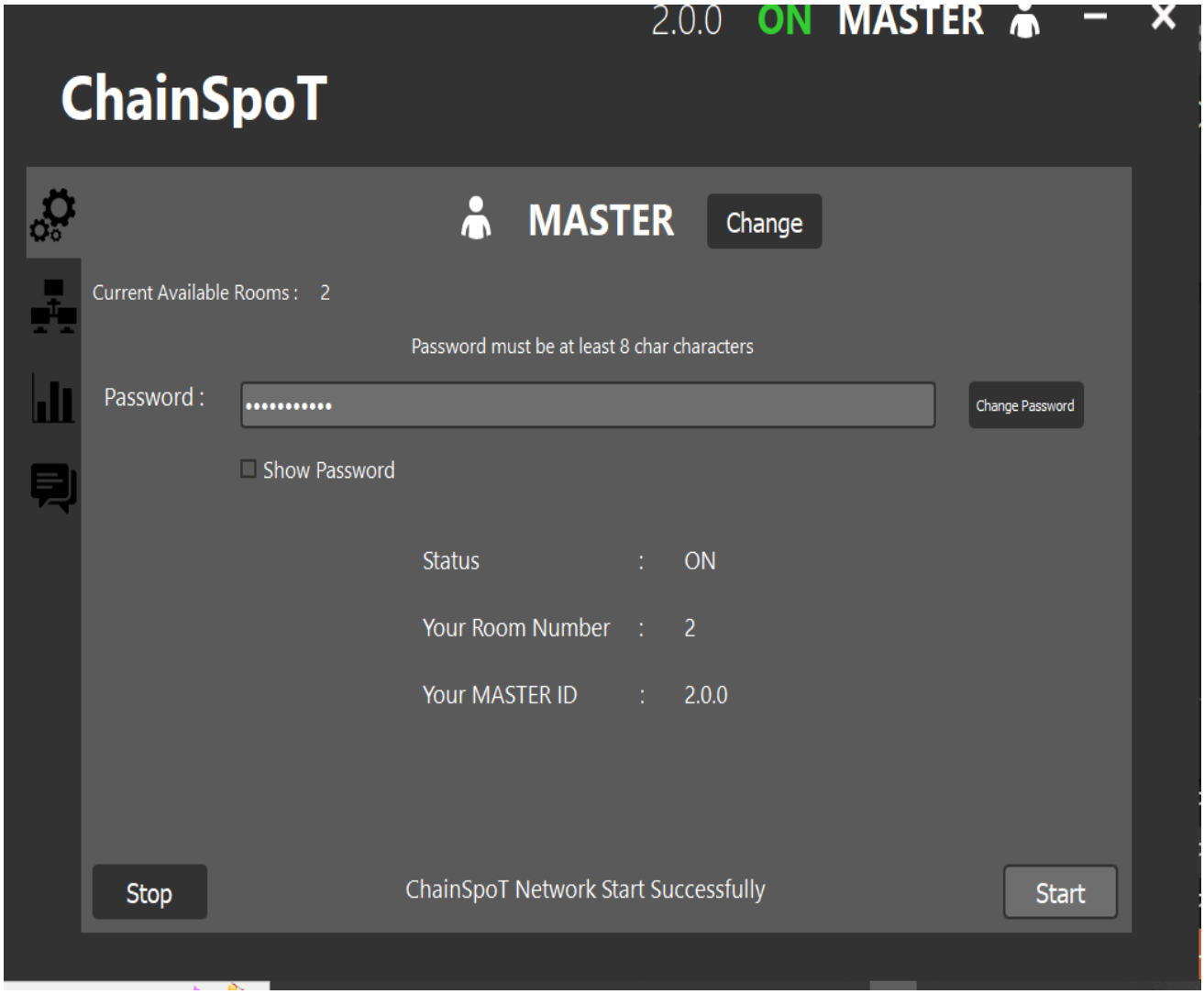
Statistics uses the tree_send() function too, but instead of show just hierarchical structure it's show all the information.



5.7 Test Case

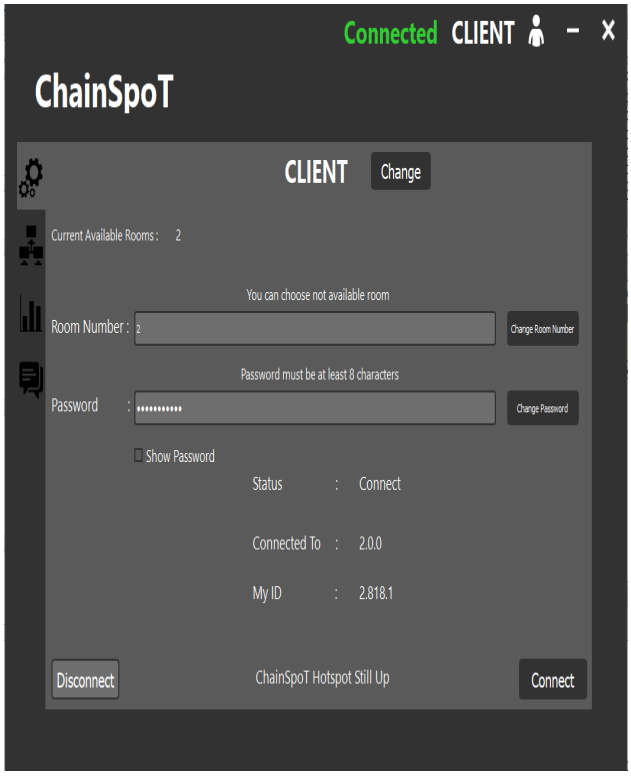
For this test case we have 5 laptops which aren't high-end laptops, it's just economic laptops with budget network cards which we use in our test to get more realistic results.

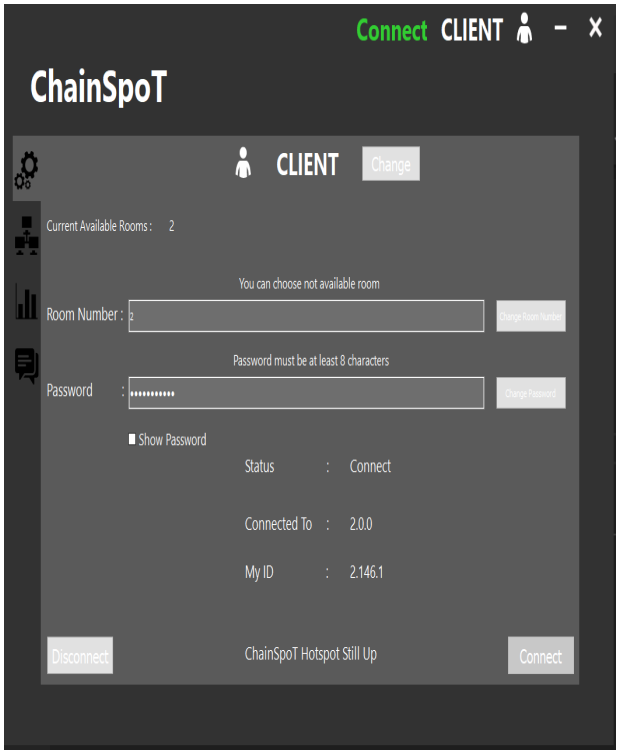
At first, the Master started the ChainSpoT network with ID "2.0.0" and waits the clients to join its room.

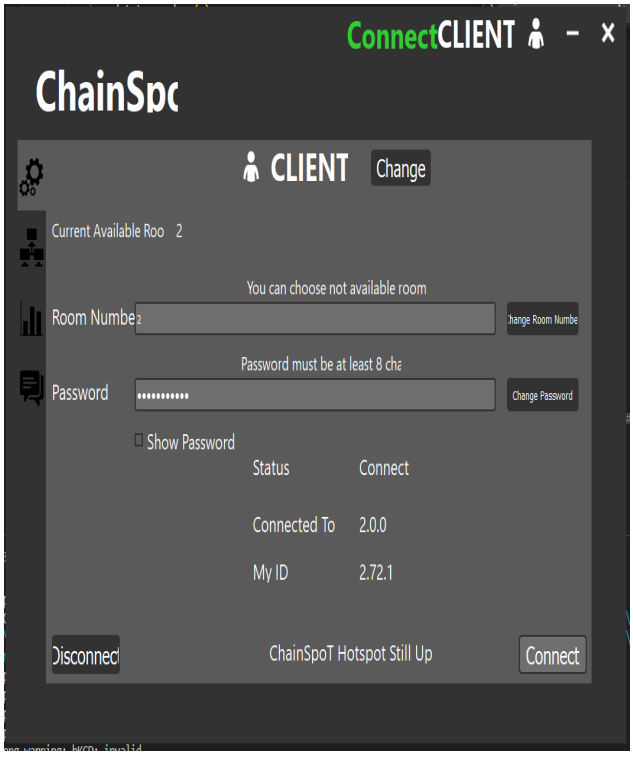


Then the clients join the master room one by one. All the 4 laptops connected directly with the Master hotspot because they all had seen the master with good power, and it always has less hub count. Each connected device could open a hotspot (with a unique node ID) for others to join the room.


ConnectCLIENT
ChainSpc
CLIENT
Change
Current Available Roo 2
You can choose not available room
Room Numbe
Password must be at least 8 cha
Password
Show Password
Status Connect
Connected To 2.0.0
My ID 2.940.1
Disconnec
ChainSpoT Hotspot Still Up
Connect


Connected CLIENT
ChainSpoT
CLIENT
Change
Current Available Rooms : 2
You can choose not available room
Room Number :
Change Room Number
Password must be at least 8 characters
Password :
Change Password
Show Password
Status : Connect
Connected To : 2.0.0
My ID : 2.818.1
Disconnect
ChainSpoT Hotspot Still Up
Connect


Connect CLIENT
ChainSpoT
CLIENT
Change
Current Available Rooms : 2
You can choose not available room
Room Number :
Password must be at least 8 characters
Password :
Show Password
Status : Connect
Connected To : 2.0.0
My ID : 2.146.1
Disconnect
ChainSpoT Hotspot Still Up
Connect


ConnectCLIENT
ChainSpc
CLIENT
Change
Current Available Roo 2
You can choose not available room
Room Numbe
Password must be at least 8 cha
Password
Change Password
Show Password
Status Connect
Connected To 2.0.0
My ID 2.72.1
Disconnec
ChainSpoT Hotspot Still Up
Connect

As soon as clients join the room, they send their information to the Master who broadcasts to all clients to draw the tree hierarchical structure and statistics as mentioned before.

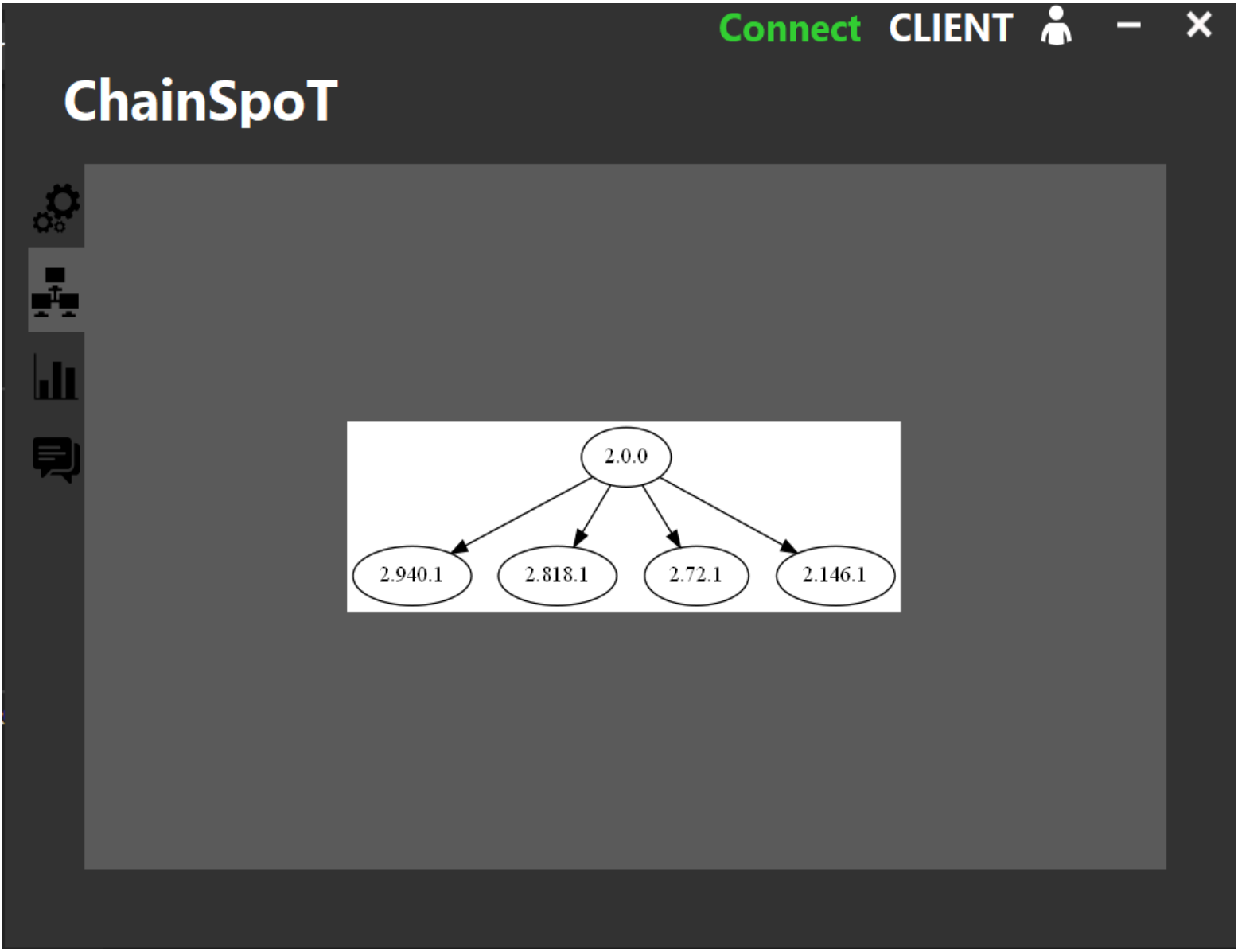


Figure 29 Tree Draw Example

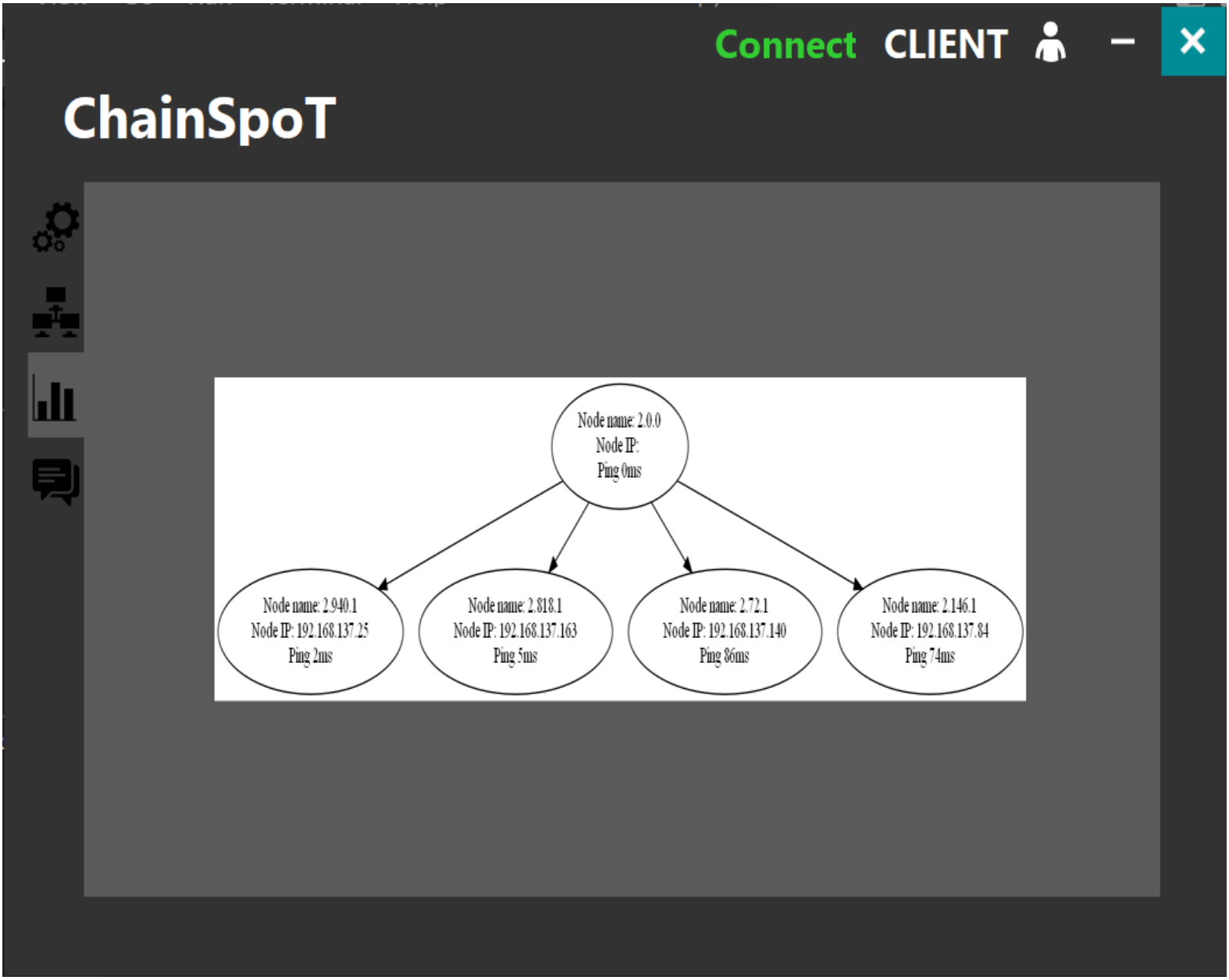
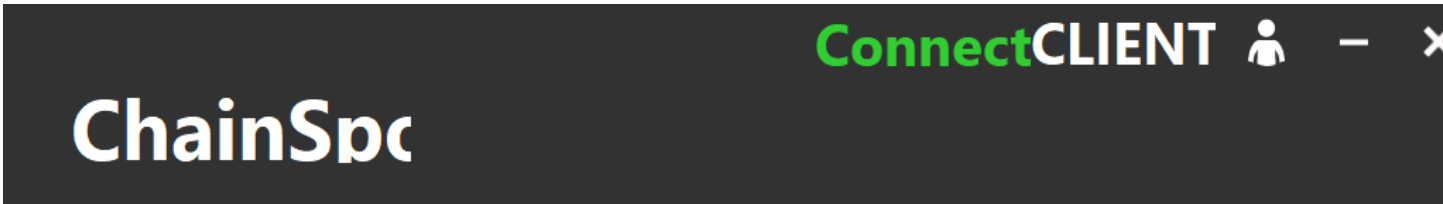


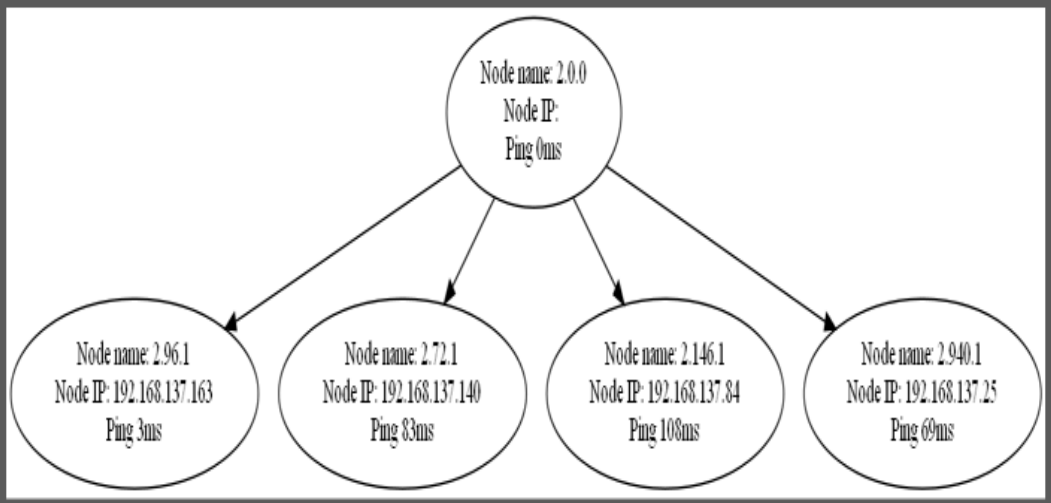
Figure 30 Statistics Draw Example

After The connection has dropped in device “2.818.1”. It reconnected again with a new ID “2.96.1” still having the same IP address “198.168.137.163”. The tree and statistics were updated as shown in the following figures.



ConnectCLIENT

ChainSpc



As shown in **Figure 31**, When new node joins the ChainSpoT network it sends a message “has joined” to the server which sends it back to All devices excepting the sending device. As you can see every node in this test case could chat with each other.

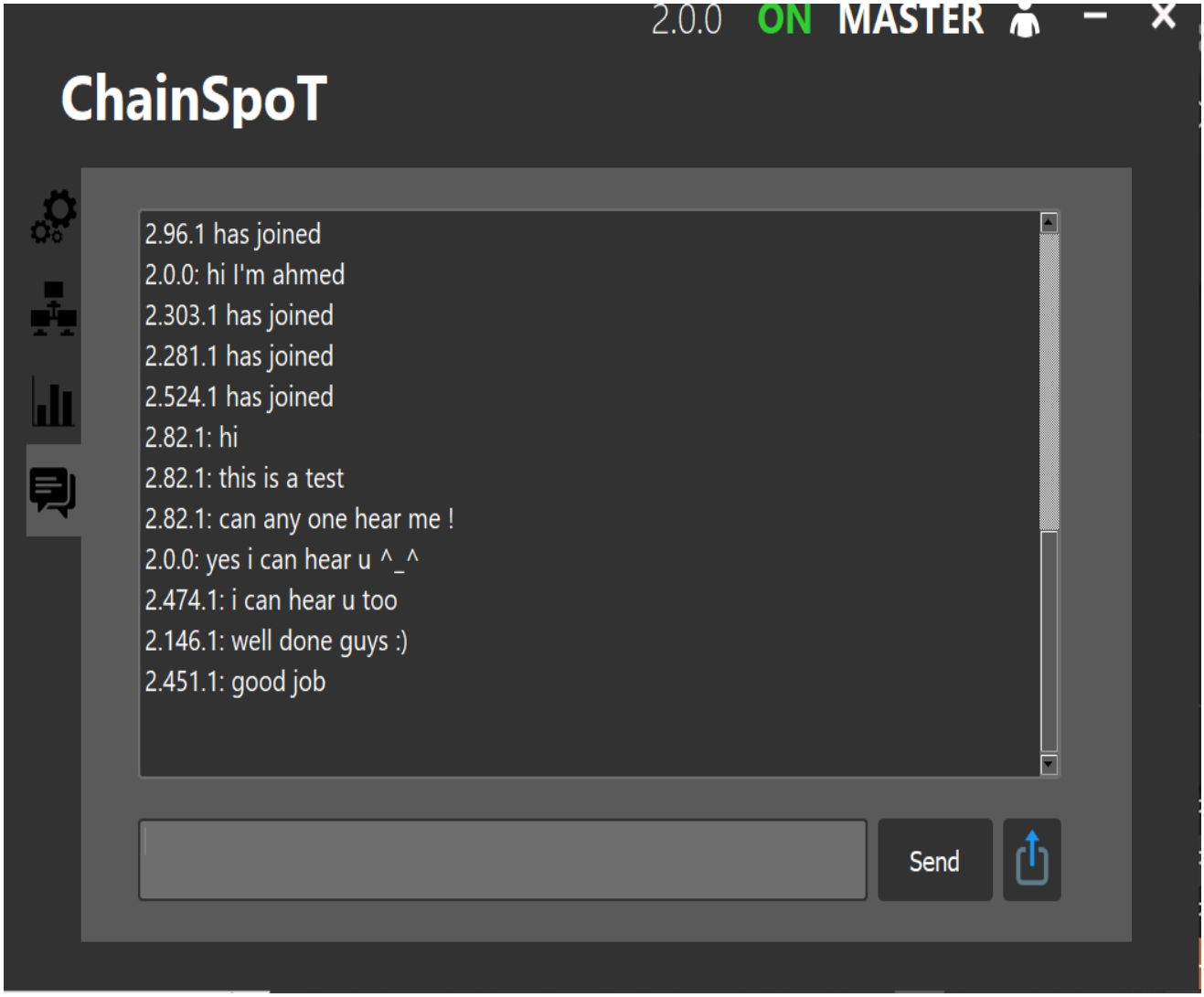


Figure 31 Chat Example

5.8 Result

- As we see in figure 29, The network tree shows us how the connection between devices takes place.
- As we see in figure 31, The connection of devices is achieved as shown in the chat and the messages took a very short time (a few milliseconds) to appear in the other devices which achieves a good communication.
- From the statistics we can determine the quality of connection by the time of the ping of each device on the master.

As we see in Figure 30, The first device with Hotspot ID 2.146.1 and IP 192.168.137.84, the distance was 2 meters from the Master and took 74 ms to ping him.

The second device with Hotspot ID 2.818.1 and IP 192.168.137.163, the distance was 1.5 meters from the Master and took 5 ms to ping him.

The third device whose Hotspot ID 2.940.1 and IP 192.168.137.25, the distance was only one meter from the Master and took 2 ms to ping him.

The fourth device with Hotspot ID 2.72.1 and IP 192.168.137.140, the distance was 2.5 meters from the Master and took 86 ms to ping him.

Different ping times depend on the distance between devices and the quality of Wi-Fi network card. The target of this test is to know our limitation and show how distance affects the quality of connection. In real life scenario the distance between devices will be more less than that and the quality of Wi-Fi network card will be better than that. It’s just the worst case to study and know our limits.

Chapter Six: Conclusion and Future Work

6.1 Conclusion

In our project, in the beginning, we aimed to create an application that establishes a network between devices without any intermediary devices (network infrastructure hardware). The network devices act as servers and clients at the same time. We could not make the devices act like servers and clients, but we could establish the network with a centralized device “Master” to connect the network. The centralized device has a loopback interface and acts like a server. When a client wants to send their data to another device, it directs it to the master and the master sends it back to all clients, except the sender. In the application, the devices are sorted in a tree shape. The tree is being displayed and upgraded with each new device that connects or disconnects the network. The devices connected to this network can chat with each other. We created a chatting application that all the network devices can group chat in. We were looking forward to creating a sharing files program as well as chatting, but we did not have time to implement it, unfortunately. Now you can start a free wide range of networks without being constrained by the number of clients and communicate with your friends or your colleagues. So, the number of your team members will not be a big issue as the ChainSpoT tree will grow until we hit a level where connection slow and cannot grow further. No need for user experience in the network. Our program is very user-friendly. Just by a few steps the net, work starts, and the client connects to the network. We also guide the users in the tutorial to teach them how they use our program. And for the superuser, we made a statistics section for the network statistics to help him in troubleshooting the network. The statistics section shows the user the node name, node Ip and ping time for the whole tree. The user can change the root ma at any time even after the connection has been done. Finally, we hope that we were able to help people, even in a small way, in communicating -with each other, and we hope that we can improve our application so that everyone can use it with the best performance and provide them with many benefits.

6.2 Future Work

In our project, we did not have time to improve the program in the best way. Unfortunately, there are still many problems that we cannot solve. Also, we can enhance our project with some features.
In this section, we will explain the problems we face in the journey of developing the project and some suggested solutions for these problems for future work.
If the master or any of the nodes changes its hostname the whole branch will lose connection to that node. the name of the node indicates the name of the network the leaves connect to. If the name is changed the hotspot will restart and the leaves will down. This leads to another critical problem.

If one of the nodes went down, every connected node would also go down Introducing the solution to this problem the program will immediately change the connected leaves of that node to another node.

Due to the design of the network the chat and shared files are broadcasted to the whole network: In developing the program, we are aiming to protect and encrypt the data. so, we can make Private chats between users that require securing data delivery.

Implementing a dynamic algorithm to make the tree of our network balanced. Thus, we can improve the performance and there is no load money device.
The old laptops and desktops will slow the network speed due to the old network cards: So, the old devices will be set up to be only leaves.
Finally, testing the performance for the whole network to achieve the best performance.
Log everything, the analysis of this logged data will improve our application.
Support the Linux, IOS and android platforms.

7 References:

[1] SOCKET PROGRAMMING HOWTO — PYTHON 3.8.13 DOCUMENTATION by ***Author*** *Gordon McMillan*

[2] - https://docs.python.org/3/library/socket.html.

[3] Data Communications and Networking by Behrouz A. Forouzan.DATA COMMUNICATIONS AND NETWORKING – BEHROUZ A. FOROUZAN.FOURTH EDITION[A4].PDF - GOOGLE DRIVE

[4] Computer Networking First-Step by Norman Laurence. An introductory guide to understanding wireless and cloud technology, basic communications services and network security for beginners, Laurence, Norman, eBook - Amazon.com Unlimited reading.

[5] https://sourcedaddy.com/networking/historical-conceptual.html sourcedaddy |at| gmail.com

[6] https://www.mobikin.com/mobile-phone/shareit-review.html

[7] https://obkio.com/blog/how-to-measure-network-performance-metrics Networks are changing. Monitoring solutions should too.Obkio is a legacy-free application designed to fill a gap within the industry.

[8]https://www.researchgate.net/publication/340050983_Efficient-Path_Selection_Scheme_Using_Optimized_Adhoc_on_Demand_Multipath_Routing_Protocol_For_Adhoc_Networks/figures?lo=1 Juan P. Macas, Recent Developments in Mobile Communications A Multidisciplinary Approach, ISBN 978-953-307-910-3, 2011 InTech

[9] https://www.lucidchart.com See and build the future with a powerful visual collaboration suite.

[10]NetworkOperatorTetheringManager.StopTetheringAsyncMethod (Windows.Networking.NetworkOperators) - Windows UWP applications | Microsoft Docs

[11] Enable Windows 10 built-in hotspot by cmd/batch/PowerShell - Stack Overflow

[12] https://ipcisco.com/lesson/routing-table/

[13] IP Address Lookup | Locate any public internet IP address (lookip.net)

[14] The Book of GNS3 by Jason C. Neumann. THE BOOK OF GNS3 PDF

[15] Python and PyQt: Building a GUI Desktop Calculator – Real Python

Appendix A. Routing Table

In networking, Routing is the process of deciding the path to the destination and directing the packets through this path. With the routing process, routers decide where to send the packets. In other words, routing is a very important part of networking. Because, based on some metrics, available paths are determined to a destination and the best path is chosen among these paths to be used in sending the packets.

Any router to know where and how to forward packets of data it receives to any specific destination uses its routing table, which is a mapping for the router, it contains different routing destinations' IP addresses with their subnet masks, and gateways addresses, and other information. it is built with static routing and with routing protocols like OSPF, BGP, EIGRP, etc.

It doesn't contain all the possible destinations but only the directly connected destinations. each interface of the router is considered a network. when the router receives a packet, it forwards it to the next network (hop) until it reaches the desired destination.

Routing table parameters

Windows allows us to get the IPv4 routing table by **route PRINT -4** command on the command prompt for example the following routing table (Figure 32).

```
===========================================================================
Active Routes:
Network Destination        Netmask          Gateway       Interface  Metric
          0.0.0.0          0.0.0.0      192.168.1.1      192.168.1.7     65
        127.0.0.0        255.0.0.0         On-link         127.0.0.1    331
        127.0.0.1  255.255.255.255         On-link         127.0.0.1    331
  127.255.255.255  255.255.255.255         On-link         127.0.0.1    331
      192.168.1.0    255.255.255.0         On-link      192.168.1.7    321
      192.168.1.7  255.255.255.255         On-link      192.168.1.7    321
    192.168.1.255  255.255.255.255         On-link      192.168.1.7    321
     192.168.56.0    255.255.255.0         On-link     192.168.56.1    281
     192.168.56.1  255.255.255.255         On-link     192.168.56.1    281
   192.168.56.255  255.255.255.255         On-link     192.168.56.1    281
        224.0.0.0        240.0.0.0         On-link         127.0.0.1    331
        224.0.0.0        240.0.0.0         On-link     192.168.56.1    281
        224.0.0.0        240.0.0.0         On-link      192.168.1.7    321
  255.255.255.255  255.255.255.255         On-link         127.0.0.1    331
  255.255.255.255  255.255.255.255         On-link     192.168.56.1    281
  255.255.255.255  255.255.255.255         On-link      192.168.1.7    321
===========================================================================
```

Figure 32 Routing Table Parameters

Each parameter of the routing table has a different meaning.

The network destination: it is the network that the router can send the packet to it. Example: 192.168.56.0

Netmask: is the Subnet ID of the destination address. Example: 255.255.255.0

The gateway is the entrance point to another network.

A default gateway is an address to which packets are sent if there is no specific gateway for a given destination listed in the routing table.

Interface: it is an outgoing interface from which the packet should go out to reach the destination network.

Metric: is a value that indicates the quality of the path and helps in choosing the best path according to these various parameters:

Hop count, Path reliability, Path speed, Load, Bandwidth, Latency and

Maximum transmission unit.

It determines the priority of the route. If there are several routes to the same destination the lowest metric

Appendix B. Loopback Interface

The loopback adapter is a dummy network card that is used as a testing tool for virtual network environments where network access is not available or when you want to isolate your testing network from your main network. You use the loopback adapter on your computer (which is the host machine) to enable your virtual machine (guest operating system) to communicate as if they are in a network without having to use an external network, such as your work network or home network.
We can make a loopback interface

1-from hdwwiz as shown in Figure 33.

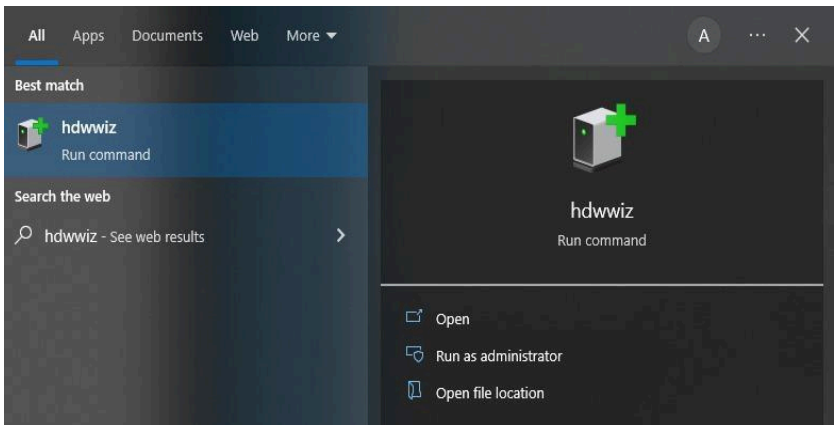


Figure 33 Install loopback interface 1

2-Then choose to install the hardware by manual selection as shown in Figure 34.

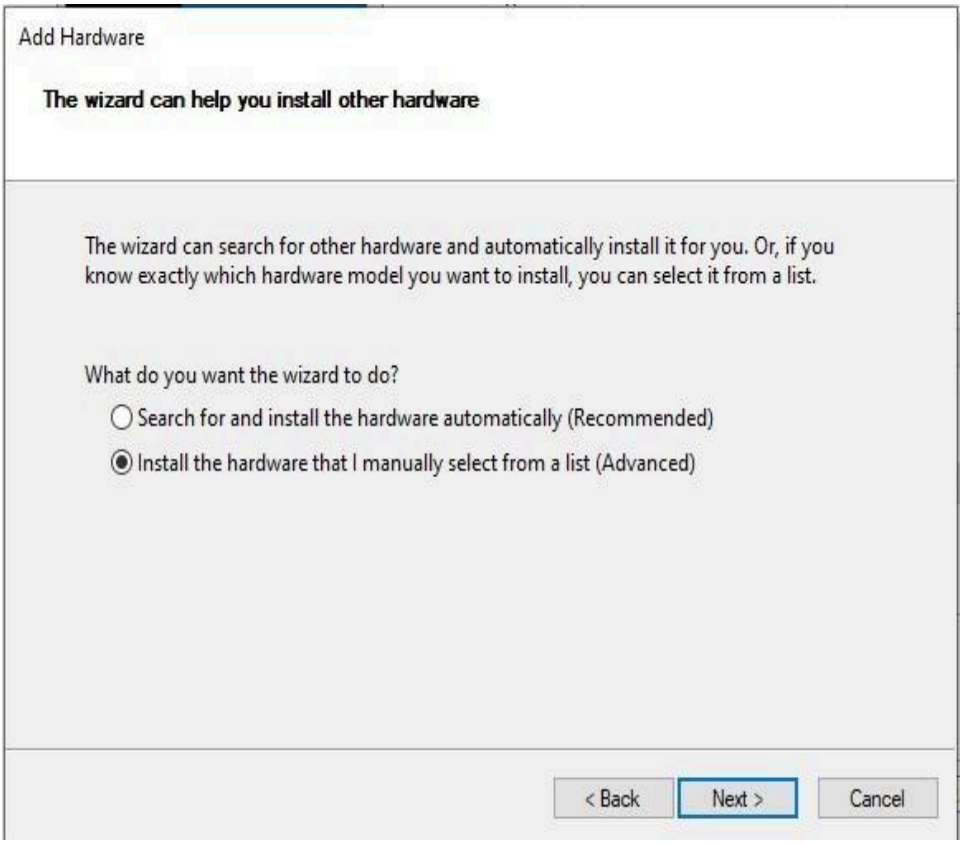


Figure 34 Install loopback interface 2

3-Then choose network adapters as shown in Figure 35.

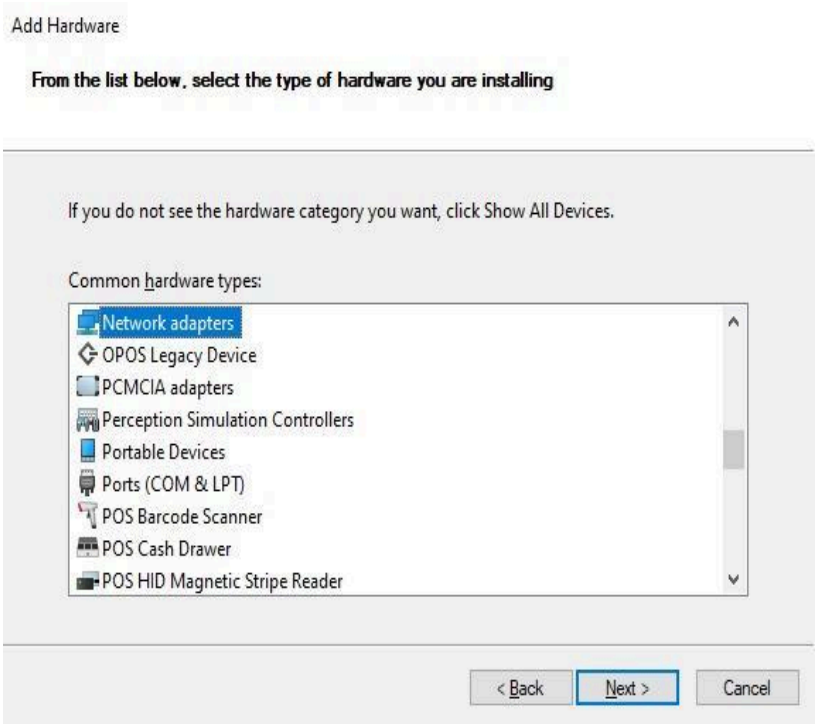


Figure 35 Install loopback interface 3

4-Then choose Microsoft and Microsoft KM loopback adapter as shown in Figure36.

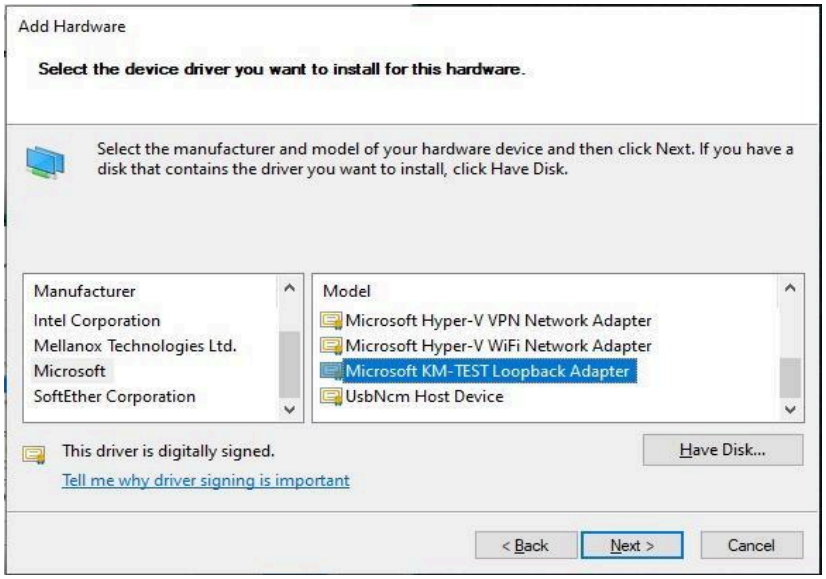


Figure 36 Install loopback interface 4

5-Then simply click next.

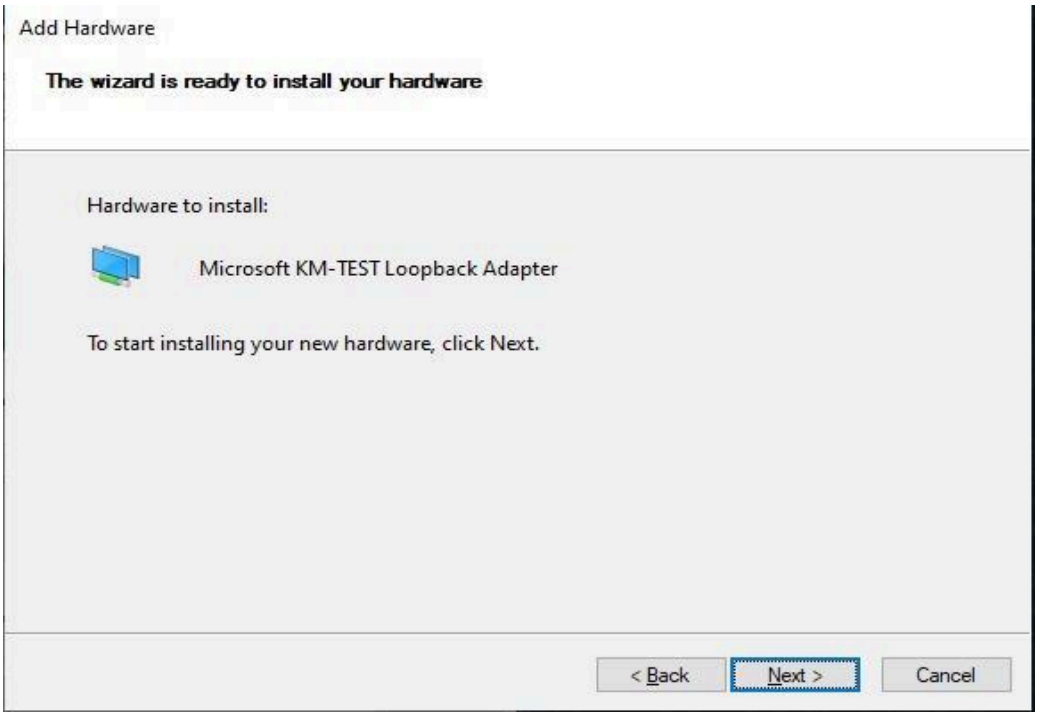


Figure 37 Install loopback interface 5

6-At last check the adapters on your device, you will see that the loopback interface is made as ethernet 2 as shown in **Figure 38**.

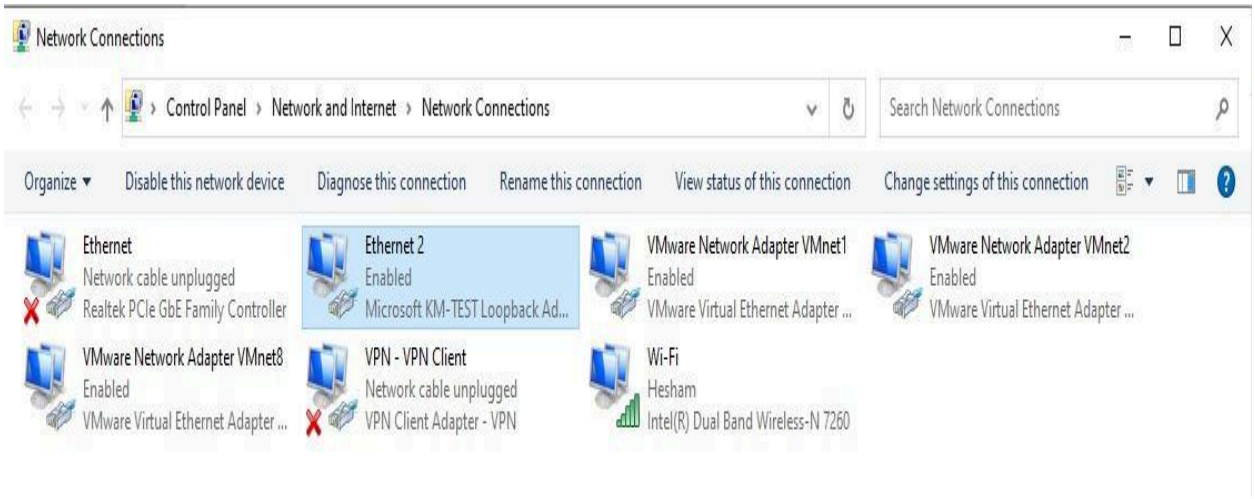


Figure 38 Loopback Interface Adapter in Network Connections

Appendix C. Sockets

The network can be a logical, local network to the computer or one that is physically connected to an external network. The obvious example is the Internet, which you connect to via your ISP.

Socket API Overview

Python's socket module provides an interface to the Berkeley Sockets API. The primary socket API functions and methods in this module are:

- `socket()`
- `bind()`
- `listen()`
- `accept()`
- `connect()`
- `connect_ex()`
- `send()`
- `recv()`
- `close()`

Python provides a convenient and consistent API that maps directly to system calls their C counterparts. As part of its standard library, Python also has classes that make using these low-level socket functions easier. There are also many modules available that implement higher-level Internet protocols like HTTP and SMTP.

One socket (node) listens on a particular port at an IP, while the other socket reaches out to the other to form a connection. The server forms the listener socket while the client reaches out to the server.

The main methods we attempted to use in the ChainSpoT chat program are:

- Sending:
  - connect (): connects to a remote socket at the address.
  - send (): sends data to the socket.
- Receiving:

- `bind()`: binds the server to a specific IP and port so that it can listen to incoming requests on that IP and port.
- listen(): puts the server into listening mode. This allows the server to listen to incoming connections.
- accept (): initiates a connection with the client.
- recv(): receives data from the socket with the size specified by buffer size (mostly 1024 bytes).
- close(): closes the connection with the client.

APPENDIX D. Gns3

It is open-source software that emulates Cisco router and switches hardware to simulate complex networks. we can use GNS3 on any computer to test with various router configurations.

it allows us to build our network and create many projects by using Cisco and non-Cisco technology, and we can access these projects anytime without needing of internet connection.

It provides us with emulated hardware devices (a variety of routers, switches, and PCs) that we can combine, and these devices run the real network operating systems such as Cisco IOS, simulated operating systems such as NX-OSv, and the ability to share resources across multiple computers. So, we can create a virtualized network using these devices.

it uses a backend hypervisor (a program used to run and manage one or more virtual machines on a computer) application to emulate the hardware that runs Cisco IOS (we run an actual IOS image file on our PC).

In addition to Cisco IOS, GNS3 can integrate Quick Emulator (QEMU) and VirtualBox virtual machines running operating systems such as Linux, BSD, or Windows.

Also, all the configuration commands and output come from a real IOS, and thus, theoretically any protocols or features that an IOS version supports are available to use in our network designs.

Configuration and building our network:

After installing GNS3, we can configure our network in some steps:

- First, we select how we would like to run our GNS3 network simulation as shown in the **Figure 39**.

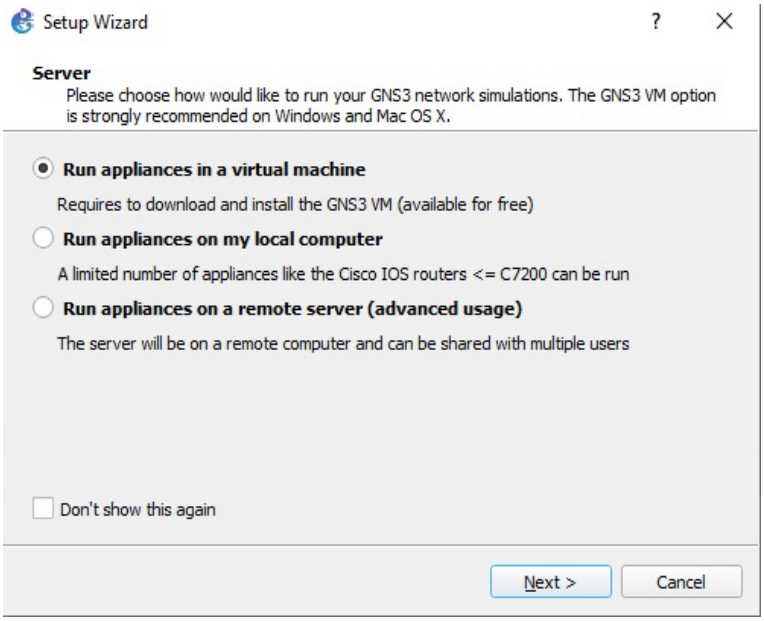


Figure 39 GNS3 Setup Wizard

We choose the virtual machine option (so we need to download and install VM-WARE) then we choose the server path, Host binding, and port.

Before we can boot up a router, we will need to install and configure at least one Cisco IOS image file in GNS3

We installed the c7200 IOS image as we will use the cisco router 7200 in our test.

Now we can select the router (c7200) and it is available for use after configuration from the console.

We want to connect a cloud (the physical PC) to the router so we must choose the server

We choose GNS3 VM as shown in **Figure 40**.

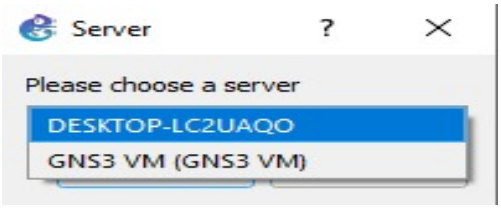


Figure 40 GNS3 Choose Server

We need to give an interface manually (loopback interface to my device) as mentioned in 3.6.1.1 .

Now we will give an IP address to the loopback interface that we created but this IP address must be in the same network with the Router so they can see each other.

We chose 192.168.0.2 as the Ip address for the loopback and 192.168.0.1 as the Ip address for the router as shown in **Figure 41**.

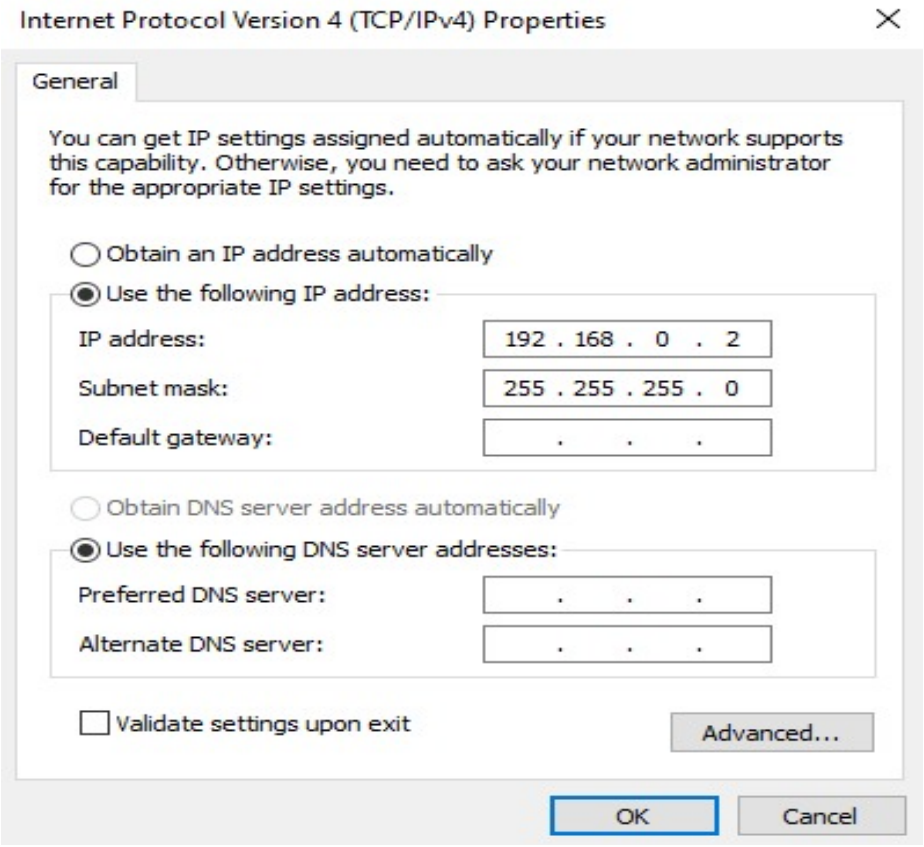


Figure 41 Loopback Static IP

To connect the cloud (the physical pc) to the router we need bridged interface in GNS3 VM. We could take bridged from ethernet or Wi-Fi interface. But after we disconnect from Ethernet or Wi-Fi. Bridged network will no longer be available. We need a more stable interface, not depending on connecting state. So, logical interface became a tempting solution. So, we use bridged from loopback interface as our logical interface and this is done by some steps on VM-ware as shown in the figures below

First, we add new interface in Virtual Network Editor in VM-Ware as shown in **Figure 42**.

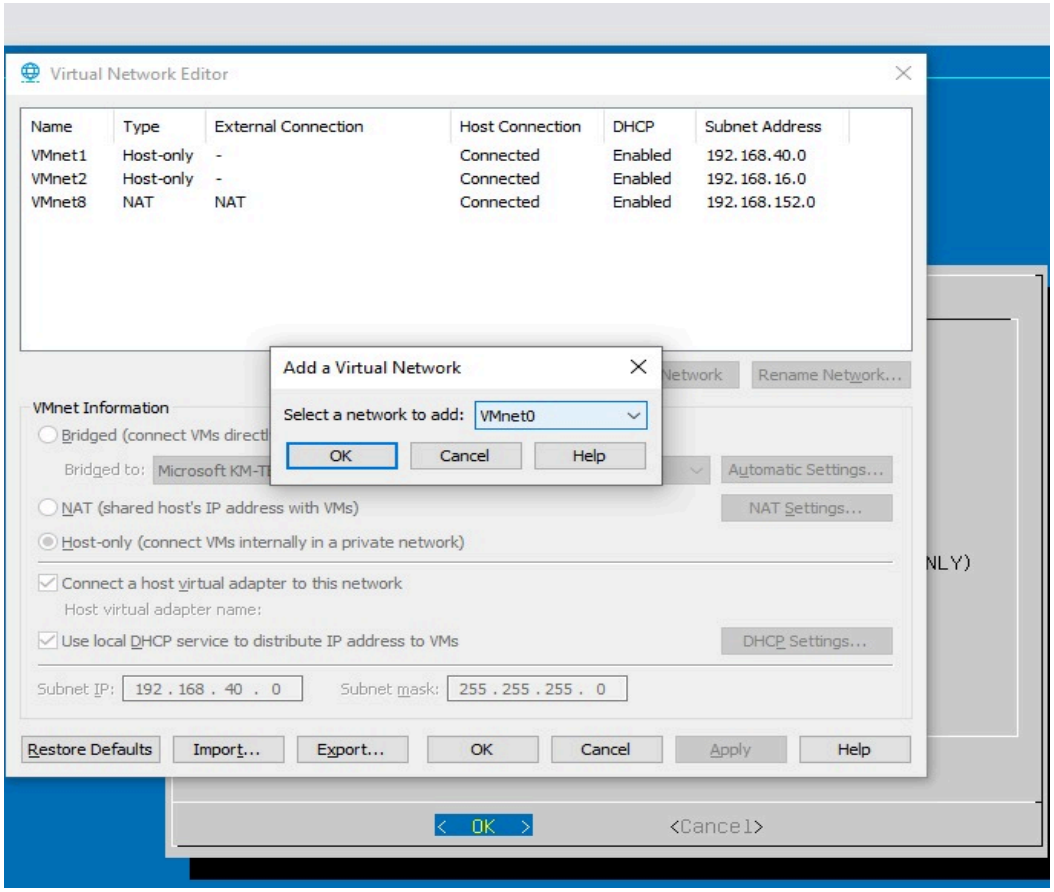


Figure 42 Add Virtual Network to Virtual Network Editor

Then, we make it bridged from Loopback interface as shown in **Figure 43**.

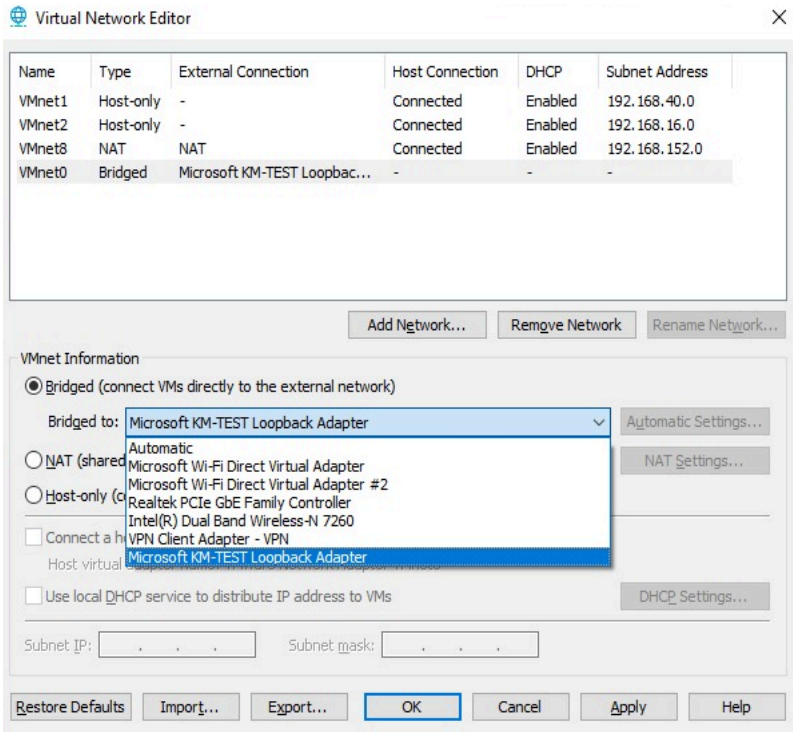
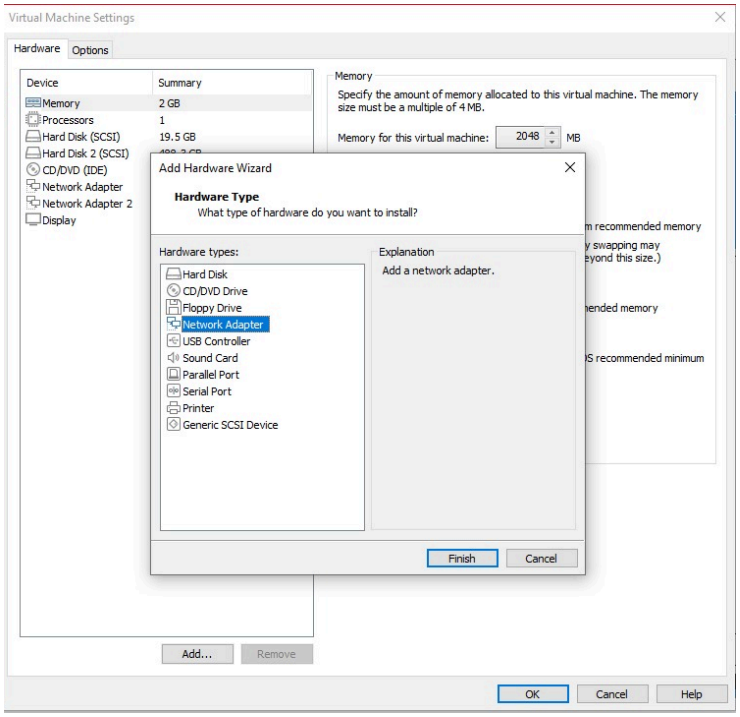


Figure 43 Add Bridged Loopback Interface to VMnet0

Then, in GNS3 VM-Ware settings we add new hardware (Network Adapter)



In this new adapter we make its network connection Bridged as shown in **Figure 44**.

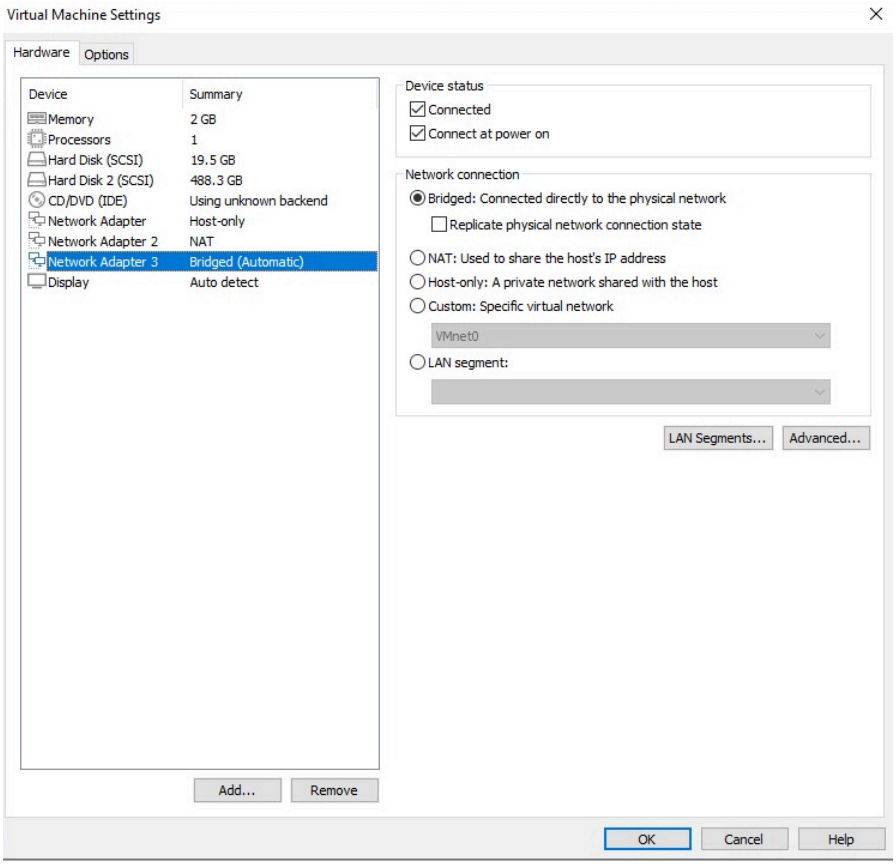


Figure 44 Add Bridged Network Adapter to GNS3 VM

configure the cloud with the new interface (eth2) as shown in **Figure 45**.

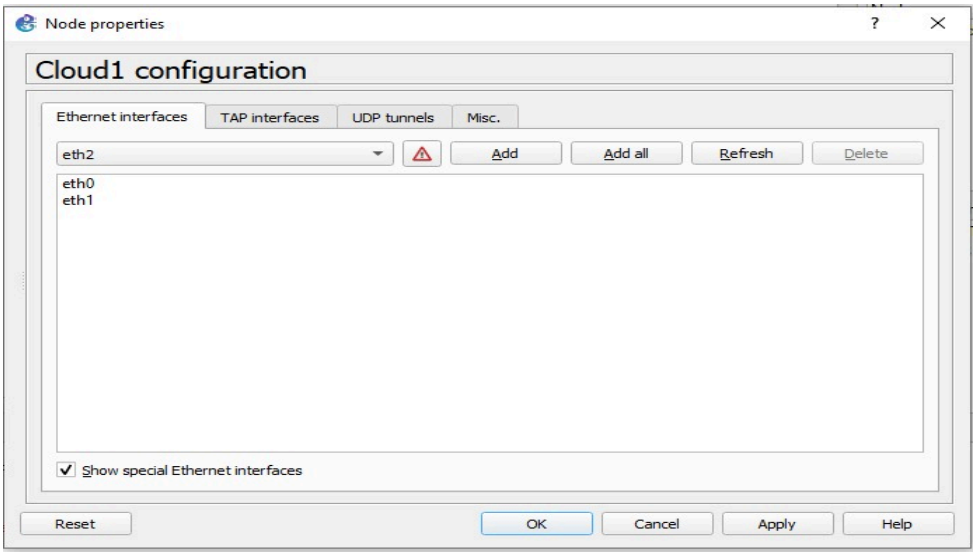


Figure 45 Configuration the Cloud with The New Interface

Now connect router to the cloud by Ethernet2

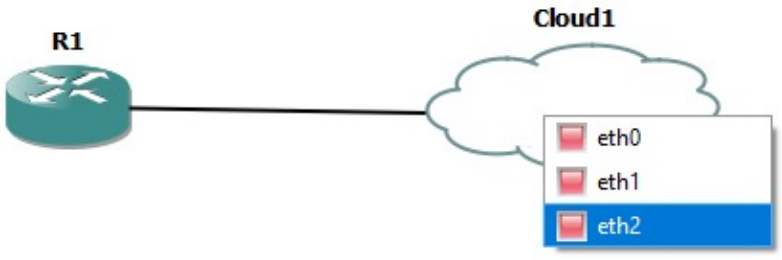
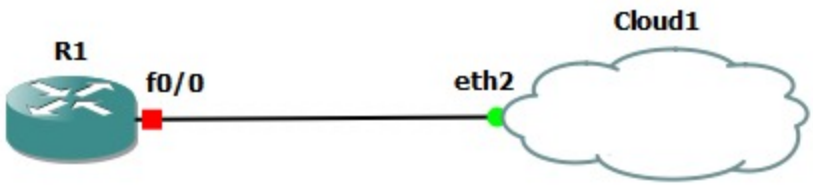


Figure 46 Connect the Router with The New Interface

The connection is done as shown in the following figure.


Cloud1
R1
f0/0
eth2