

INSTITUTO TECNOLÓGICO DE CANCÚN

LEON QUEB MIGUEL ANGEL

ISMAEL JIMENEZ SANCHEZ

FUND. TELECOMUNICACIONES

HORARIO

17:00 – 18:00



## PROYECTO - SISTEMA DE COMUNICACION

Reporte.

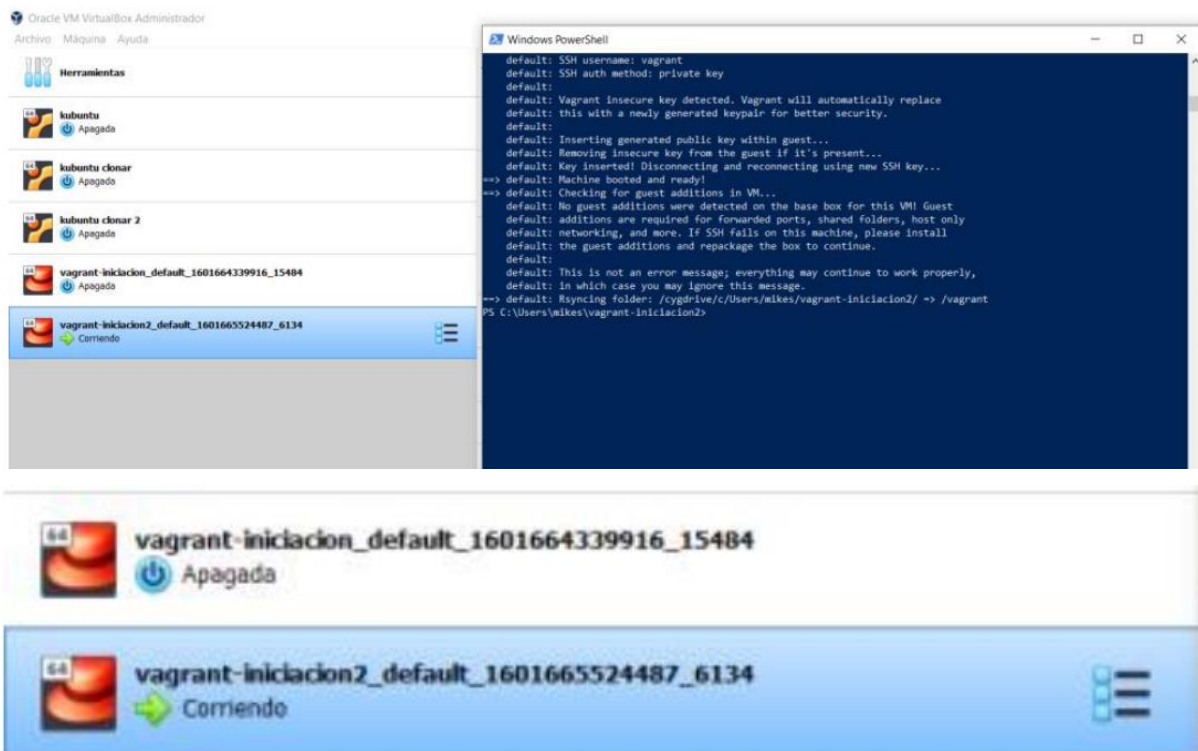
Para la realización de esta práctica se manejaron distintos programas para la creación de un sistema de comunicación en la cual se simuló la comunicación entre 2 maquinas demostrando la interacción de ambas y hacer una observación en los componentes relacionándolo con lo visto en esta primera unidad.

Programas:

- VirtualBox: para la simulación de maquinas
- Vagrant: también para la simulación de maquinas
- GNS3: para la simulación de redes entre ambas maquinas
- Python: lenguaje usado para la aplicación de los scripts dados por el profesor
- PuTTY: para el cliente de acceso remoto a las máquinas virtuales
- Wireshark: para observar el tráfico e información entre ambas maquinas

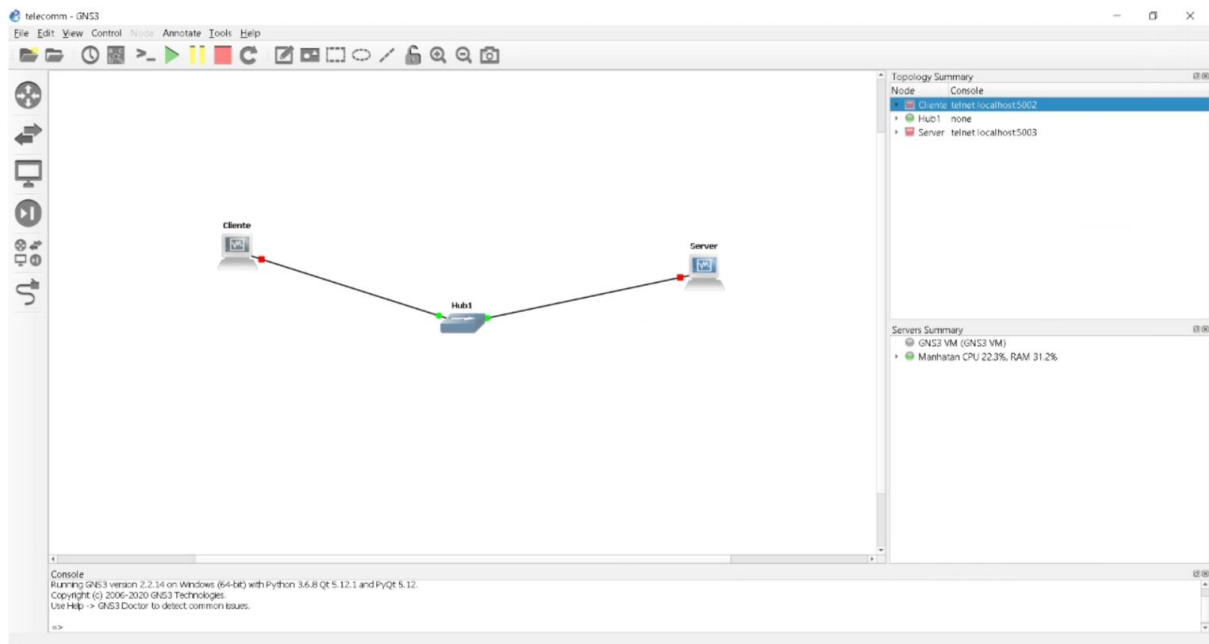
## Fase 1.

Se crearon 2 máquinas virtuales con el sistema operativo CentOS 8 asignándole a una de ellas como la maquina cliente (el emisor) y a la otra como el servidor (el receptor) para la simulación de red.



## Fase 2.

Hacer las conexiones de las máquinas virtuales en GNS3 para la simulación de redes. Una de estas siendo el cliente y la otra el servidor como ya se dijo.



Fase 3.

Utilizar PuTTY para la configuración de las redes y aplicación de los scripts dados por el profesor en el lenguaje de programación Python tanto en el cliente como servidor y de igual forma usando los scripts en su respectiva máquina para su funcionamiento.

The screenshot shows a PuTTY terminal window titled 'Cliente - PuTTY'. The terminal output displays the system boot process, including the starting of the Login Service, System Logging Service, and Dynamic System Tuning Daemon. It then shows the user 'vagrant' logging in as 'root' on a CentOS Linux 8 (Core) system. The user successfully runs the command 'python2 tcclient.py', which outputs 'GET / HTTP/1.1' and 'Host: google.com'.

```
[ OK ] Started Login Service.
[ OK ] Started System Logging Service.
[ OK ] Started Dynamic System Tuning Daemon.
[ OK ] Reached target Multi-User System.
Starting Update UTMP about System Runlevel Changes...
[ OK ] Started Update UTMP about System Runlevel Changes.

CentOS Linux 8 (Core)
Kernel 4.18.0-80.el8.x86_64 on an x86_64

localhost login: vagrant
Password:
Login incorrect

localhost login: vagrant
Password:
Last login: Fri Oct 23 18:55:35 on ttyS0
[vagrant@localhost ~]$ sudo su
[root@localhost vagrant]# python2 tcclient.py
GET / HTTP/1.1
Host: google.com

[root@localhost vagrant]#
```

```
Server - PuTTY

Starting Update UTMP about System Runlevel Changes...
[ OK ] Started Update UTMP about System Runlevel Changes.

CentOS Linux 8 (Core)
Kernel 4.18.0-80.el8.x86_64 on an x86_64

localhost login: vagrant
Password:
Last login: Fri Oct 23 18:55:46 on ttyS0
[vagrant@localhost ~]$ sudo su
[root@localhost vagrant]# python2 tcpserver2.py
Socket Ready...
Bind Ready...
Listening...
Conexion from: ('192.168.60.101', 42194)
Client says: GET / HTTP/1.1
Host: google.com

Sending: GET / HTTP/1.1
Host: google.com
```

```
import socket

target_host = "www.google.com"
target_port = 80

# create a socket object
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# connect the client
client.connect((target_host, target_port))

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

# receive some data
response = client.recv(4096)

print response
```

```
import socket
import threading

bind_ip = "0.0.0.0"
bind_port = 9999

server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

server.bind((bind_ip,bind_port))

server.listen(5)

print "[*] Listening on $s:%d" % (bind_ip,bind_port)

# this is our client-handling thread
def handle_client(client_socket):

    # print out what the client sends
    request = client_socket.recv(1024)

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

    # send back a packet
    client_socket.send("ACK!")

    client_socket.close()
```

```
while True:

    client,addr = server.accept()

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

    # spin up our client thread to handle incoming data
    client_handler = threading.Thread(target=handle_client,args=(client,))
    client_handler.start()
```

#### Fase 4.

Utilizar Wireshark para la observación y captación del tráfico de datos enviados por ambas maquinas.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	RealtekU_72:fe:6e	Broadcast	ARP	60	Who has 192.168.60.102? Tell 192.168.60.101
2	0.000000	RealtekU_72:fe:6e	RealtekU_72:fe:6e	ARP	60	192.168.60.102 is at 52:54:00:72:fe:6e
3	0.000976	192.168.60.101	192.168.60.102	TCP	74	42194 → 130 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=842705613 TSecr=0 WS=64
4	0.000976	192.168.60.102	192.168.60.101	TCP	74	130 → 42194 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=3471541164 TSecr=842705613 WS=64
5	0.001951	192.168.60.101	192.168.60.102	TCP	66	42194 → 130 [ACK] Seq=1 Ack=1 Win=29248 Len=0 TSval=842705615 TSecr=3471541164
6	0.001951	192.168.60.101	192.168.60.102	HTTP	102	GET / HTTP/1.1
7	0.001951	192.168.60.102	192.168.60.101	TCP	66	130 → 42194 [ACK] Seq=1 Ack=37 Win=28992 Len=0 TSval=3471541165 TSecr=842705616
8	0.002928	192.168.60.102	192.168.60.101	HTTP	102	GET / HTTP/1.1
9	0.002928	192.168.60.101	192.168.60.102	TCP	66	42194 → 130 [ACK] Seq=37 Ack=37 Win=29248 Len=0 TSval=842705617 TSecr=3471541166
10	0.007060	192.168.60.101	192.168.60.102	TCP	66	42194 → 130 [FIN, ACK] Seq=37 Ack=37 Win=29248 Len=0 TSval=842705620 TSecr=3471541166
11	0.007060	192.168.60.102	192.168.60.101	TCP	66	130 → 42194 [FIN, ACK] Seq=37 Ack=38 Win=28992 Len=0 TSval=3471541170 TSecr=842705620
12	0.008036	192.168.60.101	192.168.60.102	TCP	66	42194 → 130 [ACK] Seq=38 Ack=38 Win=29248 Len=0 TSval=842705621 TSecr=3471541170
13	5.022858	RealtekU_72:fe:6e	RealtekU_72:fe:6e	ARP	60	Who has 192.168.60.101? Tell 192.168.60.102
14	5.023588	RealtekU_72:fe:6e	RealtekU_72:fe:6e	ARP	60	192.168.60.101 is at 52:54:00:72:fe:6e
15	32.280901	fe80::5054:ff:fe72::	ff02::2	ICMPv6	70	Router Solicitation from 52:54:00:72:fe:6e

**[SYN][ACK][RST]**

Cada una de estas etiquetas (flags) representa una acción o acciones en simultaneo que está realizando una maquina al comunicarse con la otra. La etiqueta SYN en resumen representa la sincronización que se está llevando a cabo en ambas maquinas que se intentan comunicar. La etiqueta ACK aparece cuando este ha logrado obtener una respuesta por parte de una máquina. Es posible encontrar ambas etiquetas (SYN y ACK) en simultaneo, esto representa que el servidor o receptor ha respondido a la solicitud que hace el emisor o cliente. Por último, la etiqueta RST, significa el reinicio de la conexión entre ambas maquinas cuando esta se pierde. Esto usualmente pasa cuando un paquete es enviado a un puerto distinto y no propio de la conexión a la que se desea. Estas 3 etiquetas forman parte del triple handshake, que son los protocolos que aparecen cuando se establece una conexión entre maquinas.



Fase 5.

## Conclusión

Gracias a esta práctica, fue posible ver las interacciones con las que se puede realizar diferentes tipos de redes y también las limitaciones que se existe entre un componente switch a un hub por ejemplo. También algo interesante fue como es posible ver entre redes por Wireshark el envío de datos entre ambas maquinas, como un programa de monitoreo e interceptor de datos.