

INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO

**PRÁCTICA 1: DESCUBRIMIENTO BASADO EN
AGENTES**

ADMINISTRACIÓN DE SERVICIOS EN RED

EQUIPO 8:

+ HERNÁNDEZ PINEDA MIGUEL ANGEL

+ MONROY MARTOS ELIOTH

+ ZÚÑIGA HERNÁNDEZ CARLOS

PROFRA. TANIBET PÉREZ DE LOS SANTOS MONDRAGÓN

2 de Marzo 2019

Índice general

1. Introducción	1
2. Desarrollo y Resultados	2
2.1. Ejercicio 1	2
2.2. Ejercicio 2	3
2.3. Ejercicio 3	6
2.4. Ejercicio 4	10
2.5. Ejercicio 5	11
3. Cuestionario	13
4. Códigos	21
5. Conclusiones	31
Referencias Bibliográficas	33

Capítulo 1

Introducción

La administración de redes se dedica a establecer las actividades que deben realizarse sobre una red informática con el fin de brindar los diferentes servicios de manera eficiente y eficaz, garantizando la disponibilidad y calidad que el usuario final espera. En la administración de redes es muy usado el protocolo SNMP o protocolo simple de Administración de red, que facilita el intercambio de paquetes de información de administración entre los diferentes dispositivos de la red; existen 3 versiones de este protocolo, sin embargo, la última versión SNMPv3 tiene algunos errores de seguridad, por lo que la versión más utilizada es la versión 2.

Para que se pueda llevar a cabo la administración de una red, es necesario configurarla de manera que exista un gestor que será el que monitorizará los demás dispositivos de la misma, mejor conocidos como agentes, los que además deben ser configurados para soportar las tramas enviadas por este tipo de protocolo (SNMP) y que una vez hecho esto, permitan al gestor entrar a una entidad conocida como base de información de administración o MIB (por sus siglas en inglés), de la cual obtienen información relevante de los dispositivos y sus conexiones mediante el acceso a los objetos de esta haciendo uso de sus identificadores u OID.

Existen diversas herramientas que nos permiten llevar a cabo el monitoreo de una red como es el caso de observium, el cual es un sistema basado en Linux, además podemos acceder a dispositivos de una misma red mediante la línea de comandos y consultar los objetos de la MIB de dicho dispositivo. En este documento se detallan ciertas actividades realizadas usando el protocolo SNMP dentro de una red con dispositivos conectados, de manera que se utilizaron 3 formas distintas de consultar los objetos de la MIB, observium, línea de comandos y una herramienta hecha por los titulares del documento, de igual manera se explican los distintos procedimientos y los resultados obtenidos con el fin de establecer las características y diferencias en cada uno de los casos.

Capítulo 2

Desarrollo y Resultados

2.1. Ejercicio 1

Se nos pidió modificar la comunidad de los agentes Linux y windows que teníamos en nuestros equipos por la comunidad **comunidadEquipo8_grupo4cm1**, para el caso de Linux ingresamos al archivo SNMPD.CONF que se encuentra en la ruta **etc/snmp**, buscamos la línea del archivo que comienza con *rwcommunity* y modificamos el valor designado. Esto se puede observar en las figuras 2.1 y 2.2.

```
54 # arguments: community [default|hostname|network/bits] [oid]
55
56 rwcommunity grupo_4cm1
57
58
59
```

Figura 2.1: Línea a modificar del archivo snmpd.conf

```
54 # arguments: community [default|hostname|network/bits] [oid]
55
56 rwcommunity comunidadEquipo8_grupo4cm1
57
```

Figura 2.2: Línea modificada del archivo snmpd.conf

En el caso de Windows, entramos al apartado de servicios locales, identificamos Servicio SNMP, damos clic derecho y elegimos propiedades, ahí en la pestaña de *Seguridad*, damos agregar e ingresamos los valores solicitados como se muestra en la figura 2.3.

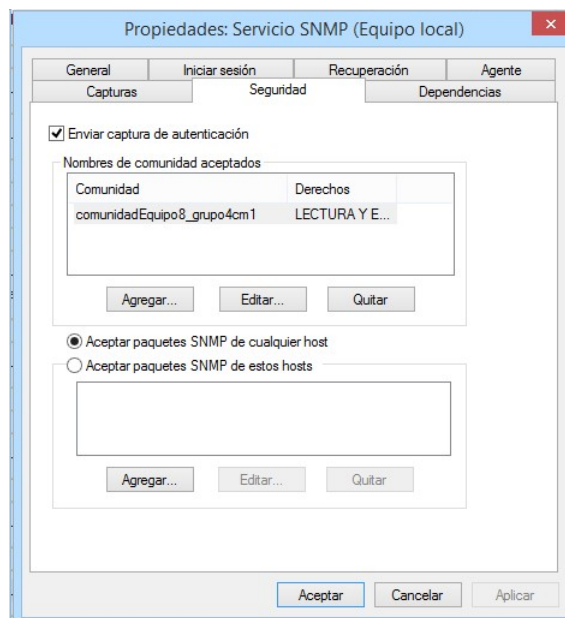


Figura 2.3: Configuración en Windows del nombre de la comunidad

En ambos casos reiniciamos el servicio SNMP para que los cambios surtan efecto, en el caso de Linux es necesario escribir en la consola la línea *service snmp restart*; para el caso de windows, damos clic en Servicio SNMP y posterior a ello damos clic en la opción *reiniciar el servicio*. Ahora para consultar el nombre de la comunidad asignada, lo único que debemos hacer, en el caso de Linux es acceder al archivo *snmpd.conf* y verificamos la información, como se muestra en la figura 2.4.

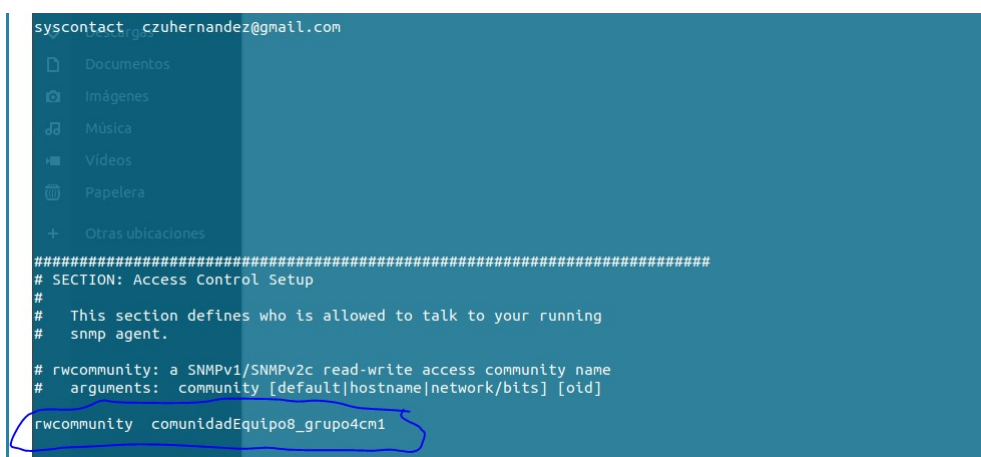


Figura 2.4: Reinicio de servicio snmp y cat a archivo snmpd.config

2.2. Ejercicio 2

Para esta sección se nos pidió monitorizar el agente de Windows mediante el uso de la herramienta Observium, para ello iniciamos en una maquina virtual dicho servicio, obtenemos la IP con la que podremos ingresar desde los agentes y una vez en la consola del sistema, accedemos a **etc/hosts** (figura 2.5) donde agregaremos la IP y el nombre de nuestro agente a monitorizar, como se muestra en la figura 2.6.

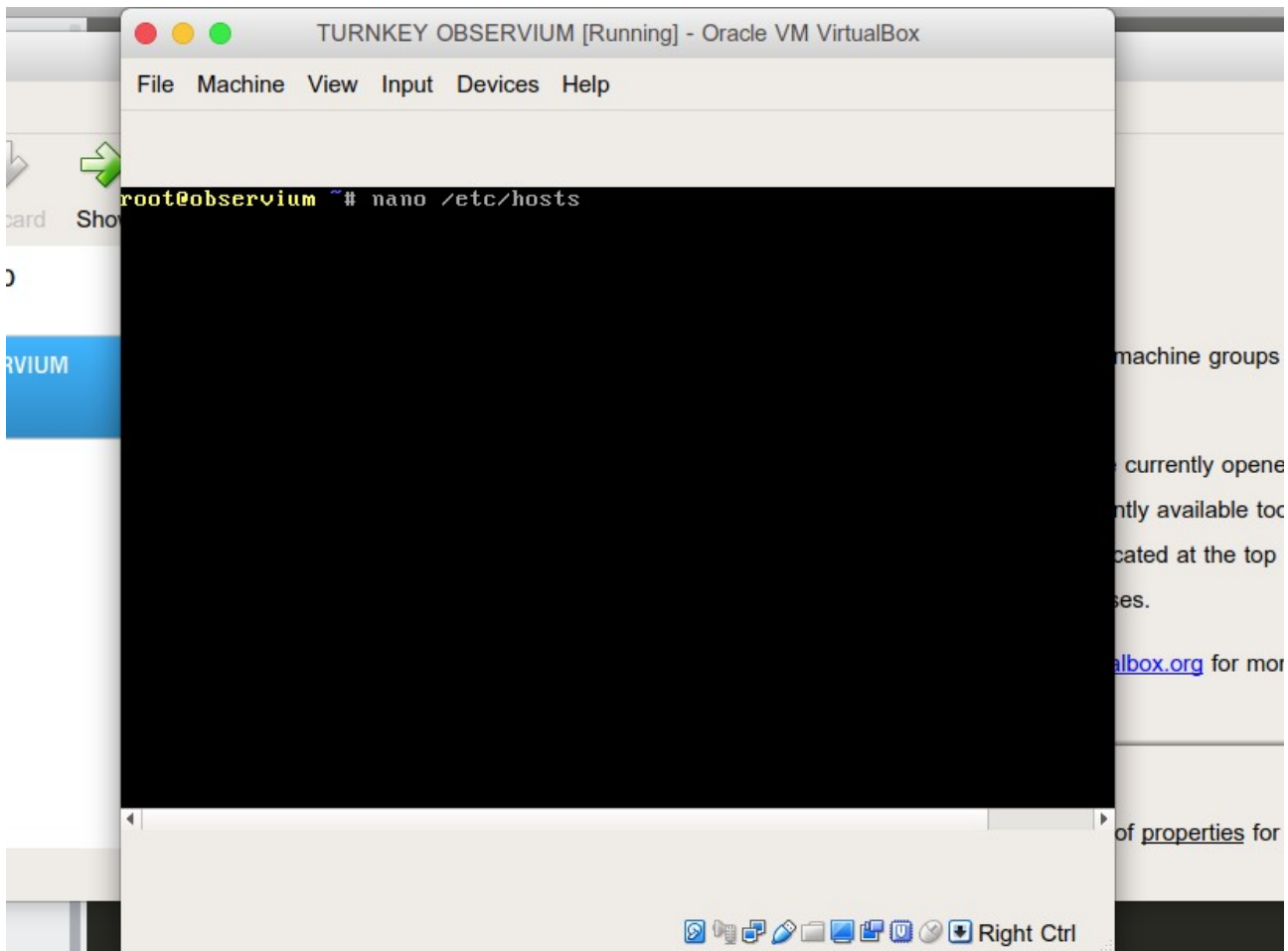


Figura 2.5: Ejecución de nano para agregar el host



Figura 2.6: Ejecución del comando nano para agregar el host

En un buscador en la plataforma Windows, ingresamos a la IP perteneciente a Observium, una vez escritas las credenciales, estaremos en la pestaña principal del gestor,

buscamos la sección *Devices* e introducimos los datos necesarios para agregar un nuevo agente, en este caso, el agente Windows, como se muestra en la figura 2.7.

The screenshot shows the 'Basic Configuration' and 'Authentication Configuration' sections of the Observium interface. In the 'Basic Configuration' section, the 'Hostname' is set to 'windows10', 'Skip PING' is unchecked, 'Protocol Version' is 'v2c', 'Transport' is 'UDP', 'Port' is '161', 'Timeout' is '1', and 'Retries' is '5'. The 'Ignore existing RRDs' checkbox is checked. In the 'Authentication Configuration' section, the 'SNMP Community' is set to 'comunidadEquipo8_grupo4cm1'.

Figura 2.7: Agregar agente desde Observium

Una vez agregado, podremos observar que en la pestaña *Devices* ahora nos aparece Windows como se muestra en las figuras 2.8.

The screenshot shows the 'Devices' page in Observium. A table lists the devices, with 'windows10' highlighted. The table has columns for 'Device / Location', 'Operating System / Hardware Platform', and 'Uptime / sysName'.

Device / Location	Operating System / Hardware Platform	Uptime / sysName
windows10	Microsoft Windows	0s Ckotron

Figura 2.8: Agente agregado a Observium

Si damos clic en el, nos aparecerá la información relevante del agente, así como algunas gráficas del trafico de información de este dispositivo en la red, como se muestra en las figuras 2.9 y 2.10.

The screenshot shows the detailed view of the 'windows10' agent. It includes a sidebar with navigation links (Overview, Graphs, Health, Ports, Inventory, Logs, Alerts) and a main content area with hardware details, system information, and performance metrics.

Hardware	Operating system	System name	Location	Uptime	Last reboot
Intel x64	Microsoft Windows 8.1, Update 1 (NT 6.3) (Multiprocessor)	ckotron	Equipo8Grupo4cm1@evaluacion1.com	41m 34s	2019-02-27 16:58:28

Processors	Memory	Storage
Unknown Processor Type x 4	Virtual Memory: 7.46GB / 15.9GB (47%) Physical Memory: 5.69GB / 7.91GB (72%)	C:\: 344GB / 670GB (51%)

Figura 2.9: Información del agente mostrada por Observium (1)



Figura 2.10: Información del agente mostrada por Observium (2)

2.3. Ejercicio 3

En esta sección tuvimos que hacer uso de nuestra herramienta para el monitoreo de los agentes Linux y Windows anteriormente configurados, esta herramienta se programó en el lenguaje Python, donde se hacen consultas a la información relevante de cada dispositivo así como el tráfico de Red para las siguientes opciones:

- Segmentos TCP (in/out)
- Tráfico de Interfaces (in/out)
- Datagramas UDP (in/out)
- Tráfico ICMP (in/out)
- Tráfico SNMP (in/out)

Al ejecutar el programa (main.py) se muestra la pantalla de inicio, la cual, en caso de no tener agentes registrados se muestra vacía como lo muestra la figura 2.11.

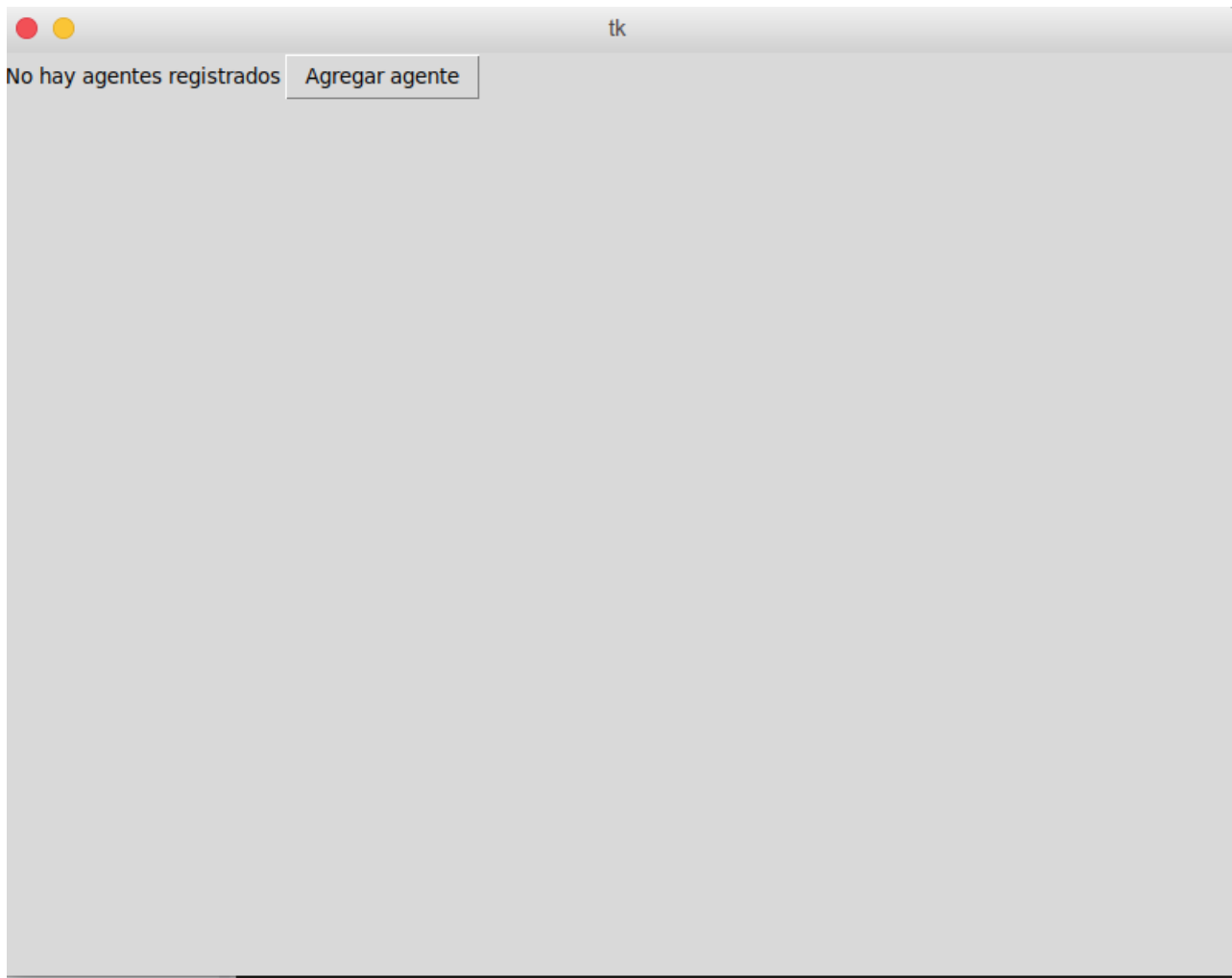
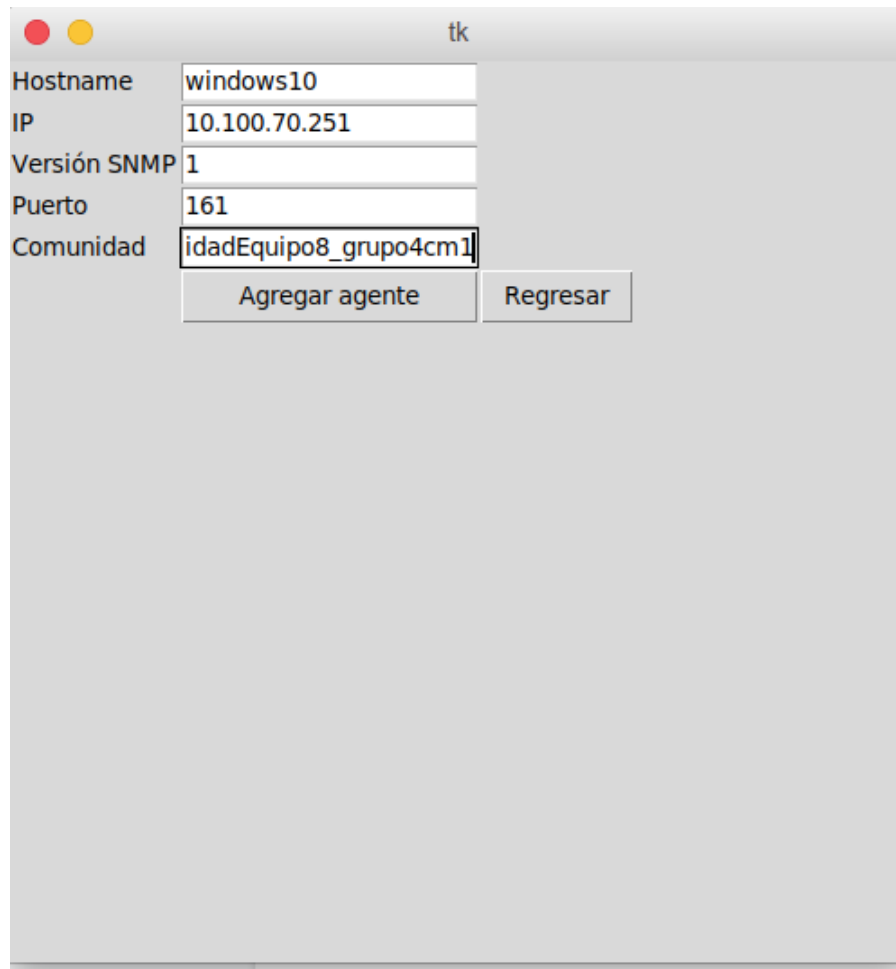


Figura 2.11: Pantalla de inicio del programa

Para agregar un nuevo agente, damos clic en la opción agregar Agente, posteriormente llenamos el formulario con los datos correspondientes y damos clic en Agregar Agente, como se muestra en la pantalla 2.12.



Hostname	windows10
IP	10.100.70.251
Versión SNMP	1
Puerto	161
Comunidad	idadEquipo8_grupo4cm1

Agregar agente Regresar

Figura 2.12: Agregar agente desde el programa

En la pantalla principal nos aparecerá el agente que agregamos, su IP y el estado del mismo, como se muestra en la pantalla 2.13.

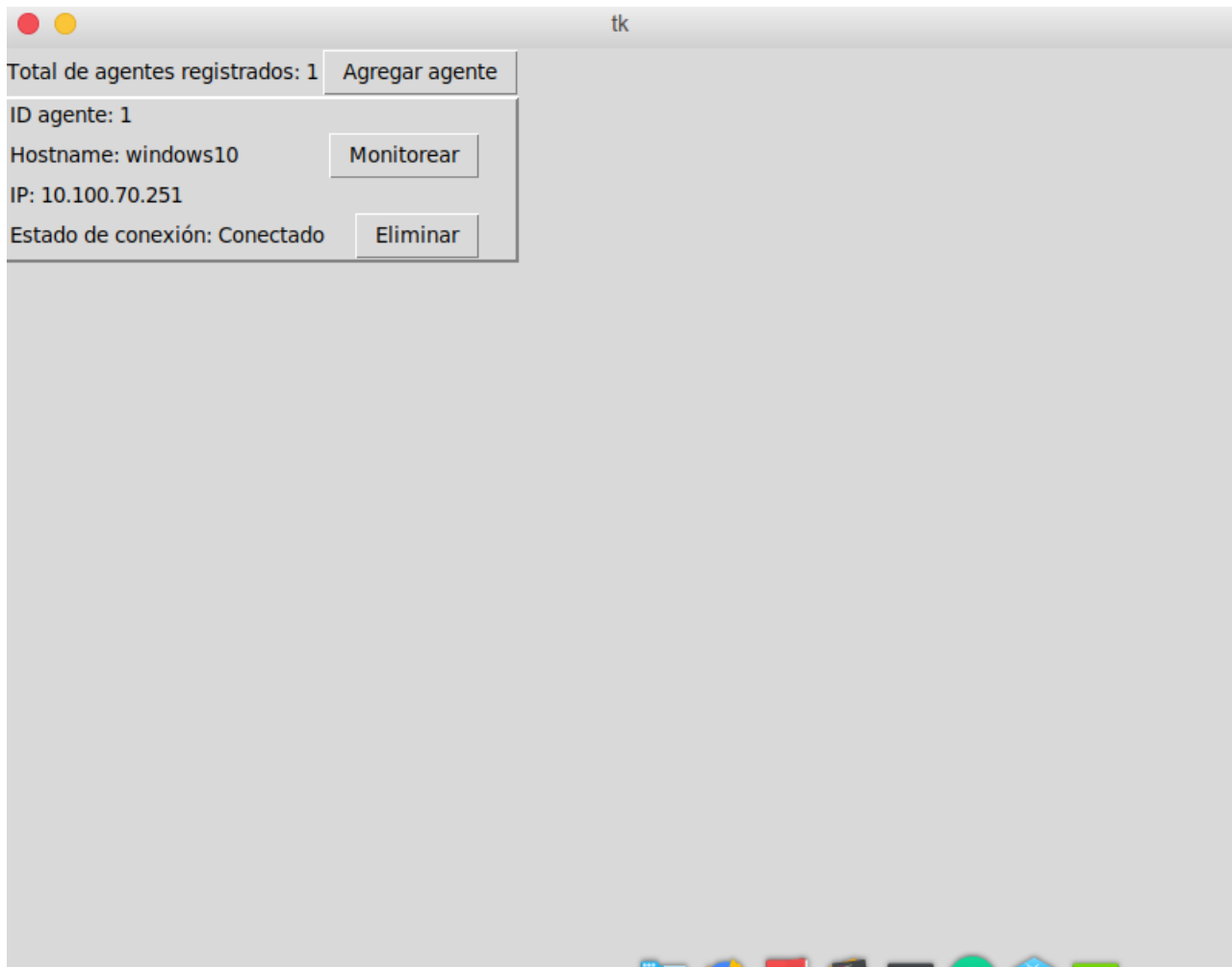
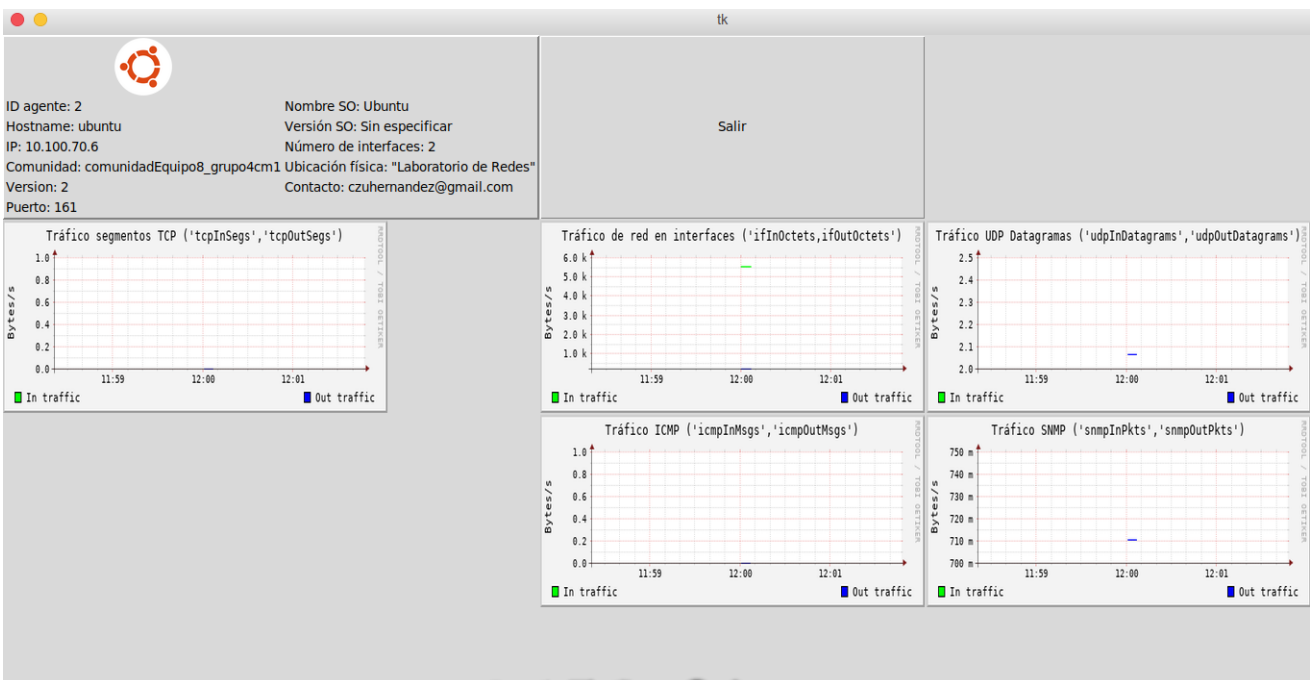


Figura 2.13: Pantalla de inicio con un agente registrado

Si damos clic en la opción *Monitorear* se desplegará toda la información del dispositivo así como las gráficas para las características antes mencionadas como se muestra en la figura 2.14.

El mismo proceso fue repetido para agregar al agente Linux. El resultado de su monitoreo se puede apreciar en la figura 2.15.



2.4. Ejercicio 4

Administración de Servicios en Red

desarrollamos un pequeño programa en Python (figura 2.16) que nos permitiera monitorizar dicho objeto y graficar el tráfico durante un lapso de tiempo de 10 minutos, el resultado se muestra en la figura 2.17.

```

crearRRDUno("prueba.rrd","N","1","600","1","1","600","600")
while 1:

    consulta=consultav2SNMP("variation/virtualtable","10.100.71.230",1,1,"IF-MIB","ifOutOctets",1024)
    valor = "N:" + str(consulta)
    print (valor)
    rrdtool.update('prueba.rrd', valor)
    rrdtool.dump('prueba.rrd','prueba.xml')
    ultimo=rrdtool.last('prueba.rrd')
    ret = rrdtool.graph( "prueba.png",
                        "--start",str(ultimo-600),
                        "--end","+100",
                        "--vertical-label=Bytes/s",
                        "DEF:valor1=prueba.rrd:valor1:AVERAGE",
                        "AREA:valor1#00FF00:In traffic")

    if ret:
        print (rrdtool.error())
        time.sleep(300)

```

Figura 2.16: Código realizado para el monitoreo del objeto MIB

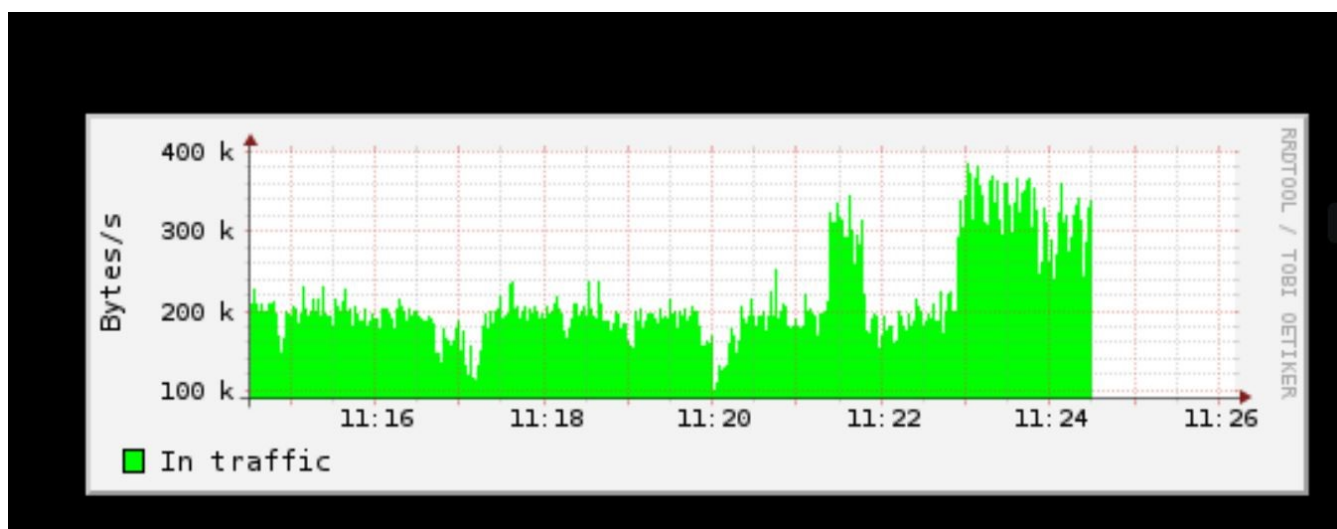


Figura 2.17: Gráfica obtenida como resultado

2.5. Ejercicio 5

En esta ultima sección tuvimos que acceder al objeto *sysLocation* de la MIB con el fin de modificar su valor y asignarle **Equipo8Grupo4cm1@evaluacion.com** para ello, primero debemos obtener el valor actual de dicho objeto haciendo uso del comando `snmpget` como se muestra en la figura 2.18.

```

carlos@carlos-VirtualBox:~$ snmpget -c comunidadEquipo8_grupo4cm1 -v 2c 10.100.70.251 1.3.6.1.2.1.1.6.0
iso.3.6.1.2.1.1.6.0 = STRING: "EDO MEX"
carlos@carlos-VirtualBox:~$

```

Figura 2.18: Pantalla de inicio con un agente registrado

Posterior a ello, haciendo uso del comando `snmpset`, realizamos los cambios deseados de la siguiente manera como se muestra en la figura 2.19.

```
carlos@carlos-VirtualBox:~$ snmpset -c comunidadEquipo8_grupo4cm1 -v 2c 10.100.70.251 1.3.6.1.2.1.1.6.0 s "Equipo8Grupo4cm1@evaluacion1.com"
iso.3.6.1.2.1.1.6.0 = STRING: "Equipo8Grupo4cm1@evaluacion1.com"
carlos@carlos-VirtualBox:~$
```

Figura 2.19: Pantalla de inicio con un agente registrado

Y finalmente, haciendo uso de nuevo del comando `snmpget`, verificamos que los cambios se realizaron de manera satisfactoria, como se muestra en la figura 2.20.

```
carlos@carlos-VirtualBox:~$ snmpget -c comunidadEquipo8_grupo4cm1 -v 2c 10.100.70.251 1.3.6.1.2.1.1.6.0
iso.3.6.1.2.1.1.6.0 = STRING: "Equipo8Grupo4cm1@evaluacion1.com"
carlos@carlos-VirtualBox:~$
```

Figura 2.20: Pantalla de inicio con un agente registrado

Capítulo 3

Cuestionario

1. ¿Cuándo fue el último reinicio (Día, Hora y Minuto) de los agentes?

Agente Windows

Comando:

```
snmpget -c comunidadEquipo8_grupo4cm1 -v 2c 192.168.1.114  
1.3.6.1.2.1.1.3.0
```

Resultado:

```
carlos@carlos-VirtualBox:~$ snmpget -c comunidadEquipo8_grupo4cm1 -v 2c 192.168.1.114 1.3.6.1.2.1.1.3.0  
iso.3.6.1.2.1.1.3.0 = Timeticks: (2633372) 7:18:53.72  
carlos@carlos-VirtualBox:~$
```

Figura 3.1: Ejecución y resultado del comando para el agente Windows

Agente Linux

Comando: `snmpget -c comunidadEquipo8_grupo4cm1 -v 2c localhost
1.3.6.1.2.1.1.3.0`

Resultado:

```
carlos@carlos-VirtualBox:~$ snmpget -c comunidadEquipo8_grupo4cm1 -v 2c localhos  
t 1.3.6.1.2.1.1.3.0  
iso.3.6.1.2.1.1.3.0 = Timeticks: (362880) 1:00:28.80  
carlos@carlos-VirtualBox:~$
```

Figura 3.2: Ejecución y resultado del comando para el agente Linux

2. ¿Cuántas interfaces ethernet tienen?

Agente Windows

Comando:

```
snmpwalk -c comunidadEquipo8_grupo4cm1 -v 2c 192.168.1.114  
1.3.6.1.2.1.2.2.1.3 | grep 6$ | wc -l
```

Resultado:

```
carlos@carlos-VirtualBox:~$ snmpwalk -c comunidadEquipo8_grupo4cm1 -v 2c 192.168.1.114 1.3.6.1.2.1.2.2.1.3 | grep 6$ | wc -l
11
carlos@carlos-VirtualBox:~$
```

Figura 3.3: Ejecución y resultado del comando para el agente Windows

Agente Linux

Comando:

```
snmpwalk -c comunidadEquipo8_grupo4cm1 -v 2c localhost
1.3.6.1.2.1.2.2.1.3 | grep 6$ | wc -l
```

Resultado:

```
carlos@carlos-VirtualBox:~$ snmpwalk -c comunidadEquipo8_grupo4cm1 -v 2c localhost 1.3.6.1.2.1.2.2.1.3 | grep 6$ | wc -l
1
carlos@carlos-VirtualBox:~$
```

Figura 3.4: Ejecución y resultado del comando para el agente Linux

Es necesario notar que grep filtra por 6, este valor corresponde a las interfaces que son de tipo ethernet.

3. ¿Cuál es la velocidad (en mbps) de esas interfaces?

Agente Windows

Comando:

```
snmpwalk -c comunidadEquipo8_grupo4cm1 -v 2c 192.168.1.114
1.3.6.1.2.1.2.2.1.5 | grep -E 5.2\ \|5.4\|5.5\|5.9\|5.1[0-6]
```

Resultado:

```
carlos@carlos-VirtualBox:~$ snmpwalk -c comunidadEquipo8_grupo4cm1 -v 2c 192.168.1.114 1.3.6.1.2.1.2.2.1.5 | grep -E 5.2\ \|5.4\|5.5\|5.9\|5.1[0-6]
iso.3.6.1.2.1.2.2.1.5.2 = Gauge32: 0
iso.3.6.1.2.1.2.2.1.5.4 = Gauge32: 0
iso.3.6.1.2.1.2.2.1.5.5 = Gauge32: 3000000
iso.3.6.1.2.1.2.2.1.5.9 = Gauge32: 1000000000
iso.3.6.1.2.1.2.2.1.5.10 = Gauge32: 1000000000
iso.3.6.1.2.1.2.2.1.5.11 = Gauge32: 1000000000
iso.3.6.1.2.1.2.2.1.5.12 = Gauge32: 1000000000
iso.3.6.1.2.1.2.2.1.5.13 = Gauge32: 0
iso.3.6.1.2.1.2.2.1.5.14 = Gauge32: 0
iso.3.6.1.2.1.2.2.1.5.15 = Gauge32: 0
iso.3.6.1.2.1.2.2.1.5.16 = Gauge32: 0
carlos@carlos-VirtualBox:~$
```

Figura 3.5: Ejecución y resultado del comando para el agente Windows

Agente Linux

Comando:

```
snmpwalk -c comunidadEquipo8_grupo4cm1 -v 2c localhost
1.3.6.1.2.1.2.2.1.5 | grep 5.2
```


Resultado:

```
carlos@carlos-VirtualBox:~$ snmpwalk -c comunidadEquipo8_grupo4cm1 -v 2c localhost 1.3.6.1.2.1.2.2.1.5 | grep 5.2
iso.3.6.1.2.1.2.2.1.5.2 = Gauge32: 1000000000
carlos@carlos-VirtualBox:~$
```

Figura 3.6: Ejecución y resultado del comando para el agente Linux

Para resolver esto para ambos agentes, se utilizó el resultado anterior para filtrar por el OID de cada interfaz.

4. ¿Cuál es la interfaz que ha recibido el mayor número de octetos?

Agente Windows

Comando:

```
snmpwalk -c comunidadEquipo8_grupo4cm1 -v 2c 192.168.1.114
1.3.6.1.2.1.2.2.1.10
```

Resultado:

```
carlos@carlos-VirtualBox:~$ snmpwalk -c comunidadEquipo8_grupo4cm1 -v 2c 192.168.1.114 1.3.6.1.2.1.2.2.1.10
iso.3.6.1.2.1.2.2.1.10.1 = Counter32: 0
iso.3.6.1.2.1.2.2.1.10.2 = Counter32: 0
iso.3.6.1.2.1.2.2.1.10.3 = Counter32: 24320268
iso.3.6.1.2.1.2.2.1.10.4 = Counter32: 0
iso.3.6.1.2.1.2.2.1.10.5 = Counter32: 0
iso.3.6.1.2.1.2.2.1.10.6 = Counter32: 0
iso.3.6.1.2.1.2.2.1.10.7 = Counter32: 0
iso.3.6.1.2.1.2.2.1.10.8 = Counter32: 0
iso.3.6.1.2.1.2.2.1.10.9 = Counter32: 0
iso.3.6.1.2.1.2.2.1.10.10 = Counter32: 0
iso.3.6.1.2.1.2.2.1.10.11 = Counter32: 0
iso.3.6.1.2.1.2.2.1.10.12 = Counter32: 0
iso.3.6.1.2.1.2.2.1.10.13 = Counter32: 0
iso.3.6.1.2.1.2.2.1.10.14 = Counter32: 0
iso.3.6.1.2.1.2.2.1.10.15 = Counter32: 0
iso.3.6.1.2.1.2.2.1.10.16 = Counter32: 0
iso.3.6.1.2.1.2.2.1.10.17 = Counter32: 24320268
iso.3.6.1.2.1.2.2.1.10.18 = Counter32: 24320268
iso.3.6.1.2.1.2.2.1.10.19 = Counter32: 24320268
iso.3.6.1.2.1.2.2.1.10.20 = Counter32: 24320268
iso.3.6.1.2.1.2.2.1.10.21 = Counter32: 24320268
iso.3.6.1.2.1.2.2.1.10.22 = Counter32: 24320268
iso.3.6.1.2.1.2.2.1.10.23 = Counter32: 0
iso.3.6.1.2.1.2.2.1.10.24 = Counter32: 0
iso.3.6.1.2.1.2.2.1.10.25 = Counter32: 0
iso.3.6.1.2.1.2.2.1.10.26 = Counter32: 0
iso.3.6.1.2.1.2.2.1.10.27 = Counter32: 0
carlos@carlos-VirtualBox:~$
```

Figura 3.7: Ejecución y resultado del comando para el agente Windows**Agente Linux**

Comando:

```
snmpwalk -c comunidadEquipo8_grupo4cm1 -v 2c localhost
1.3.6.1.2.1.2.2.1.10
```

Resultado:

```
carlos@carlos-VirtualBox:~$ snmpwalk -c comunidadEquipo8_grupo4cm1 -v 2c localhost 1.3.6.1.2.1.2.2.1.10
iso.3.6.1.2.1.2.2.1.10.1 = Counter32: 2061872
iso.3.6.1.2.1.2.2.1.10.2 = Counter32: 8388214
carlos@carlos-VirtualBox:~$
```

Figura 3.8: Ejecución y resultado del comando para el agente Linux

En el agente Windows se observa que muchas interfaces no han recibido octetos y varias han recibido el mismo número.

Para el agente Linux se sombreó el registro que corresponde a la interfaz que recibió el mayor número de octetos.

5. Indica cuál interfaz de red ha recibido el mayor número de octetos.

Se pregunta lo mismo que en la pregunta número cuatro.

6. ¿Cuál es la MAC de esa interfaz?

Agente Windows

Comando:

```
snmpget -c comunidadEquipo8_grupo4cm1 -v 2c 192.168.1.114
1.3.6.1.2.1.2.2.1.6.3
```

Resultado:

```
carlos@carlos-VirtualBox:~$ snmpget -c comunidadEquipo8_grupo4cm1 -v 2c 192.168.1.114 1.3.6.1.2.1.2.2.1.6.3
iso.3.6.1.2.1.2.2.1.6.3 = Hex-STRING: 34 E6 AD BD 39 C7
carlos@carlos-VirtualBox:~$
```

Figura 3.9: Ejecución y resultado del comando para el agente Windows

Para este agente, se tomó una de las interfaces, en este caso la

1.3.6.1.2.1.2.2.1.6.3

Agente Linux

Comando:

```
snmpget -c comunidadEquipo8_grupo4cm1 -v 2c localhost
1.3.6.1.2.1.2.2.1.6.2
```

Resultado:

```
carlos@carlos-VirtualBox:~$ snmpget -c comunidadEquipo8_grupo4cm1 -v 2c localhost 1.3.6.1.2.1.2.2.1.6.2
iso.3.6.1.2.1.2.2.1.6.2 = Hex-STRING: 08 00 27 5F 85 8C
carlos@carlos-VirtualBox:~$
```

Figura 3.10: Ejecución y resultado del comando para el agente Linux

7. ¿Cuántos mensajes ICMP ha recibido el agente?

Agente Windows

Comando:

```
snmpget -c comunidadEquipo8_grupo4cm1 -v 2c 192.168.1.114
1.3.6.1.2.1.5.1.0
```

Resultado:

```
carlos@carlos-VirtualBox:~$ snmpget -c comunidadEquipo8_grupo4cm1 -v 2c 192.168.1.114 1.3.6.1.2.1.5.1.0
iso.3.6.1.2.1.5.1.0 = Counter32: 500
carlos@carlos-VirtualBox:~$
```

Figura 3.11: Ejecución y resultado del comando para el agente Windows

Agente Linux

Comando:

```
snmpget -c comunidadEquipo8_grupo4cm1 -v 2c localhost
1.3.6.1.2.1.5.1.0
```

Resultado:

```
carlos@carlos-VirtualBox:~$ snmpget -c comunidadEquipo8_grupo4cm1 -v 2c localhost 1.3.6.1.2.1.5.1.0
iso.3.6.1.2.1.5.1.0 = Counter32: 13403
carlos@carlos-VirtualBox:~$
```

Figura 3.12: Ejecución y resultado del comando para el agente Linux

8. ¿Cuántas entradas tiene la tabla de enrutamiento IP?

Las entradas de la tabla ipv6RouterAdvertTable son 12.

Por otro lado, la tabla ipDefaultRouterTable tiene 5.

9. ¿Cuántos datagramas UDP ha recibido el agente?

Agente Windows

Comando:

```
snmpget -c comunidadEquipo8_grupo4cm1 -v 2c 192.168.1.114
1.3.6.1.2.1.7.2.0
```

Resultado:

```
carlos@carlos-VirtualBox:~$ snmpget -c comunidadEquipo8_grupo4cm1 -v 2c 192.168.1.114 1.3.6.1.2.1.7.2.0
iso.3.6.1.2.1.7.2.0 = Counter32: 3656
carlos@carlos-VirtualBox:~$
```

Figura 3.13: Ejecución y resultado del comando para el agente Windows

Agente Linux

Comando:

```
snmpget -c comunidadEquipo8_grupo4cm1 -v 2c localhost
1.3.6.1.2.1.7.2.0
```

Resultado:

```
carlos@carlos-VirtualBox:~$ snmpget -c comunidadEquipo8_grupo4cm1 -v 2c localhost 1.3.6.1.2.1.7.2.0
iso.3.6.1.2.1.7.2.0 = Counter32: 200
carlos@carlos-VirtualBox:~$
```

Figura 3.14: Ejecución y resultado del comando para el agente Linux

10. ¿El agente ha recibido mensajes TCP? ¿Cuántos?

Agente Windows

Comando:

```
snmpget -c comunidadEquipo8_grupo4cm1 -v 2c 192.168.1.114
1.3.6.1.2.1.6.10.0
```

Resultado:

```
carlos@carlos-VirtualBox:~$ snmpget -c comunidadEquipo8_grupo4cm1 -v 2c 192.168.1.114 1.3.6.1.2.1.6.10.0
iso.3.6.1.2.1.6.10.0 = Counter32: 626840
carlos@carlos-VirtualBox:~$
```

Figura 3.15: Ejecución y resultado del comando para el agente Windows

Agente Linux

Comando:

```
snmpget -c comunidadEquipo8_grupo4cm1 -v 2c localhost
1.3.6.1.2.1.6.10.0
```

Resultado:

```
carlos@carlos-VirtualBox:~$ snmpget -c comunidadEquipo8_grupo4cm1 -v 2c localhost 1.3.6.1.2.1.6.10.0
iso.3.6.1.2.1.6.10.0 = Counter32: 6915
carlos@carlos-VirtualBox:~$
```

Figura 3.16: Ejecución y resultado del comando para el agente Linux

11. ¿Cuántos mensajes EGP ha recibido el agente?

Agente Windows

Comando:

```
snmpget -c comunidadEquipo8_grupo4cm1 -v 2c 192.168.1.114
1.3.6.1.2.1.8.5.1.4
```

Resultado:

```
carlos@carlos-VirtualBox:~$ snmpwalk -c comunidadEquipo8_grupo4cm1 -v 2c 192.168.1.114 1.3.6.1.2.1.8.5.1.4
iso.3.6.1.2.1.8.5.1.4 = No Such Object available on this agent at this OID
carlos@carlos-VirtualBox:~$
```

Figura 3.17: Ejecución y resultado del comando para el agente Windows

Agente Linux

Comando:

```
snmpget -c comunidadEquipo8_grupo4cm1 -v 2c localhost
1.3.6.1.2.1.8.5.1.4
```

Resultado:

```
carlos@carlos-VirtualBox:~$ snmpwalk -c comunidadEquipo8_grupo4cm1 -v 2c localhost 1.3.6.1.2.1.8.5.1.4
iso.3.6.1.2.1.8.5.1.4 = No Such Object available on this agent at this OID
carlos@carlos-VirtualBox:~$
```

Figura 3.18: Ejecución y resultado del comando para el agente Linux

Se pudo observar que ninguno de los dos agentes pudo realizar la consulta debido a que no se reconoce el objeto descrito por el OID especificado en el comando.

12. Indica el SO que dé el agente.

Agente Windows

Comando:

```
snmpget -c comunidadEquipo8_grupo4cm1 -v 2c 192.168.1.114
1.3.6.1.2.1.1.1.0
```

Resultado:

```
carlos@carlos-VirtualBox:~$ snmpget -c comunidadEquipo8_grupo4cm1 -v 2c 192.168.1.114 1.3.6.1.2.1.1.1.0
iso.3.6.1.2.1.1.1.0 = STRING: "Hardware: Intel64 Family 6 Model 61 Stepping 4 AT/AT COMPATIBLE - Software: Windows Version 6.3 (Build 9600 Mul
tiprocessor Free)"
carlos@carlos-VirtualBox:~$
```

Figura 3.19: Ejecución y resultado del comando para el agente Windows

Agente Linux

Comando:

```
snmpget -c comunidadEquipo8_grupo4cm1 -v 2c localhost
1.3.6.1.2.1.1.1.0
```

Resultado:

```
carlos@carlos-VirtualBox:~$ snmpget -c comunidadEquipo8_grupo4cm1 -v 2c localhost 1.3.6.1.2.1.1.1.0
iso.3.6.1.2.1.1.1.0 = STRING: "Linux carlos-VirtualBox 4.15.0-45-generic #48-Ubuntu SMP Tue Jan 29 16:28:13 UTC 2019 x86_64"
carlos@carlos-VirtualBox:~$
```

Figura 3.20: Ejecución y resultado del comando para el agente Linux

13. Modifica el nombre del contacto o la ubicación del sistema de un agente.

Agente Windows

Consulta:

```
snmpget -c comunidadEquipo8_grupo4cm1 -v 2c 192.168.1.114
1.3.6.1.2.1.1.6.0
```

```
carlos@carlos-VirtualBox:~$ snmpget -c comunidadEquipo8_grupo4cm1 -v 2c 192.168.1.114 1.3.6.1.2.1.1.6.0
iso.3.6.1.2.1.1.6.0 = STRING: "Redes 3"
carlos@carlos-VirtualBox:~$
```

Figura 3.21: Ejecución y resultado del comando para el agente Windows

Modificación:

```
snmpset -c comunidadEquipo8_grupo4cm1 -v 2c 192.168.1.114  
1.3.6.1.2.1.1.6.0 s "Estado de Mexico"
```

Resultado:

```
carlos@carlos-VirtualBox:~$ snmpset -c comunidadEquipo8_grupo4cm1 -v 2c 192.168.1.114 1.3.6.1.2.1.1.6.0 s "Estado de Mexico"  
iso.3.6.1.2.1.1.6.0 = STRING: "Estado de Mexico"  
carlos@carlos-VirtualBox:~$
```

Figura 3.22: Ejecución y resultado del comando para el agente Windows

14. Dibuja la MIB del agente.

En la figura 3.23, se puede ver el uso del comando:

```
snmptranslate -Tp -OS 1.3.6.1.2.1
```

```
carlos@carlos-VirtualBox:~$ snmptranslate -Tp -OS 1.3.6.1.2.1  
+--iso(1)  
carlos@carlos-VirtualBox:~$
```

Figura 3.23: Uso de snmptranslate para dibujar la MIB de un agente

Sin embargo, no es posible dibujar todo el árbol aunque se cambie el OID por 1.3.6.1.2.1.1 o cualquier otro; este comando siempre devolverá el mismo resultado al utilizar esas opciones.

Capítulo 4

Códigos

El archivo con el que se inicia la ejecución del programa, es con main.py, el cual solo crea una instancia del objeto Gestor, el cual se encuentra dentro del archivo gestor.py. Esto para ajustarnos al desarrollo orientado a objetos.

main.py:

```
1 from gestor import Gestor
2
3 gestor=Gestor()
```

El archivo gestor.py, es el que maneja toda la interfaz principal.

gestor.py:

```
1 #https://www.tutorialspoint.com/python/python_gui_programming.htm
2 from tkinter import *
3 from agregar_agente import AgregarAgente
4 from agente import obtenerAgentes , obtenerInfoPrincipalAgente , eliminar
5 from monitor import Monitor
6 from functools import partial
7
8 class Gestor():
9     """Clase principal del programa"""
10     def __init__(self):
11         #Creamos la ventana y sus medidas
12         self.top=Tk()
13         self.top.geometry("800x600")
14         self.top.resizable(0,0)
15         #Definimos los elementos de la misma y a cada uno le hacemos un pack
16         self.b = Button(self.top, text="Agregar agente", command=self.agregarAgente)
17         self.b.grid(row=0, column=3, sticky=W+E+N+S)
18         self.obtenerAgentes()
19         #Final
20         self.top.mainloop()
21     def agregarAgente(self):
22         #Destruimos la ventana actual y traemos a AgregarAgente
23         self.top.destroy()
24         self.agente=AgregarAgente()
25     def monitorearAgente(self, id_agente):
26         monitor=Monitor(id_agente)
27         monitor.start()
28
29     def eliminarAgente(self, id_agente):
30         self.top.destroy()
31         eliminar(id_agente)
32
33     def obtenerAgentes(self):
```

```

34     ids_agentes=obtenerAgentes()
35     if len(ids_agentes)>0:
36         Label(self.top, text="Total de agentes registrados: "+ str(len(ids_agentes
37         ))) .grid(row=0, sticky=W)
38     else:
39         Label(self.top, text="No hay agentes registrados").grid(row=0, sticky=W)
40     for id_agente in ids_agentes:
41         info=obtenerInfoPrincipalAgente(id_agente)
42         frame=Frame(self.top, width = 600, height = 200, relief = 'raised',
43         borderwidth=2)
44         frame.grid(row = int(id_agente), column = 0, columnspan=6, sticky=W+E+N+S
45         )
46         Label(frame, text="ID agente: "+id_agente).grid(row=1, column=0, sticky=W)
47         Label(frame, text="Hostname: "+info[0]).grid(row=2, column=0, sticky=W)
48         Label(frame, text="IP: "+info[1]).grid(row=3, column=0, sticky=W)
49         if info[2]=="":
50             Label(frame, text="Estado de conexión: Desconectado").grid(row=4, column
51             =0, sticky=W)
52         else:
53             Label(frame, text="Estado de conexión: Conectado").grid(row=4, column=0,
54             sticky=W)
55         #Label(frame, text="").grid(row=1, column=0, sticky=W)
56         Button(frame, text="Monitorear", command=partial(self.monitorearAgente,
57         id_agente)).grid(row=2, column=3, sticky=E)
58         Button(frame, text="Eliminar", command=partial(self.eliminarAgente,
59         id_agente)).grid(row=4, column=3, sticky=E)

```

Este archivo, maneja todo lo referente a los agentes. agente.py:

```

1 import json
2 import os.path
3 from SNMP import getInfo, consultav2SNMP
4 def obtenerAgentes():
5     print("Voy a obtener los agentes")
6     with open('agentes.json','r') as f:
7         if os.path.getsize('agentes.json') > 0:
8             print("Existe al menos un agente")
9             data=json.load(f)
10            return list(data.keys())
11        else:
12            print("No hay agentes registrados")
13    return []
14 def obtenerInfoPrincipalAgente(id):
15     data={}
16     info=[]
17     with open('agentes.json','r') as f:
18         if os.path.getsize('agentes.json') > 0:
19             data=json.load(f)
20             #0->hostname
21             info.append(data[id]["hostname"])
22             #1->ip
23             info.append(data[id]["ip"])
24             #2-> estado conexión
25             info.append(consultav2SNMP(data[id]["comunidad"],data[id]["ip"],int(data[id]["
26             version"]),0,'SNMPv2-MIB','sysUpTime',int(data[id]["puerto"])))
27             info.append(data[id]["comunidad"])
28             info.append(data[id]["version"])
29             info.append(data[id]["puerto"])
30            return info
31 def obtenerInfoAgente(id):
32     data={}

```



```

33 info=[]
34 with open('agentes.json','r') as f:
35     if os.path.getsize('agentes.json') > 0:
36         data=json.load(f)
37
38 aux=getInfo(data[id][ "comunidad"],data[id][ "ip"],int(data[id][ "version"]),int(
39     data[id][ "puerto"]))
40 #obtener nombre versión y logo SO, número de interfaces de red, tiempo de
41 #actividad desde último reinicio,
42 #ubicación física e información de contacto del administrador
43 #0->nombre so
44 if "Ubuntu" in aux[1]:
45     info.append("Ubuntu")
46     info.append("ubuntu.png")
47 elif "Windows" in aux[1]:
48     info.append("Windows")
49     info.append("windows.png")
50 continuar=0
51 info.append("Sin especificar")
52 for a in aux[1].split():
53     if continuar==1:
54         info[2]=a
55         break
56     if "Version" in a:
57         continuar=1
58 info.append(aux[2])
59 info.append(int(aux[3])/100)
60 info.append(aux[4])
61 info.append(aux[5])
62 #print (info)
63 return info
64
65 def eliminar(id_agente):
66     data={}
67     with open('agentes.json','r+') as f:
68         if os.path.getsize('agentes.json') > 0:
69             data=json.load(f)
70             data.pop(id_agente,None)
71             print(data)
72             f.seek(0)
73         with open('agentes.json','w') as f:
74             json.dump(data,f,sort_keys="True",indent=4)
75 from gestor import Gestor
76 gestor=Gestor()
77
78 class Agente():
79     def __init__(self,id_agente):
80         self.id_agente=id_agente
81         self.hostname, self.ip, self.conexion, self.comunidad, self.version, self.
82         puerto = obtenerInfoPrincipalAgente(id_agente)
83         self.nombre_so, self.logo_so, self.version_so, self.num_interfaces, self.
84         tiempo_reinicio, self.ubicacion, self.contacto=obtenerInfoAgente(id_agente)

```

Este archivo, permite el registro de un nuevo agente. agregar_agente.py:

```

1 from tkinter import *
2 from tkinter import messagebox
3 import json
4 import os.path
5 #from main import Main
6
7 class AgregarAgente():

```

```

8 """docstring for Agente"""
9 def __init__(self):
10     #Creamos la ventana y sus medidas
11     self.data={}
12     self.top=Tk()
13     self.top.geometry("500x500")
14     self.top.resizable(0,0)
15     #Definimos los elementos de la misma y a cada uno los acomodamos en el grid
16     self.b = Button(self.top, text="Regresar", command=self.cancelar)
17     self.b.grid(row=10, column=2, sticky=W+E+N+S)
18     self.b = Button(self.top, text="Agregar agente", command=self.crearAgente)
19     self.b.grid(row=10, column=1, sticky=W+E+N+S)
20
21     Label(self.top, text="Hostname").grid(row=1, sticky=W)
22     self.hostname=Entry(self.top)
23     self.hostname.grid(row=1, column=1)
24     Label(self.top, text="IP").grid(row=2, sticky=W)
25     self.ip=Entry(self.top)
26     self.ip.grid(row=2, column=1)
27     Label(self.top, text="Versión SNMP").grid(row=3, sticky=W)
28     self.version=Entry(self.top)
29     self.version.grid(row=3, column=1)
30     Label(self.top, text="Puerto").grid(row=4, sticky=W)
31     self.puerto=Entry(self.top)
32     self.puerto.insert(0, '161')
33     self.puerto.grid(row=4, column=1)
34     Label(self.top, text="Comunidad").grid(row=5, sticky=W)
35     self.comunidad=Entry(self.top)
36     self.comunidad.grid(row=5, column=1)
37     #Final
38     self.top.mainloop()
39 #Regresar a la pantalla anterior
40 def cancelar(self):
41     self.top.destroy()
42     #El import lo puse aquí por que si no marca un error extraño
43     from gestor import Gestor
44     self.gestor=Gestor()
45 #Crear un agente nuevo
46 def crearAgente(self):
47     agente_id=1
48     with open('agentes.json','r+') as f:
49         if os.path.getsize('agentes.json') > 0:
50             print("No estoy vacío")
51             self.data = json.load(f)
52             f.seek(0)
53             llaves=list(self.data.keys())
54             agente_id=int(llaves[len(llaves)-1])+1
55         else:
56             print("estoy vacío")
57         nuevo_agente={
58             'hostname': self.hostname.get(),
59             'ip': self.ip.get(),
60             'version': self.version.get(),
61             'puerto': self.puerto.get(),
62             'comunidad': self.comunidad.get()
63         }
64         self.data[str(agente_id)]=nuevo_agente
65         json.dump(self.data,f,sort_keys="True",indent=4)
66         messagebox.showinfo("Éxito", "Agente registrado exitosamente")

```

Permite monitorear a un agente y además realiza la consultas correspondientes para

mostrar las gráficas. monitor.py:

```

1 from tkinter import *
2 from PIL import Image, ImageTk
3 import threading, time, calendar
4 from agente import Agente
5 from SNMP import crearRRDDos, consultaSNMP
6 import rrdtool
7 class Monitor(threading.Thread):
8     """docstring for Agente"""
9     def __init__(self, id_agente):
10         threading.Thread.__init__(self)
11         self.agente=Agente(id_agente)
12         self.data={}
13         self.top=Toplevel()
14         self.top.geometry("1500x800")
15         self.top.resizable(0,0)
16         self.b = Button(self.top, text="Salir", command=self.salir)
17         self.b.grid(row=1, column=2, sticky=W+E+N+S)
18         self.frame=Frame(self.top, width = 600, height = 200, relief = 'raised',
borderwidth=2)
19         self.frame.grid(row = 1, column = 0, columnspan=1, sticky=W+E+N+S)
20         self.continuar=True
21         self.image=[""]*5
22         self.photo=[""]*5
23         self.label=[""]*5
24         self.mostrarInfo()
25         self.crearRRDs()
26     def run(self):
27         while self.continuar:
28             #Interfaces
29             self.graficarRRD('IF-MIB', 'ifInOctets', 'ifOutOctets', "interfaz", "Tráfico
de red en interfaces ('ifInOctets,ifOutOctets')",2)
30             #ICMP ->Deprecated
31             self.graficarRRD('IP-MIB', 'icmpInMsgs', 'icmpOutMsgs', "icmp", "Tráfico ICMP
('icmpInMsgs', 'icmpOutMsgs')",0)
32             #TCP
33             self.graficarRRD('TCP-MIB', 'tcpInSegs', 'tcpOutSegs', "tcp", "Tráfico
segmentos TCP ('tcpInSegs', 'tcpOutSegs')",0)
34             #UDP
35             self.graficarRRD('UDP-MIB', 'udpInDatagrams', 'udpOutDatagrams', "udp", "
Tráfico UDP Datagramas ('udpInDatagrams', 'udpOutDatagrams')",0)
36             #SNMP
37             self.graficarRRD('SNMPv2-MIB', 'snmpInPkts', 'snmpOutPkts', "snmp", "Tráfico
SNMP ('snmpInPkts', 'snmpOutPkts')",0)
38
39             #Ahora actualizamos las imagenes
40             self.actualizarImagen(0, self.agente.hostname+"_"+self.agente.id_agente+"
_interfaz.png",3,2)
41
42             self.actualizarImagen(0, self.agente.hostname+"_"+self.agente.id_agente+"
_icmp.png",4,2)
43
44             self.actualizarImagen(0, self.agente.hostname+"_"+self.agente.id_agente+"
_tcp.png",3,0)
45
46             self.actualizarImagen(0, self.agente.hostname+"_"+self.agente.id_agente+"
_udp.png",3,3)
47
48             self.actualizarImagen(0, self.agente.hostname+"_"+self.agente.id_agente+"
_snmp.png",4,3)
49

```

```

50     time.sleep(5)
51     self.top.destroy()
52     def salir(self):
53         self.continuar=False
54     def mostrarInfo(self):
55         imagen=Image.open(self.agente.logo_so).resize((60,60),Image.ANTIALIAS)
56         photo1=ImageTk.PhotoImage(imagen)
57         label1=Label(self.frame,image=photo1)
58         label1.image=photo1
59         label1.grid(row=1, column=1, sticky=W+E+N+S)
60         Label(self.frame, text="ID agente: "+self.agente.id_agente).grid(row=2,
column=1, sticky=W)
61         Label(self.frame, text="Hostname: "+self.agente.hostname).grid(row=3, column
=1, sticky=W)
62         Label(self.frame, text="IP: "+self.agente.ip).grid(row=4, column=1, sticky=W
)
63         Label(self.frame, text="Comunidad: "+self.agente.comunidad).grid(row=5,
column=1, sticky=W)
64         Label(self.frame, text="Version: "+str((int(self.agente.version))+1)).grid(
row=6, column=1, sticky=W)
65         Label(self.frame, text="Puerto: "+self.agente.puerto).grid(row=7, column=1,
sticky=W)
66         Label(self.frame, text="Nombre SO: "+self.agente.nombre_so).grid(row=2,
column=2, sticky=W)
67         Label(self.frame, text="Versión SO: "+self.agente.version_so).grid(row=3,
column=2, sticky=W)
68         Label(self.frame, text="Número de interfaces: "+self.agente.num_interfaces).
grid(row=4, column=2, sticky=W)
69         Label(self.frame, text="Ubicación física: "+self.agente.ubicacion).grid(row
=5, column=2, sticky=W)
70         Label(self.frame, text="Contacto: "+self.agente.contacto).grid(row=6, column
=2, sticky=W)
71     def crearRRDs(self):
72         #Para tr{afico de red:
73         crearRRDDos(self.agente.hostname+"_"+self.agente.id_agente+"_interfaz.rrd","
N","1","60","1","1","100","100")
74         #Para tr{afico de IP
75         crearRRDDos(self.agente.hostname+"_"+self.agente.id_agente+"_icmp.rrd","N","
1","60","1","1","100","100")
76         #Para tr{afico de TCP segmentos
77         crearRRDDos(self.agente.hostname+"_"+self.agente.id_agente+"_tcp.rrd","N","1
","60","1","1","100","100")
78         #Para tr{afico de Datagramas UDP
79         crearRRDDos(self.agente.hostname+"_"+self.agente.id_agente+"_udp.rrd","N","1
","60","1","1","100","100")
80         #Para tr{afico de SNMP
81         crearRRDDos(self.agente.hostname+"_"+self.agente.id_agente+"_snmp.rrd","N","
1","60","1","1","100","100")
82     def graficarRRD(self, grupo, oid1, oid2, archivo, header, numero):
83         tiempo_actual=calendar.timegm(time.gmtime())
84         ultimo=rrdtool.last(self.agente.hostname+"_"+self.agente.id_agente+"_"+
archivo+".rrd")
85         #Graficar
86         consulta=consultaSNMP(self.agente.comunidad, self.agente.ip, int(self.agente.
version), numero, grupo, oid1, oid2, int(self.agente.puerto))
87         if consulta[0]!="" or consulta[1]!="":
88             valor = "N:" + str(consulta[0]) + ':' + str(consulta[1])
89             #print (valor)
90             rrdtool.update(self.agente.hostname+"_"+self.agente.id_agente+"_"+archivo+
".rrd", valor)
91             rrdtool.dump(self.agente.hostname+"_"+self.agente.id_agente+"_"+archivo+"

```

```

128         rrd", self.agente.hostname+"_"+self.agente.id_agente+"_"+archivo+".xml")
129         ret = rrdtool.graph(self.agente.hostname+"_"+self.agente.id_agente+"_"+
130             archivo+".png",
131             "--start", str(ultimo-100),
132             "--end", "+100",
133             "--vertical-label=Bytes/s",
134             "--title="+header,
135             "DEF:in="+self.agente.hostname+"_"+self.agente.
136             id_agente+"_"+archivo+".rrd:in:AVERAGE",
137             "DEF:out="+self.agente.hostname+"_"+self.agente.
138             id_agente+"_"+archivo+".rrd:out:AVERAGE",
139             "LINE1:in#00FF00:In traffic",
140             "LINE1:out#0000FF:Out traffic")
141     else:
142         pass
143 def actualizarImagen(self, index, archivo, fila, columna):
144     self.image[index]=Image.open(archivo).resize((400,200),Image.ANTIALIAS)
145     self.photo[index]=ImageTk.PhotoImage(self.image[index])
146     self.label[index]=Label(self.top, image=self.photo[index])
147     self.label[index].image=self.photo[index]
148     self.label[index].grid(row=fila, column=columna, sticky=W)

```

Archivo en donde se encuentran todas las consultas necesarias para el correspondiente monitoreo. SNMP.py:

```

1 from pysnmp.hlapi import *
2 import rrdtool
3 import time
4 #Funcion para obtener las interfaces
5 def getInterfaces(comunidad, host, version, puerto):
6     resultado=""
7     errorIndication, errorStatus, errorIndex, varBinds = next(getCmd(SnmpEngine(),
8         CommunityData(comunidad, mpModel=version), UdpTransportTarget((host, puerto)),
9         ContextData(),
10         ObjectType(ObjectIdentity('IF-MIB', 'ifNumber', 0).addAsn1MibSource('file:///usr/share/snmp', 'http://mibs.snmplabs.com/asn1/@mib@'))))
11     if errorIndication:
12         print(errorIndication)
13     elif errorStatus:
14         print('%s at %s' % (errorStatus.prettyPrint(),
15             errorIndex and varBinds[int(errorIndex)-1][0] or '?'))
16     else:
17         for varBind in varBinds:
18             VarB=('='.join([x.prettyPrint() for x in varBind]))
19             resultado=VarB.partition('=')[2]
20     return resultado
21 #Funcion para obtener el estatus de una interfaz
22 def getStatus(comunidad, host, version, interfaz, puerto):
23     errorIndication, errorStatus, errorIndex, varBinds = next(getCmd(SnmpEngine(),
24         CommunityData(comunidad, mpModel=version), UdpTransportTarget((host, puerto)),
25         ContextData(),
26         ObjectType(ObjectIdentity('IF-MIB', 'ifAdminStatus', interfaz).
27             addAsn1MibSource('file:///usr/share/snmp', 'http://mibs.snmplabs.com/asn1/@mib@'))))
28     if errorIndication:
29         print(errorIndication)
30     elif errorStatus:
31         print('%s at %s' % (errorStatus.prettyPrint(),

```

```

30         errorIndex and varBinds[int(errorIndex) - 1][0] or '?'
31     ))
32     else:
33         for varBind in varBinds:
34             VarB=(' = '.join([x.prettyPrint() for x in varBind]))
35             resultado=VarB.partition(' = ')[2]
36         return resultado
37 #Funcion para obtener informacion del estado del dispositivo
38 def getInfo(comunidad,host,version,puerto):
39     resultado=[]
40     errorIndication, errorStatus, errorIndex, varBinds = next(getCmd(SnmpEngine(),
41     CommunityData(comunidad,mpModel=version),UdpTransportTarget((host, puerto)),
42     ContextData(),
43     ObjectType(ObjectIdentity('SNMPv2-MIB','sysName',0).addAsn1MibSource('file
44     :///usr/share/snmp','http://mibs.snmplabs.com/asn1/@mib@')),
45     ObjectType(ObjectIdentity('SNMPv2-MIB','sysDescr',0).addAsn1MibSource('file
46     :///usr/share/snmp','http://mibs.snmplabs.com/asn1/@mib@')),
47     ObjectType(ObjectIdentity('IF-MIB','ifNumber',0).addAsn1MibSource('file :///
48     usr/share/snmp','http://mibs.snmplabs.com/asn1/@mib@')),
49     ObjectType(ObjectIdentity('SNMPv2-MIB','sysUpTime',0).addAsn1MibSource('file
50     :///usr/share/snmp','http://mibs.snmplabs.com/asn1/@mib@')),
51     ObjectType(ObjectIdentity('SNMPv2-MIB','sysLocation',0).addAsn1MibSource('
52     file :///usr/share/snmp','http://mibs.snmplabs.com/asn1/@mib@')),
53     ObjectType(ObjectIdentity('SNMPv2-MIB','sysContact',0).addAsn1MibSource('
54     file :///usr/share/snmp','http://mibs.snmplabs.com/asn1/@mib@'))))
55     if errorIndication:
56         print(errorIndication)
57     elif errorStatus:
58         print('%s at %s' %(errorStatus.prettyPrint(),
59         errorIndex and varBinds[int(errorIndex) - 1][0] or '?'
60         ))
61     else:
62         for varBind in varBinds:
63             VarB=(' = '.join([x.prettyPrint() for x in varBind]))
64             resultado.append(VarB.partition(' = ')[2])
65         return resultado
66
67 def getInfo2(comunidad,host,version,puerto,oids):
68     resultado=[]
69     for element in oids:
70         errorIndication, errorStatus, errorIndex, varBinds = next(getCmd(SnmpEngine
71         (),CommunityData(comunidad,mpModel=version),UdpTransportTarget((host, puerto
72         )),ContextData(),ObjectType(ObjectIdentity(element))))
73         if errorIndication:
74             print(errorIndication)
75         elif errorStatus:
76             print('%s at %s' %(errorStatus.prettyPrint(),
77             errorIndex and varBinds[int(errorIndex) - 1][0] or '
78             ?'))
79         else:
80             for varBind in varBinds:
81                 VarB=(' = '.join([x.prettyPrint() for x in varBind]))
82                 resultado.append(VarB.partition(' = ')[2])
83             return resultado
84
85 #Funcion para hacer consultas version facil (entrada y salida)
86 def consultaSNMP(comunidad,host,version,interfaz,grupo,objeto1,objeto2,puerto):
87     resultado=[]

```

```

78 errorIndication , errorStatus , errorIndex , varBinds = next(getCmd(SnmpEngine() ,
    CommunityData(comunidad , mpModel=version) , UdpTransportTarget((host , puerto)) ,
    ContextData() ,
79     ObjectType(ObjectIdentity(grupo , objeto1 , interfaz) . addAsn1MibSource('file :///
    usr / share / snmp' , 'http :// mibs . snmplabs . com / asn1 / @mib@')) ,
80     ObjectType(ObjectIdentity(grupo , objeto2 , interfaz) . addAsn1MibSource('file :///
    usr / share / snmp' , 'http :// mibs . snmplabs . com / asn1 / @mib@')))
81
82 if errorIndication :
83     print(errorIndication)
84     resultado . append('')
85     resultado . append('')
86 elif errorStatus :
87     print('%s at %s' % (errorStatus . prettyPrint() ,
88         errorIndex and varBinds[int(errorIndex) - 1][0] or '?')
89 ))
90 else :
91     for varBind in varBinds :
92         VarB=(' = ' . join([x . prettyPrint() for x in varBind]))
93         resultado . append(VarB . partition(' = ')[2])
94     return resultado
95
96 #Funcion para hacer consultas a un solo objeto con referencias al nombre
97 def consultav2SNMP(comunidad , host , version , interfaz , grupo , objeto , puerto) :
98     resultado=""
99     errorIndication , errorStatus , errorIndex , varBinds = next(getCmd(SnmpEngine() ,
    CommunityData(comunidad , mpModel=version) , UdpTransportTarget((host , puerto)) ,
    ContextData() ,
100     ObjectType(ObjectIdentity(grupo , objeto , interfaz) . addAsn1MibSource('file :///
    usr / share / snmp' , 'http :// mibs . snmplabs . com / asn1 / @mib@')))
101
102 if errorIndication :
103     print(errorIndication)
104 elif errorStatus :
105     print('%s at %s' % (errorStatus . prettyPrint() ,
106         errorIndex and varBinds[int(errorIndex) - 1][0] or '?')
107 ))
108 else :
109     for varBind in varBinds :
110         VarB=(' = ' . join([x . prettyPrint() for x in varBind]))
111         resultado=VarB . partition(' = ')[2]
112     return resultado
113
114 #Funcion para hacer consultas de un objeto con el OID
115 def consultav3SNMP(comunidad , host , version , oid , puerto) :
116     resultado=""
117     errorIndication , errorStatus , errorIndex , varBinds = next(getCmd(SnmpEngine() ,
    CommunityData(comunidad , mpModel=version) , UdpTransportTarget((host , puerto)) ,
    ContextData() ,
118     ObjectType(ObjectIdentity(oid))))
119
120 if errorIndication :
121     print(errorIndication)
122 elif errorStatus :
123     print('%s at %s' % (errorStatus . prettyPrint() ,
124         errorIndex and varBinds[int(errorIndex) - 1][0] or '?')
125 ))
126 else :
127     for varBind in varBinds :
128         VarB=(' = ' . join([x . prettyPrint() for x in varBind]))
129         resultado=VarB . partition(' = ')[2]

```



```

127     return resultado
128
129 #Funcion para hacer consultas de dos objetos con OID
130 def consultav4SNMP(comunidad, host, version, oid1, oid2, puerto):
131     resultado = []
132     errorIndication, errorStatus, errorIndex, varBinds = next(getCmd(SnmpEngine(),
133                               CommunityData(comunidad, mpModel=version), UdpTransportTarget((host, puerto)),
134                               ContextData(),
135                               ObjectType(ObjectIdentity(oid1)),
136                               ObjectType(ObjectIdentity(oid2))))
137
138     if errorIndication:
139         print(errorIndication)
140     elif errorStatus:
141         print('%s at %s' % (errorStatus.prettyPrint(),
142                               errorIndex and varBinds[int(errorIndex) - 1][0] or '?'))
143     else:
144         for varBind in varBinds:
145             VarB = (' = ').join([x.prettyPrint() for x in varBind])
146             resultado.append(VarB.partition(' = ')[2])
147     return resultado
148
149 #Funcion para crear el archivo .rrd con un valor
150 def crearRRDUno(nombre, inicio, step, tiempo, steps1, steps2, row1, row2):
151     ret = rrdtool.create(nombre, '--start', inicio, '--step', step,
152                          "DS: valor1:COUNTER:" + tiempo + ":U:U",
153                          "RRA:AVERAGE:0.5:" + steps1 + ":" + row1,
154                          "RRA:AVERAGE:0.5:" + steps2 + ":" + row2)
155     if ret:
156         print(rrdtool.error())
157
158 #Funcion para crear el archivo .rrd con dos valores
159 def crearRRDDos(nombre, inicio, step, tiempo, steps1, steps2, row1, row2):
160     ret = rrdtool.create(nombre, '--start', inicio, '--step', step,
161                          "DS: in:COUNTER:" + tiempo + ":U:U",
162                          "DS: out:COUNTER:" + tiempo + ":U:U",
163                          "RRA:AVERAGE:0.5:" + steps1 + ":" + row1,
164                          "RRA:AVERAGE:0.5:" + steps2 + ":" + row2)
165     if ret:
166         print(rrdtool.error())

```

Ejemplo de como se almacena la información sobre los agentes, para la persistencia. agentes.json:

```

1 {
2     "1": {
3         "comunidad": "comunidadEquipo8_grupo4cm1",
4         "hostname": "windows10",
5         "ip": "10.100.70.251",
6         "puerto": "161",
7         "version": "1"
8     },
9     "2": {
10        "comunidad": "comunidadEquipo8_grupo4cm1",
11        "hostname": "ubuntu",
12        "ip": "10.100.70.6",
13        "puerto": "161",
14        "version": "1"
15    }
16 }

```


Capítulo 5

Conclusiones

Hernández Pineda Miguel Angel:

Parte importante del manejo de redes, es la administración de las mismas, y como vimos a lo largo de esta práctica, el protocolo usado para hacer esto se llama SNMP, el cual facilita el intercambio de paquetes de información para la administración de la red, donde alguno de los dispositivos debe cumplir con la tarea de gestión, mientras que los demás funcionan como agentes. Parte importante de entender el concepto de SNMP es verificar su funcionamiento, para ello se nos pidió desarrollar una aplicación que mediante el uso del protocolo SNMP obtuviera información de la MIB de los dispositivos conectados, para ello fue necesario que al hacer las peticiones el usuario definiera ciertos parámetros, como la IP del dispositivo a monitorear, el puerto de conexión, el o los OID de la información a consultar así como la comunidad a la que pertenece el dispositivo; comprendido esto, podemos decir que el protocolo SNMP es una herramienta muy importante en la realización de esta tarea, pues gracias a el se puede consultar mucha información de los dispositivos conectados a una red así como definir ciertos parámetros que nosotros necesitemos.

Monroy Martos Elioth:

La administración de las redes es una parte importante de nuestro desarrollo como ingenieros, esta práctica me ha servido para aprender sobre como es que se realiza el monitoreo de las mismas a través de software especializado como Observium. Al tratar de hacer un “clon” de Observium, he podido comprender un poco mejor, como es que se realiza el monitoreo de una red, desde el que información necesito conocer sobre un agente para poder monitorearlo, hasta que herramientas puedo usar para llevar a cabo mi labor. Además, comprendí mejor como es que se compone una red en un sistema operativo, y como es que existe una especificación de objetos los cuales tienen como objetivo llevar un control de distintos aspectos de la comunicación que tiene un agente con la red (los objetos MIB). Los cuales forman parte del protocolo SNMP, sobre el cual está basado el curso, y es una herramienta muy útil para poder trabajar. En este caso, estuvimos trabajando tanto con snmp mediante consola como con una biblioteca para el lenguaje Python.

Zuñiga Hernández Carlos:

En la práctica se obtuvieron resultados tanto esperados como inesperados. Se pudo notar que obtener unos u otros radica significativamente en la buena configuración de los agentes y el gestor. Si algo de esto está erróneamente configurado, es muy probable que nada resulte bien. Así, notarlo sirve para dar importancia a la manera en que se configuran las herramientas y evitar futuros inconvenientes. También es útil para conocer las ubicaciones de los archivos que se suelen ocupar para cambiar esta configuración, permitiendo cada vez más entender como funciona el protocolo SNMP y explotarlo de acuerdo a nuestras necesidades.

Por otro lado, los comandos SNMP fueron una gran ayuda para obtener información de la MIB de los agentes. Esto fue clave para el desarrollo de la práctica.

Referencias Bibliográficas

- [1] SNMP library for Python. Febrero 20, 2019, de - Sitio web:
<http://snmplabs.com/pysnmp/index.html>
- [2] PySNMP architecture. Febrero 20, 2019, de - Sitio web:
<http://snmplabs.com/pysnmp/docs/pysnmp-architecture.html>
- [3] Library reference. Febrero 20, 2019, de - Sitio web:
<http://snmplabs.com/pysnmp/docs/api-reference.html>
- [4] K. McCloghrie. (March 1991). Management Information Base for Network Management of TCP/IP-based internets: MIB-II. Febrero 20, 2019, de IETF
Sitio web: <https://www.ietf.org/rfc/rfc1213.txt>