Instituto Politecnico Nacional

Escuela Superior de Computo

Practica 5.1 Monitoreo de una interfaz

Alumno: Javier Martinez Carranza

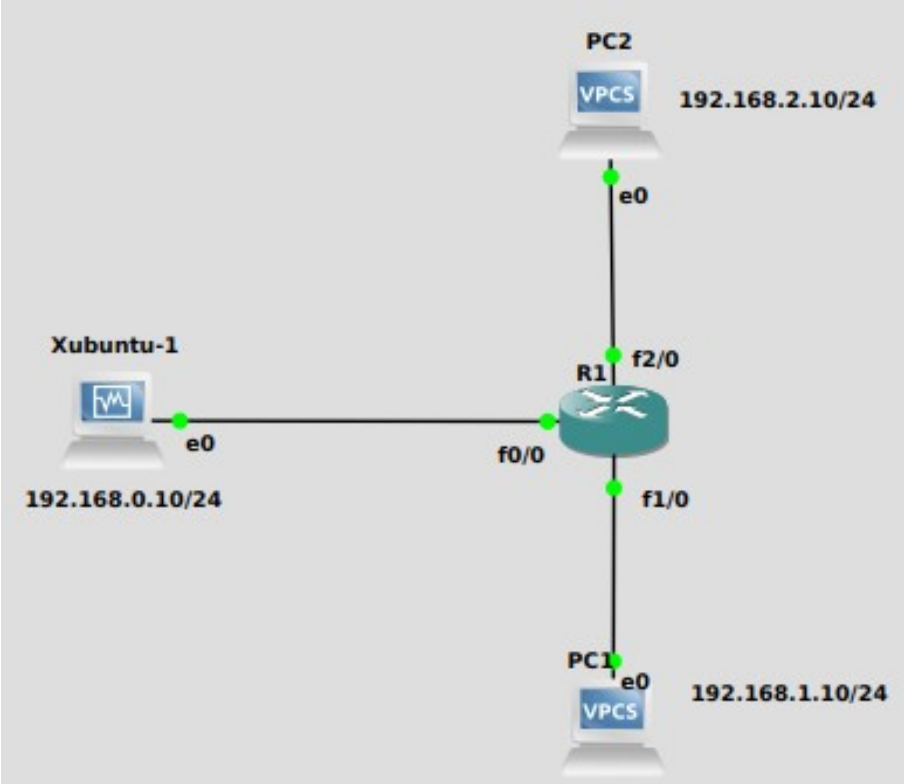
Grupo: 4CM11

Materia: Administracion de Servicios en Red

Profesor: Ricardo Martinez Rosales

Topologia que sera usada en esta practica

Configuraremos de forma basica la siguiente topologia, usaremos dos maquinas virtuales y una maquina virtual desde donde ejecutaremos nuestra aplicación.



Configuraremos a las computadoras con sus respectivas ips y el router de la siguiente manera:

```
R1#show ip interface br
R1#show ip interface brief
Interface                  IP-Address      OK? Method Status                Protocol
FastEthernet0/0            192.168.0.1     YES manual up                    up
FastEthernet1/0            192.168.1.1     YES manual up                    up
FastEthernet2/0            192.168.2.1     YES manual up                    up
FastEthernet3/0            unassigned      YES unset  administratively down down
FastEthernet4/0            unassigned      YES unset  administratively down down
R1#
```

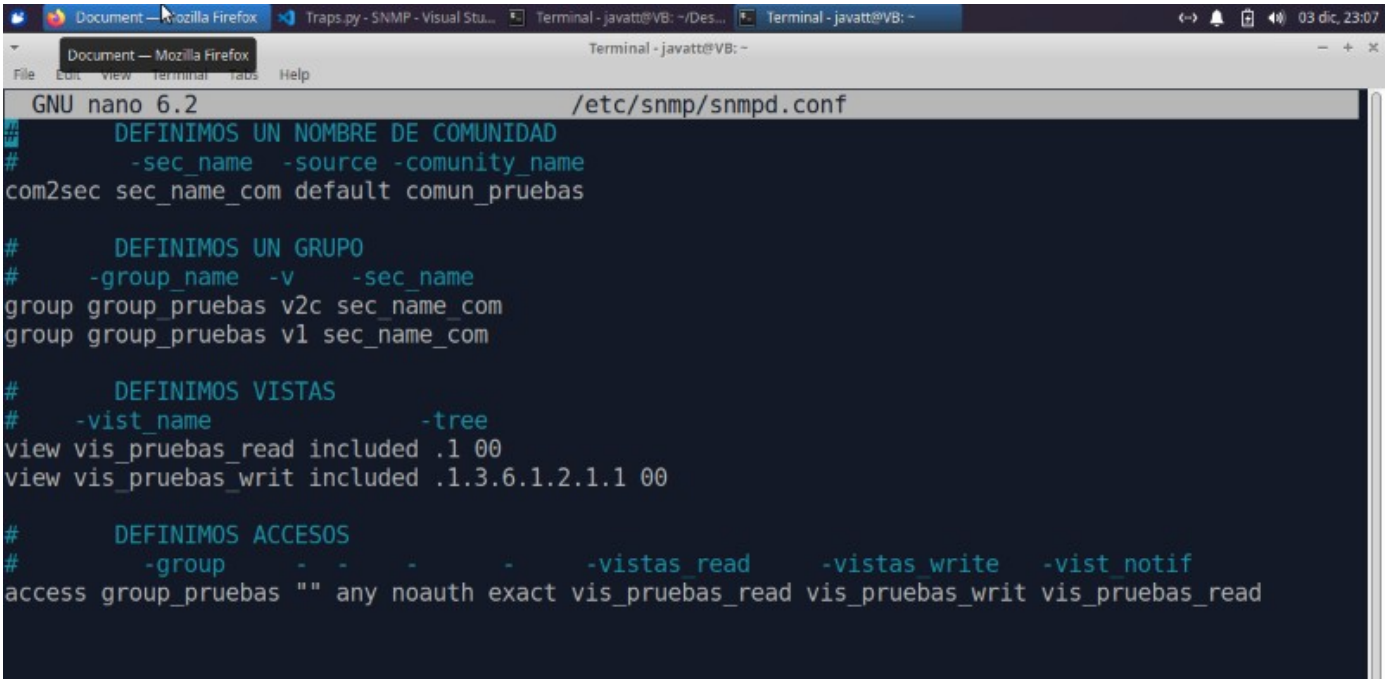
ya con las ips de gateway en cada interfaz, pasaremos a configurar la parte de SNMP:

```
R1#show running-config | i snmp
snmp-server community comun_pruebas RW
snmp-server enable traps snmp linkdown linkup
snmp-server host 192.168.0.10 version 2c comun_pruebas
R1#
```

Como podemos observar los comandos anteriores ya se encuentran configurados en el Router 1, de esta forma agregamos una comunidad “comun_pruebas” en modo de escritura y lectura, habilitamos las traps de linkdown y linkup, asi cada vez que una interfaz se de de baja o de alta, nos alertara por medio de la trap.

Finalmente anadimos un host, el cual sera nuestra maquina virtual, usaremos la version 2c con la comunidad definida previamente.

Esta configuracion debe ir tambien configurada en el archivo “snmpd.config” que tendremos en la maquina virtual:

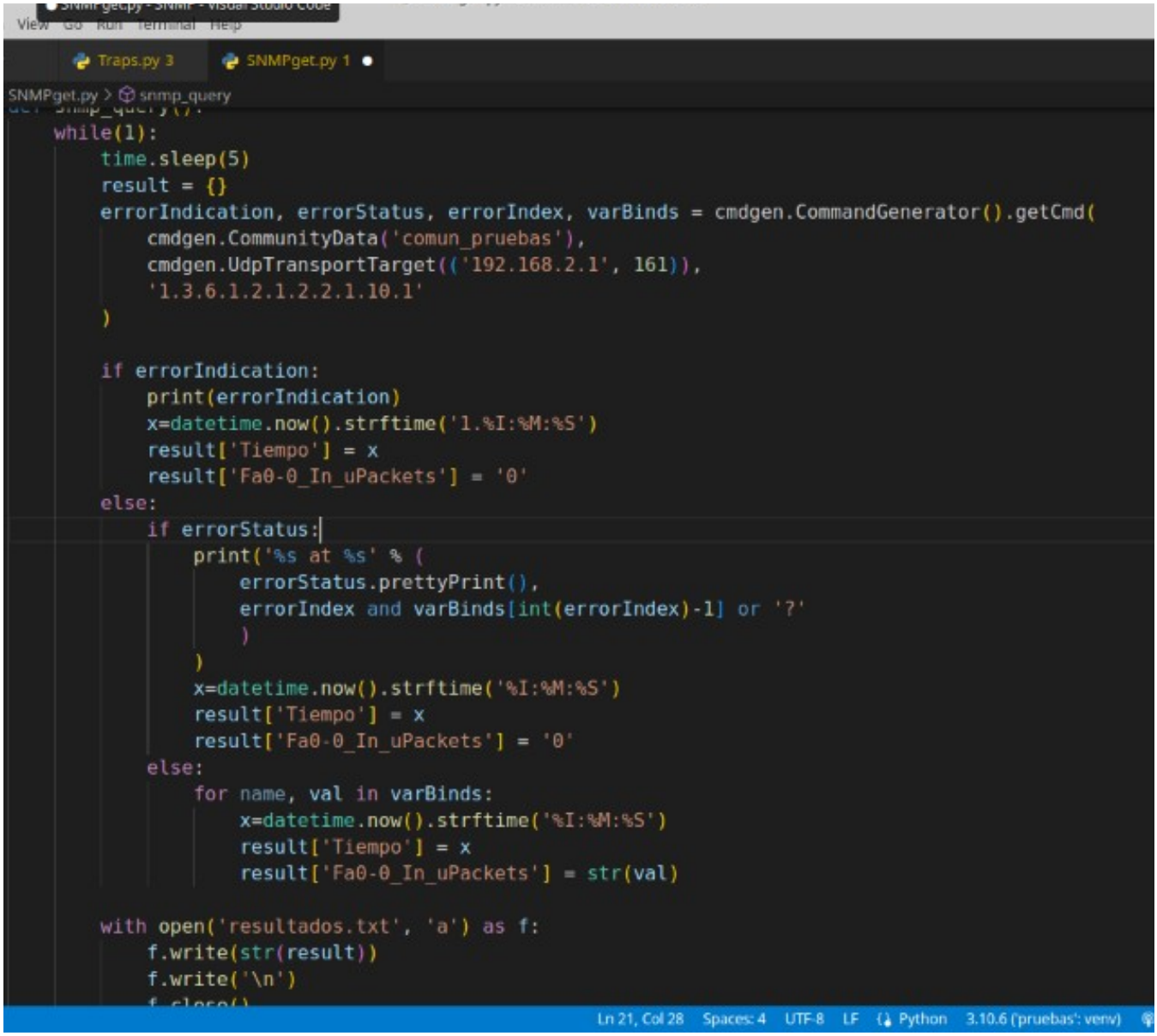


en esta configuracion definimos una comunidad “comun_pruebas”, un grupo “group_pruebas” con version 2c y v1.

En el apartado de la vista tenemos “vis_prueb_read” que incluira todo el arbol mib y “vis_prueb_writ” que solo podra modificar o escribir en la rama de system.

En la lista de acceso agregamos estas vistas, en la vista de notificaciones agregamos la “vis_prueb_read” para poder trabajar en dado coso con esas ramas.

Una vez terminemos con las configuraciones basicas, podemos comanzar nuestro programa, primero crearemos un codigo que nos permita analizar los paquetes de entrada de una interfaz, usaremos la rama ".1.3.6.1.2.1.2.2.1.10.1" que es la de "In_uPackets" usando la comunidad comun_pruebas.



Como podemos ver, hacemos una peticion get a la interfaz f2/0, donde analizaremos los paquetes de entrada, una vez obtenidos, obtendremos la hora de consulta y los paquetes en ese instante para despues guardarlos en un archivo.

Este proceso lo haremos cada 5 segundos y lo pondremos en un hilo para que este consultando sin necesidad de ordenar o ejecutar el codigo varias veces.

Con las traps podremos observar que tiempo estuvo apagada o prendida la interfaz, para esto presentamos el siguiente codigo:

```python
from pysnmp.entity import engine, config
from pysnmp.carrier.asyncore.dgram import udp
from pysnmp.entity.rfc3413 import ntfrcv

def traps():
    print("Analizando 192.168.2.1 en el puereto 162")
    snmpEngine = engine.SnmpEngine()

    config.addTransport(
        snmpEngine, udp.domainName + (1,),
        udp.UdpTransport().openServerMode(('192.168.0.10', 162))
    )

    config.addV1System(snmpEngine, 'vis_pruebas_read', 'comun_pruebas')

    def cbFun(snmpEngine, stateReference, contextEngineId, contextName, varBinds, cbCtx):
        valor = str((varBinds.pop())[-1])
        with open('status.txt', 'w') as f:
            f.write(valor)
            f.close()
        print(valor)

    ntfrcv.NotificationReceiver(snmpEngine, cbFun)
    snmpEngine.transportDispatcher.jobStarted(1)

    try:
        snmpEngine.transportDispatcher.runDispatcher()
    except:
        snmpEngine.transportDispatcher.closeDispatcher()
        raise

traps()
```

Creamos un codigo que analice el host en busca de traps, una vez que recibe una trap, checa que tipo de trap es y el estado de esta; con esta informacion , guardamos ese estado en un archivo, esto estara activo como un demonio para la escucha de traps en el host.

Aunado a esto, tenemos un programa que se encarga de leer este archivo que modifica el codigo de la trap.

Este codigo estara en un hilo, siempre leyendo el mismo archivo y consultandolo, guardando la hora de consulta y el estado, lo guardado lo usaremos despues para poder graficar. Cabe aclarar que escanea cada 3 segundos y cada 40 muestras se formatea el archivo.

```python
def imprime():
    i = 0
    while(1):
        time.sleep(3)
        with open('status.txt', 'r') as f:
            estado = f.read()
            f.close()

        result = {}
        x=datetime.now().strftime('%I:%M:%S')
        result['Tiempo'] = x
        if estado == 'up':
            result['estado'] = '1'
        else:
            result['estado'] = '0'

        if i == 40:
            with open('estado.txt', 'w') as f:
                f.write(str(result))
                f.write('\n')
                f.close()
            i = 0
        else:
            with open('estado.txt', 'a') as f:
                f.write(str(result))
                f.write('\n')
                f.close()
            i= i+1

if __name__ == '__main__':
    hilo = threading.Thread(target=imprime)
    hilo.start()
```

Para graficar usaremos la librería pygal, con esta consultaremos los archivos generados por los codigos anteriores para agregar los datos en la grafica, de la siguiente forma:

app > Graficas.py > graficar_estado

```python
import pygal

def graficar_paquetes():
    x_time = []
    in_packets = []

    with open('resultados.txt', 'r') as f:
        for line in f.readlines():
            line = eval(line)
            x_time.append(line['Tiempo'])
            in_packets.append(float(line['Fa0-0_In_uPackets']))

    line_chart = pygal.Line(x_label_rotation=25)
    line_chart.title = "R1 Fa2/0"
    line_chart.x_labels = x_time
    line_chart.add('Paq. entrada', in_packets)
    line_chart.render_to_file('/home/javatt/Desktop/SNMP/app/static/grafica_paquetes.svg')

def graficar_estado():
    x_time = []
    estado = []

    with open('estado.txt', 'r') as f:
        for line in f.readlines():
            line = eval(line)
            x_time.append(line['Tiempo'])
            estado.append(float(line['estado']))

    line_chart = pygal.StackedLine(height=300,fill=True,x_label_rotation=25)
    line_chart.title = "R1 Fa2/0"
    line_chart.x_labels = x_time
    line_chart.add('Conexion de la interfaz', estado)
    line_chart.render_to_file('/home/javatt/Desktop/SNMP/app/static/grafica_estado.svg')
```

Cada funcion graficara una grafica, consultando diferentes archivos y plantando en una imagen svg cada una.

Ya con los codigos armados, usaremos flask para juntar estos en un servicio REST.

```python
from flask import Flask , render_template
import SNMPget,Graficas,os,Lectura,threading

comand= " python3 /home/javatt/Desktop/SNMP/app/Traps.py "
app= Flask(__name__)

@app.route("/")
def hello():
    hilo1 = threading.Thread(target=Lectura.imprime)
    hilo2 = threading.Thread(target=SNMPget.snmp_query)
    hilo1.start()
    hilo2.start()
    os.popen("sudo -S %s"%(comand), 'w').write("1532")
    return "Servidor corriendo"

@app.route("/graficas")
def grafica():
    Graficas.graficar_paquetes()
    Graficas.graficar_estado()
    return render_template("index.html")


if __name__ == '__main__':
    app.run()
```

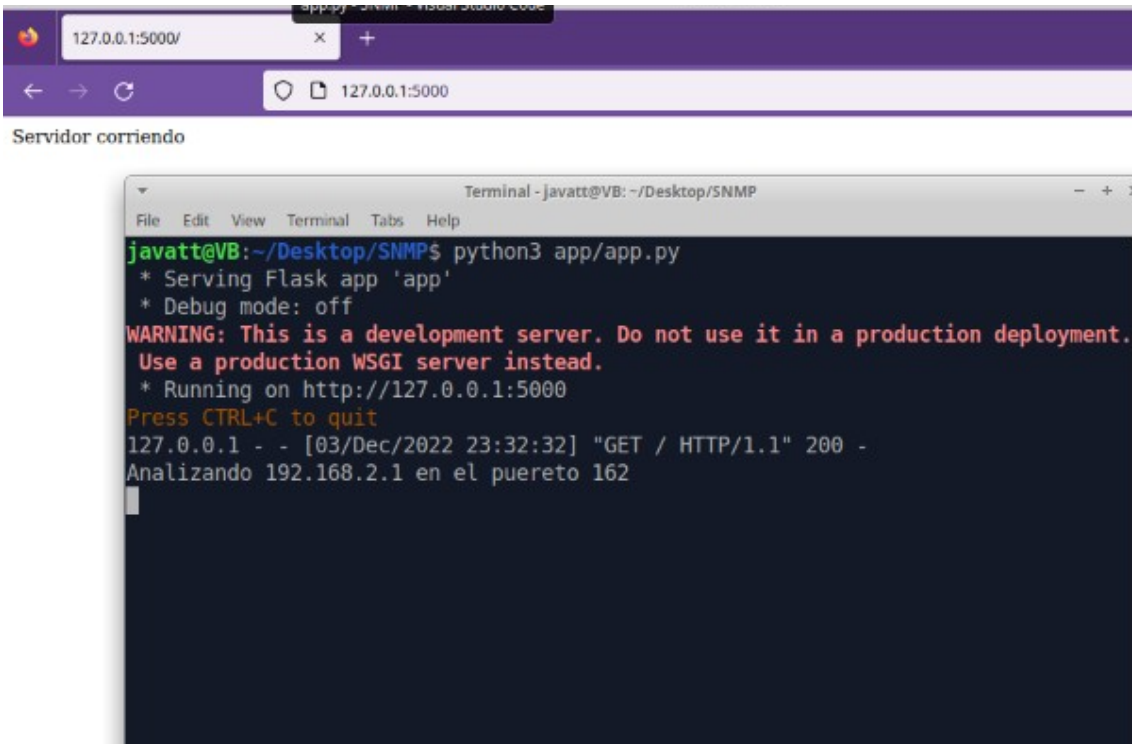
Tenemos dos rutas

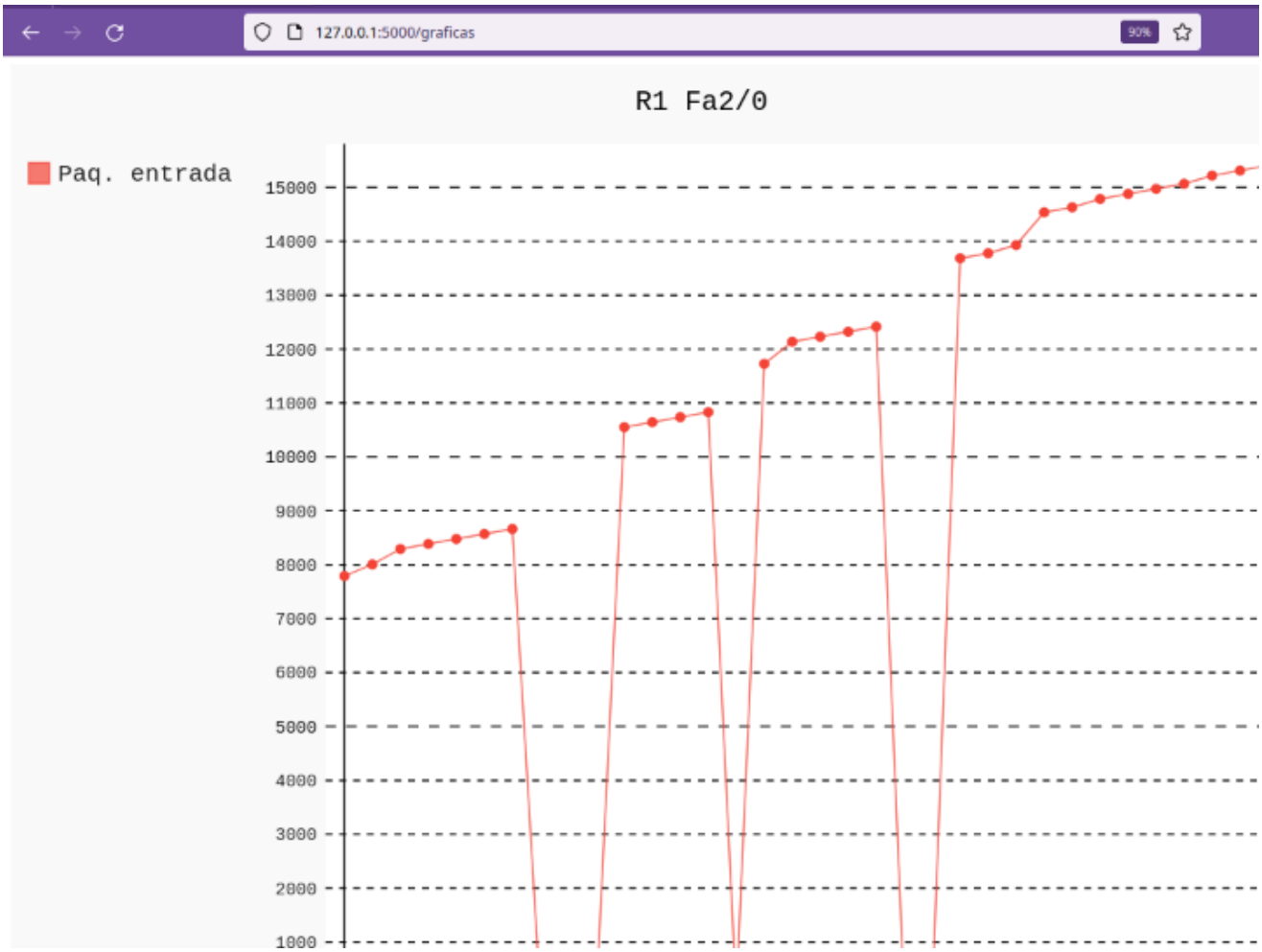
- /

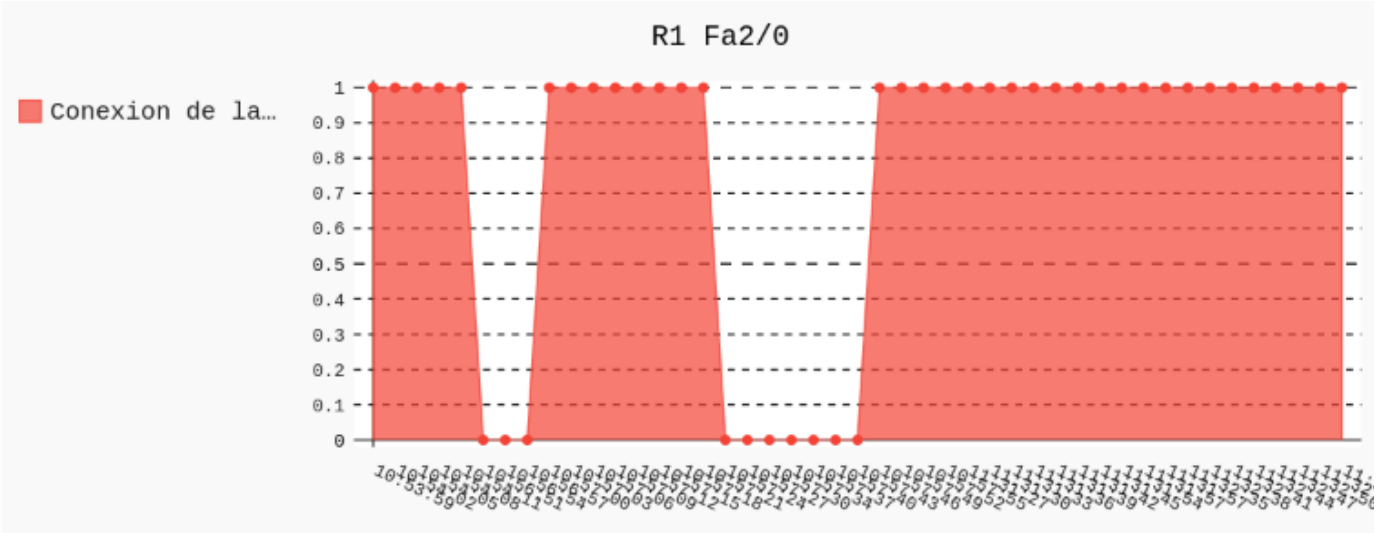
-/graficas

En la primera se crearan los hilos que estaran muestreando la informacion y ejecutando el codigo que esta a la escucha de las traps. Es importante que al iniciar el programa se vaya primero a la ruta raiz sino no se crearan estos hilos ni se lanzara el programa de las traps.

En el segundo simplemente se manda a graficar con los datos de los archivos y se devuelve en un html.




127.0.0.1:5000/graficas
90%
R1 Fa2/0
Paq. entrada
15000
14000
13000
12000
11000
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000


R1 Fa2/0
Conexion de la…
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0