PROYECTO FINAL

BIT-28 Sistemas Operativos II
Escuela de Ingeniería en Software
Periodo: III Cuatrimestre del 2024

Estudiantes:
Paulettet Cabal Orellana
Isabella Lozano Ramos
Sebastián Velázquez Guillén

Profesor:
Carlos Andrés Méndez

Tema: Simulación de Redes Virtuales con Mininet.

Implementación y Configuración de un Entorno de Simulación de Redes utilizando Mininet en Linux

Objetivo General:

- Implementar y configurar un entorno de simulación de redes utilizando Mininet en un sistema operativo Linux para analizar la escalabilidad, disponibilidad, rendimiento del sistema.

Objetivos Específicos:

- Realizar la instalación correcta de Mininet y sus dependencias en un sistema Linux
- Configurar un entorno básico de simulación de red utilizando los componentes de Mininet
- Implementar y evaluar mecanismos de alta disponibilidad para servicios de aplicación.
- Realizar pruebas de escalabilidad horizontal y vertical de servicios.
- Implementar métricas de rendimiento y análisis para servicios de aplicación.
- Documentar el proceso completo de instalación y configuración para futura referencia.

Plan del Proyecto

El desarrollo de este proyecto se llevará a cabo siguiendo un plan estructurado que garantiza una ejecución eficiente, desde la definición de objetivos hasta el análisis y la presentación de resultados obtenidos al configurar la simulación de redes utilizando Mininet en un sistema operativo Linux.

Planificación del proyecto:

1. **Definición de objetivos (Semana 12):**
    - Identificar y definir los objetivos a tomar en el proyecto.
2. **Distribución de responsabilidades (Semana 13):**
    - Asignar tareas específicas a cada integrante del equipo, teniendo en cuenta sus habilidades.
3. **Implementación de Mininet y sus dependencias (Semana 13):**
    - Instalación y configuración inicial exitosa.
    - Creación de topologías.
    - Ejecución de pruebas en las diferentes topologías.
    - Análisis de resultados.
4. **Implementación y recopilación de resultados (Semana 14):**
    - Crear las topologías de red diseñadas y ejecutar simulaciones en Mininet.
    - Recopilar métricas de rendimiento.
    - Documentar y analizar los resultados obtenidos.
5. **Presentación de resultados (Semana 15):**
    - Consolidar los hallazgos y aprendizajes del proyecto en un informe final.
    - Preparar una presentación que incluya los resultados obtenidos, lecciones aprendidas y recomendaciones.

Justificación de la Metodología

Para este proyecto se ha optado por implementar una metodología de investigación aplicada con enfoque experimental y cuantitativo. La investigación aplicada es la más apropiada debido a que el proyecto se centra en la implementación práctica de soluciones tecnológicas y la resolución de problemas específicos en el ámbito de las redes virtuales. El objetivo principal no es generar nuevo conocimiento teórico, sino aplicar el conocimiento existente.

Enfoque Experimental y Cuantitativo

El componente experimental y cuantitativo constituye un pilar fundamental en nuestra metodología, nos permite desarrollar un estudio del comportamiento de las redes virtuales. A través del enfoque experimental, podemos crear y manipular diferentes topologías de red en un entorno controlado, lo que facilita la realización de pruebas repetibles y la obtención de resultados consistentes. Esto se complementa con una perspectiva cuantitativa que nos permite recopilar información de rendimiento y realizar análisis estadísticos detallados del comportamiento de la red. La combinación de ambos enfoques nos proporciona la capacidad de validar objetivamente el funcionamiento y rendimiento de los servicios de red mediante datos medibles y verificables..

Fases de la Metodología

Fase 1: Preparación y Configuración

- Instalación del entorno Linux
- Implementación de Mininet y sus dependencias
- Verificación de la correcta instalación
- Documentación del proceso de configuración

Fase 2: Diseño Experimental

- Definición de los servicios y aplicaciones a implementar
- Establecimiento de parámetros de medición
- Diseño de casos de prueba
- Definición de criterios de éxito

Fase 3: Implementación y Pruebas

- Creación de los escenarios de servicios diseñados
- Ejecución de pruebas de rendimiento y disponibilidad
- Medición de métricas de rendimiento
- Documentación de resultados

Fase 4: Análisis y Evaluación

- Procesamiento de datos recopilados
- Análisis comparativo de resultados
- Evaluación del rendimiento del sistema
- Identificación de áreas de mejora

Herramientas para la Implementación:

- Mininet : Para simular redes virtuales
- VirtualBox: Para crear máquinas virtuales
- Python: Para automatizar pruebas y análisis de datos
- Herramientas para graficar los resultados

Sistemas operativos ligeros: Alpine Linux o Ubuntu

Documentación y Referencias:

- Manuales de Mininet
- Artículos sobre sistemas operativos ligeros y métricas de red (IEEE)

Artículos Académicos Fundamentales

1. Sobre Mininet y Virtualización de Redes

Lantz, B., Heller, B., & McKeown, N. (2010). "A network in a laptop: rapid prototyping for software-defined networks." ACM SIGCOMM Workshop on Hot Topics in Networks.

Wang, S. Y., Chou, C. L., & Yang, C. M. (2013). "EstiNet: A network simulator and emulator." IEEE Communications Magazine.

2. Implementaciones y Casos de Estudio

Oliveira, R. L. S., et al. (2014). "Using Mininet for emulation and prototyping Software-Defined Networks." IEEE Colombian Conference on Communications and Computing.

Keti, F., & Askar, S. (2015). "Emulation of Software Defined Networks Using Mininet in Different Simulation Environments." International Conference on Intelligent Systems, Modelling and Simulation.

3. Documentación y Guías

Mininet Walkthrough Commands (http://mininet.org/walkthrough/)

Mininet Python API Reference Manual (http://mininet.org/api/annotated.html)

GitHub de Mininet (https://github.com/mininet/mininet)

4. Aspectos de Rendimiento y Escalabilidad

Kaur, S., Singh, J., & Ghumman, N. S. (2014). "Network programmability using POX controller." International Conference on Communication, Computing & Systems.

Gupta, M., et al. (2013). "Network virtualization and software defined networking for cloud computing: a survey." IEEE Communications Magazine.

5. Alta Disponibilidad

Sharma, S., et al. (2011). "Implementing Quality of Service for the Software Defined Networking enabled future internet." European Workshop on Software Defined Networks.

Lin, P., et al. (2015). "A west-east bridge based SDN inter-domain testbed." IEEE Communications Magazine.

6. Métricas y Análisis de Rendimiento

Tootoonchian, A., Ghobadi, M., & Ganjali, Y. (2012). "OpenTM: Traffic matrix estimator for OpenFlow networks." Passive and Active Measurement Conference.

Curtis, A. R., et al. (2011). "DevoFlow: Scaling flow management for high-performance networks." ACM SIGCOMM Computer Communication Review.

7. Tendencias Actuales y Futuras

Kreutz, D., et al. (2015). "Software-defined networking: A comprehensive survey." Proceedings of the IEEE.

Nunes, B. A. A., et al. (2014). "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks." IEEE Communications Surveys & Tutorials.

Marco Teórico

La simulación de redes es una herramienta esencial en la investigación y desarrollo de infraestructuras de comunicación. Esta permite crear entornos supervisados donde se puedan probar, evaluar y analizar configuraciones de red, servicios y protocolos sin requerir hardware físico. Entre las herramientas más relevantes en este campo se encuentra **Mininet,** un emulador de redes virtuales creado para realizar prototipos rápidos de redes definidas por software (SDN) (Lantz, Heller & McKeown, 2010).

Mininet emula redes completas utilizando namespaces y cgroups de Linux, estableciendo un entorno de prueba auténtico en un solo sistema operativo. Este método lo convierte en la opción perfecta para evaluar topologías de red, protocolos y servicios sin necesitar una infraestructura física complicada.

Mininet es ampliamente conocido como un recurso para la emulación y el prototipo de redes SDN. En el estudio de Oliveira et al. (2014), se investiga a la aplicación como un entorno de simulación para evaluar arquitecturas SDN, resaltando su habilidad para incorporar controladores como POX y OpenDaylight. Además, Keti y Askar (2015) estudiaron de ~~que~~qué manera Mininet puede funcionar en diversos entornos de simulación, demostrando su capacidad para ajustarse a distintos escenarios, como aquellas redes de campus o entornos en la nube. La documentación oficial de Mininet, que abarca su guía interactiva y la API de Python, ofrece herramientas importantes para crear y analizar redes virtuales.

La disponibilidad continua y la estabilidad son elementos esenciales para las redes actuales. Sharma et al. (2011) indican que las redes SDN posibilitan la implementación de mecanismos de alta disponibilidad, como el balanceo de carga y la conmutación por error (failover), para asegurar un servicio continuo. Por otro lado, Lin et al. (2015) demuestran de qué forma las arquitecturas fundamentadas en SDN pueden promover la escalabilidad tanto horizontal como vertical, ajustándose a las necesidades cambiantes de la red.

Mininet, al integrarse con controladores como POX, permite evaluar estas características en entornos controlados. La capacidad de integrar o eliminar nodos y enlaces en tiempo real hace que sea una herramienta perfecta para evaluar la resiliencia y flexibilidad de las redes simuladas.

La evaluación del rendimiento es clave para medir la efectividad de las configuraciones de la red. De acuerdo con Tootoonchian et al. (2012), herramientas como OpenTM posibilitan la estimación de matrices de tráfico en redes OpenFlow, una característica que puede combinarse con Mininet. Además, Curtis et al. (2011) presenta métodos como DevoFlow, que mejoran la administración de flujos en redes de alto rendimiento, enfatizando la relevancia

de implementar mediciones exactas en entornos simulados. Entre las métricas más significativas a analizar en Mininet están la latencia, el tiempo de respuesta y el uso de recursos del sistema.

El balanceo de carga, combinado con redes definidas por software, sigue siendo un área activa de investigación. Kreutz et al. (2015) señalan que estas tecnologías no sólo maximizan la utilización de recursos, sino que también aumentan la seguridad y la adaptabilidad en redes empresariales y en la nube. También, la flexibilidad de las redes programables permite a los investigadores experimentar con nuevos algoritmos y mecanismos para mejorar la eficiencia de las redes virtuales. En este contexto, Mininet se posiciona como una herramienta fundamental para investigadores y profesionales, facilitando la exploración de estas tendencias en un ambiente accesible y altamente personalizable. Su aplicación en investigaciones como la de Nunes et al. (2014) evidencia su relevancia tanto en el entorno académico como en aplicaciones prácticas.

En conclusión, la utilización de Mininet en la simulación de redes proporciona una plataforma flexible para explorar temas como la alta disponibilidad, escalabilidad y evaluación de rendimiento. Los estudios actuales apoyan su eficacia en el desarrollo y la validación de arquitecturas de red definidas por software.

Marco Teórico del Proyecto

Introducción a la Simulación de Redes con Mininet

La simulación con Mininet brinda un entorno especializado para probar, entender y analizar topologías de diversas redes, sin necesidad de hardware físico. Esta herramienta es muy útil en escenarios académicos y profesionales, gracias a ellas se puede reducir costos y riesgos asociados con la implementación en entornos reales.

¿Qué es Mininet?

Mininet Emulador de Redes Virtuales

Mininet es una herramienta diseñada para emular redes completas utilizando namespaces y cgroups de Linux. Este emulador permite la creación de topologías de red realistas en un solo sistema operativo, ofreciendo una plataforma eficiente para evaluar protocolos, servicios y topologías.

Características Principales de Mininet

1. **Emulación de Redes Definidas por Software (SDN):** Permite prototipar rápidamente redes SDN mediante la simulación de hosts, switches y enlaces virtuales.
2. **Flexibilidad:** Facilita la creación de entornos personalizados adaptados a necesidades específicas de investigación o a desarrollar.
3. **Escalabilidad:** Ofrece soporte para topologías que van desde redes simples hasta configuraciones complejas.

Campos de Uso de Mininet

1. **Investigación académica:**
    - Evaluación de protocolos de red.
    - Desarrollo y prueba de algoritmos de encaminamiento.
2. **Entornos industriales:**
    - Validación de configuraciones antes de implementaciones reales.
    - Pruebas de resiliencia y seguridad.
3. **Educación:**
    - Capacita a estudiantes en diseño y administración de redes.
4. **Optimización de servicios:**
    - Implementación de mecanismos de calidad de servicio (QoS).
    - Pruebas de escalabilidad y alta disponibilidad.

Configuración de Mininet en Linux

La configuración de Mininet en un entorno Linux requiere:

1. **Instalación del sistema operativo base (Alpine Linux o Ubuntu):** Sistemas ligeros que garantizan un rendimiento óptimo.
2. **Instalación de Mininet y sus dependencias:** Incluyendo controladores SDN y librerías necesarias.
3. **Pruebas iniciales:** Verificar la emulación correcta de topologías básicas.
4. **Automatización:** Uso de Python para generar scripts que gestionen la configuración y pruebas de red.

Relevancia de Mininet en el Contexto Actual

En un mundo donde las redes virtuales y SDN están en auge, Mininet se posiciona como una herramienta clave para:

- Reducir costos: Evitando la necesidad de hardware físico.
- Acelerar el desarrollo: Facilitando pruebas rápidas de topologías y servicios.
- Fomentar la innovación: Permitiendo explorar nuevas arquitecturas y protocolos sin riesgos.
- Preparar para el futuro: Mejorando la comprensión de entornos virtuales y optimizando el rendimiento

Conclusión

El uso de herramientas como Mininet en la simulación de redes proporciona una plataforma versátil y accesible para el desarrollo y análisis de infraestructuras de red. Su integración con controladores SDN y su capacidad para emular topologías complejas lo convierten en un recurso invaluable en la investigación y desarrollo de redes virtuales modernas.

Paso 1: Instalación de base

- sudo apt update
- sudo apt upgrade -y
- sudo apt install -y git

```
isabella@servidorpruebas:~$ sudo apt install -y git
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
git is already the newest version (1:2.34.1-1ubuntu1.11).
git set to manually installed.
```

- sudo apt install -y htop

```
isabella@servidorpruebas:~$  sudo apt install -y htop
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
htop is already the newest version (3.0.5-7build2).
htop set to manually installed.
```

- sudo apt install -y python3-pip

```
isabella@servidorpruebas:~$ sudo apt install -y python3-pip
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
python3-pip is already the newest version (22.0.2+dfsg-1ubuntu0.5).
The following packages were automatically installed and are no longer required:
  linux-headers-5.15.0-89 linux-headers-5.15.0-89-generic linux-headers-5.15.0-91
  linux-headers-5.15.0-91-generic linux-image-5.15.0-89-generic linux-image-5.15.0-91-generic
  linux-modules-5.15.0-89-generic linux-modules-5.15.0-91-generic
  linux-modules-extra-5.15.0-89-generic linux-modules-extra-5.15.0-91-generic
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 5 not upgraded.
isabella@servidorpruebas:~$
```

Paso 2: Instalación de Mininet

- sudo apt install -y mininet

```
isabella@servidorpruebas:~$ sudo apt install -y mininet
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
mininet is already the newest version (2.3.0-1ubuntu1).
The following packages were automatically installed and are no longer required:
  linux-headers-5.15.0-89 linux-headers-5.15.0-89-generic linux-headers-5.15.0-91
  linux-headers-5.15.0-91-generic linux-image-5.15.0-89-generic linux-image-5.15.0-91-generic
  linux-modules-5.15.0-89-generic linux-modules-5.15.0-91-generic
  linux-modules-extra-5.15.0-89-generic linux-modules-extra-5.15.0-91-generic
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 5 not upgraded.
isabella@servidorpruebas:~$ _
```

- sudo apt install -y openswitch-testcontroller

```
isabella@servidorpruebas:~$ sudo apt install -y openvswitch-testcontroller
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
openvswitch-testcontroller is already the newest version (2.17.9-0ubuntu0.22.04.1).
```

- sudo service openswitch-switch start

```
isabella@servidorpruebas:~$  sudo service openvswitch-switch start
isabella@servidorpruebas:~$
```

Paso 3: Verificar la Instalación de Mininet

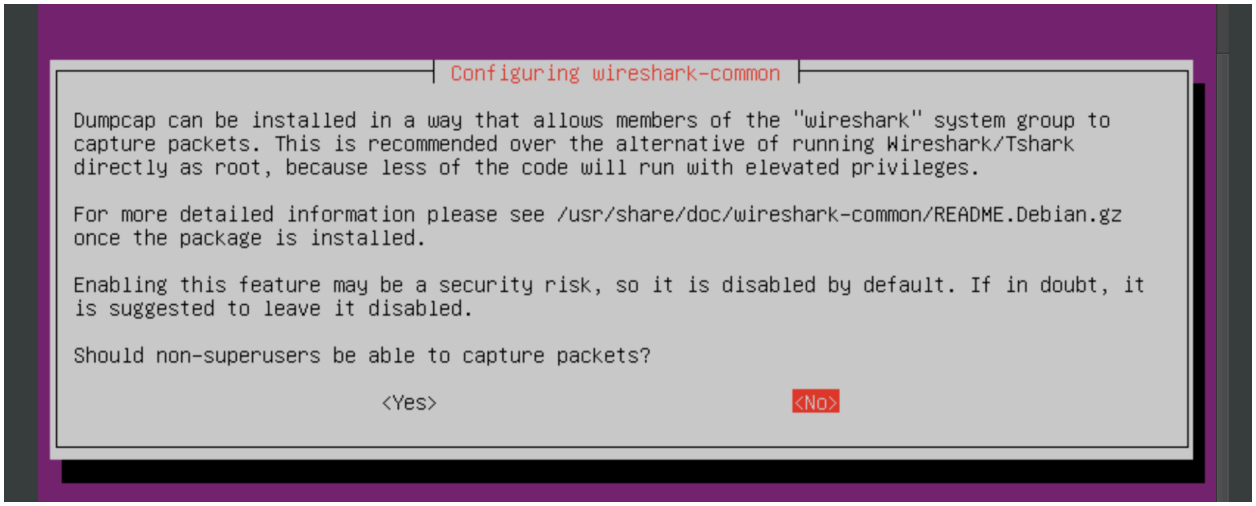
- sudo mn –test pingall

```
isabella@servidorpruebas:~$ sudo mn --test pingall
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Waiting for switches to connect
s1
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
*** Stopping 1 controllers
c0
*** Stopping 2 links
..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
completed in 4.031 seconds
isabella@servidorpruebas:~$
```

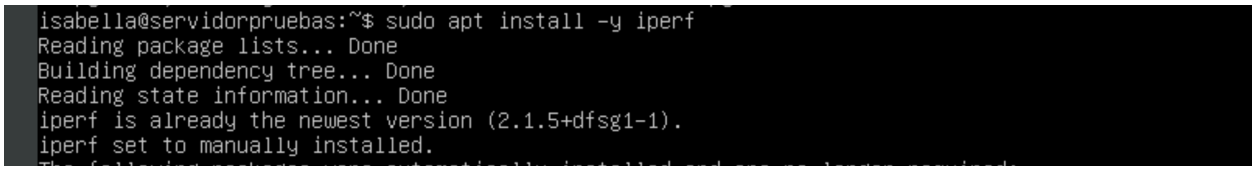
Paso 4: Instalación de herramientas adicionales de análisis

- sudo apt install -y wireshark

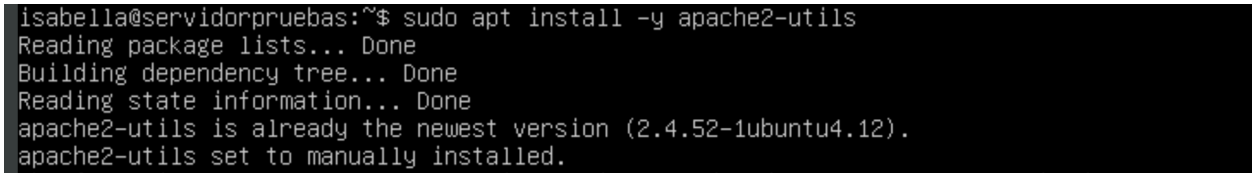
```
isabella@servidorpruebas:~$ sudo apt install -y wireshark
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
wireshark is already the newest version (3.6.2-2).
```



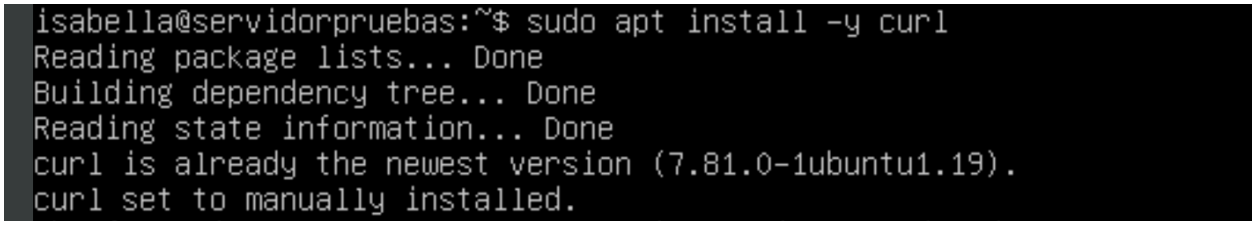
- sudo apt install -y iperf



- sudo apt install -y apache2-utils



- sudo apt install -y curl



Paso 5: Instalación de Componentes para Capa 7

- sudo apt install -y nginx haproxy

```
isabella@servidorpruebas:~$ sudo apt install -y nginx haproxy
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
haproxy is already the newest version (2.4.24-0ubuntu0.22.04.1).
nginx is already the newest version (1.18.0-6ubuntu14.5).
```

- sudo systemctl enable haproxy

```
isabella@servidorpruebas:~$ sudo systemctl enable haproxy
Synchronizing state of haproxy.service with SysV service script with /lib/systemd/systemd-sysv-insta
ll.
Executing: /lib/systemd/systemd-sysv-install enable haproxy
isabella@servidorpruebas:~$ _
```

Paso 6: Crear Directorio de Trabajo

- mkdir /mininet-tests
- cd mininet-tests

Paso 7: Implementar Scripts de Prueba

crear archivo: nano basic_topo.py

```python
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.node import Host
from mininet.link import TCLink
from mininet.log import setLogLevel
from mininet.cli import CLI

class BasicTopo(Topo):
    def build(self):
        # Switches
        s1 = self.addSwitch('s1')
        s2 = self.addSwitch('s2')

        # Hosts
        h1 = self.addHost('h1')
        h2 = self.addHost('h2')

        # Links
        self.addLink(h1, s1, bw=10, delay='5ms')
        self.addLink(h2, s2, bw=10, delay='5ms')
        self.addLink(s1, s2, bw=10)

def runTest():
    topo = BasicTopo()
    net = Mininet(topo=topo, host=Host, link=TCLink)
    net.start()

    print("Prueba de conectividad básica")
    net.pingAll()

    CLI(net)
    net.stop()

if __name__ == '__main__':
    setLogLevel('info')
    runTest()
```

Paso 8: Implementar Pruebas de Capa 7

crear archivo: nano layer7_tests.py

```
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.node import Host
from mininet.link import TCLink
from mininet.util import dumpNodeConnections
from mininet.log import setLogLevel
from mininet.cli import CLI
import time
import os

class Layer7TestTopo(Topo):
    def build(self):
        # Crear switches
        lb = self.addSwitch('lb', dpid='1')
        s1 = self.addSwitch('s1', dpid='2')
        s2 = self.addSwitch('s2', dpid='3')

        # Crear hosts
        client1 = self.addHost('client1')
        client2 = self.addHost('client2')
        web1 = self.addHost('web1')
        web2 = self.addHost('web2')
        haproxy = self.addHost('haproxy')

        # Agregar enlaces con características específicas
        # Usamos TCLink para simular condiciones de red reales
        self.addLink(client1, lb, bw=10, delay='5ms', loss=1)
        self.addLink(client2, lb, bw=10, delay='5ms', loss=1)
        self.addLink(haproxy, lb, bw=100, delay='1ms', loss=0)
        self.addLink(web1, s1, bw=100, delay='1ms', loss=0)
        self.addLink(web2, s2, bw=100, delay='1ms', loss=0)
        self.addLink(s1, lb, bw=100, delay='1ms', loss=0)
        self.addLink(s2, lb, bw=100, delay='1ms', loss=0)
```

dpid= Datapath ID

```
def setupNetwork():
    "Configurar la red y los servicios"
    topo = Layer7TestTopo()
    net = Mininet(topo=topo, host=Host, link=TCLink)
    net.start()

    print("Configurando servidores web...")
    web1, web2 = net.get('web1', 'web2')

    # Configurar servidores web
    for web in [web1, web2]:
        web.cmd('apt-get update && apt-get install -y nginx')
        web.cmd('echo "Server $(hostname)" > /var/www/html/index.html')
        web.cmd('service nginx start')

    # Configurar HAProxy
    haproxy = net.get('haproxy')
    haproxy.cmd('apt-get update && apt-get install -y haproxy')

    # Configuración de HAProxy
    haproxy_config = """
    global
        daemon
        maxconn 256

    defaults
        mode http
        timeout connect 5000ms
        timeout client 50000ms
        timeout server 50000ms

    frontend http-in
        bind *:80
        default_backend webservers

    backend webservers
        balance roundrobin
```

```
    backend webservers
        balance roundrobin
        option httpchk
        server web1 {web1_ip}:80 check
        server web2 {web2_ip}:80 check
    """.format(web1_ip=web1.IP(), web2_ip=web2.IP())

    haproxy.cmd('echo "%s" > /etc/haproxy/haproxy.cfg' % haproxy_config)
    haproxy.cmd('service haproxy restart')

    return net

def runTests(net):
    "Ejecutar pruebas de rendimiento"
    print("\n=== Iniciando pruebas de rendimiento ===")

    client1 = net.get('client1')
    haproxy = net.get('haproxy')
    web1, web2 = net.get('web1', 'web2')

    # Instalar herramientas de prueba
    client1.cmd('apt-get update && apt-get install -y apache2-utils curl')

    # Test 1: Latencia HTTP
    print("\n=== Prueba de Latencia HTTP ===")
    for i in range(5):
        result = client1.cmd('curl -s -w "%%{time_total}\n" -o /dev/null http://%s' % haproxy.IP())
        print(f"Request {i+1}: {result.strip()} segundos")

    # Test 2: Throughput
    print("\n=== Prueba de Throughput ===")
    result = client1.cmd('ab -n 1000 -c 10 http://%s/' % haproxy.IP())
    print(result)

    # Test 3: Prueba de Failover
    print("\n=== Prueba de Failover ===")
    print("Estado inicial - ambos servidores activos:")
```

```
    print(result)

    # Test 3: Prueba de Failover
    print("\n=== Prueba de Failover ===")
    print("Estado inicial - ambos servidores activos:")
    result = client1.cmd('curl -s http://%s/' % haproxy.IP())
    print(f"Respuesta: {result.strip()}")

    print("\nApagando web1...")
    web1.cmd('service nginx stop')
    time.sleep(2)

    print("Verificando disponibilidad después de falla:")
    for i in range(3):
        result = client1.cmd('curl -s http://%s/' % haproxy.IP())
        print(f"Request {i+1}: {result.strip()}")

    # Test 4: Prueba de carga
    print("\n=== Prueba de Carga ===")
    result = client1.cmd('ab -n 5000 -c 50 http://%s/' % haproxy.IP())
    print(result)

if __name__ == '__main__':
    setLogLevel('info')
    print("Iniciando configuración de red...")
    net = setupNetwork()

    print("\nRed configurada. Esperando 10 segundos para estabilización...")
    time.sleep(10)

    runTests(net)

    print("\nPruebas completadas. Iniciando CLI para pruebas manuales...")
    CLI(net)

    net.stop()
```

Paso 9: Ejecutar pruebas

Pruebas básicas:

```
isabella@servidorpruebas:~/mininet-tests$ sudo python3 basic_topo.py
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1 s2
*** Adding links:
(10.00Mbit 5ms delay) (10.00Mbit 5ms delay) (h1, s1) (10.00Mbit 5ms delay) (10.00Mbit 5ms delay) (h2, s2) (10.00Mbit) (10.00Mbit) (s1, s2)

*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...(10.00Mbit 5ms delay) (10.00Mbit) (10.00Mbit 5ms delay) (10.00Mbit)
Prueba de conectividad básica
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
*** Starting CLI:
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=30.2 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=22.6 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=22.0 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=22.8 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=22.6 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=22.3 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=21.5 ms
^C
--- 10.0.0.2 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6009ms
rtt min/avg/max/mdev = 21.499/23.416/30.161/2.782 ms
```

Prueba Capa 7:

```
isabella@servidorpruebas:~/mininet-tests$ sudo python3 layer7_tests.py
Iniciando configuración de red...
*** Creating network
*** Adding controller
*** Adding hosts:
client1 client2 haproxy web1 web2
*** Adding switches:
lb s1 s2
*** Adding links:
(10.00Mbit 5ms delay 1.00000% loss) (10.00Mbit 5ms delay 1.00000% loss) (client1, lb) (10.00Mbit 5ms delay 1.00000% loss) (10.00Mbit 5ms d
elay 1.00000% loss) (client2, lb) (100.00Mbit 1ms delay 0.00000% loss) (100.00Mbit 1ms delay 0.00000% loss) (haproxy, lb) (100.00Mbit 1ms
delay 0.00000% loss) (100.00Mbit 1ms delay 0.00000% loss) (s1, lb) (100.00Mbit 1ms delay 0.00000% loss) (100.00Mbit 1ms delay 0.00000% los
s) (s2, lb) (100.00Mbit 1ms delay 0.00000% loss) (100.00Mbit 1ms delay 0.00000% loss) (web1, s1) (100.00Mbit 1ms delay 0.00000% loss) (100
.00Mbit 1ms delay 0.00000% loss) (web2, s2)
*** Configuring hosts
client1 client2 haproxy web1 web2
*** Starting controller
c0
*** Starting 3 switches
lb s1 s2 ...(10.00Mbit 5ms delay 1.00000% loss) (10.00Mbit 5ms delay 1.00000% loss) (100.00Mbit 1ms delay 0.00000% loss) (100.00Mbit 1ms d
elay 0.00000% loss) (100.00Mbit 1ms delay 0.00000% loss) (100.00Mbit 1ms delay 0.00000% loss) (100.00Mbit 1ms delay 0.00000% loss) (100.00
Mbit 1ms delay 0.00000% loss) (100.00Mbit 1ms delay 0.00000% loss)
Configurando servidores web...

Red configurada. Esperando 10 segundos para estabilización...

=== Iniciando pruebas de rendimiento ===

=== Prueba de Latencia HTTP ===
Request 1: > 0.028381 segundos
Request 2: > 0.014286 segundos
Request 3: > 0.014614 segundos
Request 4: > 0.014119 segundos
Request 5: > 0.013692 segundos

=== Prueba de Throughput ===
This is ApacheBench, Version 2.3 <$Revision: 1879490 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/
```

Dentro del CLI de Mininet probar:

h1 ping h2

```
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=23.0 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=22.6 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=22.5 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=21.8 ms
^C
--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 21.804/22.475/23.032/0.438 ms
```

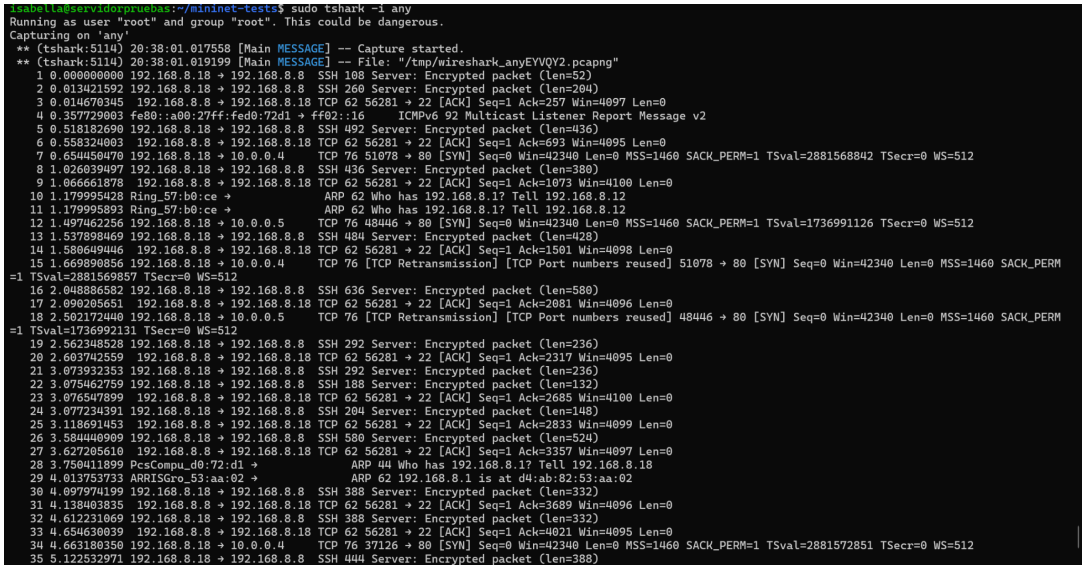
h1 iperf -c h2

```
mininet> h2 iperf -s &
mininet>  h2 netstat -tln | grep 5001
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
tcp        0      0 0.0.0.0:5001            0.0.0.0:*               LISTEN
mininet>  h1 iperf -c h2
------------------------------------------------------------
Client connecting to 10.0.0.2, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  1] local 10.0.0.1 port 53660 connected with 10.0.0.2 port 5001
[ ID] Interval       Transfer     Bandwidth
[  1] 0.0000-12.4001 sec  14.0 MBytes  9.47 Mbits/sec
mininet>
```

Paso 10: Monitoreo y Análisis

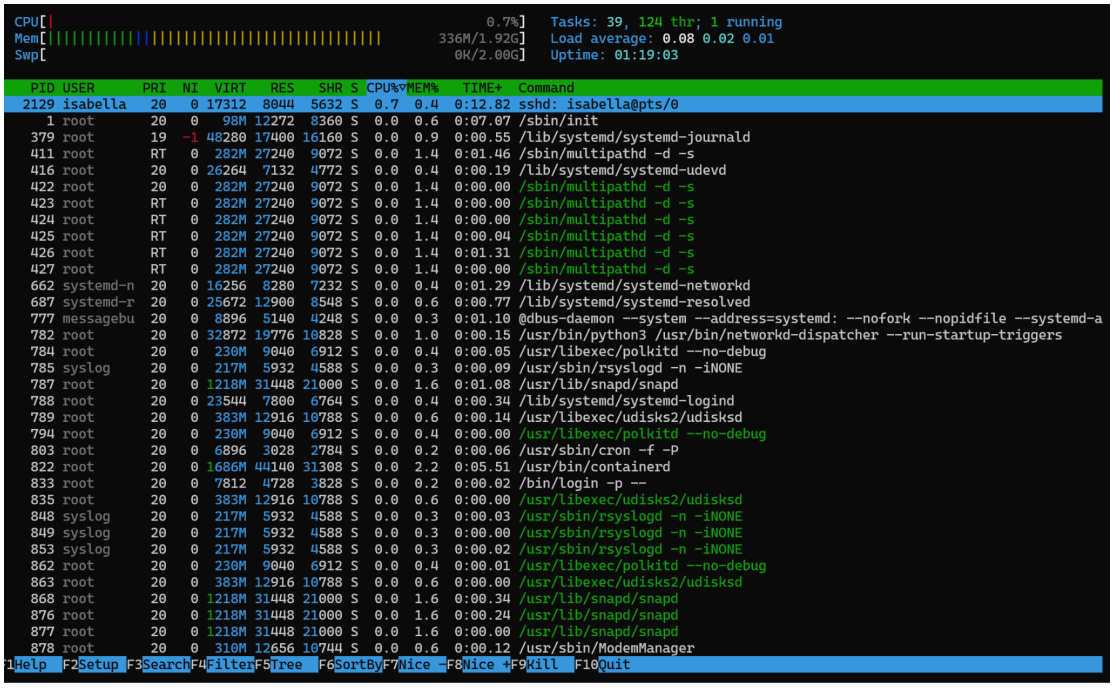
- **Para monitorear el tráfico**

  sudo tshark -i any



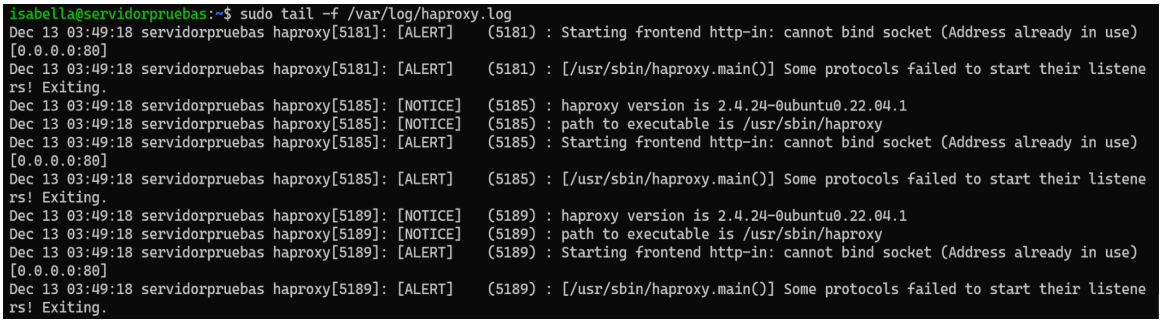
- **Para monitorear recursos**

  htop



- **Para ver los logs de HAProxy**

  sudo tail -f /var/log/haproxy.log



- **Pruebas de Rendimiento**

  Desde la CLI de Mininet:

  Latencia: client1 curl -w "%{time_total}\n" -o /dev/null http://haproxy

  Carga: client1 ab -n 1000 -c 10 http://haproxy/

- **Pruebas de Alta Disponibilidad**

  Detener un servidor web

  web1 service nginx stop

  Verificar que el servicio sigue funcionando

  client1 curl http://haproxy

Paso 11: Instalación de apache

sudo apt-get install apache2

Paso 12: Crear archivo artifact

nano apache_topology.py

```
from mininet.node import Controller
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.node import OVSKernelSwitch
import time

def createApacheTopology():
    # Create network
    net = Mininet(controller=Controller, switch=OVSKernelSwitch)

    # Add controller
    info('*** Adding controller\n')
    net.addController('c0')

    # Add switch
    info('*** Adding switch\n')
    s1 = net.addSwitch('s1')

    # Add hosts
    info('*** Adding hosts\n')
    # Apache servers
    apache1 = net.addHost('apache1')
    apache2 = net.addHost('apache2')
    # Client for testing
    client = net.addHost('client')

    # Add links
    info('*** Creating links\n')
    net.addLink(s1, apache1)
    net.addLink(s1, apache2)
    net.addLink(s1, client)

    # Start network
    info('*** Starting network\n')
    net.start()

    # Configure Apache servers
    info('*** Configuring Apache servers\n')

    # Configure apache1
    apache1.cmd('apt-get update')
    apache1.cmd('apt-get install -y apache2')
    apache1.cmd('echo "This is Apache Server 1" > /var/www/html/index.html')
    apache1.cmd('service apache2 start')
```

```
    # Configure apache2
    apache2.cmd('apt-get update')
    apache2.cmd('apt-get install -y apache2')
    apache2.cmd('echo "This is Apache Server 2" > /var/www/html/index.html')
    # Change port to avoid conflict
    apache2.cmd('sed -i \'s/Listen 80/Listen 8080/\' /etc/apache2/ports.conf')
    apache2.cmd('sed -i \'s/:80/:8080/\' /etc/apache2/sites-enabled/000-default.conf')
    apache2.cmd('service apache2 start')

    # Wait for services to start
    time.sleep(2)

    # Test connectivity
    info('*** Testing web servers\n')
    client.cmd('wget -O - apache1')
    client.cmd('wget -O - apache2:8080')

    # Start CLI
    CLI(net)

    # Cleanup
    net.stop()

if __name__ == '__main__':
    setLogLevel('info')
    createApacheTopology()
```

Paso 13: Dale permisos de ejecución

chmod +x apache_topology.py

Paso 14: Ejecutar la topología

sudo python apache_topology.py

Paso 15: Verificar que todo funcione

mininet> nodes # Ver todos los nodos

mininet> net # Ver todas las conexiones

mininet> dump # Ver información detallada

Paso 16: Monitoreo y Análisis

mininet> client wget -O - apache1

mininet> client wget -O - apache2:8080

- **Repetir pruebas de análisis (Ver paso 10)**