SDN mininet con firewall, balanceador de carga y tolerancia a fallos

Sebastián Bolaños, Andrés Ramírez, Daniela Urbano, Juan José De Los Ríos

Universidad Autónoma de Occidente
Cali, Colombia

Sebastian.bolanos_c@uao.edu.co

Andres.ramirez_leg@uao.edu.co

daniela.urbano_mue@uao.edu.co

Juan.de@uao.edu.co

En este documento se expone el uso del Software Defined Networking mediante una máquina virtual Ubuntu y el funcionamiento e implementación de un balanceo de cargas que se encargara de dirigir la petición con respecto a la decisión del balanceador de cargas utilizando además un firewall y un direccionador de errores para que la red pueda seguir en funcionamiento

***Abstract*— This document shows the use of Software Defined Networking through an Ubuntu virtual machine, the operation and implementation of a load balancer that will be in charge of directing the request with respect to the decision of the load balancer using also a firewall and an error router so that the network can continue to operate.**

I. INTRODUCCIÓN

El SDN se ha visto altamente utilizado y ha pasado de boca en boca mucho estos últimos años, las redes que son definidas por software (SDN) son las correspondientes al control del hardware para suministrarle el control a una aplicación software la cual tiene como nombre controlador, el impacto positivo del Software Defined Networking es alto, brindando una mayor velocidad, una infraestructura más ágil otorgando un dinamismo al usuario mucho mejor.

La red definida por software permite la separación del plano de datos y el plano de control gracias al control centralizado del plano de datos permitiendo que el plano de datos logre ser supervisado de forma remota a través del plano de control.

El balanceo de carga trata de distribuir el tráfico de red del trabajo de acuerdo con la disponibilidad de procesamiento y a los recursos de cada equipo en un sistema computacional. Esta distribución pretende maximizar la utilización de los recursos, posibilitando el mejor desempeño del sistema, esto se utiliza en sitios web con alto tráfico que deben atender a cientos de miles de solicitudes.

El mininet es un emulador para el despliegue de redes sobre los recursos de un ordenador sencillo o máquina virtual.

Pox es un controlador que se basa en el lenguaje de programación de Python que permite al usuario controlar una topología de red creada por mininet, además de que tiene la posibilidad de proporcionar características de seguridad en la topología dada con la ayuda de componentes, que pueden ser entendidos como funcionalidades del controlador POX.

II. DESCRIPCIÓN DEL PROBLEMA

SDN (Software Defined Networking) para atención de desastres. Se requiere implementar una red que permita la reacción efectiva frente a desastres que afecten su disponibilidad. Para esto se hará uso de conceptos de SDN (Software Defined Networking), permitiendo la adaptación, flexibilidad de la red para reconfigurarse y su rápida recuperación ante desastres.

III. ALTERNATIVAS DE SOLUCIÓN Y SELECCIÓN

Otros controladores considerados para la solución del problema:

1. **Ryu:** Controlador open source utilizado para mejorar la agilidad de la red facilitando la administración y la adaptación al tráfico en el balanceador de carga.es un framework basado en componentes hecho en Python permitiendo así la programación de varios protocolos con una curva de aprendizaje media.

2. **Floodlight:** Controlador bajo licencia de Apache 2.0, amigable con el usuario y con gran curva de aprendizaje. Este controlador permite que el SDN (el cerebro de la red) se pueda comunicar con el plano de redirección (switches, routers, etc.) para permitirles hacer los cambios de red. Una de sus mayores ventajas es la compatibilidad con APIs REST facilitando la programación por medio de interfaz con el producto.

3. **OpenDaylight:** Similar a floodlight comprende el uso de un controlador con diferentes funcionalidades agregadas. Esta desarrollado bajo una licencia EPLv1.0. Basado en Java, es un controlador que implementa el uso de Maven (automatización del proceso), OSGi (Back-end permitiendo cargas dinámicas de paquetes), JAVA y APIs

REST. Es considerado un controlador muy completo y ordenado, pero demanda mucho conocimiento acerca del manejo de sus utilidades teniendo una gran curva de aprendizaje, además posee escasa documentación por parte de sus autores dificultando un poco la tarea de ahondar en su funcionamiento.

	Ryu	Floodlight	OpenDaylight
Lenguaje	Python	Java	Java
Rendimiento	Lento	Rápido	Rápido
OpenFlow	1.0, 1.2, 1.3, 1.4, 1.5	1.0 y 1.3	1.0 y 1.3
Curva de aprendizaje	Mediana	Difícil	Difícil
¿Por qué no se eligió?	Poco conocimiento acerca del controlador y poca maestría en el uso de Python	Bajo manejo de Java y poco tiempo disponible para conocer el funcionamiento del controlador	Poco conocimiento de Java y escasa cantidad de documentación oficial distribuida por los creadores sobre el funcionamiento del controlador.

Figura 1. Tabla de comparación de alternativas no escogidas

El grupo tomo como punto de partida en la selección algunos parámetros como:

- El lenguaje de programación, en donde consideramos optar por uno que todos tuviéramos conocimientos.
- El tiempo para realizar el proyecto era definido en donde contábamos con cierta cantidad de semanas en donde se debe aprender el funcionamiento del controlador, saber programar en él, la curva de aprendizaje y la dificultad del sistema fue otro factor crucial.
- Por último, nos inclinamos por un controlador que tuviera bastante documentación, soporte y guías del docente, donde tuvimos en cuenta si era una tecnología que se pudiera implementar y así tener diferentes fuentes a las cuales acudir con respecto a los inconvenientes o pausas del proyecto.

Gracias a esto el grupo puede concluir que Mininet y Pox serían los controladores más adecuados con respecto a los parámetros que planteamos, por otro lado, satisfaciendo las necesidades del problema. Además, se tuvo en cuenta, que utilizara un lenguaje sencillo y de conocimiento para todo el grupo y así facilitar que la curva de aprendizaje sea más efectiva.

IV. PALABRAS CLAVE

- Balanceador de carga: equipo de red destinado a distribuir tráfico según uno o más algoritmos entre varios servidores.

- Switch: en una red de datos legacy o no-SDN, equipo que permite conmutar paquetes de datos a nivel de capa de enlace del modelo OSI. En una red SDN, equipo configurable mediante el controlador y con similares características físicas al legacy, aunque con una variedad mucho más amplia de funcionalidades.

- Mininet: simulador de red. Capaz de simular una red virtual conformada por múltiples hosts, switches OpenFlow y enlaces.

- Software Defined Networks (SDN): paradigma de red basado en la configuración de la misma a través de herramientas centralizadas y desacopladas del hardware

- Controlador: equipo programable en una arquitectura SDN que permite gestionar y configurar los switches

V. REQUERIMIENTOS

Este proyecto consiste en la implementación de tres requerimientos principales:

- Profundizar en la investigación de tecnologías avanzadas de servicios de telecomunicaciones y generar informe resultado de la investigación.
- Implementar prototipo de prueba de los conceptos investigados.
- Pruebas de funcionamiento del prototipo funcional implementado para la tecnología SDN.

VI. IMPLEMENTACIÓN

Para el desarrollo del proyecto se hizo uso de las herramientas necesarias de acuerdo a la estructura básica de los requerimientos en cuanto a metodología de aplicación y aplicaciones de ejecución.

Herramientas Utilizadas

- Tolerancia a fallos con Mininet
- Sistema operativo: Ubuntu 20.04
- Balanceador de Carga: POX.
- Lenguaje de programación: Python.
- Ejecución del firewall.

Preparación inicial del sistema.

Como medida inicial para la implementación de la solución, se realizó la instalación de una máquina virtual en Ubuntu 20.04

Figura 2. Escritorio Ubuntu 20.04

Instalación del Mininet

Mininet es un emulador para el despliegue de redes sobre los limitados recursos de un ordenador sencillo simple o máquina virtual. Éste utiliza

el kernel de Linux y otros recursos para emular elementos de la SDN como el controlador, los switches OpenFlow y los hosts.

A la hora de instalar el entorno de trabajo se nos presentan dos opciones, el empleo de una máquina virtual Mininet o, por el contrario, una instalación nativa. Ambas opciones están disponibles para sistemas Linux, pero podemos emplear la primera opción para emplear este entorno en otros sistemas operativos, ya sea Windows o Mac Os. A continuación, se describirán los pasos a seguir:

1. Para instalar de forma nativa desde la fuente, primero debe obtener el código fuente:

git clone https://github.com/mininet/mininet

2. Tenga en cuenta que el git comando anterior verificará la última y mejor Mininet (que recomendamos) Si desea ejecutar la última versión etiquetada/lanzada de Mininet, o cualquier otra versión, puede verificar esa versión explícitamente:

   cd mininet
   git tag # Lista de versiones disponibles
   git checkout -b mininet-2.3.1 # o la versión que desees instalar
   cd ..

3. Una vez que tenga el árbol de fuentes, el comando para instalar Mininet es:

   Mininet/util/install.sh -a

   -a: Instala todo lo que se incluye en Mininet VM, incluidas las dependencias como Open vSwitch y las adiciones como OpenFlow wireshark dissector y POX. De manera predeterminada, estas herramientas se integrarán en directorios creados en su directorio de inicio

Equilibrador de carga

POX es una plataforma de software de red escrita en Python. POX comenzó como un controlador OpenFlow, pero ahora también puede funcionar como un conmutador OpenFlow y puede ser útil para escribir software de red en general. Actualmente es compatible con OpenFlow 1.0 e incluye soporte especial para las extensiones Open vSwitch/Nicira

Para implementar el balanceador de carga utilizaremos el controlador POX y se realizan los siguientes pasos:

1. Iniciar el controlador del equilibrador de carga

Sudo ./pox.py loadbalancer --ip=10.1.2.3 --servers=10.0.0.5,10.0.0.6,10.0.0.7,10.0.0.8 --weights=2,2,3,4 --mode=rr

Aquí los pesos indican la capacidad de manejo del servidor y en el modo rr, los paquetes que se distribuyen por turnos ponderados.

2. Iniciar una minired de un solo clúster con 8 nodos:

sudo mn --topo single,8 --controller remote --mac --switch ovsk

3. Para visualizar el proceso de equilibrio de carga haciendo ping al servidor:

mininet> h1 ping -c 15 –i 1,5 10.1.2.3

Después de eso, cuando salimos de la mininet, obtenemos un gráfico de la distribución de paquetes entre los servidores.

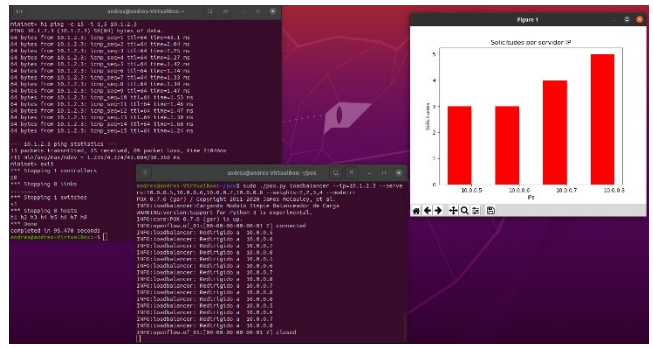

Figura 3. Ejecutando el balanceador de carga con modo weighted round robin

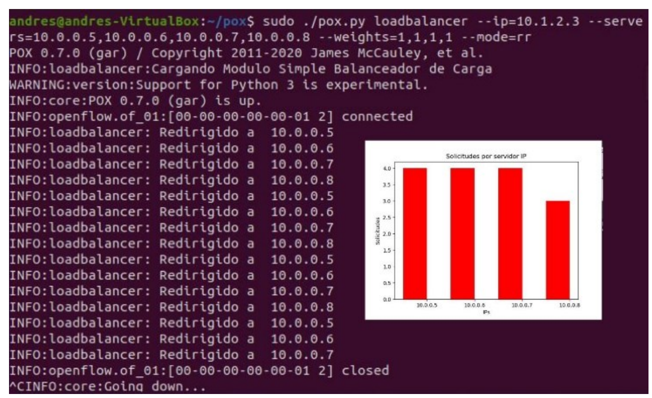

Figura 4. Ejecución del balanceador de carga con modo round robin (round robin ponderado con pesos iguales)

Configuración Firewall

Reglas de firewall predeterminadas:

Regla 1: bloquear todos los paquetes salientes desde la dirección mac 00:00:00:00:00:02
Regla 2: bloquear todos los paquetes de 10.0.0.3 a 10.0.0.4
Regla 3: bloquear todos los paquetes desde 10.0.0.1 hasta 10.0.0.3

Usaremos el controlador POX como firewall de red. A continuación, se describirán los pasos a seguir:

- Iniciar el firewall, con el comando:

  ./start_firewall.sh <Ubicación de la carpeta pox>

- Iniciar mininet con la topologia

  sudo python3 mininet-topology.py

- Probar las reglas, ejecutamos el comando:

mininet> pingall

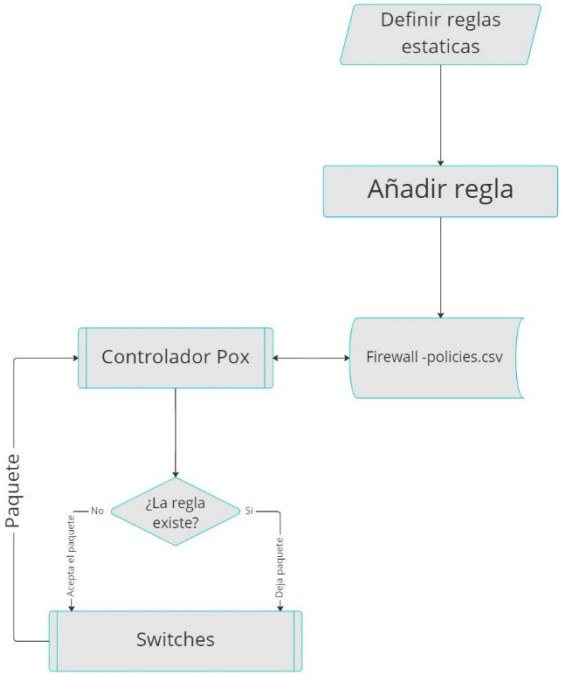

Figura 5. Diagrama de flujo del funcionamiento del firewall

En la imagen anterior describe como es el funcionamiento del firewall, primero se definen las reglas estáticas, que son añadidas dentro del Firewall-policies.csv, que están en contacto directo con el controlador pox que viene por defecto y verifica si la regla existe o no en caso de que no exista acepta el paquete y en caso contrario lo deja, esto son enviado a los conmutadores de la red que envían el resultado hacia el controlador pox.

Configuración Routing

En este caso nos centraremos en el entorno de Mininet, un emulador de redes basada en Linux, que da la posibilidad de crear redes virtuales (switches, routers, hosts, etc) bajo el dominio de un controlador basado en SDN. Tras conocer este sistema se realizará una configuración de la topología de red anti-fallas.

Con base a la problemática, se propone la creación de una arquitectura compuesta por 4 routers y los Switches de la gráfica, las H son host, las S son Switch, la R son routers y los enlaces tolerantes a fallas son R1 y R2, permitiendo que la red siga funcionando ante algún desastre.

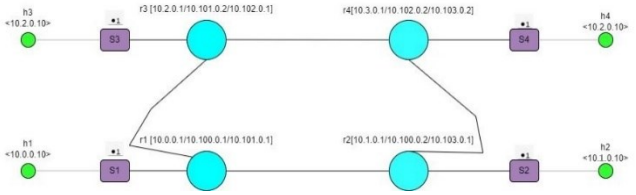

Figura 4. Conexiones entre los 4 routers y los Switches de la gráfica de SDN.

```python
#!/usr/bin/python
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.node import Node
from mininet.node import RemoteController
from mininet.log import setLogLevel, info
from mininet.cli import CLI


#Habilitar el reenvio de IP para usar un host como enrutador
class LinuxRouter(Node):
    def config(self, **params):
        super(LinuxRouter, self).config(**params)
        self.cmd('sysctl net.ipv4.ip_forward=1')

    def terminate(self):
        self.cmd('sysctl net.ipv4.ip_forward=0')
        super(LinuxRouter, self).terminate()


class NetworkTopo(Topo):
    def build(self, **_opts):
        # Agrega 4 routers en cuatro diferentes subredes
        r1 = self.addHost('r1', cls=LinuxRouter, ip='10.0.0.1/24')
        r2 = self.addHost('r2', cls=LinuxRouter, ip='10.1.0.1/24')
        r3 = self.addHost('r3', cls=LinuxRouter, ip='10.2.0.1/24')
        r4 = self.addHost('r4', cls=LinuxRouter, ip='10.3.0.1/24')

        # Agrega 4 switches
        s1 = self.addSwitch('s1')
        s2 = self.addSwitch('s2')
        s3 = self.addSwitch('s3')
        s4 = self.addSwitch('s4')

        # Agregar vinculos de host-switch(conmutador) en la misma subred relativa
        self.addLink(s1,
                     r1,
                     intfName2='r1-eth1',
                     params2={'ip': '10.0.0.1/24'})

        self.addLink(s2,
                     r2,
                     intfName2='r2-eth1',
                     params2={'ip': '10.1.0.1/24'})

        self.addLink(s3,
                     r3,
                     intfName2='r3-eth1',
```

Figura 6. Habilitar el reenvió de IP para usar un host como enrutar y agregar vínculos de host switch

```python
                     r3,
                     intfName2='r3-eth1',
                     params2={'ip': '10.2.0.1/24'})

        self.addLink(s4,
                     r4,
                     intfName2='r4-eth1',
                     params2={'ip': '10.3.0.1/24'})

        # Agregar vinculos enrutador-enrutador en nuevas subredes para las conexiones enrutador-enrutador
        self.addLink(r1,
                     r2,
                     intfName1='r1-eth2',
                     intfName2='r2-eth2',
                     params1={'ip': '10.100.0.1/24'},
                     params2={'ip': '10.100.0.2/24'})

        self.addLink(r1,
                     r3,
                     intfName1='r1-eth3',
                     intfName2='r3-eth2',
                     params1={'ip': '10.101.0.1/24'},
                     params2={'ip': '10.101.0.2/24'})

        self.addLink(r3,
                     r4,
                     intfName1='r3-eth3',
                     intfName2='r4-eth2',
                     params1={'ip': '10.102.0.1/24'},
                     params2={'ip': '10.102.0.2/24'})

        self.addLink(r2,
                     r4,
                     intfName1='r2-eth3',
                     intfName2='r4-eth3',
                     params1={'ip': '10.103.0.1/24'},
                     params2={'ip': '10.103.0.2/24'})

        # Agregando hosts, especificando la ruta por defecto
        h1 = self.addHost(name='h1',
                          ip='10.0.0.10/24',
                          defaultRoute='via 10.0.0.1')
        h2 = self.addHost(name='h2',
                          ip='10.1.0.10/24',
                          defaultRoute='via 10.1.0.1')

        h3 = self.addHost(name='h3',
                          ip='10.2.0.10/24',
```

Figura 7. Agregar vínculos enrutador-enrutador en nuevas subredes y agregar hosts, especificando la ruta.

```
    # Agregar rutas Para llegar a redes que no estan conectadas directamente: "metrica" especifica la
prioridad
    info(net['r1'].cmd("ip route add 10.1.0.0/24 via 10.100.0.2 dev r1-eth2 metric 100"))
    info(net['r1'].cmd("ip route add 10.1.0.0/24 via 10.101.0.2 dev r1-eth3 metric 200"))
    info(net['r1'].cmd("ip route add 10.2.0.0/24 via 10.101.0.2 dev r1-eth3"))
    info(net['r1'].cmd("ip route add 10.3.0.0/24 via 10.101.0.2 dev r1-eth3 metric 100"))
    info(net['r1'].cmd("ip route add 10.3.0.0/24 via 10.100.0.2 dev r1-eth2 metric 200"))
    info(net['r1'].cmd("ip route add 10.102.0.0/24 via 10.101.0.2 dev r1-eth3"))
    info(net['r1'].cmd("ip route add 10.103.0.0/24 via 10.100.0.2 dev r1-eth2 metric 100"))
    info(net['r1'].cmd("ip route add 10.103.0.0/24 via 10.101.0.2 dev r1-eth3 metric 200"))

    info(net['r2'].cmd("ip route add 10.0.0.0/24 via 10.100.0.1 dev r2-eth2 metric 100"))
    info(net['r2'].cmd("ip route add 10.0.0.0/24 via 10.103.0.2 dev r2-eth3 metric 200"))
    info(net['r2'].cmd("ip route add 10.3.0.0/24 via 10.103.0.2 dev r2-eth3"))
    info(net['r2'].cmd("ip route add 10.2.0.0/24 via 10.103.0.2 dev r2-eth3 metric 100"))
    info(net['r2'].cmd("ip route add 10.2.0.0/24 via 10.100.0.1 dev r2-eth2 metric 200"))
    info(net['r2'].cmd("ip route add 10.102.0.0/24 via 10.103.0.2 dev r2-eth3"))
    info(net['r2'].cmd("ip route add 10.101.0.0/24 via 10.100.0.1 dev r2-eth2 metric 100"))
    info(net['r2'].cmd("ip route add 10.101.0.0/24 via 10.103.0.2 dev r2-eth3 metric 200"))

    info(net['r3'].cmd("ip route add 10.0.0.0/24 via 10.101.0.1 dev r3-eth2"))
    info(net['r3'].cmd("ip route add 10.3.0.0/24 via 10.102.0.2 dev r3-eth3"))
    info(net['r3'].cmd("ip route add 10.1.0.0/24 via 10.102.0.2 dev r3-eth3 metric 100"))
    info(net['r3'].cmd("ip route add 10.1.0.0/24 via 10.101.0.1 dev r3-eth2 metric 200"))
    info(net['r3'].cmd("ip route add 10.103.0.0/24 via 10.102.0.2 dev r3-eth3"))
    info(net['r3'].cmd("ip route add 10.100.0.0/24 via 10.101.0.1 dev r3-eth2"))
    #info(net['r3'].cmd("ip route add 10.102.0.0/24 via 10.102.0.2 dev r3-eth3 metric 200"))
    #info(net['r3'].cmd("ip route add 10.102.0.0/24 via 10.100.0.1 dev r3-eth2 metric 100"))

    info(net['r4'].cmd("ip route add 10.1.0.0/24 via 10.103.0.1 dev r4-eth3"))
    info(net['r4'].cmd("ip route add 10.2.0.0/24 via 10.102.0.1 dev r4-eth2"))
    info(net['r4'].cmd("ip route add 10.0.0.0/24 via 10.102.0.1 dev r4-eth2 metric 100"))
    info(net['r4'].cmd("ip route add 10.0.0.0/24 via 10.103.0.1 dev r4-eth3 metric 200"))
    info(net['r4'].cmd("ip route add 10.101.0.0/24 via 10.102.0.1 dev r4-eth2"))
    info(net['r4'].cmd("ip route add 10.103.0.0/24 via 10.103.0.1 dev r4-eth3"))
    #info(net['r4'].cmd("ip route add 10.101.0.0/24 via 10.101.0.2 dev r4-eth3 metric 100"))
    #info(net['r4'].cmd("ip route add 10.101.0.0/24 via 10.103.0.1 dev r4-eth3 metric 200"))

    net.start()
    CLI(net)
    net.stop()


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

Figura 8. Podemos visualizar las direcciones que envían a cada uno de los router las métricas especifican la prioridad son las que permiten que la red soporte la caída del enlace.

Use el controlador POX para reenviar el tráfico como un enrutador. Sigue estos pasos:

- Abra una terminal y ejecute el controlador POX con:

python3 home/pox/pox.py log.level --DEBUG samples.pretty_log forwarding.l2_pairs

```
andres@andres-VirtualBox: ~/Downloads/SDN-proyecto/fire...
andres@andres-VirtualBox:~/Downloads/SDN-proyecto/firewall$ python3 /home/andres
/pox/pox.py log.level --DEBUG samples.pretty_log forwarding.l2_pairs
POX 0.7.0 (gar) / Copyright 2011-2020 James McCauley, et al.
[forwarding.l2_pairs    ] Pair-Learning switch running.
[core                   ] POX 0.7.0 (gar) going up...
[core                   ] Running on CPython (3.8.10/Jun 22 2022 20:18:18)
[core                   ] Platform is Linux-5.15.0-52-generic-x86_64-with-glibc2
.29
[version                ] Support for Python 3 is experimental.
[core                   ] POX 0.7.0 (gar) is up.
[openflow.of_01         ] Listening on 0.0.0.0:6633
[openflow.of_01         ] [00-00-00-00-00-01 2] connected
[openflow.of_01         ] [00-00-00-00-00-04 3] connected
[openflow.of_01         ] [00-00-00-00-00-03 4] connected
[openflow.of_01         ] [00-00-00-00-00-02 5] connected
[forwarding.l2_pairs    ] Installing 72:14:d2:a1:a8:94 <-> 3a:a5:4c:17:43:00
[forwarding.l2_pairs    ] Installing de:7c:0d:8f:01:08 <-> 62:3a:d7:75:8a:08
[forwarding.l2_pairs    ] Installing 2a:7d:2a:5f:ba:9f <-> 86:38:69:c5:14:27
[forwarding.l2_pairs    ] Installing e2:ce:35:dd:f0:05 <-> 62:f2:d5:25:1a:4a
[openflow.of_01         ] 1 connection aborted
[openflow.of_01         ] [00-00-00-00-00-01 2] closed
[openflow.of_01         ] [00-00-00-00-00-04 3] closed
[openflow.of_01         ] [00-00-00-00-00-03 4] closed
[openflow.of_01         ] [00-00-00-00-00-02 5] closed
```

Figura 9. Ejecutando controlador POX

- Abra una nueva terminal y muévase a la carpeta de enrutadores
- Inicie la topología Mininet con:

sudo python3 router_topo.py

```
andres@andres-VirtualBox: ~/Downloads/SDN-proyecto/tol...
andres@andres-VirtualBox:~/Downloads/SDN-proyecto/tolerancia a fallos$ sudo pyth
on2 router_topo.py
[sudo] password for andres:
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Connecting to remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3 h4 r1 r2 r3 r4
*** Adding switches:
s1 s2 s3 s4
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (h4, s4) (r1, r2) (r1, r3) (r2, r4) (r3, r4) (s1, r1)
 (s2, r2) (s3, r3) (s4, r4)
*** Configuring hosts
h1 h2 h3 h4 r1 r2 r3 r4
RTNETLINK answers: File exists
*** Starting controller
c0
*** Starting 4 switches
s1 s2 s3 s4 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
```

Figura 10. Iniciando topología Mininet

- En la CLI de Mininet, ejecute:

Pingall

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 r1 r2 r3 r4
h2 -> h1 h3 h4 r1 r2 r3 r4
h3 -> h1 h2 h4 r1 r2 r3 r4
h4 -> h1 h2 h3 r1 r2 r3 r4
r1 -> h1 h2 h3 h4 r2 r3 r4
r2 -> h1 h2 h3 h4 r1 r3 r4
r3 -> h1 h2 h3 h4 r1 r2 r4
r4 -> h1 h2 h3 h4 r1 r2 r3
```

Figura 11. Resultados del pingall

VI. RESULTADOS

Los resultados obtenidos por la red SDN nos permite evidenciar que, si hay un error entre un enlace de los enrutadores, la red puede seguir funcionando correctamente para restablecer la conexión entre ellas.

Al ejecutar la topología con python2 lo primero que comprobamos es la conectividad en toda la red con el comando pingall, paralelamente ejecutamos el pox que viene por defecto en la carpeta pox. Cuando hay una falla en el en el enlace entre los enrutadores r1 y r2 la red sigue manteniendo todos sus enlaces adaptándose a partir del enlace roto, se puede comprobar que la ruta que tenia para acceder desde el host 1 al host 2 a cambiado porque la ruta mas corta (r1-r2) ha sido cortada pero el sistema permite que se recupere con esta falla, lo cual se puede comprobar haciendo nuevamente el comando pingall.

```
mininet> link r1 r2 down
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 r1 r2 r3 r4
h2 -> h1 h3 h4 r1 r2 r3 r4
h3 -> h1 h2 h4 r1 r2 r3 r4
h4 -> h1 h2 h3 r1 r2 r3 r4
r1 -> h1 h2 h3 h4 r2 r3 r4
r2 -> h1 h2 h3 h4 r1 r3 r4
r3 -> h1 h2 h3 h4 r1 r2 r4
r4 -> h1 h2 h3 h4 r1 r2 r3
*** Results: 0% dropped (56/56 received)
mininet> h1 traceroute h2
traceroute to 10.1.0.10 (10.1.0.10), 30 hops max, 60 byte packets
 1  10.0.0.1 (10.0.0.1)  0.380 ms  0.313 ms  0.301 ms
 2  10.101.0.2 (10.101.0.2)  0.290 ms  0.271 ms  0.258 ms
 3  10.102.0.2 (10.102.0.2)  0.243 ms  0.223 ms  0.208 ms
 4  10.103.0.1 (10.103.0.1)  0.194 ms  0.170 ms  0.153 ms
 5  10.1.0.10 (10.1.0.10)  0.136 ms  0.106 ms  0.087 ms
mininet> exit
*** Stopping 1 controllers
c0
*** Stopping 12 links
............
*** Stopping 4 switches
s1 s2 s3 s4
*** Stopping 8 hosts
h1 h2 h3 h4 r1 r2 r3 r4
```

Figura 12. Resultados del SDN

Los resultados obtenidos con el firewall permiten ver la aplicación de las reglas presentes dentro de su configuración que se puede ver en general como quedaron configurado con el comando pingall.

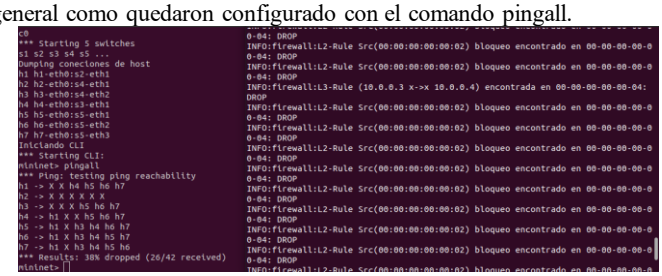

Figura 13. Resultados del firewall

Los resultados obtenidos con el balanceador de la carga utilizando el método de round robin que selecciona todos los paquetes enviados desde el mininet de manera equitativa y con un orden racional, que inicia por el primer host de la red hasta llegar al último y empezando nuevamente desde el primer host, hasta que todos los paquetes hayan sido recibidos.

Esta grafica se puede ver al salirse del mininet una grafica de como quedo distribuidos los paquetes de la red creada, utilizando pesos iguales 1,1,1,1 enviando 15 paquetes,

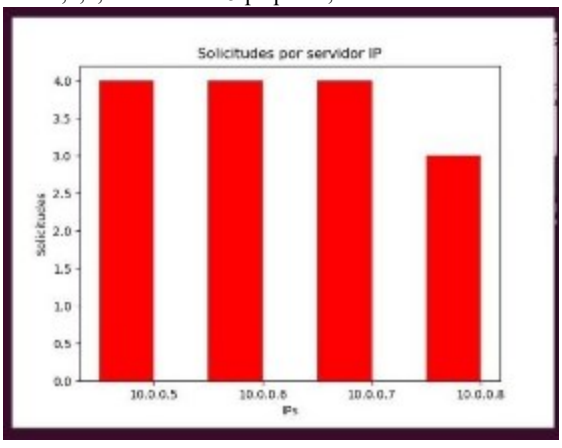

Figura 14. Resultados del balanceo de carga

VII. CONCLUSIONES

Gracias a la indagación sobre los temas relacionados para la elaboración de este proyecto como grupo logramos conocer la importancia de las tecnologías que ayudan a la ejecución de los procesos, tomando encuentra esto con el trabajo realizado se logró implementar el balance de cargas en la plataforma de ubuntu y mininet con ayuda del lenguaje Python en donde gracias al balanceo de carga se logra repartir los paquetes o peticiones que un usuario o varios usuarios pueden realizar y lo reparte de forma efectiva y eficiente.

Gracias a el buen número de documentación, información, repositorios y videos el equipo pudo contar con suficientes bases para la elaboración del proyecto, este proyecto nos ayudó a profundizar en temas estudiados en asignaturas diferentes a la de servicios telemáticos, además de que el proyecto tiene muchas más vías, mejoras e implementaciones diferentes que pueden utilizarse en pro de la mejora del servicio

VIII. REFERENCIAS

[1] S. M. Metev and V. P. Veiko, *Laser Assisted Microtechnology*, 2nd ed., R. M. Osgood, Jr., Ed. Berlin, Germany: Springer-Verlag, 1998.

[2] J. Breckling, Ed., *The Analysis of Directional Time Series: Applications to Wind Speed and Direction*, ser. Lecture Notes in Statistics. Berlin, Germany: Springer, 1989, vol. 61.

[3] F. Rivera, "Balanceo de carga en redes IPv4, un enfoque de Redes Definidas por Software", resumen extendido de, Universidad ORT Uruguay, Uruguay, 2015. [En línea]. Disponible: https://dspace.ort.edu.uy/bitstream/handle/20.500.11968/3200/Material%20completo.pdf?sequence=-1&isAllowed=y

[4] "A short walk-through of Mininet and POX". NUS Computing - Home. https://www.comp.nus.edu.sg/~tbma/teaching/cs4226y16_past/tutorial-Mininet-POX.pdf