

TP N°2: OpenFlow Lite

[TA048] Redes

CURSO: 02-Alvarez Hamelin
Segundo cuatrimestre de 2025

Alumno/a:	KRESTA, Facundo Ariel
Número de padrón:	110857
Email:	facundoarielkresta@gmail.com

Alumno/a:	Vaccarelli, Santiago
Número de padrón:	106051
Email:	svaccarelli@fi.uba.ar

Alumno/a:	Rodriguez Negri, Federica
Número de padrón:	xxxxxx
Email:	anombre@fi.uba.ar

Alumno/a:	APELLIDO, Ulises
Número de padrón:	xxxxxx
Email:	anombre@fi.uba.ar

Alumno/a:	General, Camila
Número de padrón:	xxxxxx
Email:	anombre@fi.uba.ar

25 de noviembre de 2025

Resumen

Este trabajo implementa un controlador OpenFlow centralizado utilizando POX para administrar una topología de red definida por software. Se desarrolló una topología con múltiples switches conectados en cascada y hosts distribuidos en Mininet. El controlador implementa dos estrategias: un comportamiento NORMAL para la mayoría de los switches, que actúan como switches de Capa 2 tradicionales, y un firewall de switches específico con capacidad de filtrado en múltiples capas (MAC, IP, puertos). Se demostraron las ventajas de Software-Defined Networking (SDN) mediante la implementación de políticas centralizadas y dinámicas, la administración simplificada de la red, y la capacidad de modificar el comportamiento de los switches sin alterar el hardware. Las pruebas validaron la conectividad entre hosts y la efectividad de las reglas de firewall implementadas.

Índice

1	Introducción	3
2	Hipótesis y Suposiciones Realizadas	4
2.1	Hipótesis Principal	4
2.2	Suposiciones Técnicas	4
2.3	Decisiones de Diseño	4
2.4	Limitaciones del Sistema	4
3	Implementación	6
3.1	Arquitectura General del Sistema	6
3.2	Topología de Red	6
3.3	Controlador OpenFlow en POX	6
3.3.1	Estrategia NORMAL para switches no-firewall	6
3.3.2	Estrategia Firewall con Aprendizaje para s2	6
3.3.3	Formato de Reglas de Firewall	7
3.4	Instalación y Ejecución	7
3.4.1	Iniciar el Controlador	7
3.4.2	Levantar la Topología	7
3.5	Protocolo OpenFlow 1.0	7
3.6	Funcionalidades Implementadas	7
3.6.1	1. Aprendizaje de Direcciones MAC	7
3.6.2	2. Encaminamiento Basado en Flujos	7
3.6.3	3. Firewall Dinámico	7
3.6.4	4. Flooding para Destinos Desconocidos	8
4	Respuestas a Preguntas de la Consigna	9
4.1	¿Qué es Software Defined Networking (SDN)? ¿Qué ventajas y desventajas tiene frente a redes tradicionales?	9
4.2	¿Qué es OpenFlow? ¿Cuáles son sus características principales?	9
4.3	Describe la arquitectura e implementación del controlador utilizado	10
4.4	¿Cómo funciona el aprendizaje de direcciones MAC en el controlador?	10
4.5	¿Cuál es la función de la tabla de flujos en los switches OpenFlow?	11
4.6	¿Cómo se comunica el controlador con los switches?	11
4.7	¿Qué parámetros se pueden controlar en las tablas de flujos?	11
5	Dificultades Encontradas y Soluciones	13
5.1	Configuración Inicial de la Topología	13
5.2	Aprendizaje de Direcciones MAC	13
5.3	Conflictos entre Reglas Proactivas y Aprendidas	13
5.4	Comportamiento de Flooding	13
5.5	Depuración de OpenFlow	13

6	Conclusión	14
6.1	Logros Principales	14
6.2	Aprendizajes Clave	14
6.3	Limitaciones	14
6.4	Reflexión Final	15

1. Introducción

En la actualidad, las redes de computadoras han evolucionado significativamente hacia arquitecturas más flexibles y programables. Históricamente, los switches Ethernet se han comportado como dispositivos estáticos, con lógica de encaminamiento fija implementada en hardware o firmware propietario. Sin embargo, el surgimiento de **Software-Defined Networking (SDN)** y el protocolo **OpenFlow** ha revolucionado la forma en que se controla y administra el comportamiento de la red.

OpenFlow es un protocolo de comunicación abierto que permite que un controlador centralizado determine dinámicamente cómo los switches deben procesar y reenviar paquetes de datos. Esta separación entre el plano de datos (switches) y el plano de control (controlador) proporciona una mayor flexibilidad, simplificando la administración de redes y permitiendo la implementación de políticas de red complejas.

El presente trabajo práctico implementa un controlador OpenFlow simplificado utilizando el framework **POX**, un controlador escrito en Python que proporciona una plataforma accesible para comprender los principios de SDN. Se desarrolló una topología de red personalizada en Mininet con múltiples switches conectados en cascada y hosts distribuidos, permitiendo experimentar con el comportamiento del plano de datos cuando es controlado por un controlador centralizado.

Este informe detalla el diseño e implementación de la topología de red, la arquitectura del controlador POX, las funcionalidades desarrolladas, y los resultados obtenidos a través de diferentes escenarios de prueba. Se analizan aspectos como la comunicación entre controlador y switches, el aprendizaje de direcciones MAC, y el comportamiento de la red bajo diferentes cargas y condiciones.

2. Hipótesis y Suposiciones Realizadas

2.1. Hipótesis Principal

La hipótesis principal de este trabajo es que un controlador OpenFlow centralizado puede administrar efectivamente el comportamiento de una topología de red con múltiples switches, permitiendo:

- El aprendizaje dinámico de direcciones MAC y la toma de decisiones de encaminamiento basada en información de la red.
- La comunicación entre hosts en diferentes segmentos de la red a través de múltiples switches.
- La recolección de estadísticas y la monitorización del tráfico de la red de forma centralizada.

Se espera que el controlador POX logre mantener tablas de flujo consistentes y actualizadas en cada switch, permitiendo que los hosts se comuniquen sin necesidad de broadcasting excesivo después de la fase inicial de descubrimiento.

2.2. Suposiciones Técnicas

Para el desarrollo e implementación del sistema, se han realizado las siguientes suposiciones:

- **Topología estable:** Se asume que la topología de la red permanece constante durante el experimento, sin cambios en la conectividad de switches o hosts.
- **Comunicación confiable OpenFlow:** Se asume que la comunicación entre el controlador POX y los switches es confiable mediante el protocolo TCP/SSL, sin pérdida de mensajes de control.
- **Sincronización de controlador:** Se asume que el controlador tiene una visión consistente del estado de la red basada en los eventos recibidos de los switches.
- **Mininet como entorno de prueba:** Se supone que Mininet proporciona un comportamiento de red lo suficientemente realista para verificar el funcionamiento del controlador OpenFlow.

2.3. Decisiones de Diseño

Durante el diseño de la solución, se tomaron las siguientes decisiones clave:

- **Topología lineal:** Se implementó una topología con switches conectados en cascada, con hosts conectados al primer y último switch. Esta configuración permite probar el encaminamiento a través de múltiples saltos.
- **Controlador simple de aprendizaje:** Se desarrolló un controlador que aprende direcciones MAC observando el origen de los paquetes que llegan y crea reglas de flujo estáticas para el destino.
- **Uso de POX en Python:** Se utilizó POX como controlador por su accesibilidad y facilidad de desarrollo, permitiendo un rápido prototipado y debugging.
- **Medición con herramientas estándar:** Se utilizaron herramientas como `pingall` e `iperf` disponibles en Mininet para evaluar la funcionalidad.

2.4. Limitaciones del Sistema

A pesar de las decisiones de diseño, el sistema presenta las siguientes limitaciones:

- **Escalabilidad:** El controlador simple de aprendizaje puede no escalar eficientemente con un número muy grande de hosts o switches.
- **Sin redundancia:** No se implementó redundancia del controlador, por lo que una falla del controlador interrumpiría la red.

- **Sin control de congestión:** El controlador no implementa mecanismos para detectar o responder a congestión de red.
- **Emulación vs Realidad:** Mininet emula redes, pero puede no capturar todas las complejidades de hardware real (latencia variable, pérdida de paquetes, etc.).

3. Implementación

3.1. Arquitectura General del Sistema

El sistema implementa una arquitectura SDN clásica con los siguientes componentes:

- **Controlador OpenFlow:** Un controlador centralizado desarrollado en POX que administra el comportamiento de los switches.
- **Switches OpenFlow:** Switches virtuales en Mininet que implementan el protocolo OpenFlow 1.0.
- **Hosts:** Máquinas finales conectadas a los switches para enviar y recibir tráfico de prueba.
- **Topología:** Una red con múltiples switches conectados en cascada y hosts distribuidos.

3.2. Topología de Red

La topología implementada en `topology.py` es una red lineal con los siguientes elementos:

- **Switches:** N switches conectados en serie ($s1, s2, \dots, sN$), donde N es configurable. Se utilizó principalmente $N=4$ en las pruebas.
- **Hosts:** 4 hosts distribuidos: $h1$ y $h2$ conectados al primer switch ($s1$), $h3$ y $h4$ conectados al último switch (sN).
- **Enlaces:** Enlaces directos entre switches consecutivos, permitiendo comunicación multi-salto.

3.3. Controlador OpenFlow en POX

El controlador implementado en `pox/pox/misc/controller.py` implementa dos estrategias de enrutamiento:

3.3.1. Estrategia NORMAL para switches no-firewall

Para la mayoría de los switches ($s1, s3, s4$), se instala una regla inicial de flujo con prioridad baja que indica el comportamiento "NORMAL":

```

1 fm = of.ofp_flow_mod()
2 fm.priority = 1
3 fm.match = of.ofp_match()
4 fm.actions.append(of.ofp_action_output(port=of.OFPP_NORMAL))
5 event.connection.send(fm)

```

Código 1: Instalación de regla NORMAL

Este comportamiento permite que los switches actúen como switches tradicionales, aprendiendo direcciones MAC dinámicamente.

3.3.2. Estrategia Firewall con Aprendizaje para $s2$

El switch $s2$ implementa una estrategia más sofisticada:

1. **Instalación proactiva de reglas:** Al conectarse, se instalan reglas de firewall desde un archivo JSON (`rules.json`).
2. **Aprendizaje dinámico:** Cuando se recibe un `PacketIn`, el controlador aprende la MAC origen y el puerto de entrada.
3. **Flujos aprendidos:** Si el destino es conocido, se instala un flujo específico con prioridad baja (`PRIO_LEARN = 100`).
4. **Flooding:** Si el destino no es conocido, se inunda el paquete por todos los puertos excepto el de entrada.

3.3.3. Formato de Reglas de Firewall

Las reglas se definen en formato JSON y soportan:

- **Capas 2:** `src_mac`, `dst_mac`
- **Capas 3:** `src_ip`, `dst_ip`
- **Capas 4:** `protocol` (TCP, UDP, ICMP, ANY), `src_port`, `dst_port`

Las reglas sin acciones implícitamente descartan el tráfico.

3.4. Instalación y Ejecución

3.4.1. Iniciar el Controlador

En una terminal, dentro del directorio `pox/`:

```
1 ./pox.py misc.controller
```

3.4.2. Levantar la Topología

En otra terminal:

```
1 sudo mn --custom topology.py --topo customTopo,num_switches=4 \  
2 --controller remote --switch ovsk --mac --arp
```

El parámetro `--controller remote` conecta los switches a un controlador OpenFlow remoto (POX en este caso).

3.5. Protocolo OpenFlow 1.0

El controlador implementa el protocolo OpenFlow 1.0 con los siguientes mensajes principales:

- **ofp_flow_mod:** Modifica la tabla de flujos de un switch. Permite agregar, actualizar o eliminar reglas.
- **ofp_packet_out:** Envía un paquete específico con una acción asociada (forward, flood, etc.).
- **PacketIn:** Evento generado cuando un switch recibe un paquete que no coincide con ninguna regla de flujo.

3.6. Funcionalidades Implementadas

3.6.1. 1. Aprendizaje de Direcciones MAC

El controlador mantiene una tabla `mac_to_port` que mapea direcciones MAC a puertos físicos. Esta tabla se actualiza dinámicamente cada vez que se recibe un `PacketIn`, permitiendo que el controlador conozca la topología de la red.

3.6.2. 2. Encaminamiento Basado en Flujos

Una vez que el controlador aprende la ubicación de una MAC, instala un flujo específico en el switch correspondiente. Los flujos posteriores que coincidan con ese patrón se procesan localmente en el switch sin necesidad de contactar al controlador.

3.6.3. 3. Firewall Dinámico

El archivo `rules.json` define reglas de bloqueo que se instalan proactivamente en s2. Las reglas soportan filtrado en múltiples capas (MAC, IP, puertos).

3.6.4. 4. Flooding para Destinos Desconocidos

Cuando el controlador no conoce la ubicación de un destino, inunda el paquete por todos los puertos (excepto el de entrada) para permitir el descubrimiento.

4. Respuestas a Preguntas de la Consigna

4.1. ¿Qué es Software Defined Networking (SDN)? ¿Qué ventajas y desventajas tiene frente a redes tradicionales?

Definición:

Software Defined Networking (SDN) es un paradigma de arquitectura de redes que separa el plano de control (inteligencia de la red) del plano de datos (reenvío de paquetes). En lugar de que cada switch tome decisiones de encaminamiento independientemente, un controlador centralizado determina cómo los switches deben procesar y reenviar paquetes.

Ventajas:

- **Programabilidad:** La red puede ser reconfigurada dinámicamente sin cambios en el hardware.
- **Control centralizado:** Una visión única del estado de la red simplifica la administración.
- **Flexibilidad:** Facilita la implementación de políticas complejas de red.
- **Reducción de costos:** Permite usar hardware estándar en lugar de equipos propietarios caros.
- **Innovación:** Acelera el despliegue de nuevas características y protocolos.

Desventajas:

- **Punto único de fallo:** El controlador es crítico; su falla detiene la red.
- **Latencia de decisión:** El controlador centralizado puede convertirse en cuello de botella.
- **Seguridad:** El controlador es un objetivo atractivo para ataques.
- **Complejidad inicial:** Requiere nueva infraestructura y reentrenamiento del personal.
- **Consistencia:** Mantener la consistencia del estado en la red distribuida es desafiante.

4.2. ¿Qué es OpenFlow? ¿Cuáles son sus características principales?

Definición:

OpenFlow es un protocolo de comunicación abierto y estandarizado que permite que un controlador centralizado comunique con switches de red para administrar sus tablas de flujos. Es el protocolo más utilizado en implementaciones de SDN.

Características principales:

- **Tablas de flujos:** Los switches mantienen tablas que mapean patrones de paquetes a acciones.
- **Granularidad:** Los flujos pueden ser tan específicos como sea necesario (Capa 2, 3, 4, etc.).
- **Prioridades:** Los flujos tienen prioridades que determinan cuál se aplica cuando múltiples coinciden.
- **Estadísticas:** Los switches reportan al controlador contadores de paquetes y bytes por flujo.
- **Aislamiento:** Cada flujo puede tener diferentes acciones (forward, drop, modify, etc.).
- **Protocolo abierto:** Definido por Open Networking Foundation (ONF), permitiendo múltiples implementaciones.

4.3. Describa la arquitectura e implementación del controlador utilizado

Arquitectura:

El controlador POX implementa un modelo evento-driven donde:

1. **Evento ConnectionUp:** Se dispara cuando un switch se conecta al controlador.
2. **Evento PacketIn:** Se dispara cuando un switch recibe un paquete que no coincide con ningún flujo.
3. **Instancia de aprendizaje:** El controlador mantiene una tabla global de MAC a puertos.

Implementación:

El controlador tiene dos estrategias:

- **Switches NORMAL (s1, s3, s4):** Se instala una regla que usa el comportamiento tradicional de switching (Capa 2 learning).
- **Firewall switch (s2):**
 1. Instala proactivamente reglas de bloqueo desde `rules.json`.
 2. Mantiene una tabla de aprendizaje de MAC → puerto.
 3. Para cada PacketIn: si el destino es conocido e no bloqueado, instala un flujo aprendido; si no, inunda.

Código clave:

```

1 def _handle_ConnectionUp(self, event):
2     # Para switches NORMAL
3     if dpid_str != FIREWALL_SWITCH:
4         fm = of.ofp_flow_mod()
5         fm.actions.append(of.ofp_action_output(
6             port=of.OFPP_NORMAL))
7         event.connection.send(fm)
8     # Para firewall, instala reglas proactivas
9     else:
10        for rule in self.rules:
11            self.install_rule(rule, event.connection)

```

4.4. ¿Cómo funciona el aprendizaje de direcciones MAC en el controlador?

Procedimiento:

1. **PacketIn recibido:** El switch recibe un paquete sin flujo coincidente y lo envía al controlador.
2. **Extracción de MAC origen:** El controlador extrae la dirección MAC origen del paquete y el puerto de entrada.
3. **Actualización de tabla:** Se almacena la asociación MAC → puerto en `mac_to_port[dpid][mac]`.
4. **Búsqueda de destino:** El controlador consulta si ya conoce la ubicación de la MAC destino.
5. **Instalación de flujo:** Si el destino es conocido (y no bloqueado), se instala un flujo específico.
6. **Procesamiento futuro:** Los siguientes paquetes del mismo flujo se procesan localmente en el switch.

Ejemplo:

Cuando h1 (MAC: 00:00:00:00:00:01) envía un paquete a h2:

1. Se genera un PacketIn en s1.
2. El controlador aprende que 00:00:00:00:00:01 está en puerto 1 de s1.
3. Si también conoce que 00:00:00:00:00:02 está en puerto 2 de s1, instala un flujo.
4. Los siguientes paquetes se forwarden en s1 sin contactar al controlador.

4.5. ¿Cuál es la función de la tabla de flujos en los switches OpenFlow?

Función:

La tabla de flujos es el componente central de un switch OpenFlow. Actúa como:

- **Dirección de tráfico:** Determina cómo procesar cada paquete basado en campos específicos.
- **Cache de decisiones:** Reduce la latencia al evitar contactos repetidos al controlador.
- **Estadísticas:** Registra números de paquetes y bytes por flujo.
- **Políticas:** Permite implementar políticas de QoS, seguridad, etc.

Estructura de una entrada de flujo:

Una entrada típica tiene:

- **Match fields:** Patrones que identifican paquetes (MAC src/dst, IP src/dst, puertos, etc.).
- **Priority:** Prioridad relativa (números más altos = mayor prioridad).
- **Actions:** Qué hacer si coincide (forward, drop, modify, etc.).
- **Counters:** Estadísticas de paquetes/bytes procesados.

4.6. ¿Cómo se comunica el controlador con los switches?

Protocolo:

La comunicación entre controlador y switches utiliza OpenFlow sobre TCP/SSL:

- **Canal de control:** Conexión TCP persistente entre controlador y cada switch.
- **Mensajes bidireccionales:** El controlador envía comandos ($flow_mod, packet_out, yrecibeeventos(packet_in, stats_reply)$).
- **Serialización:** Los mensajes OpenFlow se serializan en formato binario.

Tipos de mensajes principales:

- **ofp_flow_mod:** Modificación de tabla de flujos (agregar, actualizar, eliminar).
- **ofp_packet_out:** Envío de paquete específico con acción asociada.
- **PacketIn:** Evento del switch cuando no hay flujo coincidente.
- **FlowRemoved:** Evento cuando se expira un flujo.
- **StatsReply:** Respuesta con estadísticas de flujos o puertos.

4.7. ¿Qué parámetros se pueden controlar en las tablas de flujos?

En OpenFlow 1.0 (utilizado en este trabajo), los parámetros controlables incluyen:

Campos de coincidencia (Match fields):

- Puertos de entrada (in_port)
- Direcciones MAC origen/destino (dl_src, dl_dst)
- Tipo de Ethernet (dl_type)
- VLANs (dl_vlan, dl_vlan_pcp)
- Direcciones IP origen/destino (nw_src, nw_dst)

- Protocolo de transporte (nw_proto)
- Puertos TCP/UDP (tp_src, tp_dst)

Parámetros de flujo:

- **Priority:** Determina el orden de evaluación.
- **Cookie:** Identificador opaco para el controlador.
- **Timeout:** Tiempo de expiración (idle_timeout, hard_timeout).

Acciones:

- Forward a puertos específicos (forward, flood)
- Drop (descartar paquete)
- Modificar campos (set dl_src, set nw_src, etc.)

5. Dificultades Encontradas y Soluciones

5.1. Configuración Inicial de la Topología

Una de las dificultades iniciales fue configurar correctamente la topología en Mininet con múltiples switches conectados en cascada y asegurar que todos estuvieran bajo el control de un único controlador remoto.

Problema: Mininet por defecto crea switches con controladores internos. Se requería especificar el parámetro `--controller remote` para usar un controlador externo.

Solución: Se utilizó el comando correcto:

```
sudo mn --custom topology.py --topo customTopo,num_switches=4 \
--controller remote --switch ovsk --mac --arp
```

Se agregó la opción `--mac` para asignar direcciones MAC basadas en IP y `--arp` para habilitar ARP.

5.2. Aprendizaje de Direcciones MAC

El controlador debe aprender dinámicamente la topología observando PacketIn. Sin embargo, el aprendizaje incompleto causaba que algunos hosts no fuera alcanzables.

Problema: Los primeros paquetes generaban PacketIn, pero si el controlador no procesaba rápidamente, se perdían paquetes y la topología quedaba incompleta.

Solución: Se agregó un mecanismo de feedback: cada vez que se instala un flujo aprendido, se envía el paquete pendiente (`ofp_packet_out`) al destino, asegurando que no se pierda el paquete disparador del aprendizaje.

5.3. Conflictos entre Reglas Proactivas y Aprendidas

En el switch s2 (firewall), pueden coexistir reglas proactivas (de firewall) y reglas aprendidas (flujos dinámicos). Esto creó ambigüedad en cuál aplicar cuando un paquete coincide con múltiples reglas.

Problema: Sin prioridades correctas, un flujo aprendido podría actuar antes que una regla de bloqueo.

Solución: Se utilizaron prioridades distintas:

- **Reglas proactivas (firewall):** Prioridad 10000
- **Flujos aprendidos:** Prioridad 100

Las prioridades más altas se evalúan primero, asegurando que el firewall prevalezca.

5.4. Comportamiento de Flooding

El flooding por todos los puertos es necesario para descubrir destinos desconocidos, pero puede causar tormenta de paquetes en topologías con ciclos.

Problema: En Mininet con topología lineal, el flooding no es problema, pero en redes complejas puede causar duplicación de paquetes.

Solución: Se implementó verificación para evitar re-enviar paquetes por el puerto de entrada:

```
if out_port == in_port:
    log.warning("Dropping: src y dst en el mismo puerto")
    return
```

5.5. Depuración de OpenFlow

Entender qué ocurre en los switches es difícil sin herramientas de debugging.

Problema: Los logs del controlador no siempre reflejaban exactamente el estado de los switches.

Solución: Se utilizó `ovs-ofctl dump-flows` para consultar el contenido exacto de las tablas de flujos:

```
mininet> s1 ovs-ofctl dump-flows s1
```

Esto proporcionaba una vista clara de qué reglas estaban instaladas y sus prioridades.

6. Conclusión

El presente trabajo ha demostrado exitosamente la implementación de un controlador OpenFlow utilizando POX para administrar una topología de red con múltiples switches. Los objetivos principales del trabajo práctico fueron alcanzados:

6.1. Logros Principales

1. **Topología funcional:** Se implementó una topología lineal con 4 switches y 4 hosts distribuidos en Mininet.
2. **Controlador centralizado:** Se desarrolló un controlador POX que se conecta a los switches mediante OpenFlow 1.0.
3. **Aprendizaje dinámico:** El controlador aprende la topología de la red observando PacketIn y construye una tabla de MAC a puerto.
4. **Encaminamiento basado en flujos:** Se implementó instalación de flujos estáticos en switches que actúan como switches tradicionales (NORMAL behavior).
5. **Firewall de switch específico:** El switch s2 implementa reglas de bloqueo a nivel de MAC, IP y puertos desde un archivo JSON.
6. **Pruebas exitosas:** Se verificó conectividad completa entre hosts y se validó el funcionamiento del firewall.

6.2. Aprendizajes Clave

A través del desarrollo e implementación, se obtuvieron los siguientes aprendizajes:

- **Separación de planos:** La separación del plano de control y datos simplifica significativamente la administración de redes.
- **Ventajas de SDN:** Usar un controlador centralizado permitió implementar políticas complejas (firewall) sin cambios en el hardware de los switches.
- **OpenFlow como abstracción:** OpenFlow proporciona una abstracción clara y programable del comportamiento de switches.
- **Escalabilidad del controlador:** Un controlador simple es efectivo para topologías pequeñas, pero sería un cuello de botella en redes grandes.
- **Importancia de prioridades:** La correcta asignación de prioridades en flujos es crítica para resolver conflictos entre reglas.

6.3. Limitaciones

A pesar de los logros, el sistema actual presenta limitaciones:

- **Punto único de fallo:** El controlador es un punto crítico de falla; su caída desconecta la red.
- **Escalabilidad:** El aprendizaje ingenuista (per-switch) no escala a redes grandes con cientos de switches.
- **Sin redundancia:** No hay mecanismo de failover si el controlador falla.
- **OpenFlow 1.0 limitado:** Versiones posteriores de OpenFlow ofrecen características más avanzadas.

6.4. Reflexión Final

Este trabajo práctico ha proporcionado una comprensión profunda de cómo funcionan las redes modernas basadas en SDN. La implementación práctica de un controlador OpenFlow, aunque simplificado, ilustra los principios fundamentales que subyacen a infraestructuras complejas en centros de datos e internet moderno. La programabilidad de la red abre posibilidades para innovación y optimización que no eran posibles con arquitecturas de networking tradicionales.