

Trabajo Práctico 2

Software-Defined Networks -

Grupo 9

[75.43] Introducción a los Sistemas Distribuidos
Segundo cuatrimestre de 2025

ALUMNO	PADRON	CORREO
BARTOCCI, Camila	105781	cbartocci@fi.uba.ar
PATIÑO, Franco	105126	fpatino@fi.uba.ar
RETAMOZO, Melina	110065	mretamozo@fi.uba.ar
SAGASTUME, Matias	110530	csagastume@fi.uba.ar
SENDRA, Alejo	107716	asendra@fi.uba.ar

Índice

1. Introducción	2
2. Herramientas utilizadas	2
3. Implementación	2
3.1. Arquitectura General	2
3.2. Conexión Switch-Controlador	3
3.3. Topología Parametrizable	3
3.3.1. Diseño	3
3.3.2. Caso Especial: $N = 1$	4
3.3.3. Implementación Técnica	4
3.3.4. Topologías de Ejemplo	4
3.4. Controlador SDN	5
3.4.1. L2 Learning	5
3.4.2. Firewall	6
3.5. Reglas del Firewall Implementadas	6
3.5.1. Regla 1: Bloqueo de Puerto 80	7
3.5.2. Regla 2: Bloqueo Específico UDP	7
3.5.3. Regla 3: Bloqueo Bidireccional	7
3.6. Estructura del Proyecto	8
3.7. Decisiones de Diseño	8
3.7.1. Uso de L2 Learning Existente	8
3.7.2. Validación Extensiva	9
3.8. Scripts de Automatización	9
4. Pruebas y Validación	9
4.1. Configuración del Entorno de Pruebas	9
4.2. Verificación Básica: Conectividad	10
4.2.1. Prueba 1: PingAll	10
4.3. Pruebas de Firewall	10
4.3.1. Prueba 2: Bloqueo de Puerto 80 (HTTP) - Regla 1 . .	10
4.3.2. Prueba 3: Bloqueo UDP Específico - Regla 2	11
4.3.3. Prueba 4: Bloqueo Bidireccional - Regla 3	12
4.4. Pruebas con Diferentes Topologías	12
4.4.1. Topología con $N=1$	12
4.4.2. Topología con $N=3$	13
4.5. Análisis de Tráfico con Wireshark	13
4.6. Logs del Controlador	13
4.6.1. Inicialización del Firewall	13

4.6.2. Conexión de Switches	14
4.7. Resumen de Resultados	14
5. Preguntas a responder	15
5.1. ¿Cuál es la diferencia entre un switch y un router? ¿Qué tienen en común?	15
6. Conclusión	16

1. Introducción

2. Herramientas utilizadas

- **Mininet**: Emulación de la topología de red
- **POX**: Controlador SDN con L2 learning y firewall
- **iperf**: Generación y medición de tráfico TCP/UDP
- **Wireshark**: Captura y análisis de paquetes
- **ping**: Verificación de conectividad ICMP
- **curl/http.server**: Pruebas de tráfico HTTP

3. Implementación

La implementación del proyecto se dividió en tres componentes principales: la topología de red, el controlador SDN y las reglas del firewall. A continuación se detallan los aspectos técnicos de cada uno.

3.1. Arquitectura General

El sistema implementado sigue el paradigma de Software-Defined Networking (SDN), separando el plano de control del plano de datos. La arquitectura consta de:

- **Plano de Datos**: Switches OpenFlow emulados en Mininet, responsables del forwarding de paquetes según las reglas instaladas.
- **Plano de Control**: Controlador POX que implementa la lógica de L2 learning y firewall.

- **Canal de Comunicación:** Protocolo OpenFlow sobre TCP (puerto 6633).

3.2. Conexión Switch-Controlador

La comunicación entre los switches y el controlador se establece mediante el protocolo OpenFlow:

1. El controlador POX inicia un servidor TCP en el puerto 6633 (puerto estándar de OpenFlow).
2. Al crear la topología, Mininet configura cada switch con la dirección del controlador (127.0.0.1:6633).
3. Cada switch establece automáticamente una conexión TCP al controlador.
4. Una vez conectado, el controlador recibe un evento `ConnectionUp`, momento en el cual los módulos registrados (L2 learning y firewall) pueden instalar sus reglas en el switch.

Este canal de control se mantiene activo durante toda la simulación.

3.3. Topología Parametrizable

3.3.1. Diseño

Se implementó una topología de cadena (*chain topology*) parametrizable, donde el número de switches (N) puede ser definido por el usuario. La topología cumple con las siguientes características:

- **Switches:** N switches conectados linealmente formando una cadena:
 $S_1 - S_2 - \dots - S_N$
- **Hosts:** 4 hosts totales distribuidos en los extremos
 - h_1 y h_2 conectados al switch S_1 (primer extremo)
 - h_3 y h_4 conectados al switch S_N (último extremo)
- **Direcciones IP:** Asignación secuencial en la red 10.0.0.0/24
 - h_1 : 10.0.0.1
 - h_2 : 10.0.0.2
 - h_3 : 10.0.0.3
 - h_4 : 10.0.0.4

3.3.2. Caso Especial: $N = 1$

Cuando $N = 1$, todos los hosts se conectan al único switch disponible, formando una topología de estrella simple.

3.3.3. Implementación Técnica

La topología se implementó en Python usando la API de Mininet, definiendo una clase `ChainTopology` que hereda de `Topo`.

La clase maneja dinámicamente la creación de switches y sus enlaces.

3.3.4. Topologías de Ejemplo

A continuación se muestran diagramas de las topologías para diferentes valores de N :

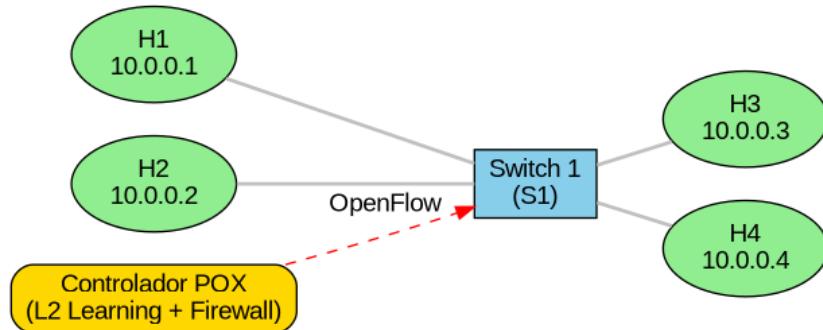


Figura 1: Topología con $N=1$ switches

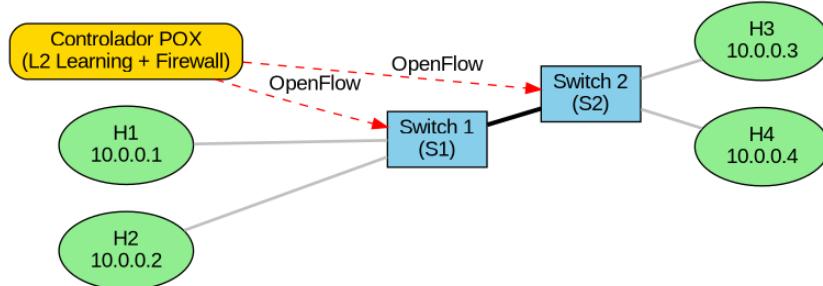


Figura 2: Topología con $N=2$ switches

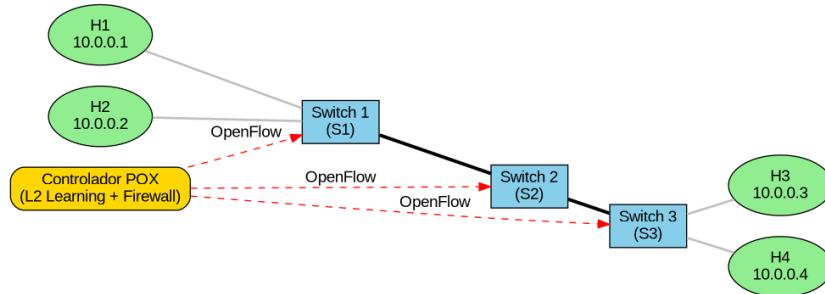


Figura 3: Topología con N=3 switches

3.4. Controlador SDN

El controlador se implementó usando POX, un framework de controladores SDN en Python. La implementación consta de dos módulos principales:

3.4.1. L2 Learning

Se utiliza el módulo `forwarding.l2_learning` de POX, que implementa el algoritmo de aprendizaje de direcciones MAC:

1. Al recibir un *PacketIn*, el switch no sabe cómo reenviar el paquete.
2. El controlador aprende la asociación MAC origen ↔ puerto.
3. Si conoce la MAC destino, instala una regla de flujo.
4. Si no la conoce, realiza *flooding* en todos los puertos (excepto el de entrada).

Este mecanismo permite que los switches aprendan dinámicamente la topología sin configuración manual.

3.4.2. Firewall

El firewall se implementó como un módulo POX custom en `controller/firewall.py`. Sus características principales son:

Instalación Proactiva de Reglas:

- Las reglas se instalan al momento de la conexión del switch (evento `ConnectionUp`).
- Se utiliza el mensaje `ofp_flow_mod` de OpenFlow para crear entradas en la tabla de flujos.

Estructura de Reglas:

Cada regla define un `ofp_match` con los siguientes campos posibles:

- `dl_type`: Tipo de Ethernet (0x0800 para IPv4)
- `nw_src / nw_dst`: Direcciones IP origen/destino
- `nw_proto`: Protocolo (TCP=6, UDP=17, ICMP=1)
- `tp_src / tp_dst`: Puertos TCP/UDP origen/destino

Las reglas sin acciones asociadas resultan en *DROP* implícito.

Validación de Reglas:

Se implementó un sistema robusto de validación en `controller/utils.py`:

- Validación de formato de direcciones IPv4
- Verificación de protocolos válidos (TCP, UDP, ICMP)
- Validación de rangos de puertos (1-65535)
- Verificación de requisitos de OpenFlow (ej: puerto requiere protocolo)
- Detección de errores comunes (ej: ICMP no puede tener puertos TCP/UDP)
- Logging detallado de reglas inválidas

Las reglas inválidas son ignoradas automáticamente, permitiendo que el sistema continúe funcionando.

3.5. Reglas del Firewall Implementadas

Las reglas se definen en el archivo `controller/firewall_rules.json`. Se implementaron las siguientes reglas según el enunciado:

3.5.1. Regla 1: Bloqueo de Puerto 80

Descripción: Bloquear todos los mensajes cuyo puerto destino sea 80.

Implementación: Se crearon dos reglas separadas, una para TCP y otra para UDP:

```
{
  "description": "Bloquear todo el tráfico al puerto 80",
  "protocol": "TCP",
  "dst_port": 80
}
```

Esta regla previene el acceso HTTP en cualquier dirección.

3.5.2. Regla 2: Bloqueo Específico UDP

Descripción: Bloquear mensajes desde host 1 (10.0.0.1) con puerto destino 5001 y protocolo UDP.

```
{
  "description": "Bloquear UDP desde host 1 con puerto destino 5001",
  "src_ip": "10.0.0.1",
  "protocol": "UDP",
  "dst_port": 5001
}
```

Esta regla es más específica, bloqueando solo el tráfico UDP desde un host particular a un puerto específico.

3.5.3. Regla 3: Bloqueo Bidireccional

Descripción: Impedir comunicación en ambas direcciones entre host 2 (10.0.0.2) y host 3 (10.0.0.3).

Implementación: Se requieren dos reglas para bloquear ambas direcciones:

```
{
  "description": "Bloquear comunicación de host 2 a host 3",
  "src_ip": "10.0.0.2",
  "dst_ip": "10.0.0.3"
},
{
  "description": "Bloquear comunicación de host 3 a host 2",
  "src_ip": "10.0.0.3",
  "dst_ip": "10.0.0.2"
}
```

Esto asegura el bloqueo total de comunicación entre ambos hosts, sin importar quién inicie la conexión.

3.6. Estructura del Proyecto

El código fuente se organizó de la siguiente manera:

```
 proyecto/
  |-- controller/          # Módulos SDN custom
```

```
|   |-- __init__.py
|   |-- firewall.py      # Implementación del firewall
|   |-- utils.py         # Validación de reglas
|   '-- firewall_rules.json # Reglas del firewall
|-- pox/                 # POX (instalado externamente)
|-- topology.py        # Topología Mininet
|-- run_controller.sh   # Script para ejecutar POX
|-- run_topology.sh     # Script para ejecutar Mininet
|-- install_pox.sh       # Script de instalación
`-- README.md           # Documentación
```

3.7. Decisiones de Diseño

3.7.1. Uso de L2 Learning Existente

Se decidió utilizar el módulo `forwarding.l2_learning` de POX en lugar de implementar uno desde cero. Esta decisión se basó en:

- Cumplimiento de requisitos (el enunciado permite usar módulos de POX)
- Robustez y optimización del módulo oficial
- Enfoque en la implementación del firewall (objetivo principal)
- Reducción de complejidad y tiempo de desarrollo

3.7.2. Validación Extensiva

Se implementó un sistema de validación completo para prevenir errores comunes:

- Evita reglas que generen warnings de OpenFlow
- Proporciona retroalimentación clara sobre errores
- Permite que el sistema continúe funcionando con reglas válidas
- Facilita el debugging y la depuración

3.8. Scripts de Automatización

Se desarrollaron scripts bash para facilitar la ejecución:

- `install_pox.sh`: Clona e instala POX desde GitHub
- `run_controller.sh`: Ejecuta POX con los módulos necesarios
- `run_topology.sh`: Lanza Mininet con la topología parametrizable

Estos scripts incluyen validaciones y mensajes de error informativos.

4. Pruebas y Validación

Esta sección describe las pruebas realizadas para verificar el correcto funcionamiento de la topología, el controlador SDN y las reglas del firewall.

4.1. Configuración del Entorno de Pruebas

Todas las pruebas se realizaron siguiendo el siguiente procedimiento:

1. Iniciar el controlador (Terminal 1):

```
./run_controller.sh
```

2. Iniciar la topología (Terminal 2):

```
./run_topology.sh 2      # Para N=2 switches
```

3. Esperar a que los switches se conecten y las reglas se instalen.

Importante: El controlador debe iniciarse *antes* que la topología para que los switches puedan conectarse al momento de su creación.

4.2. Verificación Básica: Conectividad

4.2.1. Prueba 1: PingAll

Objetivo: Verificar que todos los hosts pueden comunicarse entre sí (excepto donde el firewall lo impide).

Comando:

```
mininet> pingall
```

Resultado Esperado:

- $h_1 \leftrightarrow h_2$: Conectividad exitosa
- $h_1 \leftrightarrow h_3$: Conectividad exitosa
- $h_1 \leftrightarrow h_4$: Conectividad exitosa
- $h_2 \leftrightarrow h_3$: Bloqueado (Regla 3)
- $h_2 \leftrightarrow h_4$: Conectividad exitosa
- $h_3 \leftrightarrow h_4$: Conectividad exitosa

Logs del Controlador:

[Insertar captura de logs mostrando instalación de reglas]

Captura de Wireshark:

[Insertar captura de Wireshark mostrando paquetes ICMP]

4.3. Pruebas de Firewall

4.3.1. Prueba 2: Bloqueo de Puerto 80 (HTTP) - Regla 1

Objetivo: Verificar que todo el tráfico al puerto 80 es bloqueado.

Configuración:

```
# Terminal h1 - Iniciar servidor HTTP en puerto 80
mininet> h1 python3 -m http.server 80 &
```

```
# Terminal h4 - Intentar acceder desde otro host
mininet> h4 curl -m 5 http://10.0.0.1:80
```

Resultado Esperado: Timeout (conexión bloqueada por firewall)

Resultado Obtenido:

[Insertar salida del comando curl mostrando timeout]

Verificación con iperf:

```
# Servidor en h1
mininet> h1 iperf -s -p 80 &

# Cliente en h4
mininet> h4 iperf -c 10.0.0.1 -p 80 -t 5
```

Resultado: No se establece conexión.

Prueba de Control (puerto no bloqueado):

```
mininet> h1 python3 -m http.server 8000 &
mininet> h4 curl http://10.0.0.1:8000
```

Resultado: Conexión exitosa (el puerto 8000 no está bloqueado).

4.3.2. Prueba 3: Bloqueo UDP Específico - Regla 2

Objetivo: Verificar que el tráfico UDP desde h_1 al puerto 5001 es bloqueado.

Configuración:

```
# Servidor UDP en h4
mininet> h4 iperf -s -u -p 5001 &

# Cliente desde h1 (bloqueado)
mininet> h1 iperf -c 10.0.0.4 -u -p 5001 -t 5

# Cliente desde h2 (permitido)
mininet> h2 iperf -c 10.0.0.4 -u -p 5001 -t 5
```

Resultados Esperados:

- Desde h_1 : 0 % de paquetes recibidos (bloqueado)
- Desde h_2 : 100 % de paquetes recibidos (permitido)

Resultados Obtenidos:

[Insertar salida de iperf mostrando los porcentajes]

4.3.3. Prueba 4: Bloqueo Bidireccional - Regla 3

Objetivo: Verificar que h_2 y h_3 no pueden comunicarse en ninguna dirección.

Test 1: ICMP (ping)

```
mininet> h2 ping -c 4 h3  
mininet> h3 ping -c 4 h2
```

Resultado Esperado: 100% packet loss en ambas direcciones.

Test 2: TCP (iperf)

```
mininet> h3 iperf -s &  
mininet> h2 iperf -c 10.0.0.3 -t 5
```

Resultado Esperado: No se establece conexión TCP.

Resultado Obtenido:

[Insertar salida mostrando que no hay conectividad]

Prueba de Control:

```
mininet> h2 ping -c 4 h1      # Debería funcionar  
mininet> h3 ping -c 4 h4      # Debería funcionar
```

Resultado: Conectividad exitosa (solo está bloqueada la comunicación entre h_2 y h_3).

4.4. Pruebas con Diferentes Topologías

4.4.1. Topología con N=1

Configuración:

```
./run_topology.sh 1
```

Características: Todos los hosts conectados al mismo switch (topología estrella).

Resultado: Las reglas del firewall se aplican correctamente independientemente de la topología.

4.4.2. Topología con N=3

Configuración:

```
./run_topology.sh 3
```

Características: Cadena de 3 switches, aumenta la distancia entre hosts extremos.

Resultado:

- L2 learning funciona correctamente en topología extendida
- Firewall mantiene las reglas en todos los switches
- Latencia aumenta proporcionalmente con el número de saltos

4.5. Análisis de Tráfico con Wireshark

[Insertar captura de Wireshark mostrando PacketIn]

Figura 4: Mensaje PacketIn de OpenFlow al controlador

[Insertar captura de Wireshark mostrando FlowMod]

Figura 5: Mensaje FlowMod instalando regla de firewall

5. Preguntas a responder

5.1. ¿Cuál es la diferencia entre un switch y un router? ¿Qué tienen en común?

Los switches y los routers son dispositivos esenciales para que una red funcione. Ambos reciben paquetes, toman decisiones de reenvío y permiten que distintos equipos se comuniquen.

Sin embargo, presentan diferencias claras:

1. **Capa en la que operan:** Un switch trabaja en la capa de enlace de datos, utiliza direcciones MAC y una tabla de conmutación para decidir por qué puerto enviar cada frame. Un router opera en la capa de red, utiliza direcciones IP y una tabla de enrutamiento para determinar la mejor ruta hacia otras redes.

2. **Tipo de comunicación que habilitan:** El switch conecta dispositivos dentro de la misma red local (LAN). El router conecta diferentes redes entre sí (LAN-LAN, LAN-Internet), permitiendo alcanzar destinos externos.
3. **Reenvío de datos:** El switch aprende qué dirección MAC está asociada a cada puerto y envía el frame únicamente al puerto correspondiente. El router requiere configuración IP y emplea algoritmos de enrutamiento para decidir la ruta óptima. Al procesar información de la capa de red, el reenvío suele ser más costoso computacionalmente.
4. **Funciones de seguridad y control:** El switch ofrece controles básicos a nivel de puerto. El router puede aplicar reglas más avanzadas (como NAT o firewall), gracias a que analiza información de las capas de red y transporte.

En cuanto a sus similitudes:

1. Ambos son dispositivos de conmutación de paquetes (store-and-forward).
2. Ambos toman decisiones de reenvío utilizando una tabla interna (MAC table / routing table).
3. Forman parte crítica de la infraestructura de red.
4. En un entorno SDN, ambos pueden administrarse desde un controlador central que define cómo deben manejar el tráfico.

6. Conclusión