Redes

(TA048) Curso Alvarez Hamelin

Trabajo Práctico 2
OpenFlow

27 de noviembre de 2025

Máximo Augusto Calderón Vasil
111810

Alen Monti
108081

Ulises Valentín Tripaldi
111919

Jose Ignacio Adelardi
111701

1. Introducción

El crecimiento exponencial de Internet en las últimas décadas ha planteado desafíos sin precedentes para la gestión y administración de las redes de comunicación. Desde sus inicios en los años 60, la infraestructura de Internet se ha basado en un paradigma relativamente rígido, donde routers y switches funcionaban siguiendo reglas predefinidas, principalmente orientadas a la dirección destino de los paquetes. Sin embargo, la aparición masiva de dispositivos móviles, la proliferación de contenidos multimedia y la consolidación de grandes centros de datos han generado un aumento significativo del tráfico y una mayor diversidad en los requerimientos de red. Esta evolución ha puesto de manifiesto la necesidad de redes más flexibles, programables y capaces de adaptarse dinámicamente a las políticas de control definidas por los administradores.

En el marco de la evolución constante de las redes de comunicación, la creciente demanda de flexibilidad, escalabilidad y control motivó la aparición de nuevas arquitecturas de gestión basadas en software. En este contexto surge el concepto de Software-Defined Networking (SDN), una aproximación cuyo objetivo principal es desacoplar el plano de control del plano de datos para permitir una gestión centralizada, programable y dinámica de la infraestructura de red. A diferencia de las arquitecturas tradicionales, donde los dispositivos como switches y routers operan de forma autónoma aplicando lógicas preestablecidas por sus fabricantes y dependen casi exclusivamente de su hardware para la toma de decisiones, las SDN permiten que dichos dispositivos sean programados mediante controladores externos que dictan cómo deben manejarse los flujos de datos. Esta separación no solo incrementa la flexibilidad y eficiencia en la gestión de la red, sino que también habilita nuevas aplicaciones como la implementación de firewalls, balanceadores de carga y políticas de enrutamiento avanzadas, sin necesidad de modificar el hardware subyacente. Además, otorga un grado de adaptación y automatización antes inalcanzable, permitiendo definir estrategias de encaminamiento, seguridad y tratamiento del tráfico mediante software.

La tecnología OpenFlow constituye uno de los pilares fundamentales del paradigma SDN, al proporcionar un protocolo estandarizado mediante el cual un controlador remoto puede interactuar directamente con las tablas de flujo de los switches. Estas tablas contienen entradas que representan flujos de tráfico y definen las acciones que deben ejecutarse sobre los paquetes que coinciden con dichos patrones. OpenFlow permite modificar dinámicamente esas tablas, habilitando la aplicación de políticas específicas como el bloqueo de tráfico (por ejemplo, mediante IP blackholing), la reescritura de cabeceras o el balanceo de carga. Esto abre la posibilidad de implementar estrategias avanzadas de ingeniería de tráfico y mecanismos de seguridad, tales como firewalls flexibles, detección de anomalías o tratamiento diferenciado por flujo, basados en características más allá de los campos tradicionales como la dirección IP destino. Tales capacidades resultan prácticamente imposibles de alcanzar en redes convencionales basadas exclusivamente en hardware.

El presente trabajo práctico tiene como objetivo comprender los fundamentos del paradigma SDN y su interacción con los distintos componentes involucrados —Mininet como simulador de red, POX como controlador y OpenFlow como protocolo de control— mediante la construcción de topologías dinámicas y parametrizables, así como la implementación de un firewall. A través del diseño e implementación de un controlador personalizado capaz de gestionar flujos de datos mediante reglas definidas dinámicamente. Se realizarán una serie de pruebas, mediante las cuales se medirá el funcionamiento de la red (con la topología implementada) bajo condiciones normales y frente a escenarios de bloqueo, utilizando herramientas como Wireshark e Iperf para la verificación y análisis de los resultados.

2. Consideraciones

Para el desarrollo y análisis del presente trabajo práctico se establecen ciertas consideraciones con el objetivo de definir un marco de análisis controlado y garantizar la coherencia en la interpretación de los resultados. Se considera que todos los switches y hosts involucrados en la simulación operan de manera confiable, sin generar pérdidas de paquetes ni retrasos significativos que puedan alterar los resultados. Las posibles variaciones en el tiempo de procesamiento de los paquetes

se limitan a lo esperado en un entorno de simulación, y no se contemplan fallas de hardware o interrupciones inesperadas. Asimismo, se asume que la API de POX proporciona todas las funcionalidades necesarias para implementar la topología de la red, instalar reglas en las tablas de flujo de los switches y recolectar registros completos para su posterior análisis con herramientas como Wireshark. Se presupone que el controlador opera de manera continua y puede manejar múltiples eventos simultáneamente sin pérdida de información.

En cuanto a los enlaces de red, se asume que son simétricos, con latencias equivalentes en ambas direcciones, y que el ancho de banda disponible es suficiente para evitar cuellos de botella durante las pruebas de conectividad (pingall) y rendimiento (iperf). No se consideran interferencias externas ni limitaciones físicas que puedan afectar el comportamiento de la topología. De la misma manera, se asume que no existen ataques externos ni situaciones de red anómalas que puedan comprometer la correcta ejecución de las reglas definidas en las tablas de flujo del firewall. Todo el tráfico que circula por la topología es generado y controlado dentro del entorno de simulación, garantizando que las políticas de filtrado y control de flujos se apliquen de manera íntegra y predecible.

Estas consideraciones permiten establecer un entorno de laboratorio controlado, centrando el análisis en el comportamiento de las SDN y OpenFlow bajo condiciones ideales, y asegurando que los resultados obtenidos reflejen fielmente el funcionamiento de las herramientas y políticas implementadas.

3. Implementación

La solución desarrollada para este trabajo práctico se estructuró en torno a la integración de Mininet con un controlador OpenFlow basado en POX, siguiendo el paradigma de Software-Defined Networks (SDN). La topología de la red se define en el módulo topología.py, donde se implementa una estructura en cadena parametrizable (ChainTopo). Esta topología consta de dos hosts en cada extremo y un número variable de switches intermedios conectados en serie. Cada host se instancia con una dirección IP única, mientras que los switches se conectan entre sí y con los hosts mediante enlaces definidos en el constructor de la topología.

En cuanto a la decisión de dónde situar el firewall, optamos por elegir el switch que se conecta con los hosts hL1 y hL2 y con el siguiente switch en la cadena (en caso de haberlo). Esta decisión nos forzó a establecer el mínimo de switches de la cadena en una unidad, fijando así que estos dos hosts sean los que no se puedan comunicar entre sí. De esta forma, cualquier tráfico entre hL1 y hL2 pasa necesariamente por el switch que contiene el firewall, no habiendo forma de que se puedan comunicar entre ellos.

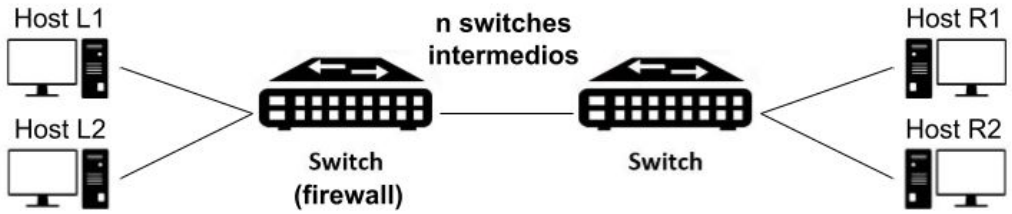

Figura 1: Topología de la red implementada.

El script init_mininet.py se encarga de inicializar la simulación en Mininet, creando la topología mediante ChainTopo y conectando los switches a un controlador remoto (RemoteController) ejecutándose en 127.0.0.1:6633, donde se ejecuta POX por defecto. Una vez levantada la red, se realiza un test de conectividad general mediante el comando pingAll y se habilita la interfaz de línea de comandos de Mininet (CLI) para permitir la interacción manual con la topología, la ejecución de pruebas y la observación del comportamiento del tráfico.

El comportamiento del firewall se implementó en pox_firewall.py. Este módulo funciona como una aplicación del controlador POX y gestiona las tablas de flujo de los switches de forma dinámica,

aplicando las reglas definidas en el archivo firewall_rules.json. Al iniciarse, el firewall carga la configuración y, al recibir la conexión de un switch, verifica si este está seleccionado para aplicar las políticas. Posteriormente, se insertan las reglas en la tabla de flujo, permitiendo bloquear o permitir tráfico según los campos de la cabecera y ejecutando las acciones definidas en el archivo de configuración. Esto asegura un control centralizado y dinámico del flujo de paquetes en la red simulada.

Para la validación y prueba de las reglas de firewall se desarrolló el script test_firewall.py. Este script automatiza pruebas de conectividad y de filtrado de tráfico utilizando iperf y ping, verificando que los paquetes sean bloqueados o permitidos de acuerdo con las políticas definidas. Se implementan distintos escenarios, incluyendo el bloqueo de tráfico hacia puertos específicos, la restricción de comunicación entre hosts determinados y la validación de tráfico permitido hacia puertos abiertos.

4. Pruebas

Con el objetivo de testear el correcto funcionamiento tanto de nuestra implementación de la topología como del firewall, realizamos una serie de pruebas sobre las distintas reglas que fueron aplicadas al firewall utilizando la aplicación mininet para poder simular la red. Para corroborar el éxito, hicimos uso de Wireshark para visualizar los paquetes que llegan/salen de un host e iperf para verificar que el tráfico permitido fluya correctamente mientras que el bloqueado no genere transferencia de datos.

Para aplicar la siguiente regla solicitada: *"Se deben elegir dos hosts cualesquiera y los mismos no deben poder comunicarse de ninguna forma."*, optamos por elegir arbitrariamente que los hosts hL1 y hL2 no se puedan comunicar de ninguna manera. Dicha restricción es corroborada mediante la realización de una prueba de ping entre ambos hosts. A continuación se observa como se muestra en la siguiente captura de la terminal de mininet la primera de ellas:

```
mininet> hL1 ping -c 3 hL2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.

--- 10.0.0.2 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2042ms
```

Figura 2: Captura de la terminal de mininet corriendo un ping desde hL1 hasta hL2.

Como se puede observar en las sucesivas dos capturas de Wireshark desde el host hL1 salieron tres paquetes ICMP que fueron enviados por el ping con flag -c para limitar la cantidad de paquetes. Sin embargo en el host hL2 no se ve ningún arribo de paquetes desde 10.0.0.1 (IP de hL1), por lo que se comprueba que la comunicación desde hL1 hasta hL2 está bloqueada.

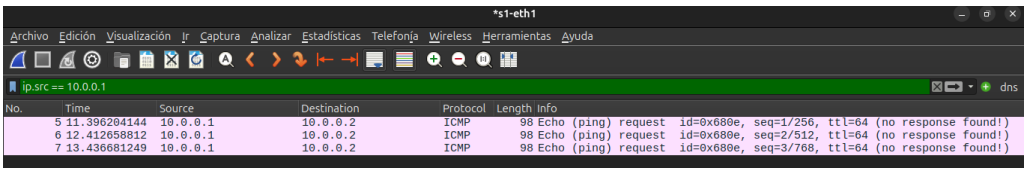

Figura 3: Captura de Wireshark del host hL1 durante la prueba.

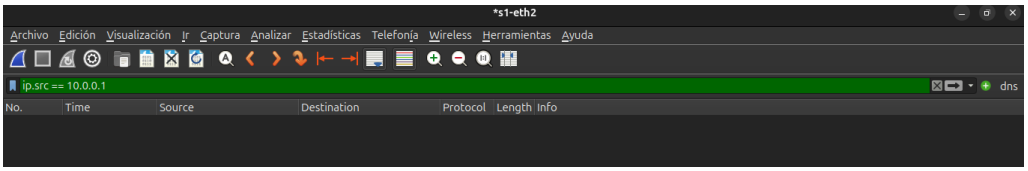

Figura 4: Captura de Wireshark del host hL2 durante la prueba.

La prueba análoga enviando tres paquetes ICMP de manera inversa, es decir desde hL2 a hL1, se puede observar en las siguientes imágenes. Las mismas arrojan el mismo resultado satisfactorio.

```
mininet> hL2 ping -c 3 hL1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.

--- 10.0.0.1 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2053ms
```

Figura 5: Captura de la terminal de mininet corriendo un ping desde hL2 hasta hL1.

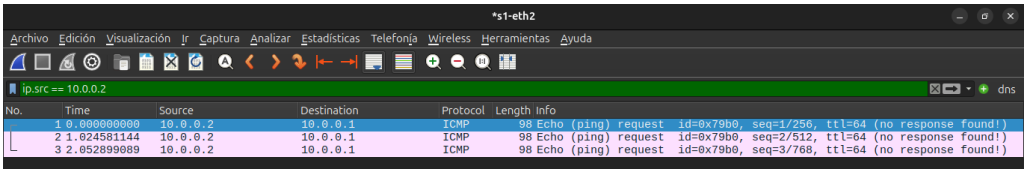

Figura 6: Captura de Wireshark del host hL2 durante la prueba.

Figura 7: Captura de Wireshark del host hL1 durante la prueba.

Para corroborar la siguiente regla solicitada: *"Se deben descartar todos los mensajes cuyo puerto destino sea 80."*, optamos por elegir dos hosts que no fueran hL1 y hL2, ni hR1 y hR2 en pareja, dado que el firewall se encuentra en el switch que une a hL1 y hL2, que no es utilizado en la ruta que conecta a hR1 con hR2. Entonces elegimos realizar la prueba sobre hL1 y hR1 (IP 10.0.0.3), enviando paquetes tcp syn al puerto 80, haciendo uso de la herramienta iperf y seteando un timeout de cinco segundos. A continuación se observan tanto la salida de línea de comandos, que muestra que se está intentando iniciar la conexión tcp mediante el 3-way-handshake, como las capturas de Wireshark evidenciando los paquetes syn y retransmission enviados por hL1 que no arribaron al destino y por lo tanto no obtuvieron respuesta.

```
mininet> hL1 timeout -s KILL 5s iperf -c 10.0.0.3 -p 80 -t 4
------------------------------------------------------------
Client connecting to 10.0.0.3, TCP port 80
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
```

Figura 8: Captura de la terminal de mininet corriendo iperf desde hL1 hasta 10.0.0.3 (IP de hR1).

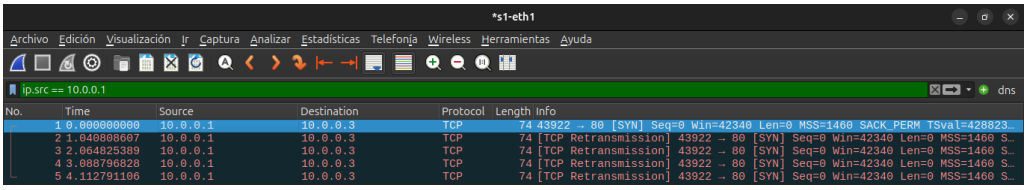

Figura 9: Captura de Wireshark del host hL1 durante la prueba.

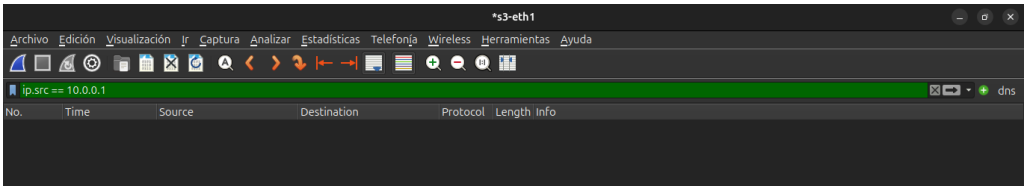

Figura 10: Captura de Wireshark del host hR1 durante la prueba.

Para mayor completitud, se muestran las capturas correspondientes a la prueba análoga a la anterior, invirtiendo el origen y el destino de la conexión, pero utilizando el protocolo de transporte UDP.

```
mininet> hR1 timeout -s KILL 5s iperf -c 10.0.0.1 -p 80 -u -t 4
------------------------------------------------------------
Client connecting to 10.0.0.1, UDP port 80
Sending 1470 byte datagrams, IPG target: 11215.21 us (kalman adjust)
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  1] local 10.0.0.3 port 60996 connected with 10.0.0.1 port 80
[ ID] Interval       Transfer     Bandwidth
[  1] 0.0000-4.0152 sec   517 KBytes  1.05 Mbits/sec
[  1] Sent 361 datagrams
```

Figura 11: Captura de la terminal de mininet corriendo iperf desde hR1 hasta 10.0.0.1 (IP de hL1).

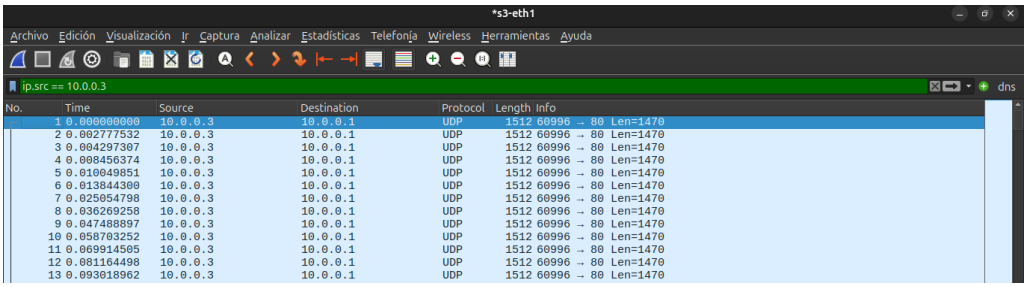

Figura 12: Captura de Wireshark del host hR1 durante la prueba.

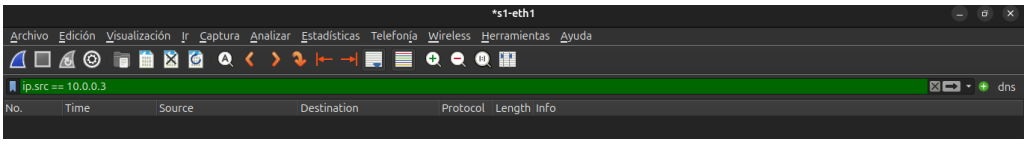

Figura 13: Captura de Wireshark del host hL1 durante la prueba.

Luego de realizar de ambas pruebas de la presente regla, podemos concluir que sin importar el protocolo de capa de transporte utilizado ni el origen ni destino (excluyendo los casos previamente mencionados de hosts conectados al mismo switch, por las restricciones de la topología), los paquetes con puerto destino 80 son descartados por el firewall.

Para corroborar la siguiente regla solicitada: *"Se deben descartar todos los mensajes que provengan del host 1 (hL1), tengan como puerto destino el 5001, y estén utilizando el protocolo UDP."*, optamos por elegir dos hosts que no fueran hL1 y hL2, dado que el firewall mediante otra regla ya bloquea el tráfico entre ellos. Entonces elegimos realizar la prueba sobre hL1 y hR2 (IP 10.0.0.4), enviando paquetes UDP al puerto 5001, haciendo uso de la herramienta iperf y seteando un timeout de cinco segundos. A continuación se aprecian las imágenes que convalidan lo planteado, observándose tráfico saliente de hL1 que jamás arriba al host hR2.

```
mininet> hL1 timeout -s KILL 5s iperf -c 10.0.0.4 -p 5001 -u -t 4
------------------------------------------------------------
Client connecting to 10.0.0.4, UDP port 5001
Sending 1470 byte datagrams, IPG target: 11215.21 us (kalman adjust)
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  1] local 10.0.0.1 port 34271 connected with 10.0.0.4 port 5001
[ ID] Interval       Transfer     Bandwidth
[  1] 0.0000-4.0152 sec   517 KBytes  1.05 Mbits/sec
[  1] Sent 361 datagrams
```

Figura 14: Captura de la terminal de mininet corriendo iperf desde hL1 hasta 10.0.0.4 (IP de hR2).

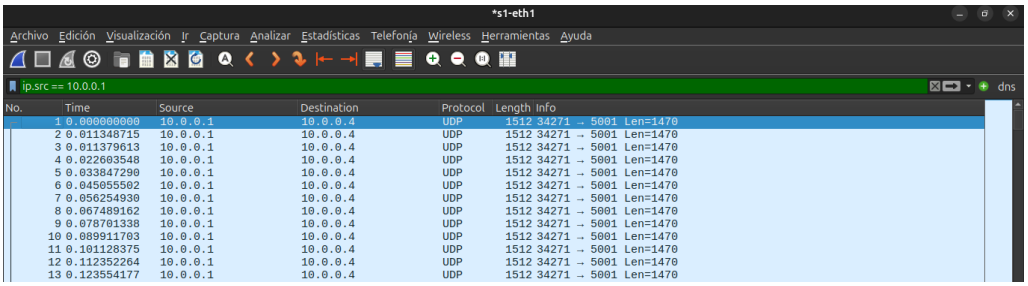

Figura 15: Captura de Wireshark del host hL1 durante la prueba.

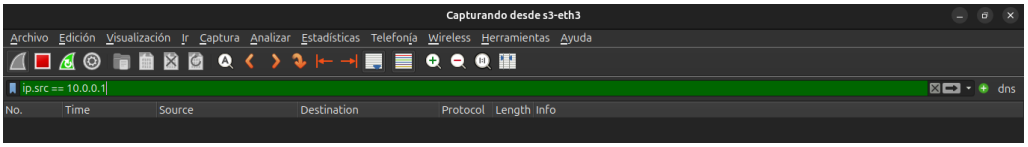

Figura 16: Captura de Wireshark del host hR2 durante la prueba.

Finalmente, adjuntamos algunos casos donde el firewall no bloquea el tráfico entre los hosts, a modo de demostración del correcto funcionamiento del tráfico no abarcado en las reglas predefinidas.

En el primer caso, ejemplificamos una conexión tcp entre hL1 y hR2 a través del puerto destino 5001. Como se puede dilucidar, el firewall no bloquea dicho envío, por lo que el syn enviado desde el origen es recibido en el destino. A pesar de que el puerto 5001 sí es una de las restricciones de las reglas planteadas, se pone en juego cuando se conjuga con el protocolo de capa de transporte udp y el host de origen hL1.

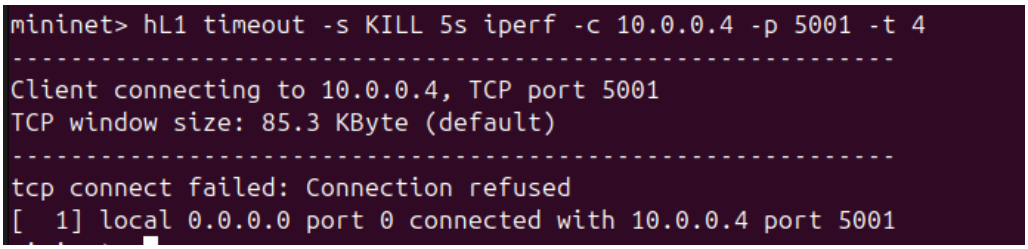

Figura 17: Captura de la terminal de mininet corriendo iperf desde hL1 hasta 10.0.0.4 (IP de hR2).

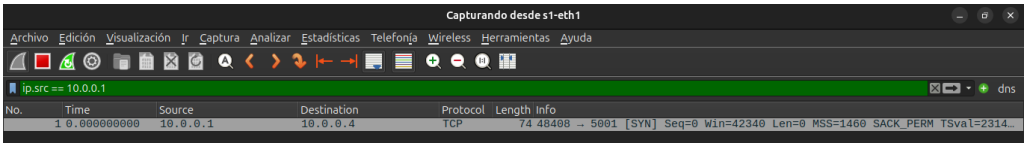

Figura 18: Captura de Wireshark del host hL1 durante la prueba.

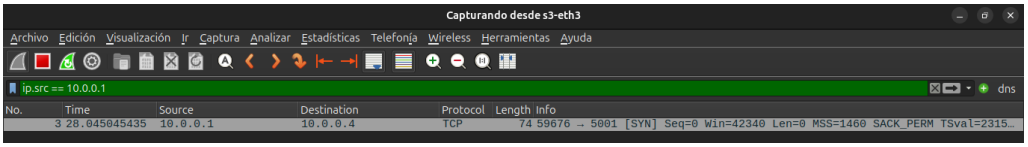

Figura 19: Captura de Wireshark del host hR2 durante la prueba.

Por último, probaremos el envío de paquetes de hR1 hacia hR2 (IP 10.0.0.4). Dadas las reglas y dónde está ubicado el firewall, ningún tipo de tráfico entre hR1 y hR2 debería ser bloqueado. A continuación se demuestra un caso particular de dicha situación. En la misma, se utiliza el protocolo de capa de transporte udp y el puerto de destino 100.

```
mininet> hR1 timeout -s KILL 5s iperf -c 10.0.0.4 -p 100 -u -t 4
------------------------------------------------------------
Client connecting to 10.0.0.4, UDP port 100
Sending 1470 byte datagrams, IPG target: 11215.21 us (kalman adjust)
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  1] local 10.0.0.3 port 34246 connected with 10.0.0.4 port 100
[ ID] Interval       Transfer     Bandwidth
[  1] 0.0000-4.0152 sec   517 KBytes  1.05 Mbits/sec
[  1] Sent 361 datagrams
read failed: Connection refused
```

Figura 20: Captura de la terminal de mininet corriendo iperf desde hR1 hasta 10.0.0.4 (IP de hR2).

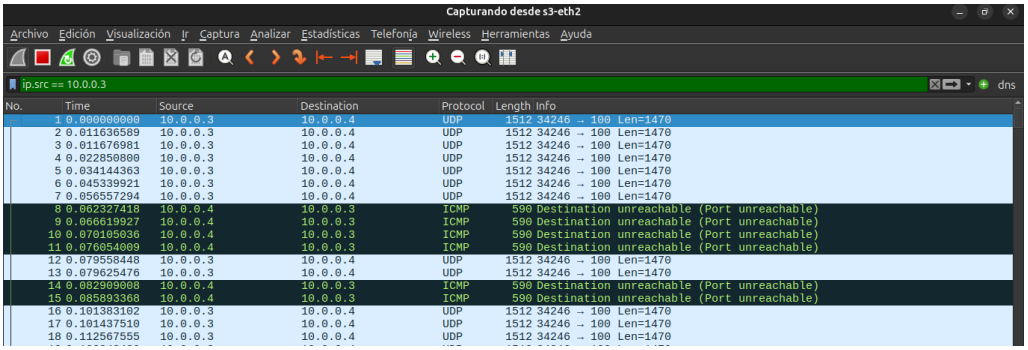

Figura 21: Captura de Wireshark del host hR1 durante la prueba.

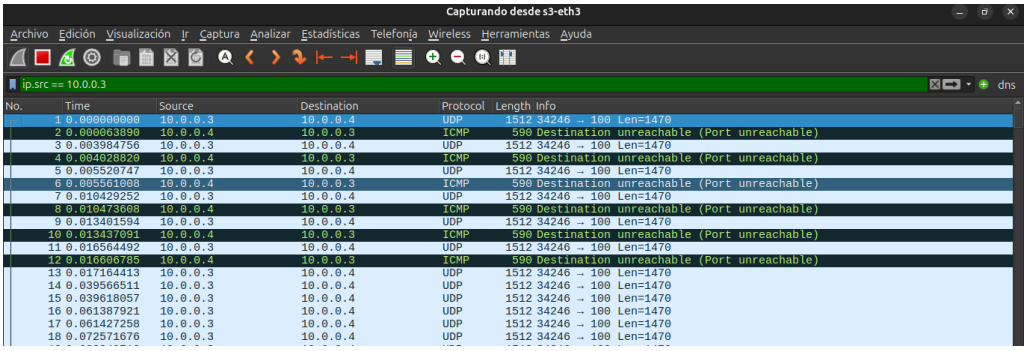

Figura 22: Captura de Wireshark del host hR2 durante la prueba.

5. Preguntas a responder

1. **¿Cuál es la diferencia entre un Switch y un router? ¿Qué tienen en común?**

   Los **switches** y los **routers** son dispositivos fundamentales en las redes de comunicaciones.

Aunque ambos cumplen la función general de **interconectar dispositivos y permitir el flujo de información**, su diseño, objetivo y ámbito de operación difieren considerablemente.

Un switch opera principalmente en la **Capa de Enlace (Capa 2)**. Se encarga de reenviar tramas utilizando las **direcciones MAC** de los dispositivos conectados. Su funcionamiento se basa en un mecanismo de "aprendizaje" automático: cuando recibe una trama, inspecciona la dirección MAC de origen y la asocia con el puerto por el cual ingresó. A medida que recibe tráfico, construye una tabla interna que permite enviar las tramas únicamente por el puerto correspondiente, evitando difundirlas innecesariamente.

El switch permite segmentar la red y reducir colisiones, contribuyendo a mejorar el rendimiento respecto a un hub. Su uso está orientado al **diseño de redes locales (LAN)**, donde los dispositivos comparten una subred IP común.

El router opera principalmente en la **Capa de Red (Capa 3)**. Utiliza **direcciones IP** para decidir a qué red enviar los paquetes. Además, incorpora funciones avanzadas como:

- Determinación de rutas mediante protocolos dinámicos (OSPF, RIP, IS-IS, BGP).
- Traducción de direcciones (NAT).
- Seguridad (ACLs, filtrado avanzado).
- Conexión entre redes heterogéneas.

A diferencia del switch, el router es indispensable cuando se necesita **interconectar redes diferentes**, especialmente cuando hay saltos entre subredes, proveedores o dominios administrativos. Su uso es clave en **Internet** y redes WAN.

Hoy en día existen **switches de Capa 3**, que incorporan capacidades básicas de ruteo. Sin embargo, no suelen reemplazar completamente a un router tradicional, especialmente cuando se requiere manejo de tráfico interdominio o en configuraciones de Internet.

Estas principales diferencias se pueden observar resumidamente en la siguiente tabla:

Dispositivo	Capa principal	Identificador que usa	Función
Switch	Enlace	MAC	Reenvío de tramas dentro de una LAN
Router	Red	IP	Interconexión entre redes y selección de rutas óptimas

Cuadro 1: Principales diferencias entre Routers y Switches

2. **¿Cuál es la diferencia entre un Switch convencional y un Switch OpenFlow?**

La diferencia fundamental radica en donde se encuentra la lógica de control y **cómo se toman las decisiones de reenvío**.

En un switch tradicional, la inteligencia de la red está embebida en el hardware. El dispositivo toma decisiones autónomamente, siguiendo algoritmos y protocolos estándar (como aprendizaje MAC, spanning tree, VLAN). El administrador tiene poco control sobre su comportamiento interno.

- No se puede programar fácilmente.
- Las reglas de reenvío están determinadas por su firmware.
- La red es **distribuida** (cada switch actúa en forma independiente).
- Se configura manualmente, con protocolos poco flexibles.

Por otro lado un switch OpenFlow (dentro del paradigma SDN) es un dispositivo **programable** que separa la funcionalidad en dos capas o planos.

Por un lado el Plano de datos (data plane), que se encarga de reenviar paquetes y se encuentra físicamente en el switch. Por otro lado el Plano de control (control plane), centralizado en un controlador SDN (como POX), donde se programa el comportamiento de la red.

Las reglas de reenvío no se aprenden dinámicamente, sino que se instalan mediante mensajes OpenFlow enviados desde el controlador. Esto permite modificar la red en tiempo real, responder a eventos, automatizar comportamientos y aplicar políticas con precisión.

Además, OpenFlow permite definir reglas de forwarding basadas en un conjunto amplio de campos de coincidencia (match fields) que el switch puede inspeccionar en los paquetes que atraviesan la red. En las versiones del estándar, estas reglas pueden utilizar hasta 15 campos distintos, que incluyen información desde la capa Física hasta la capa de Red y Transporte. Entre ellos se encuentran: el puerto de entrada al switch, las direcciones MAC de origen y destino, el tipo de EtherType, el identificador de VLAN y su prioridad, las direcciones IP de origen y destino, el protocolo de nivel de red, los puertos TCP/UDP de origen y destino, y el tipo de servicio de IP (IP-ToS). Esta granularidad permite aplicar políticas muy específicas, como priorizar cierto tráfico, redirigir flujos particulares, implementar firewalls distribuidos o realizar ingeniería de tráfico, algo que no es posible en un switch convencional cuyo análisis está limitado a campos propios de la capa de Enlace.

Característica	**Switch Convencional**	**Switch OpenFlow**
Toma de Decisiones	Interna (autónoma)	Externa (controlador SDN)
Programable	NO	SI
Control Centralizado	NO	SI
Arquitectura	Distribuida	Centralizada
Flexibilidad	Baja	Alta
Dependencia de Protocolos Tradicionales (STP, VLAN)	Alta	Baja
Ideal para	Redes tradicionales	Redes dinámicas, visualización, Data Centers

Cuadro 2: Principales características y diferencias entre Switches Convencionales y OpenFlow

3. ¿Se pueden reemplazar todos los routers de la Intenet por Switches OpenFlow? Piense en el escenario interASes para elaborar su respuesta

En términos **teóricos**, sí podría plantearse un escenario donde toda la infraestructura de Internet funcione sobre dispositivos basados en OpenFlow o bajo el paradigma SDN (Software Defined Networking). Al fin y al cabo, un switch OpenFlow puede implementar lógicas de reenvío avanzadas en Capa 3 o incluso tomar decisiones basadas en múltiples campos del paquete, lo cual se asemeja funcionalmente a lo que realiza un router.

Sin embargo, **en la práctica actual este reemplazo no es viable**, por **limitaciones técnicas, operativas, políticas y de arquitectura de Internet**, especialmente cuando se analiza el funcionamiento del ecosistema inter-AS (inter Autonomous Systems). A continuación se detalla el análisis desde ambos enfoques.

¿Por qué no es posible hoy?

3.1. Internet está concebida como una arquitectura distribuida

Cada AS (Sistema Autónomo) administra de forma independiente su infraestructura y decisiones de ruteo. No existe un controlador centralizado autorizado para manipular políticas de tránsito entre ASes.

Cambiar todos los routers de Internet por switches OpenFlow equivaldría a centralizar la toma de decisiones, eliminando la autonomía de los organismos que participan en Internet. Eso rompería el modelo organizacional y administrativo actual.

3.2. El enrutamiento inter-AS depende de BGP

- BGP no es un protocolo de simple reenvío, sino que incorpora decisiones políticas, comerciales y administrativas.

- OpenFlow podría implementar la mecánica de forwarding, pero **no reemplaza la funcionalidad de negociación entre ASes**.
- Tampoco provee mecanismos robustos de estabilidad ante cambios constantes (retiro de rutas, multipath, políticas de preferencia, etc.)
- Un router BGP no solo enrruta, también decide qué rutas aceptar y anunciar, basándose en acuerdos entre organizaciones. Eso no está contemplado en OpenFlow.

3.3. **Escalabilidad y complejidad**

- Internet enruta millones de prefijos BGP, más múltiples rutas alternativas.
- Las entradas OpenFlow son gestionadas como reglas de flujo. La cantidad de estados por mantener sería enorme.
- Los controladores SDN tienen límites de procesamiento; un fallo provocaría pérdida de conectividad global.

3.4. **Dependencia del controlador**

En SDN tradicional, los switches dependen del controlador. Si este cae:

- Un router tradicional sigue operando.
- Un switch OpenFlow dejaría de tomar decisiones y podría quedar inutilizado.

En el contexto de Internet, la tolerancia a fallos es fundamental.

3.5. **Políticas y seguridad de red**

- Los ASes operan bajo normas y acuerdos comerciales confidenciales.
- El reemplazo por un modelo SDN obligaría a compartir la lógica de control.
- También abriría ventanas de seguridad: un ataque al controlador afectaría a múltiples redes.
- En Internet, la descentralización y la redundancia es también una estrategia de seguridad.

Aunque técnicamente un switch OpenFlow puede implementar funciones equivalentes a las de un router y podría reemplazarlo desde una perspectiva funcional, en la práctica no es posible reemplazar todos los routers de la Internet pública por switches OpenFlow debido a motivos de arquitectura distribuida, independencia administrativa entre ASes, escalabilidad, seguridad y la necesidad de protocolos como BGP.

En consecuencia, los switches OpenFlow son una excelente alternativa para redes privadas, datacenters o infraestructuras controladas por un único administrador, pero no resultan adecuados para reemplazar los routers tradicionales en el plano inter-AS de Internet, donde se requiere autonomía, estabilidad y modelos de ruteo distribuidos.

6. Dificultades encontradas

A lo largo del desarrollo de este trabajo, hemos abordado distintas tecnologías y herramientas de Software. En el marco de este informe, y acompañando el uso de dichas herramientas, hemos identificado ciertas dificultades que serán detalladas a continuación.

Para comenzar, hemos identificado el software a utilizar y primaramente configuramos el mismo. En el mismo se incluyen POX, iperf, Python en su versión 2.7. Lo más complicado, en este marco, fue identificar correctamente con qué versión de POX y Python trabajar, ya que ciertas versiones de dichas herramientas cuentan con dificultades para conjugarse correctamente entre sí. Luego de una extensa prueba de distintas versiones, dimos con una que funcionaba correctamente para el alcance propuesto para este trabajo. Las mismas son para POX la rama 'dart' del siguiente repositorio (`https://github.com/noxrepo/pox/tree/dart`) y Python 2.7. En cuanto a iperf, no encontramos mayores dificultades en cuanto a la instalación y utilización de la misma.

En lo que respecta al desarrollo mismo de la topología y el firewall en concreto, la mayor dificultad fue integrar el funcionamiento de la topología simulada con Mininet y el controlador de OpenFlow. Asimismo, investigar cómo utilizar y aplicar las distintas reglas del firewall fue una de las partes centrales de este trabajo más complicadas.

Consideramos que hemos podido abordar todas las dificultades mencionadas satisfactoriamente, dando como resultado un trabajo que engloba todos los requerimientos y condiciones planteadas en el enunciado del mismo.

7. Conclusiones

El concepto de redes definidas por software (SDN) representa una fuerte evolución en la gestión de infraestructuras de red permitiendo mayor flexibilidad, control y automatización. Sin embargo, su aplicabilidad depende del entorno. Mientras que en redes privadas permite mejorar la administración del tráfico, automatizar configuraciones y facilitar la movilidad y virtualización, en redes públicas como Internet se continúa dependiendo de arquitecturas distribuidas y protocolos de enrutamiento tradicionales. La coexistencia de ambos paradigmas es actualmente la tendencia dominante, más que el reemplazo completo.

En conjunto, el proyecto permitió observar el potencial real de SDN en escenarios controlados y validar que, con herramientas como POX y Mininet, se pueden diseñar e implementar políticas avanzadas de control de tráfico, reforzando la comprensión de los principios que guían esta arquitectura emergente.