Redes de Computadoras

Obligatorio 1

Facultad de Ingeniería
Instituto de Computación
Departamento de Arquitectura de Sistemas

Nota previa - IMPORTANTE

Se debe cumplir íntegramente el "Reglamento del Instituto de Computación ante Instancias de No Individualidad en los Laboratorios", disponible en el EVA.

En particular está prohibido utilizar documentación de otros estudiantes, de otros años, de cualquier índole, o hacer público código a través de cualquier medio (EVA, news, correo, papeles sobre la mesa, etc.).

Introducción

Forma de entrega

Una clara, concisa y descriptiva documentación es clave para comprender el trabajo realizado. La entrega de la tarea consiste en un único archivo obligatorio1GrupoGG.tar.gz que deberá contener los siguientes archivos:

- Un documento llamado Obligatorio1GrupoGG.pdf donde se documente todo lo solicitado en la tarea. GG es el número del grupo. La documentación deberá describir claramente su solución, las decisiones tomadas, los problemas encontrados y posibles mejoras, y las pruebas realizadas.
- El código fuente del programa (**en lenguaje Python**) e instrucciones claras de cómo ejecutar el sistema.

La entrega se realizará en el sitio del curso, en la plataforma EVA.

Fecha de entrega

Los trabajos deberán ser entregados **antes del viernes 12/09/2025 a las 23:59 horas**. No se aceptará ningún trabajo pasada la citada fecha y hora. En particular, no se aceptarán trabajos enviados por e-mail a los docentes del curso.

Objetivo del Trabajo

- Familiarizarse con conceptos básicos sobre redes e Internet y manejar herramientas para diagnóstico y *debug* de la red.
- Aplicar los conceptos teóricos de capas de aplicación y transporte, la utilización de la API de sockets TCP, y la arquitectura de aplicaciones cliente-servidor.

Descripción general del problema

Se desea implementar una biblioteca de llamadas a procedimientos remotos (Remote Procedure Calls, RPC) así como una aplicación cliente-servidor donde

probar la biblioteca desarrollada.

Una biblioteca es un conjunto de funciones, que ofrece una interfaz bien definida para la funcionalidad que se invoca. Las herramientas RPC permiten a una aplicación cliente llamar directamente a un procedimiento ubicado en un programa servidor remoto. Es decir, se desea implementar una biblioteca que contenga las funciones necesarias para que un servidor pueda “publicar” sus procedimientos y que un cliente puede invocar estos procedimientos.

Entorno de trabajo

Se podrá realizar la tarea en el entorno de trabajo que considere mas cómodo y adecuado, por ejemplo su computadora personal o las PCUnix de la Facultad. Sin embargo, debe tener en cuenta que:

- Para resolver la **Parte 1** puede utilizar las herramientas disponibles en su máquina.
- Para la **Parte 2** necesitará tener instalado Python 3.5 o superior y también se recomienda que utilice un ambiente Linux para evitar problemas con la API de sockets. Las máquinas de las salas PCUnix cumplen con este requisito.
- Para la **Parte 3** deberá utilizar el emulador Mininet, el cual puede ser instalado de forma local en un ambiente Linux. Sin embargo, como parte de los materiales del obligatorio, se provee una máquina virtual con Mininet ya instalado y con los archivos de configuración necesarios para la ejecución de la tarea. En la Parte 3 de esta tarea se ofrece una guía detallada para su ejecución y uso.

Parte 1

El objetivo de esta primera parte es conocer las herramientas necesarias para resolver y evaluar las partes siguientes.

Análisis de paquetes y protocolos

1. Investigar las herramientas `tcpdump` [1], `wireshark` [2] y `tshark` [3], y crear un reporte técnico (una carilla, máximo dos) mencionando sus principales características y utilidad. Debe concluir si son equivalentes, en que sistemas operativos funcionan, si permiten capturar paquetes en vivo, si se pueden guardar para ser analizados en otro momento, si se puede elegir qué paquetes capturar (acá pueden existir muchos criterios de filtro), si es posible seleccionar en que interfaces de red capturar tráfico, si pueden interoperar entre ellas, entre otras.

2. Ejecute `tcpdump` o `tshark` con las opciones necesarias para capturar tráfico únicamente en la interfaz de loopback, IP 127.0.0.1 y el puerto 5001, y guarde el resultado para análisis posterior en un archivo `incognito.pcap`. En otra terminal ejecute el achivo `servidor_incognito.py` [4] y en una tercera terminal ejecute el comando `telnet 127.0.0.1 5001` para abrir una sesión con el servicio. En la sesión abierta con telnet, envíe la siguiente secuencia de mensajes de a uno a la vez: `xazf`, `xazf`, `xazf`, `bb`, `bb`, `bb`, `bb`, `affhex`, `affhex`, `zzz`. Interrumpa la captura y abra en wireshark el archivo `incognito.pcap`. Descubra la funcionalidad del servidor mediante el análisis de la captura, primero identificando los mensajes dentro de cada paquete, y luego utilizando la funcionalidad “Following Protocol Streams” [5].

3. Capture tráfico (guardarlo en `https_web.pcap`) mientras accede desde un navegador a `https://www.fing.edu.uy` y luego analice la captura con `wireshark`. Agregue las columnas de puerto de origen y puerto de destino a la visualización (si no están visibles) y luego cree un “Display Filter” para quedarse solo con los paquetes que contienen la dirección IP del servidor y el puerto 443. Realice el mismo análisis de la nueva captura que en el punto anterior, y explique lo siguiente: ¿por qué si la especificación del protocolo HTTP [6] dice que los mensajes son compuestos por texto, no los puede ver?. Repita el experimento con `http://www.columbia.edu/~fdc/family/ssamerica2.jpg` y explique el motivo por el que ahora sí puede ver parte del protocolo, pero aún se ven partes de los mensajes en un formato que no es legible. Identifique los encabezados `Content-Type` y `Content-Length` en la respuesta al pedido, explique el significado general de ambos, y justifique el valor que tienen en la captura.

Comando ping

Investigue el principio de funcionamiento de la utilidad `ping` [7]. Debe ser capaz de entender la finalidad de su uso, el protocolo utilizado, descripción de encabezados, qué mensajes se envían y cuáles se reciben. Para esto se recomienda hacer capturas de ejecuciones a diferentes sitios e identificar en las capturas todo lo que mencione en su estudio. Finalmente en base a lo anterior debería ser capaz de entender como se calculan los resultados que el comando muestra en pantalla.

Comando traceroute

Investigue el principio de funcionamiento del comando `traceroute` [8] o `tracepath` [9] (equivalente en Windows: `tracert` [10]). Debe ser capaz de entender la finalidad de uso, el análisis de los dos protocolos que pueden ser usados por esas herramientas, descripción de encabezados, qué mensajes se envían y cuáles se reciben. Es fundamental que entienda con exactitud la forma en que la herramienta sabe que el destino fue alcanzado (como se traduce eso a mensajes de los protocolos). Para esto se recomienda hacer capturas de ejecuciones a diferentes sitios e identificar en las capturas todo lo que mencione en su estudio. Finalmente en base a lo anterior debería ser capaz de entender como se calculan los resultados que el comando muestra en pantalla.

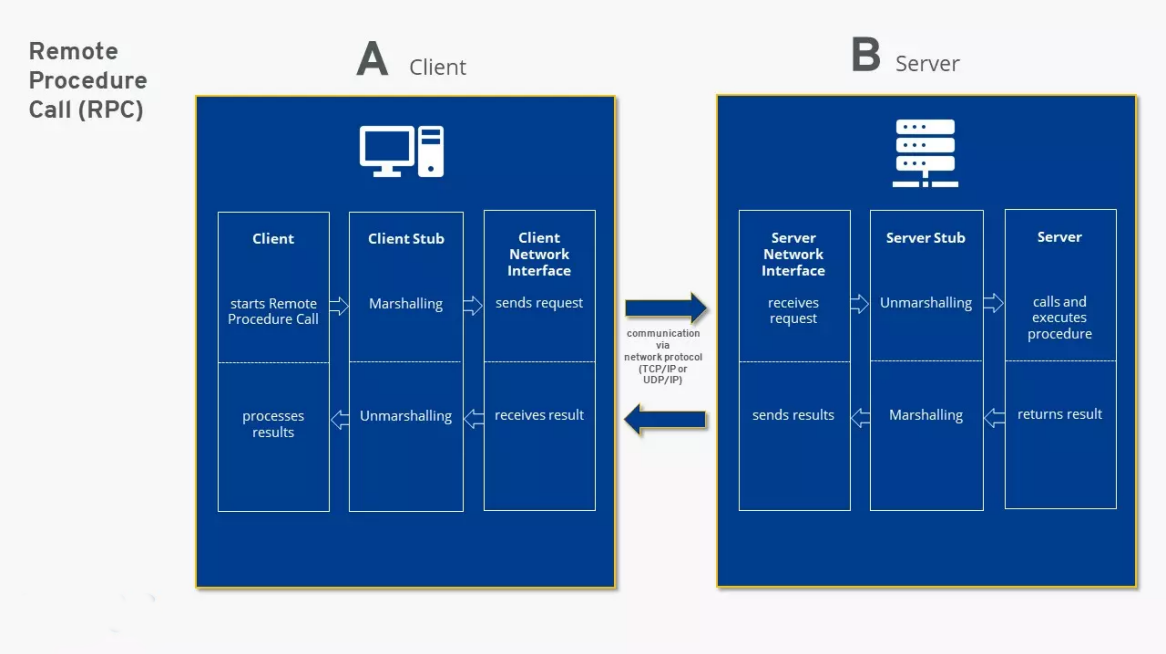
Reflexión

Suponga que de ahora en más solo puede utilizar solo una de las herramientas ping o traceroute para hacer diagnósticos de red, ¿cual será su elección? Detalle todas las ventajas que respalden su elección.

Parte 2

Descripción general de la arquitectura RPC

A continuación, se ofrece una descripción de alto nivel del funcionamiento de RPC. Más adelante se detallará lo solicitado específicamente para esta sección.



Supongamos un cliente que desea ejecutar un procedimiento que no se encuentra implementado localmente, sino que está disponible en un servidor remoto. Como muestra la ilustración, la aplicación cliente llama a un procedimiento *stub* local en lugar del código real que implementa dicho procedimiento. Los *stubs* son provistos por la biblioteca RPC y se compilan y enlazan con la aplicación cliente. En lugar de contener el código real del procedimiento remoto, el *stub* del cliente:

- Recupera los parámetros necesarios para la invocación.
- Traduce los parámetros necesarios a un formato estándar (definido por el protocolo RPC) para su transmisión a través de la red.
- Durante la transferencia, se realiza una serialización, en la que los datos estructurados se traducen a una forma de representación secuencial. Este proceso también se conoce como *marshalling* (del inglés *marshal*, que significa presentar u ordenar).
- Llama a funciones de la capa de transporte para enviar la solicitud y sus parámetros al servidor.
- Queda a la espera de la respuesta.

Cuando el mensaje llega al destino, el *stub* del servidor acepta la solicitud proveniente de la capa de transporte y llama al procedimiento encargado de procesar los mensajes. El *stub* del servidor realiza el llamado *demarshalling* o *unmarshalling*, es decir, desempaqueta los parámetros contenidos en el mensaje (de acuerdo con el protocolo RPC). Luego, transfiere los parámetros decodificados y ejecuta localmente la llamada al procedimiento en el servidor.

A continuación, el procedimiento remoto se ejecuta, generando posiblemente parámetros de salida y un valor de retorno. Al finalizar, se repite una secuencia similar de pasos para devolver los datos al cliente. Es decir, el valor resultante se comunica al *stub* del servidor, que genera un mensaje acorde al protocolo RPC y lo transmite mediante las funciones de la capa de transporte.

La capa de transporte del servidor transmite los datos a través de la red hacia el cliente.

La capa de transporte del cliente recibe los valores de retorno del procedimiento remoto y los entrega al stub del cliente. El stub convierte los datos al formato utilizado por la aplicación cliente y devuelve el resultado al programa que realizó la llamada. El procedimiento de llamada continúa como si hubiera sido ejecutado localmente.

Cabe destacar que los stubs son proporcionados por la biblioteca RPC y se compilan y enlazan con las aplicaciones tanto del cliente como del servidor.

Formato y Capa de Transporte

Para esta tarea se pide que se implemente una biblioteca en base a la especificación de XML-RPC (Extensible Markup Language Remote Procedure Call) [11]. XML-RPC es un protocolo de RPC que se basa en el formato de datos XML [12] y utiliza el protocolo HTTP para el envío de los mensajes. Cuenta con una especificación donde se definen varias estructuras de datos y las reglas en torno a su procesamiento. En esta propuesta de protocolo, HTTP controla la transferencia de datos, y XML se utiliza para representarlos.

Un mensaje XML-RPC es una solicitud HTTP-POST. El cuerpo de la solicitud está en formato XML. Un procedimiento se ejecuta en el servidor y el valor que devuelve también está formateado en XML. Los parámetros del procedimiento pueden ser escalares, números, cadenas, fechas, etc.; y también pueden ser estructuras complejas de registros y listas.

Su implementación deberá seguir el formato de mensajes de solicitud y de respuesta descritos en la especificación.

Además, deberá usted implementar el cliente y servidor HTTP que utilizará la biblioteca a través de la API de sockets de POSIX para el envío de mensajes por la red. El servidor a implementar solo debe procesar solicitudes de tipo POST y puede asumir que solo recibirá mensajes de tipo XML-RPC.

Por lo tanto, la biblioteca deberá gestionar el procesamiento de los mensajes y su formato y también se deberá encargar del envío de los mensajes por la red.

Para el manejo de los errores, devueltos en el formato de *Fault example*, presentado en la especificación [11], se deben considerar los siguientes errores:

FaultCode	faultString
1	Error parseo de XML
2	No existe el método invocado
3	Error en parámetros del método invocado
4	Error interno en la ejecución del método
5	Otros errores

Implementación de XMLRPC_REDES

La biblioteca a implementar se denominará xml`rpc_redes` y su funcionamiento se describe a continuación.

Cliente
Para el lado cliente, debe ofrecer una clase `Client` con la siguiente funcionalidad a través de una única función pública:

`conn = connect('address', 'port')` la cual devuelve un elemento que representa la conexión con el servidor remoto.

Luego, este elemento se utilizará de la siguiente forma para llamar al procedimiento remoto `proc1(arg1, arg2)`:

```
result = conn.proc1(arg1, arg2)
```

Para implementar la funcionalidad de que se pueda llamar a un procedimiento no definido previamente en la clase Client recomendamos evaluar las siguientes funciones de Python:

`__getattr__` [13] y `__call__` [14].

Servidor
Para el lado servidor, debe ofrecer una clase `Server` con las siguientes funcionalidades mínimas:

- Obtener un objeto servidor: `server = Server(('address', 'port'))`
- Agregar un procedimiento: `server.add_method(proc1)`
- Ejecutar el servidor: `server.serve() # bloqueante`

Implementación de aplicación de pruebas

Para probar la biblioteca desarrollada se pide implementar un cliente y dos servidores que ofrezcan distintos procedimientos. Cada servidor deberá contar con al menos tres procedimientos y cada uno con una cantidad de argumentos mayor a uno.

El cliente deberá realizar llamadas al servidor, probar distintos casos, incluyendo casos de error como ser el llamado a un procedimiento remoto no existente, y error en los parámetros pasados en la invocación.

Resumen

En resumen, se pide que:

a) Diseñe y documente la biblioteca `xmlrpc_redes` utilizando las primitivas de la API de sockets del curso [15].
b) Implemente en lenguaje Python, la biblioteca solicitada; debe contar al menos con un archivo para las funciones del servidor y otra para las funciones del cliente.
c) Implemente en lenguaje Python la aplicación de pruebas solicitada.

Observaciones

Se aceptará el uso de bibliotecas de estructuras de datos, de parseo de XML, de hilos, de hashing, etc.

No se acepta el uso de bibliotecas que implementen el protocolo HTTP y las que oculten el uso de sockets; debe usar la API de sockets directamente.

Sobre la implementación del protocolo HTTP, se pide las funciones básicas requeridas por XML-RPC (aceptar un POST y devolver su respuesta).

Se recomienda que durante la implementación y pruebas iniciales utilice el comando *curl* para generar mensajes POST con solicitudes para el servidor.

Parte 3

En esta parte se espera que combine los conocimientos adquiridos en la Parte 1 con su implementación de la Parte 2.
Para esto deberá definir un set de pruebas, ejecutar su aplicación cliente y servidor en Mininet y ejecutar las pruebas definidas. Es importante que las pruebas contemplen todos los casos posibles de uso de la especificación XML-RPC así como de los errores definidos.

Se espera que haga uso de `tcpdump` y `wireshark` para mostrar el correcto funcionamiento de la biblioteca, incluyendo el formato de los mensajes y el correcto uso de TCP.

Emulación de una red con Mininet

Dado que esta parte deberá desarrollarla sobre Mininet, en esta sección veremos como utilizarlo. Mininet le permite emular una topología de red en una sola máquina. Proporciona el aislamiento necesario entre los nodos emulados de forma de contar con nodos *host* y nodos *router* y permite que su nodo *router* pueda procesar y reenviar tramas Ethernet reales entre los *hosts* como un *router* real. No tiene que saber cómo funciona Mininet para completar este obligatorio, pero puede encontrar más información sobre Mininet (si tiene curiosidad) en [16].

Para facilitar la tarea de instalación y configuración, se provee una máquina virtual que ya incluye Mininet instalado y configurado.

Máquina virtual

La máquina virtual se encuentra en [17], descarguela e importela utilizando VirtualBox.

La VM contiene un sistema operativo Ubuntu sin interfaz gráfica. La configuración de red es NAT, con el objetivo de que tenga acceso a Internet (aunque no debería ser necesario para la tarea). Además, tiene configurado una regla de "Port Forwarding" para que pueda acceder desde su máquina Host.

Para acceder a la VM por ssh puede ejecutar:
```
ssh -X -p 2522 osboxes@127.0.0.1
```

y para transferir archivos a la VM o desde la VM puede usar:
```
scp -P 2522 <origen> <destino>
```

Por ejemplo, para transferir el archivo `miSol.py` desde su máquina local a la VM puede hacer:
```
scp -P 2522 ./miSol.py osboxes@127.0.0.1:
```

El usuario configurado es `osboxes` y la password `osboxes.org`.

Manejo de Máquina Virtual en los equipos de FING

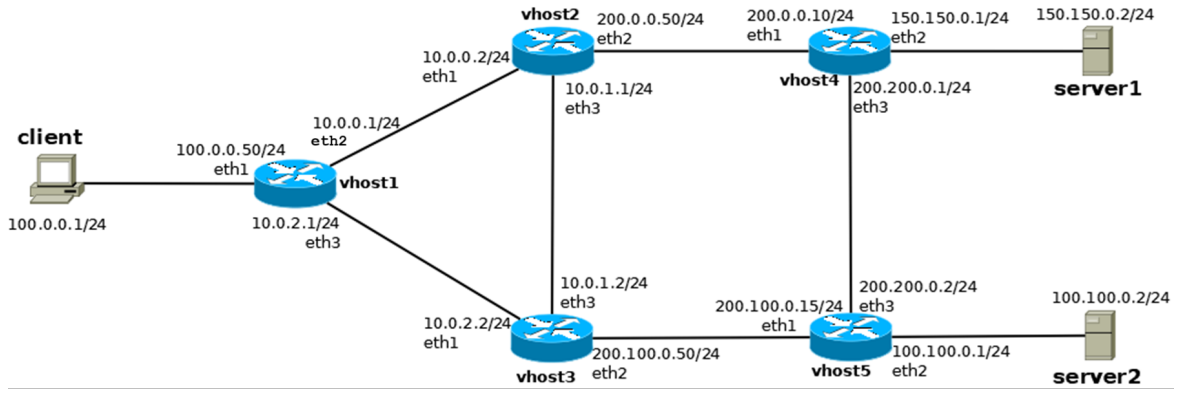
Como el almacenamiento provisto para los usuarios no es suficiente para mantener el estado de las máquinas virtuales creadas, se agregó a cada grupo

un espacio temporal con éste fin.
Este es accesible solamente por los integrantes del grupo, en el directorio /ens/devel01/redesGG (siendo redesGG el grupo).

Previo a la importación se debe generar un directorio donde almacenar las máquinas a importar.
Para esto se deberá generar un directorio en /ens/devel01/redesGG.
Es importante que cuando importe la máquina virtual seleccione este directorio para la importación.

Topología de pruebas

En la siguiente figura se muestra la topología donde se realizarán las pruebas durante el proyecto. Esta topología ya se encuentra configurada y será inicializada al iniciar Mininet.



Configure e inicie el emulador

Recomendamos fuertemente que en lugar de utilizar la terminal de la VM directamente, use su terminal local y se conecte por ssh a la VM. Puede abrir una nueva sesión ssh para cada paso que sea necesario.

1) Configure el entorno ejecutando el archivo config.sh

```
> cd ~/redes2025_ob1/
> ./config.sh
```

2) Inicie la emulación de Mininet utilizando el siguiente comando

```
> ./run_mininet.sh
```

Debería poder ver algunos resultados como el siguiente:

```
*** Shutting down stale RPCServers
server1 150.150.0.2
server2 100.100.0.2
client 100.0.0.1
vhost1-eth1 100.0.0.50
vhost1-eth2 10.0.0.1
vhost1-eth3 10.0.2.1
vhost2-eth1 10.0.0.2
vhost2-eth2 200.0.0.50
```

```
vhost2-eth3 10.0.1.1
vhost3-eth1 10.0.2.2
vhost3-eth2 200.100.0.50
vhost3-eth3 10.0.1.2
vhost4-eth1 200.0.0.10
vhost4-eth2 150.150.0.1
vhost4-eth3 200.200.0.1
vhost5-eth1 200.100.0.15
vhost5-eth2 100.100.0.1
vhost5-eth3 200.200.0.2
*** Successfully loaded ip settings for hosts
 {'vhost5-eth1': '200.100.0.15', 'vhost4-eth1': '200.0.0.10',
'vhost4-eth3': '200.200.0.1', 'vhost4-eth2': '150.150.0.1',
'vhost1-eth2': '10.0.0.1', 'vhost1-eth3': '10.0.2.1', 'server2':
'100.100.0.2', 'server1': '150.150.0.2', 'vhost3-eth3':
'10.0.1.2', 'vhost3-eth1': '10.0.2.2', 'vhost3-eth2':
'200.100.0.50', 'client': '100.0.0.1', 'vhost2-eth3': '10.0.1.1',
'vhost2-eth2': '200.0.0.50', 'vhost2-eth1': '10.0.0.2', 'vhost5-
eth2': '100.100.0.1', 'vhost5-eth3': '200.200.0.2', 'vhost1-eth1':
'100.0.0.50'}
*** Creating network
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
client server1 server2
*** Adding switches:
vhost1 vhost2 vhost3 vhost4 vhost5
*** Adding links:
(client, vhost1) (server1, vhost4) (server2, vhost5) (vhost2,
vhost1) (vhost2, vhost4) (vhost3, vhost1) (vhost3, vhost2)
(vhost3, vhost5) (vhost4, vhost5)
*** Configuring hosts
client server1 server2
*** Starting controller
c0
*** Starting 5 switches
vhost1 vhost2 vhost3 vhost4 vhost5 ...
*** setting default gateway of host server1
server1 150.150.0.1
*** setting default gateway of host server2
server2 100.100.0.1
*** setting default gateway of host client
client 100.0.0.50
*** Starting CLI:
```

3) Mantenga este terminal abierto, ya que necesitará la línea de comandos de mininet para hacer debug. Ahora, use otra terminal para continuar con el siguiente paso. (No presione ctrl-z.)

Mininet requiere un controlador, que implementamos en POX. Para ejecutar el controlador, use el siguiente comando:

```
> cd ~/redes2025_ob1/
>./run_pox.sh
```

Debería ver una salida como la siguiente:

```
POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.
server1 150.150.0.2
server2 100.100.0.2
client 100.0.0.1
vhost1-eth1 100.0.0.50
vhost1-eth2 10.0.0.1
vhost1-eth3 10.0.2.1
vhost2-eth1 10.0.0.2
vhost2-eth2 200.0.0.50
vhost2-eth3 10.0.1.1
vhost3-eth1 10.0.2.2
vhost3-eth2 200.100.0.50
vhost3-eth3 10.0.1.2
vhost4-eth1 200.0.0.10
vhost4-eth2 150.150.0.1
vhost4-eth3 200.200.0.1
vhost5-eth1 200.100.0.15
vhost5-eth2 100.100.0.1
vhost5-eth3 200.200.0.2
INFO:.home.osboxes.redes2024_ob1.pox_module.pwospf.srhandler:creat
ed server
INFO:core:POX 0.5.0 (eel) is up.
```

Tenga en cuenta que debe esperar a que Mininet se conecte al controlador POX antes de continuar con el siguiente paso. Una vez que Mininet se haya conectado, verá la siguiente salida:

```
INFO:openflow.of_01:[00-00-00-00-00-01 4] connected
{'vhost1': {'eth1': ('100.0.0.50', '02:98:89:f5:48:6f', '10Gbps',
1)}}
{'vhost1': {'eth2': ('10.0.0.1', 'd6:c4:73:33:47:53', '10Gbps',
2), 'eth1': ('100.0.0.50', '02:98:89:f5:48:6f', '10Gbps', 1)}}
{'vhost1': {'eth3': ('10.0.2.1', 'c2:94:ab:66:67:13', '10Gbps',
3), 'eth2': ('10.0.0.1', 'd6:c4:73:33:47:53', '10Gbps', 2),
'eth1': ('100.0.0.50', '02:98:89:f5:48:6f', '10Gbps', 1)}}
INFO:openflow.of_01:[00-00-00-00-00-04 2] connected
{'vhost4': {'eth1': ('200.0.0.10', '7a:1a:e2:fd:a7:f6', '10Gbps',
1)}}
{'vhost4': {'eth2': ('150.150.0.1', 'da:06:d8:f8:3e:9b', '10Gbps',
2), 'eth1': ('200.0.0.10', '7a:1a:e2:fd:a7:f6', '10Gbps', 1)}}
{'vhost4': {'eth3': ('200.200.0.1', '72:c3:41:94:46:f8', '10Gbps',
3), 'eth2': ('150.150.0.1', 'da:06:d8:f8:3e:9b', '10Gbps', 2),
'eth1': ('200.0.0.10', '7a:1a:e2:fd:a7:f6', '10Gbps', 1)}}
INFO:openflow.of_01:[00-00-00-00-00-02 3] connected
{'vhost2': {'eth1': ('10.0.0.2', '72:8b:3a:e3:2a:df', '10Gbps',
1)}}
{'vhost2': {'eth2': ('200.0.0.50', '0e:1c:e6:59:62:fe', '10Gbps',
2), 'eth1': ('10.0.0.2', '72:8b:3a:e3:2a:df', '10Gbps', 1)}}
{'vhost2': {'eth3': ('10.0.1.1', '16:28:10:2c:20:cf', '10Gbps',
3), 'eth2': ('200.0.0.50', '0e:1c:e6:59:62:fe', '10Gbps', 2),
'eth1': ('10.0.0.2', '72:8b:3a:e3:2a:df', '10Gbps', 1)}}
INFO:openflow.of_01:[00-00-00-00-00-05 1] connected
{'vhost5': {'eth1': ('200.100.0.15', '36:c9:0a:f8:03:05',
'10Gbps', 1)}}
{'vhost5': {'eth2': ('100.100.0.1', 'ba:78:39:e9:9b:dd', '10Gbps',
2), 'eth1': ('200.100.0.15', '36:c9:0a:f8:03:05', '10Gbps', 1)}}
```

```
{'vhost5': {'eth3': ('200.200.0.2', '8a:07:90:ba:d1:a9', '10Gbps',
3), 'eth2': ('100.100.0.1', 'ba:78:39:e9:9b:dd', '10Gbps', 2),
'eth1': ('200.100.0.15', '36:c9:0a:f8:03:05', '10Gbps', 1)}}
INFO:openflow.of_01:[00-00-00-00-00-03 5] connected
{'vhost3': {'eth1': ('10.0.2.2', '3e:7f:6f:5b:92:86', '10Gbps',
1)}}
{'vhost3': {'eth2': ('200.100.0.50', '96:ee:76:fa:ca:28',
'10Gbps', 2), 'eth1': ('10.0.2.2', '3e:7f:6f:5b:92:86', '10Gbps',
1)}}
{'vhost3': {'eth3': ('10.0.1.2', 'c2:40:94:fd:48:96', '10Gbps',
3), 'eth2': ('200.100.0.50', '96:ee:76:fa:ca:28', '10Gbps', 2),
'eth1': ('10.0.2.2', '3e:7f:6f:5b:92:86', '10Gbps', 1)}}
```

4) Mantenga POX en funcionamiento. Ahora, abra 5 nuevas terminales para continuar con el siguiente paso y en cada una colóquese en el directorio `redes2025_ob1`.

En la máquina virtual se incluye un binario que implementa funcionalidades de *routing* y carga una tabla de ruteo estática en cada *router*. En cada máquina *router* deberemos ejecutar este binario. Para esto, ejecute en cada terminal uno de los siguientes comandos:

```
./run_sr.sh 127.0.0.1 vhost1
./run_sr.sh 127.0.0.1 vhost2
./run_sr.sh 127.0.0.1 vhost3
./run_sr.sh 127.0.0.1 vhost4
./run_sr.sh 127.0.0.1 vhost5
```

Debería ver una salida como la siguiente:

```
Using VNS sr stub code revised 2009-10-14 (rev 0.20)
Loading routing table from server, clear local routing table.
Loading routing table
---------------------------------------------
Destination     Gateway         Mask    Iface
100.0.0.1               100.0.0.1       255.255.255.255 eth1
100.0.0.0               0.0.0.0 255.255.255.0   eth1
10.0.0.0                0.0.0.0 255.255.255.0   eth2
10.0.2.0                0.0.0.0 255.255.255.0   eth3
10.0.1.0                10.0.0.2        255.255.255.0   eth2
200.0.0.0               10.0.0.2        255.255.255.0   eth2
200.100.0.0             10.0.2.2        255.255.255.0   eth3
---------------------------------------------
Client osboxes connecting to Server 127.0.0.1:8888
Requesting topology 300
successfully authenticated as osboxes
Loading routing table from server, clear local routing table.
Loading routing table
---------------------------------------------
Destination     Gateway         Mask    Iface
100.0.0.1               100.0.0.1       255.255.255.255 eth1
100.0.0.0               0.0.0.0 255.255.255.0   eth1
10.0.0.0                0.0.0.0 255.255.255.0   eth2
10.0.2.0                0.0.0.0 255.255.255.0   eth3
10.0.1.0                10.0.0.2        255.255.255.0   eth2
200.0.0.0               10.0.0.2        255.255.255.0   eth2
200.100.0.0             10.0.2.2        255.255.255.0   eth3
```

```
-----------------------------------------
Router interfaces:
eth3    HWaddrae:25:15:6f:73:f1
        inet addr 10.0.2.1
eth2    HWaddrf6:8a:02:32:3d:f6
        inet addr 10.0.0.1
eth1    HWaddr1a:35:f9:5b:d6:ef
        inet addr 100.0.0.50
 <-- Ready to process packets -->
```

Pruebas básicas

Ahora, haremos unas pruebas básicas de la configuración.
Probaremos la conectividad entre cliente y servidor. Para esto utilizaremos las herramientas vistas anteriormente.

1) Colóquese nuevamente en la terminal donde se ejecuta Mininet. Para emitir un comando en el *host* emulado, escriba el nombre del *host* seguido del comando en la consola Mininet. Por ejemplo, el siguiente comando emite 3 pings del `client` al `server1`.

```
mininet> client ping -c 3 150.150.0.2
```

Debería ver una salida como la siguiente:

```
PING 150.150.0.2 (150.150.0.2) 56(84) bytes of data.
64 bytes from 150.150.0.2: icmp_seq=1 ttl=62 time=230 ms
64 bytes from 150.150.0.2: icmp_seq=2 ttl=62 time=97.2 ms
64 bytes from 150.150.0.2: icmp_seq=3 ttl=62 time=116 ms

--- 150.150.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2008ms
rtt min/avg/max/mdev = 97.247/148.216/230.462/58.710 ms
```

2) También puede realizar `traceroute` para ver la ruta entre el `client` y el `server2`:

```
mininet> client traceroute -n 100.100.0.2
```

Debería ver la siguiente salida:

```
client traceroute -n 100.100.0.2
traceroute to 100.100.0.2 (100.100.0.2), 30 hops max, 60 byte
packets
 1  100.0.0.50  30.212 ms  74.506 ms  79.207 ms
 2  10.0.2.2  322.304 ms  322.313 ms  322.299 ms
 3  200.100.0.15  490.767 ms  490.759 ms  490.759 ms
 4  100.100.0.2  490.727 ms  490.709 ms  490.693 ms
```

Evaluación de la implementación

Deberá copiar sus archivos a la VM, tanto la implementación de su biblioteca como sus casos de prueba. Para ejecutar sus aplicaciones cliente y servidor de pruebas en el cliente y los servers puede hacer en mininet lo siguiente (`nohup` y `&` permite liberar la terminal):

```
mininet> server1 nohup python3.8 ./myServer.py &
mininet> server2 nohup python3.8 ./myServer.py &
mininet> client python3.8 ./myClient.py
```

Por otro lado, si quiere capturar el tráfico en un host o en un nodo intermedio puede utilizar tcpdump de la siguiente forma:

```
mininet> server1 nohup sudo tcpdump -n -i server1-eth0 -w server1.pcap &

mininet> vhost1 nohup sudo tcpdump -n -i vhost1-eth1 -w vhost1.pcap
```

Al terminar, debe detener tcpdump para que se escriba todo el archivo de salida:

```
mininet> server1 killall tcpdump
mininet> vhost1 killall tcpdump
```

Luego, el archivo generado lo puede mover a su máquina local y analizarlo con Wireshark.

Referencias y Bibliografía Recomendada

[1] https://www.tcpdump.org/manpages/tcpdump.1.html. Última visita: Agosto 2025
[2] Analizador de Tráfico Wireshark. Accesible en línea: http://www.wireshark.org/. Última visita: Agosto 2025
[3] https://www.wireshark.org/docs/man-pages/tshark.html Última visita: Agosto 2025
[4] https://eva.fing.edu.uy/pluginfile.php/558434/mod_folder/content/0/servidor_incognito.py?forcedownload=1 Última visita: Agosto 2025
[5] https://www.wireshark.org/docs/wsug_html_chunked/ChAdvFollowStreamSection.html Última visita: Agosto 2025
[6] https://datatracker.ietf.org/doc/html/rfc2616 Última visita: Agosto 2025
[7] ping(8) - Linux man page. Accesible en línea: https://linux.die.net/man/8/ping Última visita: Agosto 2025.
[8] traceroute(8) - Linux man page. Accesible en línea: https://linux.die.net/man/8/traceroute Última visita: Agosto 2025.
[9] tracepath(8) - Linux man page. Accesible en línea: https://linux.die.net/man/8/tracepath. Última visita: Agosto 2025.
[10] tracert Documentation. Accesible en línea: https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/tracert Última visita: Agosto 2025.
[11] https://xmlrpc.com/spec.md Última visita: Agosto 2025
[12] https://www.w3schools.com/XML/xml_whatis.asp Última visita: Agosto 2025
[13] https://python-reference.readthedocs.io/en/latest/docs/dunderattr/getattr.html. Última visita: Agosto 2025
[14] https://www.geeksforgeeks.org/__call__-in-python/. Última visita: Agosto 2025
[15] API de sockets para un lenguaje de alto nivel https://eva.fing.edu.uy/pluginfile.php/254645/mod_resource/content/0/cartilla_sockets.pdf. Última visita: Agosto 2025
[16] https://mininet.org/
[17] https://finguy-my.sharepoint.com/:u:/g/personal/giachino_fing_edu_uy/EQPsSwr-n-dMmjE23PceRnwBD-CKe_nPZ3c_MPew9jiwEQ