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 lunes 16/09/2024 a las 09:00 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

Usted podrá realizar la tarea en el entorno de trabajo que considere mas cómodo y adecuado siendo este tanto su computador personal como las PCUnix de la Facultad.
Sin embargo, debe tener en cuenta que:

- Algunas de las herramientas a ejecutar en la **Parte 1** (por ej. tcpdump) solo pueden ser ejecutadas en ambientes Linux con permisos de administrador. En este sentido, si no cuenta con un ambiente Linux en su máquina personal. puede utilizar una máquina virtual con una instalación de Linux o el ambiente WSL2 de Windows.
- 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 Facultad 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 1 de esta tarea se ofrece una guía detallada para su ejecución y uso.

Parte 1

El objetivo de esta primera parte es familiarizarse con las herramientas necesarias para ejecutar y evaluar la posterior implementación.

Captura de tráfico con tcpdump y Wireshark

1. Investigue para qué sirve y cómo se utilizan las herramientas `tcpdump` [1] y `wireshark` [2]. En particular, investigue como utilizar `tcpdump` para capturar todo el tráfico de una interfaz, almacenar la captura en un archivo `.pcap` y cargar el archivo en `wireshark` para su visualización.

2. Utilizaremos `tcpdump` y `wireshark` para estudiar la transferencia de datos de varias aplicaciones durante una sesión de trabajo típica. Para cada caso realice las siguientes actividades (además de las especificadas en cada una de ellas):
   - Identifique el o los protocolos participantes de cada una de las capas involucradas.
   - ¿Qué nodos participan de la sesión? ¿Qué rol cumplen? Identifique cuáles están en la Red Local, y cuáles están en Internet.
   - ¿Puede acceder a los datos transferidos analizando la captura?

   a) Utilizando `tcpdump` capture (y almacene en un archivo `.pcap`) el tráfico generado mientras abre una consola SSH a su usuario en facultad, ejecuta algunos comandos tales como `ls` y `top`, y cierre la sesión. Abra el archivo en `wireshark`. ¿Puede deducir algo de los comandos ejecutados observando la traza capturada?

   b) Capture el tráfico generado por un navegador al descargar las páginas `http://www.columbia.edu/~fdc/sample.html` y `https://www.fing.edu.uy`. Cuando sea posible, responda las siguientes preguntas:
      1. ¿Qué objetos HTTP son transferidos durante la descarga? Identifique el media-type de cada objeto.
      2. ¿Qué versión de HTTP es usada?
      3. ¿Qué user-agent declara su cliente?
      4. ¿Qué *encodings* acepta el cliente, y cuáles usa el servidor?
      5. Identifique el mecanismo usado para marcar la posición del final de un objeto dentro del *stream*.

Comando ping

1. Investigue el principio de funcionamiento de la utilidad `ping` [3]. Una respuesta completa incluye el protocolo utilizado, descripción de encabezados, qué mensajes se envían y cuales se reciben. Finalmente en base a lo anterior debería ser capaz de entender como se calculan los resultados que el comando muestra en pantalla.

2. Pruebe los siguientes comandos y analice las salidas. Describa las conclusiones a las que puede arribar con respecto a cada sitio analizado.

```
ping -c 5 www.antel.com.uy
ping -c 5 www.google.com
ping -c 5 registro.br
ping -c 5 zadna.org.za
```

3. Utilizando `wireshark` analice los paquetes intercambiados entre origen y destino al hacer un `ping`. Identifique los protocolos utilizados y relacione con su investigación realizada en el apartado 1.

Comando traceroute

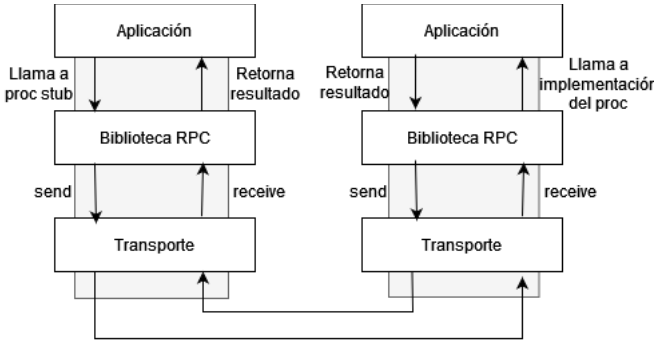
1. Investigue el principio de funcionamiento del comando `traceroute` [4] o `tracepath` [5] (equivalente en Windows: `tracert` [6]). Una respuesta completa incluye el protocolo utilizado, descripción de encabezados, qué mensajes se envían y cuales se reciben. Finalmente en base a lo anterior debería ser capaz de entender como se calculan los resultados que el comando muestra en pantalla.

2. Utilice `wireshark` para validar el análisis anterior.

3. ¿Cómo podría replicar la funcionalidad de `traceroute` usando únicamente comandos `ping`?

Parte 2

Descripción general de la arquitectura RPC



A continuación ofrecemos una descripción de alto nivel del funcionamiento de RPC, mas adelante se describe en detalle lo solicitado para esta parte.

Supongamos un cliente que desea ejecutar un procedimiento pero que el mismo 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 el procedimiento. Los *stubs* son ofrecidos por la biblioteca RPC y se compilan y enlazan con la aplicación cliente. En lugar de contener el código real que implementa el procedimiento remoto, el código *stub* del cliente:

- Recupera los parámetros necesarios para la invocación.
- Traduce los parámetros necesarios a un formato estándar para su transmisión a través de la red.
- Llama a funciones de la capa de transporte para enviar la solicitud y sus parámetros al servidor.
- Queda a la espera de la respuesta.

Por su parte, el servidor realiza los siguientes pasos para llamar al procedimiento remoto.

- Las funciones de la biblioteca RPC del servidor aceptan la solicitud proveniente de la capa de transporte y llaman al procedimiento de procesamiento de mensajes.
- Este procedimiento recupera los parámetros del mensaje recibido y los convierte del formato de transmisión de red al formato que necesita el servidor.
- El procedimiento del servidor llama por último al procedimiento real en el servidor.

A continuación, el procedimiento remoto se ejecuta, generando posiblemente parámetros de salida y un valor de retorno. Cuando el procedimiento remoto finaliza, una secuencia similar de pasos devuelve los datos al cliente.

- El procedimiento remoto devuelve sus datos al procedimiento de transmisión de la biblioteca RPC del servidor.
- El procedimiento del servidor convierte los parámetros de salida al formato requerido para la transmisión a través de la red y los envía a las funciones de la capa de transporte

- Las funciones de la capa de transporte del servidor transmiten los datos por la red al cliente.

El cliente completa el proceso aceptando los datos a través de la red y devolviéndolos a la función de llamada.

- La capa de transporte del cliente recibe los valores de retorno del procedimiento remoto y los devuelve al *stub* del cliente.
- El *stub* cliente convierte los datos al formato utilizado por el cliente. El *stub* devuelve el resultado al programa de llamada en el cliente.
- El procedimiento de llamada continúa como si el procedimiento hubiera sido llamado en el mismo equipo.

Formato y Capa de Transporte

Para esta tarea se pide que se implemente una biblioteca en base a la especificación de JSON-RPC 2.0 [7]. JSON-RPC es un protocolo de RPC que se basa en el formato de datos JSON [8]. Cuenta con una especificación donde se definen varias estructuras de datos y las reglas en torno a su procesamiento. Una característica principal es que es agnóstico en cuanto al transporte, ya que los conceptos se pueden utilizar dentro del mismo proceso, a través de sockets, a través de HTTP o en diversos entornos de paso de mensajes.

En nuestro caso, se deberá utilizar TCP a través de la API de sockets de POSIX para el envío de mensajes por la red.

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.

Implementación de JSONRPC_REDES

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

Cliente
Para el lado cliente, deberá contar con 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)
```

La implementación deberá permitir el envío de “notificaciones” tal cual se define en la especificación. Para esto se sugiere el siguiente formato:

```
conn.proc1(arg1, arg2, notify=True)
```

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__` [9] y `__call__` [10].

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`

Importante: No se requiere la implementación del soporte para solicitudes Batch.

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 a ambos servidores, probar distintos casos, y además deberá probar todos los casos de error definidos en la especificación.

Por otro lado, junto con los materiales del obligatorio, se incluye un cliente y servidor de pruebas así como la biblioteca ya implementada (con el código fuente ofuscado). Utilice este material tanto para entender el funcionamiento esperado de la biblioteca así como para realizar pruebas cruzadas. Es decir, puede probar un cliente/servidor con su biblioteca y el servidor/cliente utilizando la biblioteca entregada.

Resumen

En resumen, se pide que:

a) Diseñe y documente la biblioteca JSONRPC-TCP utilizando las primitivas de la API de sockets del curso [9].
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 JSON, de hilos, de hashing, etc.
No se acepta el uso de bibliotecas que oculten el uso de sockets, debe usar la API de sockets directamente.

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 sus dos aplicaciones servidor en Mininet y ejecutar las pruebas definidas. Es importante que las pruebas contemplen todos los casos posibles de uso de la especificación JSONRPC así como de los errores definidos.

En particular 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 la 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 proyecto, pero aquí encontrará más información sobre Mininet (si tiene curiosidad).

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 :
https://mega.nz/file/l7xWSY7L#lOK8_7N2kahnvnZlRxy9yLGYuh4cn0jqAR5qxxmZcEg.

Descargue el aplicativo e importe la máquina 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:
```

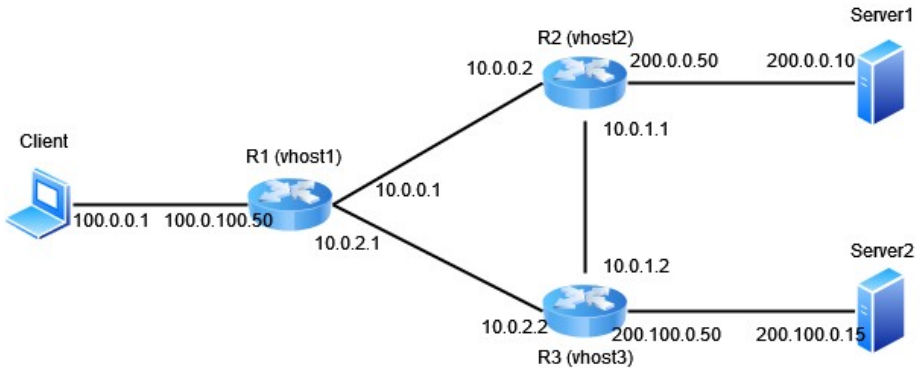
El usuario configurado es `osboxes` y la password `osboxes.org`.

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 ~/redes2024_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 200.0.0.10
server2 200.100.0.15
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
*** Successfully loaded ip settings for hosts
 {'vhost1-eth2': '10.0.0.1', 'vhost1-eth3': '10.0.2.1', 'server2':
'200.100.0.15', 'server1': '200.0.0.10', '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', '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
*** Adding links:
(client, vhost1) (server1, vhost2) (server2, vhost3) (vhost2,
vhost1) (vhost3, vhost1) (vhost3, vhost2)
*** Configuring hosts
client server1 server2
*** Starting controller
c0
*** Starting 3 switches
vhost1 vhost2 vhost3 ...
*** setting default gateway of host server1
server1 200.0.0.50
*** setting default gateway of host server2
server2 200.100.0.50
*** setting default gateway of host client
client 100.0.0.50
*** Starting JSONRPC Server on host server1
*** Starting JSONRPC Server on host server2
*** 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 ~/redes2024_ob1/
>./run_pox.sh
```

Debería ver una salida como la siguiente:

```
POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.
server1 200.0.0.10
server2 200.100.0.15
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
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:[00-00-00-00-00-02 1] connected
{'vhost2': {'eth1': ('10.0.0.2', '12:cc:fe:2a:b4:44', '10Gbps',
1)}}
{'vhost2': {'eth2': ('200.0.0.50', 'ae:6c:77:70:79:03', '10Gbps',
2), 'eth1': ('10.0.0.2', '12:cc:fe:2a:b4:44', '10Gbps', 1)}}
{'vhost2': {'eth3': ('10.0.1.1', 'd6:66:06:ef:8a:84', '10Gbps',
3), 'eth2': ('200.0.0.50', 'ae:6c:77:70:79:03', '10Gbps', 2),
'eth1': ('10.0.0.2', '12:cc:fe:2a:b4:44', '10Gbps', 1)}}
INFO:openflow.of_01:[00-00-00-00-00-03 3] connected
{'vhost3': {'eth1': ('10.0.2.2', '02:42:c7:d5:90:1d', '10Gbps',
1)}}
{'vhost3': {'eth2': ('200.100.0.50', 'ea:11:55:9e:16:97',
'10Gbps', 2), 'eth1': ('10.0.2.2', '02:42:c7:d5:90:1d', '10Gbps',
1)}}
{'vhost3': {'eth3': ('10.0.1.2', 'ba:a2:d4:c2:cc:68', '10Gbps',
3), 'eth2': ('200.100.0.50', 'ea:11:55:9e:16:97', '10Gbps', 2),
'eth1': ('10.0.2.2', '02:42:c7:d5:90:1d', '10Gbps', 1)}}
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
{'vhost1': {'eth1': ('100.0.0.50', '1a:35:f9:5b:d6:ef', '10Gbps',
1)}}
{'vhost1': {'eth2': ('10.0.0.1', 'f6:8a:02:32:3d:f6', '10Gbps',
2), 'eth1': ('100.0.0.50', '1a:35:f9:5b:d6:ef', '10Gbps', 1)}}
{'vhost1': {'eth3': ('10.0.2.1', 'ae:25:15:6f:73:f1', '10Gbps',
3), 'eth2': ('10.0.0.1', 'f6:8a:02:32:3d:f6', '10Gbps', 2),
'eth1': ('100.0.0.50', '1a:35:f9:5b:d6:ef', '10Gbps', 1)}}
```

4) Mantenga POX en funcionamiento. Ahora, abra 3 nuevas terminales para continuar con el siguiente paso y en cada una coloquese en el directorio `redes2024_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
```

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 y también probaremos un ejemplo de cliente y servidor RPC ya implementados.

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 3 200.0.0.10
```

Debería ver una salida como la siguiente:

```
PING 200.0.0.10 (200.0.0.10) 56(84) bytes of data.
64 bytes from 200.0.0.10: icmp_seq=1 ttl=62 time=230 ms
64 bytes from 200.0.0.10: icmp_seq=2 ttl=62 time=97.2 ms
64 bytes from 200.0.0.10: icmp_seq=3 ttl=62 time=116 ms
```

```
--- 200.0.0.10 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 `server1`:

```
mininet> client traceroute -n 200.0.0.10
```

Debería ver la siguiente salida:

```
traceroute to 200.0.0.10 (200.0.0.10), 30 hops max, 60 byte
packets
 1  100.0.0.50  189.929 ms  189.695 ms  115.362 ms
 2  10.0.0.2  443.759 ms  443.809 ms  443.809 ms
 3  200.0.0.10  395.513 ms  404.499 ms  404.294 ms
```

3) Finalmente, probaremos el cliente y servidor RPC de pruebas:

```
mininet> client cd dist/; python3.8 test-client.py
```

Debería ver la siguiente salida:

```
============================
Iniciando pruebas de casos sin errores.
CLIENT | REQUEST: {"jsonrpc": "2.0", "method": "echo", "params":
["Testing!"], "id": "25c7c05a-370d-4d35-9eec-ee6d88160df4"}
CLIENT | RESPONSE: {"jsonrpc": "2.0", "result": "Testing!", "id":
"25c7c05a-370d-4d35-9eec-ee6d88160df4"}
Test simple completado.
CLIENT | REQUEST: {"jsonrpc": "2.0", "method": "echo", "params":
{"message": "No response!"}}
CLIENT | RESPONSE:
Test de notificación completado.
CLIENT | REQUEST: {"jsonrpc": "2.0", "method": "echo_concat",
"params": ["a", "b", "c", "d"], "id": "10d5f94d-6e66-407e-a318-
bdd844ad3346"}
CLIENT | RESPONSE: {"jsonrpc": "2.0", "result": "abcd", "id":
"10d5f94d-6e66-407e-a318-bdd844ad3346"}
Test de múltiples parámetros completado
CLIENT | REQUEST: {"jsonrpc": "2.0", "method": "echo_concat",
"params": {"msg1": "a", "msg2": "b", "msg3": "c", "msg4": "d"},
"id": "c0241150-8370-4786-ac9d-4877a12419be"}
CLIENT | RESPONSE: {"jsonrpc": "2.0", "result": "abcd", "id":
"c0241150-8370-4786-ac9d-4877a12419be"}
Test de múltiples parámetros con nombres completado
CLIENT | REQUEST: {"jsonrpc": "2.0", "method": "echo", "params":
{"message": 5}, "id": "8266c726-3050-406c-8f08-d914999c65ef"}
CLIENT | RESPONSE: {"jsonrpc": "2.0", "result": 5, "id":
"8266c726-3050-406c-8f08-d914999c65ef"}
Otro test simple completado.
CLIENT | REQUEST: {"jsonrpc": "2.0", "method": "sum", "params":
[1, 2, 3, 4, 5], "id": "23c6982e-1277-4918-946b-6a3ae858e832"}
CLIENT | RESPONSE: {"jsonrpc": "2.0", "result": 15, "id":
"23c6982e-1277-4918-946b-6a3ae858e832"}
Test de suma completado.
```

```
CLIENT | REQUEST: {"jsonrpc": "2.0", "method": "sum", "params":
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10], "id": "73d14602-b6e8-4109-ab09-
ed1ab101e366"}
CLIENT | RESPONSE: {"jsonrpc": "2.0", "result": 55, "id":
"73d14602-b6e8-4109-ab09-ed1ab101e366"}
Segundo test de suma con 10 parámetros completado
============================
Pruebas de casos sin errores completadas.
============================
Iniciando pruebas de casos con errores.
CLIENT | REQUEST: {"jsonrpc": "2.0", "method": "echo", "id":
"12facf41-6463-4e3e-9574-836f4fd321aa"}
CLIENT | RESPONSE: {"jsonrpc": "2.0", "error": {"message":
"Internal error.", "code": -32603}, "id": "12facf41-6463-4e3e-
9574-836f4fd321aa"}
Llamada incorrecta sin parámetros. Genera excepción necesaria.
-32603 Internal error.
CLIENT | REQUEST: {"jsonrpc": "2.0", "method": "foobar", "params":
[5, 6], "id": "7a8057ec-165b-440d-830c-61511e8984d3"}
CLIENT | RESPONSE: {"jsonrpc": "2.0", "error": {"message": "Method
not found.", "code": -32601}, "id": "7a8057ec-165b-440d-830c-
61511e8984d3"}
Llamada a método inexistente. Genera excepción necesaria.
-32601 Method not found.
CLIENT | REQUEST: {"jsonrpc": "2.0", "method": "echo_concat",
"params": ["a", "b", "c"], "id": "013e62a0-fd9b-4d00-8545-
f565123e3719"}
CLIENT | RESPONSE: {"jsonrpc": "2.0", "error": {"message":
"Internal error.", "code": -32603}, "id": "013e62a0-fd9b-4d00-
8545-f565123e3719"}
Llamada incorrecta genera excepción interna del servidor.
-32603 Internal error.
Llamada incorrecta genera excepción en el cliente.
JSON spec allows positional arguments OR keyword arguments, not
both.
============================
Pruebas de casos con errores completadas.
```

Evaluación de la implementación

Como mencionamos anteriormente, como parte de la VM se entrega un cliente y servidor de pruebas junto con una implementación de la biblioteca RPC que puede utilizar para probar su implementación.

Para realizar las pruebas de su implementación primero deberá detener los servidores de prueba que se levantan de forma automática al iniciar Mininet. Esto lo puede hacer con el comando:

```
> pkill -9 -f test-server
```

Luego, 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: Julio 2024
[2] Analizador de Tráfico Wireshark. Accesible en línea: http://www.wireshark.org/. Última visita: Julio 2024
[3] ping(8) - Linux man page. Accesible en línea: https://linux.die.net/man/8/ping. Última visita: Julio 2024.
[4] traceroute(8) - Linux man page. Accesible en línea: https://linux.die.net/man/8/traceroute Última visita: Julio 2024.
[5] tracepath(8) - Linux man page. Accesible en línea: https://linux.die.net/man/8/tracepath. Última visita: Julio 2024.
[6] tracert Documentation. Accesible en línea: https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/tracert. Última visita: Julio 2024.
[7] https://www.jsonrpc.org/specification. Última visita: Julio 2024
[8] https://www.ietf.org/rfc/rfc4627.txt. Última visita: Julio 2024
[9] https://python-reference.readthedocs.io/en/latest/docs/dunderattr/getattr.html. Última visita: Julio 2024
[10] https://www.geeksforgeeks.org/__call__-in-python/. Última visita: Julio 2024
[11] 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: Julio 2024