Redes de Computadoras

Obligatorio 2

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 obligatorio2GrupoGG.tar.gz que deberá contener los siguientes archivos:

- Un documento llamado Obligatorio2GrupoGG.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 de su solución.

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

Fecha de entrega

Los trabajos deberán ser entregados **antes del lunes 18/11/2024 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

- Comprender los conceptos básicos vistos en el teórico del curso de reenvío (*forwarding)* y enrutamiento *(routing)*, así como de numeración IPv4, direcciones MAC, protocolo ARP y protocolos de enrutamiento.
- Entrenarse en el uso de una herramienta de emulación, en este caso: mininet.

Descripción general del problema

El obligatorio consiste en el desarrollo de las funcionalidades de reenvío (*forwarding)* y enrutamiento (*routing)* de un enrutador (*router*) IP. Para esto, el obligatorio se dividirá en 2 partes, la primera consiste en el desarrollo de algunas

de las funcionalidades necesarias para implementar el reenvío de paquetes en el *router* a partir de una tabla de enrutamiento estática definida por el usuario. La segunda parte consistirá en la implementación del protocolo de enrutamiento PWOSPF de forma de poder intercambiar información entre los routers y construir una tabla de enrutamiento dinámica.

Entorno de trabajo

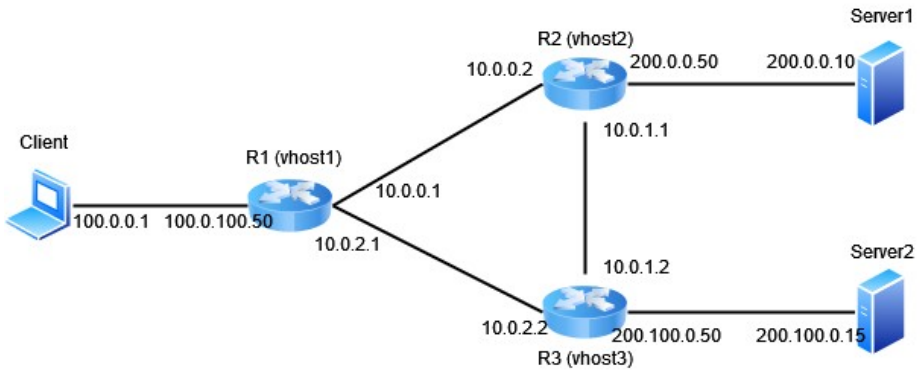
El obligatorio se deberá desarrollar utilizando el emulador Mininet (ya visto en el obligatorio 1), el cual puede ser instalado de forma local en un ambiente Linux. Sin embargo, como parte de los materiales del obligatorio, se provee una máquina virtual con Mininet ya instalado y con los archivos de configuración necesarios para la ejecución de la tarea.

Parte 1

Descripción general

Como se mencionó previamente, en esta parte el objetivo es desarrollar la función de reenvío de un *router* con una tabla de enrutamiento estático. Su *router* recibirá tramas *raw* Ethernet (es decir, toda la trama en bruto) y procesará los paquetes como un *router* real para luego reenviarlos a la interfaz de salida correcta. Su trabajo será crear la lógica de reenvío para que los paquetes vayan a la interfaz correcta.

El escenario de pruebas a utilizar será el ya utilizado en el Obligatorio 1, el cual consiste en una topología con 3 enrutadores, 2 servidores y 1 cliente (ver figura). Deberá implementar el software que ejecutará en los 3 enrutadores de forma que puedan enrutar los paquetes reales recibidos desde y hacia el cliente y los servidores. Los servidores de aplicaciones ejecutan cada uno un servidor RPC desarrollado en el Obligatorio 1.



Cuando haya programado el reenvío en los *routers*, debería poder acceder a estos servidores utilizando su cliente RPC. Además, debe poder hacer **ping** y **traceroute** hacia y a través de los *routers*. Si los *routers* funcionan correctamente, todas las siguientes operaciones deberían funcionar:

- Hacer ping desde el cliente a cualquiera de las interfaces de cualquier *router*.
- Hacer traceroute desde el cliente a cualquiera de las interfaces de cualquier *router*.
- Hacer ping desde el cliente a cualquiera de los servidores.
- Hacer traceroute desde el cliente a cualquiera de los servidores.
- Hacer una llamada RPC desde el cliente a cualquiera de los servidores.

Funcionalidad requerida

- Los *routers* deben enrutar correctamente los paquetes entre el cliente y los servidores.
- Los *routers* deben manejar correctamente las solicitudes y respuestas ARP.
- Los *routers* deben manejar correctamente los mensajes de traceroute a través de ellos (donde el router no es el *host* final) y hacia ellos (donde el router es el *host* final).

- Los *routers* deben responder correctamente a las solicitudes de echo ICMP.
- Los *routers* deben manejar los paquetes TCP/UDP enviados a una (cualquiera) de sus interfaces. En este caso, los *routers* deben responder con un mensaje ICMP *port unreachable*.
- El router no debe descartar paquetes innecesariamente (por ejemplo, cuando espera una respuesta ARP)

Lógica de reenvío

El reenvío se puede dividir en dos actividades principales: manejo de ARP y reenvío de IP. El manejo de ARP, aunque parte fundamental del reenvío, no será parte de las funcionalidades a resolver en este obligatorio.

En cuanto al reenvío IP, dada una trama raw Ethernet, si la trama contiene un paquete IP que no está destinado a una de las interfaces del *router* se debe:

- Controlar la "sanidad" del paquete (cumple con la longitud mínima y tiene la suma de verificación correcta).
- Disminuir el TTL en 1 y volver a calcular la suma de comprobación del paquete sobre el encabezado modificado.
- Encontrar qué entrada en la tabla de reenvío tiene la coincidencia de prefijo más largo (*longest prefix match*) con la dirección IP de destino.
- Verificar el caché ARP para encontrar la dirección MAC del siguiente salto correspondiente a la IP del siguiente salto. Si está allí, se puede enviar. De lo contrario, se debe enviar una solicitud ARP para la IP del siguiente salto (si no se envió una en el último segundo) y agregar el paquete a la cola de paquetes que esperan esta solicitud ARP.

Esta es una versión muy simplificada del proceso de reenvío, cuyos detalles deberá ir entendiendo y desarrollando mientras realiza la implementación. Por ejemplo, si se produce un error en cualquiera de los pasos anteriores, se deberá enviar un mensaje ICMP al remitente para notificarle un error. También puede obtener una solicitud o respuesta ARP, que tiene que interactuar con la caché ARP correctamente.

Protocolos a entender

Ethernet

Se le proporciona una trama raw Ethernet y tiene que enviar tramas raw Ethernet. Debe comprender las direcciones MAC de origen y destino y la idea de que reenviamos la trama en un enlace cambiando la dirección MAC de destino (la dirección MAC de la interfaz entrante del siguiente salto).

Internet Protocol (IP)

Antes de operar en un paquete IP, debe verificar su suma de verificación y asegurarse de que cumpla con la longitud mínima de un paquete IP. Debe comprender cómo encontrar la coincidencia de prefijo más larga de una dirección IP de destino en la tabla de reenvío. Si determina que se debe reenviar un datagrama, debe disminuir correctamente el campo TTL del encabezado y volver a calcular la suma de verificación sobre el encabezado modificado antes de reenviarlo al siguiente salto.

Internet Control Message Protocol (ICMP)

ICMP es un protocolo simple que puede enviar información de control a un *host*. En esta tarea, su enrutador utilizará ICMP para enviar mensajes de vuelta a un *host* emisor. Deberá generar correctamente los siguientes mensajes ICMP (incluida la suma de comprobación del encabezado ICMP) en respuesta al *host* emisor en las siguientes condiciones:

- Echo reply (type 0): Enviado en respuesta a una “echo request” (ping) a una de las interfaces del enrutador. (Esto es solo para echo requests a cualquiera de las IP del enrutador. Un echo request enviado a otro lugar debe enviarse a la dirección del siguiente salto como de costumbre).
- Destination net unreachable (type 3, code 0): Enviado si hay una ruta inexistente a la IP de destino (no hay una entrada coincidente en la tabla de enrutamiento al reenviar un paquete IP).
- Destination host unreachable (type 3, code 1): Enviado si se enviaron cinco solicitudes ARP a la IP del siguiente salto sin respuesta.
- Port unreachable (type 3, code 3): Enviado si un paquete IP que contiene una carga útil UDP o TCP se envía a una de las interfaces del enrutador. Esto es necesario para que traceroute funcione.
- Time exceeded (type 11, code 0): Enviado si un paquete IP se descarta durante el procesamiento porque el campo TTL es 0. Esto también es necesario para que el traceroute funcione. La dirección de origen de un mensaje ICMP puede ser la dirección de origen de cualquiera de las interfaces entrantes, como se especifica en RFC 792. Como se mencionó anteriormente, el único mensaje ICMP entrante destinado a las IP del enrutador que debe procesar explícitamente son ICMP echo requests.

Protocolo de Resolución de Dirección (ARP)
Se necesita ARP para determinar la dirección MAC del siguiente salto que corresponde a la dirección IP del siguiente salto almacenada en la tabla de reenvío. Sin la capacidad de generar una solicitud ARP y procesar respuestas ARP, su enrutador no podría completar el campo de dirección MAC de destino de la trama raw Ethernet que está enviando a través de la interfaz saliente. Análogamente, sin la capacidad de procesar solicitudes ARP y generar respuestas ARP, ningún otro enrutador podría enviar sus tramas Ethernet. Por lo tanto, su enrutador debe generar y procesar solicitudes y respuestas ARP. Como se mencionó previamente, el manejo interno de ARP será proporcionado por los docentes.

Para disminuir el número de solicitudes ARP enviadas, se almacenan en caché las respuestas ARP. Al reenviar un paquete a una dirección IP del siguiente salto, el enrutador primero debe verificar la caché ARP para la dirección MAC correspondiente antes de enviar una solicitud ARP. En el caso de no encotrarse la dirección en caché, debe enviarse una solicitud ARP a la dirección IP de destino. Si la solicitud ARP se envía cinco veces sin respuesta, se debe retornar un código ICMP de destino inalcanzable a la IP de origen como se indicó anteriormente.

En el caso de recibir una solicitud ARP, se debe enviar una respuesta ARP si la dirección IP de destino es una de las direcciones IP de su enrutador. En el caso de una respuesta ARP, solo debe almacenar en caché la entrada si la dirección IP de destino es una de las direcciones IP de su enrutador.

Tenga en cuenta que las solicitudes ARP se envían a la dirección MAC de difusión (ff-ff-ff-ff-ff-ff). Las respuestas ARP se envían directamente a la dirección

MAC del solicitante.

Destinos de paquetes IP

Un paquete IP entrante puede estar destinado a una de las direcciones IP de su enrutador, o puede estar destinado a otra parte. Si se envía a una de las direcciones IP de su enrutador, debe realizar las siguientes acciones:

- Si el paquete es un ICMP echo request y su suma de comprobación es válida, envíe una respuesta ICMP echo reply al host emisor.
- Si el paquete contiene una carga útil TCP o UDP, envíe un mensaje ICMP port unreachable al host emisor. De lo contrario, ignore el paquete. Los paquetes destinados a otros lugares deben reenviarse utilizando su lógica de reenvío normal.

Descripción general del código

Su enrutador recibe una trama r*aw Ethernet* y envía tramas r*aw Ethernet* cuando envía una respuesta al host emisor o reenvía la trama al siguiente salto. Las funciones básicas para manejar estas funciones son:

```
void sr_handlepacket (struct sr_instance* sr, uint8_t* packet,
unsigned int len, char* interface)
```

Este método, ubicado en `sr_router.c`, es llamado por el enrutador cada vez que se recibe un paquete. El argumento "`packet`" apunta al búfer de paquetes que contiene el paquete completo, incluido el encabezado de ethernet. El nombre de la interfaz de recepción también se pasa al método.

```
int sr_send_packet (struct sr_instance* sr, uint8_t* buf, unsigned
int len, const char * iface)
```

Este método, ubicado en `sr_vns_comm.c`, enviará un paquete arbitrario de longitud, `len`, a la red fuera de la interfaz especificada por `iface`.

No debe liberar el búfer que se le dio en `sr_handlepacket` (es por eso que el búfer está etiquetado como "`lent`" en los comentarios). Usted es responsable de hacer una gestión correcta de la memoria en los buffers que `sr_send_packet` le presta (es decir, `sr_send_packet` no liberará la memoria de los buffers que le pase).

```
void sr_arpcache_sweepreqs (struct sr_instance* sr)
```

Esta función envía una solicitud ARP aproximadamente una vez por segundo hasta que vuelva una respuesta o se hayan enviado cinco solicitudes. Esta función se define en `sr_arpcache.c` y se llama cada segundo, e itera a través de la cola de solicitudes ARP y reenvía cualquier solicitud ARP pendiente que no se haya enviado en el último segundo. Si una solicitud ARP se ha enviado 5 veces sin respuesta, se debe enviar un mensaje de “destination host unreachable” a todos los remitentes de paquetes que esperaban una respuesta a esta solicitud ARP.

El uso del cache ARP se describe en el siguiente pseudocódigo:

```
/* Al momento de enviar un paquete IP */
   /*buscar si la IP está en la cache ARP*/
   entry = sr_arpcache_lookup(next_hop_ip)

   /*si está en la cache*/
   if entry:
       usar mapping next_hop_ip->mac en entry para enviar el
paquete
       free entry
   else:  /*si no está*/
       /*agrego el paquete a la cola de hasta que se resuelva el
ARP y obtengo el request*/
       req = sr_arpcache_queuereq(next_hop_ip, packet, len)

      /* paso el request a la función que decide cuando se debe
enviar */
       handle_arpreq(req)
```

El código que procesa los ARP reply debe mover las entradas desde la cola de ARP request a la cache:

```
/* Cuando se recibe un ARP reply con un mapeo IP->MAC*/
   /* Inserto el mapeo en la cache. */
   req = arpcache_insert(ip, mac)
   /* La función me devuelve la lista de paquetes pendientes que
esperaban por ese reply*/

   if req:
       send all packets on the req->packets linked list
       sr_arpreq_destroy(req)
```

Estructuras de datos

El enrutador (sr_router.h):
El contexto completo del enrutador se encuentra en la estructura `sr_instance` (sr_router.h). `sr_instance` contiene información sobre la topología para la que el enrutador está enrutando, así como la tabla de enrutamiento y la lista de interfaces.

Interfaces (sr_if.c/h):
Mantiene una lista encadenada de interfaces en la instancia del enrutador en el miembro `if_list`. Los métodos de utilidad para manejar la lista de interfaces se pueden encontrar en sr_if.c/h.

La tabla de enrutamiento (sr_rt.c/h):
La tabla de enrutamiento en el código entregado se lee desde un archivo (el nombre de archivo predeterminado es "rtable" pero se puede cambiar con la opción de línea de comando -r) y se almacena en una lista encadenada de entradas de enrutamiento en la instancia de enrutamiento actual (routing_table).

La caché ARP y la cola de solicitudes ARP (sr_arpcache.c/h):
Deberá agregar solicitudes ARP y paquetes en espera de respuestas a esas solicitudes ARP a la cola de solicitudes ARP. Cuando llegue una respuesta ARP, deberá eliminar la solicitud ARP de la cola y colocarla en la caché ARP,

reenviando los paquetes que estaban esperando esa solicitud ARP. El pseudocódigo para estas operaciones se proporciona en sr_arpcache.h. El código base ya crea un hilo que agota las entradas de caché ARP 15 segundos después de que se agreguen para usted. Debe completar la función sr_arp_request_send en sr_router.c que se llama cuando es necesario enviar una solicitud ARP.

Encabezados de protocolo (sr_protocol.h):
Dentro de la tarea, usted se ocupará directamente de los paquetes raw Ethernet. El código entregado ya proporciona algunas estructuras de datos en sr_protocols.h que puede usar para manipular fácilmente los encabezados.

Los detalles de cada encabezado los puede encontrar en las RFC correspondinetes: para ARP [RFC826] (**http://www.ietf.org/rfc/rfc826.txt**), IP [RFC791] (**http://www.ietf.org/rfc/rfc791.txt**) e ICMP [RFC792] (http://www.ietf.org/rfc/rfc792.txt).

Ejemplo de creación de un paquete
A continuación se muestra un código de ejemplo que crea un paquete ARP e inicializa los encabezados Ethernet y ARP. Eso es parte del código ya entregado.

```
int arpPacketLen = sizeof(sr_ethernet_hdr_t) +
sizeof(sr_arp_hdr_t);
uint8_t *arpPacket = malloc(arpPacketLen);

sr_ethernet_hdr_t *ethHdr = (struct sr_ethernet_hdr *) arpPacket;
memcpy(ethHdr->ether_dhost, <host destino>, ETHER_ADDR_LEN);

memcpy(ethHdr->ether_shost, <host origen>, sizeof(uint8_t) *
ETHER_ADDR_LEN);
ethHdr->ether_type = htons(ethertype_arp);

sr_arp_hdr_t *arpHdr = (sr_arp_hdr_t *) (arpPacket +
sizeof(sr_ethernet_hdr_t));
arpHdr->ar_hrd = htons(1);
arpHdr->ar_pro = htons(2048);
arpHdr->ar_hln = 6;
arpHdr->ar_pln = 4;
arpHdr->ar_op = htons(arp_op_request);
memcpy(arpHdr->ar_sha, <sender address>, ETHER_ADDR_LEN);
memcpy(arpHdr->ar_tha, <target address>, ETHER_ADDR_LEN);
arpHdr->ar_sip = <sender IP>;
arpHdr->ar_tip = <target IP>;
```

Funciones de debbuging
Le proporcionamos algunas funciones básicas de debbuging en sr_utils.h, sr_utils.c. Siéntase libre de usarlos para imprimir la información del encabezado de red de sus paquetes. A continuación se presentan algunas funciones que puede encontrar útiles:

- `print_hdrs (uint8_t * buf, uint32_t length)`: imprime todos los encabezados posibles a partir del encabezado Ethernet en el paquete
- `print_addr_ip_int (uint32_t ip)`: imprime una dirección IP formateada desde un uint32_t. Asegúrese de pasar la dirección IP en el orden de bytes correcto.

El código entregado

El código de esta parte se encuentra en:
https://gitlab.fing.edu.uy/mrichart/redes2024_ob2

Para descargarlo puede hacer:
```
> git clone
https://gitlab.fing.edu.uy/mrichart/redes2024_ob2.git
```

El código incluye los archivos necesario para ejecutar mininet (como en el Obligatorio 1) y en el directioro `reenvio` los arvhivos iniciales para su implementación de la Parte 1.

Puede compilar el código de la siguiente manera:

```
> cd ~/redes2024_ob2/reenvio
> make
```

Esto genera un ejecutable `sr`. Para ejecutarlo en cada router deberá copiarlo al directiorio `redes2024_ob2`, modificar el script `run_sr.sh` para que utilice su ejecutable y ejecutar de la misma forma que lo hacía en el Obligatorio 1 (recuerde que antes debe seguir todos los pasos para levantar mininet):

```
> ./run_sr.sh 127.0.0.1 vhost[1|2|3]
```

Puede registrar los paquetes recibidos y generados por su programa `sr` utilizando el parámetro "`-l`" con su programa `sr`. El archivo estará en formato `pcap`, es decir, puede usar wireshark o tcpdump para leerlo. Para esto debe modificar el script `run_sr.sh`.

Pruebas

Recomendamos definir un set básico de pruebas para utilizar durante el desarrollo de su solución que contemple la evalaución de las funcionalidades requeridas así como de otros casos que haya considerado (por ejemplo mensajes ICMP de error, paquetes o destinos no soportados, etc).
Para esto puede hacer uso de su aplicación RPC del Obligatorio 1 así como de las utilidades `ping` y `traceroute`. Documente las pruebas realizadas.

Parte 2

Descripción general

Como se mencionó previamente, en esta parte el objetivo es desarrollar la función de enrutamiento de un *router*, es decir, encontrar de forma dinámica y autónoma las rutas a los distintos destinos de la red. El objetivo es desarrollar un protocolo de enrutamiento dinámico simple, PWOSPF, para que el router pueda generar su tabla de reenvío automáticamente basándose en las rutas anunciadas por otros routers de la red. Al final de este proyecto, se espera que su router sea capaz de construir su tabla de reenvío a partir de anuncios de estado de enlace enviados desde otros routers, y enrutar el tráfico a través de topologías complejas que contienen múltiples nodos de forma de tener exactamente la misma funcionalidad que en la Parte 1.

Puede considerar esta parte como una extensión de la primera parte, donde deberá mantener toda su solución de reenvío y solo deberá agregar lo necesario para implementar la nueva funcionalidad. Para esta parte utilizaremos el mismo entorno y topología que en la parte anterior.

Su tarea es implementar PWOSPF dentro de su router para que su router sea capaz de hacer lo siguiente:

- Construir las tablas de reenvío correctas en la topología dada.
- Detectar cuando los routers se unen o abandonan la topología y corregir las tablas de reenvío correctamente.

PWOSPF

El protocolo de enrutamiento que implementará es un protocolo de estado de enlace que se basa vagamente en OSPFv2. Se encuentra disponible la especificación completa de PWOSPF aquí. Recomendamos estudiar detalladamente la especificación antes de comenzar a implementar. Tenga en cuenta que, aunque PWOSPF se basa en OSPFv2, es lo suficientemente diferente como para que referirse a OSPFv2 como referencia no sea de mucha ayuda y, por el contrario, pueda confundirle o inducirle a error.

Descripción del código

Para la implementación de esta segunda parte tendrá a su disposición código modificado que crea una estructura de subsistema PWOSPF dentro de `sr_instance` e inicia un hilo que puede utilizarse para implementar la infraestructura de temporizador necesaria. El código también incluye un archivo de cabecera `pwospf_protocol.h` que contiene definiciones útiles. Este código así como una explicación mas detallada de lo que deberá implementar estará disponible a partir del 14/10.

Algunas consideraciones a tener en cuenta

Cómo tratar las rutas

Cuando se implementa PWOSPF, se debe tener especial cuidado para asegurar que las rutas son tratadas correctamente por su router. Antes de comenzar el desarrollo de PWOSPF, le sugerimos que primero se asegure de que su router maneja correctamente las rutas por defecto, los *gateways* y las subredes.

Rutas estáticas vs. dinámicas

Durante el funcionamiento, la tabla de enrutamiento debe contener tanto rutas estáticas como dinámicas. Las rutas estáticas se leen desde la tabla de enrutamiento (rtable) y nunca deben ser eliminadas o modificadas. Las rutas dinámicas son añadidas/eliminadas por su implementación PWOSPF. Por lo tanto, debe mantener un registro de qué rutas son estáticas. Puede manejar esto como quiera, por ejemplo manteniendo dos tablas separadas, o marcando las rutas como estáticas/dinámicas. Para mantener la coherencia, el router debe seleccionar siempre las rutas dinámicas sobre las estáticas, independientemente del tamaño del prefijo.

Qué rutas anunciar

El router es responsable de anunciar todas sus rutas estáticas junto con las subredes conectadas a cada una de sus interfaces. Dado cualquier enrutador en la red, agregando toda esta información correctamente se debería crear un grafo a partir del cual se puede calcular la tabla de enrutamiento correcta para ese enrutador hacia cada una de las subredes anunciadas. Exactamente cómo hacer esto es parte del desafío de la tarea.

No deberá añadir una ruta en su tabla de enrutamiento para subredes conectadas directamente a una de sus interfaces, incluso si estas subredes están siendo anunciadas (en la mayoría de los casos lo estarán, por ejemplo, si hay otro enrutador en la subred).

Condiciones de carrera

Es casi seguro que necesitará utilizar hilos (*threads*) en esta tarea para soportar las actualizaciones periódicas (por ejemplo, paquetes HELLO). En este sentido, se deberá tener cuidado de evitar condiciones de carrera. La tabla de enrutamiento tendrá que ser bloqueada adecuadamente durante las actualizaciones.

El código entregado

El código de esta parte se encuentra en:
https://gitlab.fing.edu.uy/mrichart/redes2024_ob2

Para descargarlo puede hacer:

```
> git clone
https://gitlab.fing.edu.uy/mrichart/redes2024_ob2.git
```

El código incluye los archivos necesario para ejecutar mininet (como en el

Obligatorio 1) y en el directioro `enrutamiento` los arvhivos iniciales para su implementación de la Parte 2.

Puede compilar el código de la siguiente manera:

```
> cd ~/redes2024_ob2/enrutamiento
> make
```

Esto genera un ejecutable `sr`. Para ejecutarlo en cada router deberá copiarlo al directiorio `redes2024_ob2`, modificar el script `run_sr.sh` para que utilice su ejecutable y ejecutar de la misma forma que lo hacía en el Obligatorio 1 (recuerde que antes debe seguir todos los pasos para levantar mininet):

```
> ./run_sr.sh 127.0.0.1 vhost[1|2|3]
```

Es importante tener en cuenta que para esta segunda parte deberá modificar las tablas de enrutamiento `rtable.vhost` para que solo incluyan las rutas necesarias y las demás sean agregadas por el algoritmo PWOSPF.

Puede registrar los paquetes recibidos y generados por su programa `sr` utilizando el parámetro "`-l`" con su programa `sr`. El archivo estará en formato `pcap`, es decir, puede usar wireshark o tcpdump para leerlo. Para esto debe modificar el script `run_sr.sh`.

Pruebas

Las pruebas de esta parte deben diseñarse cuidadosamente, ya que luego de la convergencia, no necesariamente habrá nuevos cambios. Por lo tanto, sugerimos añadir código a su *router* para que pueda desactivar una interfaz en tiempo de ejecución. Puede utilizar esto para probar si su implementación converge después de que un enlace se caiga.

El *router* debería ser capaz de construir las tablas de enrutamiento correctas y enrutar el tráfico entre el cliente y los servidores en la topología dada. Específicamente, deberá iniciar tres instancias de su *router* con la única ruta estática siendo la ruta por defecto en *vhost1* y luego ser capaz de alcanzar todos los nodos de la red dentro de una cantidad razonable de tiempo. El router debería ser capaz de corregir las tablas de enrutamiento si un enlace se cae.

En resumen, recomendamos definir un set básico de pruebas para utilizar durante el desarrollo de su solución que contemple la evalaución de las funcionalidades requeridas así como de otros casos que haya considerado (por ejemplo mensajes ICMP de error, paquetes o destinos no soportados, caída de nodos o enlaces, etc.). Para esto puede hacer uso de su aplicación RPC del Obligatorio 1 así como de las utilidades `ping` y `traceroute`.

Parte 3

Esta parte consiste en la evaluación de su solución completa (Parte 1 y Parte 2). Deberá definir un set de pruebas, ejecutar su aplicación RPC del Obligatorio 1 en Mininet y ejecutar las pruebas definidas.
Las pruebas deberán ser exahustivas, evaluando todas las funcionalidades requeridas en las partes anteriores así como los casos de error que haya contemplado. Deberá documentar las pruebas desarrolladas y su resultados. En particular se espera que haga uso de tcpdump y wireshark para mostrar el correcto funcionamiento de su implementación de PWOSPF. Para esto, deberá no solo considerar el caso de convergencia inicial sino también la caída y aparición de nodos y enlaces. También recomendamos estudiar la posibilidad de modificar la topología inicial, por ejemplo agregando nuevos routers.