

Informe Actividad 5

Mini-Internet

Nombre: Vicente Olivares Gómez

Diferencia entre tablas de ruta reales y utilizadas


El mini-Internet implementado en la actividad trabaja con tablas de ruta que no son como las que tiene un router en realidad. Tanto las tablas de la actividad como las reales contienen información sobre la red de destino y sobre el siguiente salto, pero solo las reales tienen información sobre la interfaz de salida. Además, en las tablas reales se utiliza la IP para identificar un router y nunca trabajan con el puerto, mientras que las de la actividad utilizan el par (IP, puerto) para identificar un router. Esto último es debido a que en la actividad solo se utiliza la IP de localhost y es el puerto el que diferencia un router de otro.

Tabla de rutas con *round-robin*

A la hora de hacer *forwarding*, si existe más de una ruta para redirigir a un paquete, se usa *round-robin* para decidir cual ruta tomar. Para manejar esto, se decidió trabajar con clases, particularmente con `RoutesTable`. Esta clase contiene la información de la tabla de rutas en la lista `port_ranges` y en el diccionario `round_robin_logic`. La lista `port_ranges` contiene pares que representan a los rangos de puertos que corresponden a cada ruta de la tabla de rutas, sin repeticiones, y el diccionario `round_robin_logic` asocia a cada rango de puertos de `port_ranges` a un diccionario con una lista de las rutas que le corresponde, el índice de la última ruta utilizada de la lista y el índice de la siguiente ruta a utilizar. Para que se mantenga la lógica de *round-robin*, cada vez que se utiliza una ruta de un rango de puertos, se actualizan los índices de última ruta utilizada y siguiente ruta a utilizar.

Router default

Para agregar un router default que sirva como siguiente salto para aquellas direcciones que no se encuentran en la tabla de rutas, este router debe incluir a todos los puertos en su rango. En esta implementación el rango de la ruta del router default incluye desde el puerto 0 hasta 10000. Como esta ruta puede redirigir cualquier puerto, se debe posicionar en la última fila de la tabla de rutas, puesto que esta se revisa en orden de arriba para abajo y si existiese una ruta en una fila inferior a la del router default, esta nunca sería utilizada, pues siempre se llegaría al router default antes. En las siguientes imágenes se puede observar la inclusión de una ruta default.



127.0.0.1	8880	8880	127.0.0.1	8880
127.0.0.1	8880	8880	127.0.0.1	8882
127.0.0.1	8882	8886	127.0.0.1	8880
127.0.0.1	8882	8886	127.0.0.1	8882

127.0.0.1	8880	8880	127.0.0.1	8880
127.0.0.1	8880	8880	127.0.0.1	8882
127.0.0.1	8882	8886	127.0.0.1	8880
127.0.0.1	8882	8886	127.0.0.1	8882
127.0.0.1	0	10000	127.0.0.1	7000

Pruebas Mini-Internet sin TTL

En caso de que se configure mal una de las tablas de rutas, es posible que se redirija un paquete IP a un router que no corresponda. Particularmente, si se introduce el cambio mencionado en el enunciado al archivo *rutas_R2_v2.txt*, los paquetes con destino final R3, en R2 se redireccionarán a R1 en vez de a R2, y como el destino del paquete sigue siendo R3, R1 lo redireccionará hacia R2. Esto genera un loop de envíos entre R1 y R2, y R3 nunca recibirá paquetes desde R1 ni R2.

Al probar la estructura de 5 routers, enviando varios paquetes de R1 a R5, se puede observar que la cantidad de saltos que dan los paquetes varían y son 3, que es la distancia mínima entre R1 y R5, o 5. Finalmente, debido al uso de *round-robin* los paquetes dan entre la cantidad mínima de saltos y casi el doble de saltos que el mínimo.

Al repetir las pruebas anteriores con la estructura de 7 routers, enviando paquetes de R1 a R5, se puede observar que la cantidad de saltos que dan los paquetes van desde 3, que es el mínimo, hasta 11 saltos, casi el cuádruple del mínimo. A continuación se pueden ver las tablas de rutas de la estructura utilizada.

```
127.0.0.1 8881 8886 127.0.0.1 8881
127.0.0.1 8881 8886 127.0.0.1 8882
```

Figura 1: Rutas R0

```
127.0.0.1 8880 8880 127.0.0.1 8880
127.0.0.1 8880 8880 127.0.0.1 8882
127.0.0.1 8882 8886 127.0.0.1 8880
127.0.0.1 8882 8886 127.0.0.1 8882
```

Figura 2: Rutas R1

```
127.0.0.1 8880 8881 127.0.0.1 8880
127.0.0.1 8880 8881 127.0.0.1 8881
127.0.0.1 8883 8886 127.0.0.1 8883
127.0.0.1 8883 8886 127.0.0.1 8884
127.0.0.1 8883 8886 127.0.0.1 8886
```

Figura 3: Rutas R2

```
127.0.0.1 8880 8882 127.0.0.1 8882
127.0.0.1 8880 8882 127.0.0.1 8885
127.0.0.1 8880 8882 127.0.0.1 8886
127.0.0.1 8884 8886 127.0.0.1 8882
127.0.0.1 8884 8886 127.0.0.1 8885
127.0.0.1 8884 8886 127.0.0.1 8886
```

Figura 4: Rutas R3

```
127.0.0.1 8880 8883 127.0.0.1 8882
127.0.0.1 8880 8883 127.0.0.1 8885
127.0.0.1 8885 8886 127.0.0.1 8882
127.0.0.1 8885 8886 127.0.0.1 8885
```

Figura 5: Rutas R4

```
127.0.0.1 8880 8884 127.0.0.1 8883
127.0.0.1 8880 8884 127.0.0.1 8884
127.0.0.1 8886 8886 127.0.0.1 8883
127.0.0.1 8886 8886 127.0.0.1 8884
```

Figura 6: Rutas R5

Pruebas Mini-Internet con TTL

Para poner a prueba el Mini-Internet con TTL se llevaron a cabo 2 tests, el primero consistía en repetir la prueba con una tabla de rutas mal configurada en una estructura de 3 routers e identificar diferencias en los resultados teniendo ahora TTL incorporado. Para el segundo test se enviaron las líneas de un archivo de texto en paquetes separados, utilizando la estructura de 5 routers. Estos paquetes fueron enviados desde el router R1 con destino al router R5.

Para la primera prueba se utilizó *TTL* = 10 y se observó que se sigue formando un loop entre los routers R1 y R2, sin embargo, este ya no es infinito, pues luego de que el paquete fuera redireccionado 10 veces, el valor de *TTL* del paquete llega a cero y es ignorado por uno de los routers.

En el segundo test, se utilizó un archivo de texto con 10 líneas y cada una era el número de línea repetido varias veces, comenzando desde 1 y considerando la línea 10 como 0. Esto se hizo para poder identificar con facilidad el orden original de las líneas. Al recibir todas las líneas del archivo en el router R5, se notó que estas no llegaron en orden.

Código de la Actividad

El código de esta actividad está dividido en 5 archivos: *prueba_router.py*, *router.py*, *Router_class.py*, *RoutesTable.py* y *tests.py*.

Archivo *prueba_router.py*

Para ejecutar este archivo es necesario entregar 3 argumentos, los headers IP de los paquetes que se enviarán, la dirección IP y el puerto del router que simula el código. Además, es necesario tener el archivo de texto *many_lines_file.txt* en el mismo directorio. Al ejecutar el código, se leen las líneas del archivo de texto y se encapsulan en los headers IP recibidos como argumento. Luego se crea un objeto de la clase Router para enviar las líneas encapsuladas, usando *round-robin*.

Archivo *RoutesTables.py*

En este archivo se encuentra definida la clase *RoutesTables*, que se utiliza para representar una tabla de rutas y manejar la lógica de *round-robin* a la hora de elegir una ruta. Esta clase tiene los siguientes atributos:

- ***file_name***: Nombre del archivo de texto que contiene la información de la tabla de rutas.
- ***port_ranges***: Lista de pares con los rangos de puerto de cada ruta de la tabla. Sigue el mismo orden que la tabla y no tiene pares repetidos.
- ***round_robin_logic***: Diccionario que asocia cada par de la lista *port_ranges* con un diccionario con la lista de rutas de tal rango de puertos, el índice de la última ruta usada y el índice de la próxima ruta a usar para redirigir.

Al llamar el constructor de la clase, se parsea la tabla de rutas dentro del archivo de texto y se guarda la información en *port_ranges* y en *round_robin_logic*. La clase tiene solo un método:

- ***check_routes(destination_address)***: Retorna la dirección (IP, puerto) del siguiente salto indicada por la tabla de rutas para llegar a la dirección de destino recibida. Si hay varias rutas disponibles, se utiliza *round_robin* para decidir cual usar.

Este método obtiene la dirección desde *port_ranges* y *round_robin_logic*, y actualiza los índices de última ruta utilizada y próxima ruta a utilizar.

Archivo *Router_class.py*

En este archivo se encuentra definida la clase *Router*, que se usa guardar la información y funcionalidades de un router. La clase *Router* tiene 5 atributos:

- ***IP***: Un string con la dirección IP del router.
- ***port***: El número de puerto del router
- ***dir***: La dirección (IP, puerto) del router
- ***routes_table***: Una instancia de la clase *RoutesTable* que representa la tabla de rutas del router.
- ***Socket***: El socket UDP utilizado para enviar paquetes IP a otros routers.

La clase tiene los siguientes métodos:

- ***parse_packet(IP_packet)***: Parsea el paquete IP recibido y retorna un diccionario con los diferentes headers y el contenido del paquete.
- ***create_packet(IP_packet_dict)***: Recibe un diccionario con headers IP y un área de datos, y retorna un paquete IP equivalente en bytes.
- ***check_routes(destination_address)***: Consulta a la tabla de rutas que ruta tomar para llegar la dirección de destino recibida.
- ***receive_packet()***: Recibe un paquete IP. Se ignora el paquete si su TTL llegó a cero. Si este router es el destino final del paquete, imprime su contenido, en caso contrario intenta redireccionar al paquete según la tabla de rutas.
- ***send(packet)***: Envía el paquete IP recibido. Se obtiene la dirección de destino desde los headers IP del paquete.

Archivo *router.py*

Este archivo sirve para simular un router. Para ejecutarlo, se requiere que se le entreguen 3 argumentos, la dirección IP del router, el puerto del router y el nombre del archivo de texto con la información de la tabla de rutas del router. Al ejecutar el archivo se crea un router con una instancia de la clase *Router* y se reciben paquetes IP hasta que se detenga la ejecución del código.

Archivo tests.py

Este archivo contiene tres funciones, *parameter_reception_test*, *parse_create_packet_test* y *check_routes_test*. Cada una ejecuta el test de alguna funcionalidad de los routers:

- ***parameter_reception_test***: imprime la dirección IP, el puerto y el nombre del archivo de la tabla de rutas recibidos. Esta función permite verificar que *router.py* recibe correctamente los argumentos que se le entregan.
- ***parse_create_packet_test***: Verifica que si se parsea un paquete IP y luego se crea un paquete IP a partir del diccionario resultante, se llega al mismo paquete IP.
- ***check_routes_test***: Verifica que la función *check_routers* entregue correctamente la ruta a tomar.

Al ejecutar este archivo, se invocan los test de las funciones *parse_packet*, *create_packet* y *check_routes*. El test de recepción correcta de parámetros se llama al ejecutar el archivo *router.py*