Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ciencias de la Computación

Redes

Código del curso: CC4303

Informe de Desarrollo C3

Actividad Forwarding Básico

Estudiante: Valeria Franciscangeli

Profesora: Ivana Bachmann
Prof. Auxiliar: Vicente Videla
Ayudantes: Andres Basaez
Esperanza Díaz
Franz Widerstrom
Valentina Esteban
Pablo V. Mascaró

Fecha de entrega: 24 de noviembre de 2023

Índice de Contenidos

1 Introducción **2**

2 Cómo ejecutar **2**

3 Sobre el código **2**

4 Forwarding Sin TTL **3**
4.1 Desarrollo . . . 3
4.1.1 Tabla de rutas . . . 3
4.1.2 Archivo router.py . . . 3
4.1.3 Archivo utils.py . . . 4
4.1.3.1 Funciones parse_packet y create_packet . . . 4
4.1.3.2 Función leer_archivo . . . 4
4.1.3.3 Función check_routes . . . 4
4.1.4 Configuración de 7 routers . . . 5
4.1.5 Envío de paquetes fuera de la red . . . 5
4.2 Tests de funcionalidad . . . 6

5 Forwarding Con TTL **8**
5.1 Desarrollo . . . 8
5.1.1 Modificaciones al código base de la Parte 2 . . . 8
5.1.2 Archivo prueba_router.py . . . 8
5.2 Tests de funcionalidad . . . 9

1. Introducción

En este informe se detalla el proceso de creación de un **Mini-Internet**, que emplea una versión básica de Forwarding con y sin Time To Live (TTL). El objetivo de este mini-internet es simular el funcionamiento de la Capa de Red, operando de manera local y utilizando diversos puertos locales en lugar de distintas direcciones IP de destino.

A continuación, se exponen las respuestas a algunas preguntas planteadas en el enunciado y se presentan los resultados obtenidos al realizar las pruebas solicitadas en las dos partes de la actividad.

2. Cómo ejecutar

Para inicializar un router en este proyecto, se debe emplear el comando descrito en el enunciado:

```
% python3 router.py router_IP router_puerto router_rutas.txt

% # Ejemplo: Router R1
% python3 router.py 127.0.0.1 8881 rutas_R1_v1.txt
```

Se puede verificar que el router se ha iniciado correctamente cuando aparece el mensaje .ºesperando mensaje...".

Para ejecutar el archivo prueba_router.py, se debe utilizar el comando descrito en el enunciado:

```
% python3 prueba_router.py headers IP_router_inicial puerto_router_inicial

% Ejemplo: Enviando desde R1 a R5
% python3 prueba_router.py 127.0.0.1,8885,10 127.0.0.1 8881
```

3. Sobre el código

Dado que la actividad se divide en dos partes, se entrega una carpeta llamada Act5-Pt1 que abarca el desarrollo hasta la conclusión de la parte 1 (sin TTL). La parte 2, que se basa en la parte 1 e incorpora las modificaciones solicitadas, se encuentra en la carpeta Act5-Pt2. Por lo tanto, para revisar el código final de esta actividad, se recomienda dirigirse directamente a la carpeta correspondiente a la parte 2.

4. Forwarding Sin TTL

4.1. Desarrollo

En esta sección se abordará el flujo de funcionamiento del código y las decisiones de diseño adoptadas para su implementación en el caso **sin** TTL.

4.1.1. Tabla de rutas

En esta actividad, la tabla de rutas utilizada difiere en el manejo de rangos de direcciones en comparación con una tabla real. Dado que trabajaremos de manera local, en lugar de emplear un rango de direcciones IP de destino, se utilizará únicamente una dirección fija correspondiente al *localhost*. Para simular lo que sería un rango de diferentes direcciones IP, trabajaremos con diversos puertos locales. Posteriormente, el redireccionamiento de los paquetes sigue la misma lógica que una tabla real. Si la dirección a la que deseamos reenviar un paquete se encuentra en la sección "Network Destination"de la tabla, podemos determinar la dirección a la que se debe reenviar el paquete o "Gateway". En el caso de esta actividad, al recibir un paquete dirigido a la dirección (*localhost*, *puerto_X*), este será redirigido a la dirección (*localhost*, *puerto_Y*), según lo indique la tabla de rutas, siempre que X se encuentre dentro del rango de puertos inicial y final descrito.

4.1.2. Archivo router.py

En el archivo `router.py` se implementa todo el código base requerido. Al inicio, el router revisa los parámetros recibidos por consola para asignar la dirección `(IP, puerto)` en la que debe escuchar (`direccion_router_actual`) y guardar el archivo de la tabla de rutas que le corresponde. Se define un tamaño de buffer de 4096 bytes. Luego, se inicializa un socket no orientado a conexión que llamamos `‘router”` y se enlaza a la dirección recibida. Se implementa un ciclo `while` infinito para que el router se mantenga en funcionamiento indefinidamente. La forma de cerrar el router es terminar el proceso desde la consola (`ctrl+C`). Dentro de este ciclo infinito, se tiene otro ciclo `while` infinito que se encarga de revisar constantemente si llegó algún mensaje al router. Cuando el resultado de la función de `socket.recvfrom` no sea vacío, significa que se recibió un mensaje, el cual se muestra, y luego se sale del ciclo para continuar con el procesamiento del mensaje.

Ya que el `router` ha recibido un mensaje, utilizamos la función `parse_packet` para transformar la cadena recibida a la estructura de paquetes IP elegida. De esta manera, podemos extraer cuál era la dirección a la que se destinaba el mensaje. Siguiendo lo propuesto en el mensaje, si el mensaje era para nosotros, lo mostramos. En caso contrario, se busca una dirección para reenviar el paquete a través de la función `check_routes`. Si se encuentra un *Gateway*, mostramos un aviso de reenvío en pantalla y hacemos *forward* del paquete a esta dirección utilizando `socket.sendto`. En caso de no haber encontrado una dirección, se muestra un mensaje indicando que no se encontraron rutas hacia el destino.

4.1.3. Archivo utils.py

En el archivo `utils.py` se implementan todas las funciones solicitadas en el enunciado y se utilizan en el programa `router.py` para organizar el código de manera más ordenada.

4.1.3.1. Funciones parse_packet y create_packet

Debido a que en esta actividad los paquetes IP tienen pocos headers, se modelaron como un diccionario, cuyas claves son los nombres de los headers. De esta forma, en `create_packet` se recibe un diccionario que representa el paquete IP y se transforma a un string con el formato `[IP],[puerto],[mensaje]`. Por otro lado, `parse_packet` recibe un paquete IP en forma de string codificado. Aquí se separan las secciones del paquete utilizando métodos de string y se traspasan al diccionario antes descrito.

IMPORTANTE: Dado que el test proporcionado en el enunciado utilizaba coma simple "," como separador de las secciones del paquete IP, este carácter se utilizó como separador en todo el código.

4.1.3.2. Función leer_archivo

Esta función recibe el nombre del archivo de rutas de un router y traspasa cada fila de la tabla a un diccionario que separa los valores en claves según la columna que corresponda, quedando así:

```
{
   "red": [Red (CIDR)],
   "puerto_inicial": [Puerto_inicial],
   "puerto_final": [Puerto_final],
   "ip_llegar": [IP_para_llegar],
   "puerto_llegar": [Puerto_para_llegar]
}
```

Luego, cada uno de estos diccionarios se agrega a una cola circular, la cual se describe más adelante. Finalmente, la función retorna la cola con todas las rutas.

4.1.3.3. Función check_routes

La idea detrás de la implementación de Round Robin para determinar la ruta a la cual reenviar un paquete IP consiste en utilizar una cola circular que almacene todas las posibles rutas descritas en el archivo. La cola circular, implementada en `cola_circular.py`, permite almacenar un número infinito de elementos, revisar si la cola está vacía y retornar el primer elemento de la cola sin eliminarlo (se mueve al final de la cola).

Al buscar una ruta, examinamos la cola circular donde almacenamos todos los diccionarios que representan las rutas del archivo de rutas del router. Iteramos sobre la cola revisando el primer valor (`get_first`). Cuando uno de los valores coincida con el destino buscado, se guarda como la ruta a usar y la misma cola lo llevará hacia el final. Si volvemos a buscar una ruta para el mismo destino, la cola ya tiene la ruta anteriormente usada al final de esta, por

lo que, si hay otra ruta que coincida, esta se encontrará antes que la anteriormente usada. Si no hay otra ruta, la cola llevará al primer lugar la ruta que usamos anteriormente. Si se revisa toda la cola y no se encuentra una ruta que sirva, podemos concluir que no existe la ruta.

Este algoritmo implementa Round Robin ya que revisamos las rutas en un orden fijo, otorgando la misma posibilidad a todas las rutas de ser seleccionadas. Cuando una ruta se selecciona, esta no se puede volver a seleccionar inmediatamente ya que está al final de la cola, a menos que sea la única válida. Debido a la naturaleza de la estructura, no es necesario llevar un contador explícito de cuántas veces se ha utilizado una ruta.

Con esto en consideración, se guarda en el archivo `utils.py` una variable global `cola_de_rutas`, que representa la cola circular donde se almacenarán las rutas de la tabla de rutas. La función `check_routes` revisa en primera instancia si la cola está vacía. Si está vacía, utiliza la función `leer_archivo` para construir la cola de rutas. Si no estaba vacía, no se realiza ninguna acción, ya que significa que `check_routes` ya se utilizó previamente y la cola ya se cargó con las rutas. Si se seleccionó previamente alguna de ellas, ya fue enviada al final de la cola (de esta manera, se le da la misma posibilidad de salir a todas las rutas para la misma dirección de destino).

En cualquiera de los dos casos, calculamos la longitud de la cola para conocer el total de rutas que contiene y así evitar iterar infinitamente sobre ella. Luego, por cada valor que se extrae de la cola, se compara la dirección que estamos buscando con los valores de "Network Destination"de la ruta. Si la red (dirección IP) coincide con la IP de la dirección buscada y el puerto buscado se encuentra dentro del rango de puertos de la ruta, se retorna una tupla que representa la dirección a la que se debe reenviar el paquete. En caso de haber iterado por toda la cola y no haber encontrado la dirección buscada, se retorna `None`.

4.1.4. Configuración de 7 routers

En el segundo test del ítem 8 se pide agregar un router R0 y R6 a la red. Se agregaron sus tablas de rutas respectivas y se modificaron las tablas de los routers 1 a 5. Los resultados se pueden revisar en la carpeta `Conf_7_routers`.

4.1.5. Envío de paquetes fuera de la red

En el ítem 9 se pide añadir un router default en la dirección (127.0.0.1, 7000) que maneje los envíos de paquetes fuera de la red. Este router se conecta a todos los routers anteriores. Los resultados de las nuevas tablas para cada router se encuentran en la carpeta `Conf_8_routers`.

La tabla de rutas del router default queda como sigue:

```
1 # rutas_RD_v4.txt
2 127.0.0.1 8880 8886 127.0.0.1 8880
3 127.0.0.1 8880 8886 127.0.0.1 8881
```

```
127.0.0.1 8880 8886 127.0.0.1 8882
127.0.0.1 8880 8886 127.0.0.1 8883
127.0.0.1 8880 8886 127.0.0.1 8884
127.0.0.1 8880 8886 127.0.0.1 8885
127.0.0.1 8880 8886 127.0.0.1 8886
```

Esto se debe a que se conecta a todos los routers de la red, por lo que cualquier paquete dirigido a cualquier puerto puede ser reenviado por cualquiera de los routers. Se agregan a la tabla en orden ascendente de puerto de llegada.

En la tabla de ruta del resto de los routers, se agregó la ruta default al final. Esto se debe a que es menos probable que se intente acceder a una ruta fuera de la red y porque es la convención que encontré al investigar sobre el tema. Dado que los puertos de red pueden ir desde el 0 al 65536, se dividió el intervalo en 2 líneas, ya que entre]8879, 8887[se encuentran los puertos de los routers en la red. Concretamente, las 2 líneas agregadas al final de cada tabla de rutas fueron:

```
# final de rutas_Rx_v4.txt, x = [0, 6]
127.0.0.1 0 8879 127.0.0.1 7000
127.0.0.1 8887 65536 127.0.0.1 7000
```

Con todo esto, cuando se desea enviar un paquete fuera de la red, se reenvía directamente al router default, el cual muestra el mensaje de que no existe ruta hacia ese paquete.

4.2. Tests de funcionalidad

1. Usando las rutas del ejemplo 2 de la sección anterior, pruebe qué ocurre si alguien configura mal una de las tablas de rutas y coméntelo brevemente en su informe. Para ello, cambie la configuración de la tabla de rutas del archivo rutas_R2_v2.txt por:

```
127.0.0.1 8881 8881 127.0.0.1 8881
127.0.0.1 8883 8883 127.0.0.1 8881
```

   - Resultado: Se copió la carpeta `Conf_3_routers` a `Conf_3_routers_mala` para modificar el archivo `rutas_R2_v2.txt` aquí y no perder el original. Este cambio provoca la desconexión de la red, por lo que no hay forma de llegar al router 3. Al intentar enviar un paquete desde R1 a R3, este queda rebotando en la red, enviándose entre R1 y R2 infinitamente.

2. Usando las rutas de la estructura del paso 6 (creo que se refiere realmente al 7), pruebe su código enviando paquetes a R1 con destino R5 y vea que su función de rutas efectivamente usa round-robin, es decir, va alternando entre rutas. Haga varias pruebas ¿cuántos saltos dan los paquetes? ¿siempre dan la misma cantidad de saltos?¿cómo se compara la cantidad de saltos que dan los paquetes versus la cantidad de saltos mínima?

Anote sus observaciones en el informe.

- Resultado: La cantidad mínima de saltos sería 3, ya sea tomando la ruta R1->R2->R3->R5 o R1->R2->R4->R5. Después de realizar el experimento 10 veces, se observó que en 4 ocasiones el paquete dio 5 saltos y en 6 ocasiones dio 3 saltos. Las rutas se alternaban, y nunca se repitió una ruta inmediatamente después de ser usada. En general, la cantidad de saltos promedio fue muy cercana a la cantidad mínima. Dado que no hay una ruta directa entre R1 y R5 o R2 y R2, la cantidad de saltos no podría ser menor a 3.

3. Repita las pruebas del punto 2 utilizando la estructura que se le pidió crear en el Test 2 del paso 7 (creo que se refiere realmente al 8) y escriba sus observaciones en su informe. Añada en su informe los contenidos de los distintos archivos de rutas que debió crear.

   - Resultado: En este caso, la cantidad mínima de saltos sigue siendo 3, ya que la configuración base incluye la del punto 3. Al agregar otros 2 routers, se incrementan las posibilidades de rutas. Las nuevas rutas creadas ya se detallaron en la sección "Configuración de 7 routers". Después de realizar el experimento 10 veces, se registró un mínimo de 4 saltos y un máximo de 25, resultando en un promedio de 14.9 saltos. En este caso, la diferencia con la cantidad mínima de saltos es hasta casi 7 veces mayor.

5. Forwarding Con TTL

5.1. Desarrollo

En esta sección, se describirá el flujo de funcionamiento del código y las decisiones de diseño tomadas para su implementación en el caso **con** TTL.

5.1.1. Modificaciones al código base de la Parte 2

En esta segunda parte de la actividad, se realizó una modificación en el código explicado en la sección anterior. En primer lugar, se consideró la necesidad de agregar un nuevo header para almacenar el Time To Live (TTL) del paquete IP. Esto condujo a realizar modificaciones en la función ‘parse_packet‘ del archivo ‘utils.py‘, donde se añadió la clave ‘TTL‘ al diccionario que representa el paquete. Además, en la función ‘create_packet‘, se incorporó el campo ‘TTL‘ al string de salida.

Posteriormente, en el programa ‘router.py”, se introdujo una condición adicional para procesar el paquete IP solo si su TTL es mayor a 0. Esta modificación implicó encapsular todo el código que previamente verificaba si el paquete estaba destinado al router actual o debía ser reenviado dentro de la nueva condición. En caso de que la condición no se cumpliera, se imprimiría un mensaje indicando que se recibió un paquete con TTL igual a 0. Por otro lado, si el TTL es mayor a 0, se ajusta el campo correspondiente en el diccionario del paquete, disminuyendo su valor en 1 unidad antes de crear el paquete que será reenviado.

5.1.2. Archivo prueba_router.py

En el ítem 4 se solicita la creación de un archivo de prueba que envíe cada línea de un archivo hacia un router inicial con destino a un router final. Para llevar a cabo esta prueba, primero se generó un archivo de texto denominado .ªrchivo.txt". Este archivo consta de 10 líneas, numeradas del 1 al 10 para facilitar los experimentos. En el script "prueba_router.py", cada línea de este archivo es leída y enviada desde un socket UDP con dirección ("localhost", 8080) hacia el puerto inicial, el cual se recibe los argumentos desde la línea de comandos (sys.argv).

En este programa se define una variable llamada tiempo_entre_paquetes que representa la cantidad de segundos que transcurrirán entre el envío de cada línea del archivo. Luego, se reciben los argumentos desde la consola. También se tiene una variable llamada archivo_a_enviar, la cual está fijada en el archivo de texto creado. Después, se inicia un socket no orientado a conexión en la dirección establecida. Este socket se utilizará exclusivamente para enviar datos. Posteriormente, se lee cada línea del archivo de texto, y con su contenido, se crea un paquete IP. Este paquete se envía a través del socket hacia el router inicial. Después de enviar una línea, se utiliza time.sleep(tiempo_entre_paquetes) para realizar una pausa entre el envío de cada línea, si así se desea. Se manejan los errores en caso de que se presenten problemas al enviar el paquete o al leer el archivo.

5.2. Tests de funcionalidad

1. Repita la primera prueba de la parte 1 (donde indujo error en la tabla de rutas) y vea qué ocurre. Use un TTL inicial TTL = 10 ¿Qué diferencias observa? Anote sus observaciones brevemente en su informe.

   - Resultado: Al igual que en la parte anterior, tener este archivo de rutas mal configurado provoca la desconexión de los routers 1 y 2 respecto a R3. La diferencia ahora, al implementar el campo TTL, es que el paquete no queda circulando infinitamente entre R1 y R2. Cuando el TTL alcanza el valor de 0, se detiene el reenvío del paquete.

2. Cree un código que lea un archivo línea por línea, encapsule cada línea en headers IP y luego envíe cada paquete a un router usando un socket UDP. Haga que este código reciba los headers y la dirección del router inicial de destino como argumentos con sys.argv de la siguiente forma:

   python3 prueba_router.py headers IP_router_inicial puerto_router_inicial

   Con esto si queremos enviar línea a línea un archivo a través de nuestro mini-Internet desde el router R1 al router R5, se debe ejecutar:

   python3 prueba_router.py 127.0.0.1;8885;10 127.0.0.1 8881

   - Resultado: Estaba esta pregunta pero no se pregunta nada realmente. Ya se explicó anteriormente como funciona el archivo prueba_router.py.

3. Considere la configuración de 5 routers vista en la parte 1. Use el código implementado en el paso 4 con un archivo grande (de varias líneas) ¿Qué ocurre con el orden de los paquetes? Anote sus observaciones en el informe.

   - Resultado: Al realizar las pruebas, los paquetes llegan en desorden. Si observamos el ejemplo del archivo de texto (txt) que utilicé para las pruebas, cada línea está numerada. Cuando se envía una línea tras otra de manera inmediata, las líneas llegan desordenadas. Por ejemplo, en un resultado obtenido se observa el siguiente orden: 2, 3, 4, 6, 7, 8, 10, 1, 5, 9, y en otro intento, el orden fue: 3, 6, 4, 10, 9, 7, 1, 5, 8, 2. Posteriormente, realicé otra prueba para determinar si al demorar más tiempo en enviar cada línea se solucionaba este problema, utilizando la función `time.sleep(n)` con $n > 0$ después de enviar cada línea. Al aumentar n, fue menos probable que las líneas se mezclaran. Con solo utilizar $n = 1$, las líneas llegaban en el orden correcto.