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

Actividad 1

Forwarding básico

Implementación de un Mecanismo de Forwarding IP con Round-Robin y TTL

Integrantes: Leonardo Rikhardsson
Tomás Ubilla

Profesores: Ivana Bachmann
Rodrigo Arenas

Auxiliares: Cristian Romero
Vicente Videla
Francisco Almeida

Ayudantes: Cristian Galleguillos
Cristóbal Suazo Ortiz
Paula Guerrero
Felipe Alfaro
Mario Benavente
Paz Catrilaf

Fecha de realización: 20 de Septiembre de 2025
Fecha de entrega: 21 de Noviembre de 2025
Santiago, Chile

Índice

1. Ejecución código

Para iniciar un router: `python router.py <IP> <puerto> <archivo_rutas>`

Para iniciar router con TTL: `python router_ttl.py <IP> <puerto> <archivo_rutas>`

Ejemplo con 3 routers:

```
python router.py 127.0.0.1 8881 rutas_R1_v2.txt
python router.py 127.0.0.1 8882 rutas_R2_v2.txt
python router.py 127.0.0.1 8883 rutas_R3_v2.txt
```

2. Tabla de rutas - Comparación

La similitud es que nuestra tabla de rutas igualmente indica el camino que debe tomar un mensaje para llegar a su destino. Esta tabla es a quien los routers consultan para forwardear mensajes que no estaban destinados para ellos.

La principal diferencia es que en nuestro mini-Internet, todos los routers tienen la misma IP y se diferencian por los puertos. En la vida real las redes se componen de varias direcciones IP, y son estas las que describen el rango que simulamos con puertos.

Formato de tabla de rutas:

```
[IP_red] [Puerto_min] [Puerto_max] [IP_nexthop] [Puerto_nexthop]
```

3. Implementación de Round-Robin

El Round-Robin se implementó usando un diccionario global que mantiene el índice de la última ruta utilizada para cada puerto destino:

```
round_robin_counter = {}  # {puerto_destino: índice_actual}
```

Funcionamiento: La primera vez que pasa por un router hacia un destino se forma la lista de caminos posibles. Si ya existe la lista, se selecciona el camino en la posición del contador, se usa, y se mueve el contador a la siguiente posición (módulo cantidad de rutas).

Ventajas:
- Cada puerto destino mantiene su propio contador independiente
- Funciona con cualquier número de rutas alternativas
- Si el router tiene acceso a diferentes áreas de la red, cada área mantiene su contador independiente

4. Router Default (Paso 9)

Al realizar los cambios correspondientes a las tablas de rutas para agregar el router default (puerto 7000), se generó mayor conectividad entre los routers. Las tablas quedaron de la siguiente forma:

```
#rutas_R0_Test2.txt
127.0.0.1 8881 8886 127.0.0.1 8881
127.0.0.1 8881 8886 127.0.0.1 8882
127.0.0.1 0 9999 127.0.0.1 7000

#rutas_R1_Test2.txt
127.0.0.1 8880 8880 127.0.0.1 8880
127.0.0.1 8882 8886 127.0.0.1 8882
127.0.0.1 0 9999 127.0.0.1 7000

#rutas_R2_Test2.txt
127.0.0.1 8880 8880 127.0.0.1 8880
127.0.0.1 8881 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 0 9999 127.0.0.1 7000

#rutas_R3_Test2.txt
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 8884 8886 127.0.0.1 8882
127.0.0.1 8884 8886 127.0.0.1 8885
127.0.0.1 0 9999 127.0.0.1 7000

#rutas_R4_Test2.txt
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 8885
127.0.0.1 0 9999 127.0.0.1 7000

#rutas_R5_Test2.txt
127.0.0.1 8880 8884 127.0.0.1 8884
127.0.0.1 8880 8884 127.0.0.1 8883
127.0.0.1 8886 8886 127.0.0.1 8883
127.0.0.1 0 9999 127.0.0.1 7000

#rutas_R6_Test2.txt
127.0.0.1 8880 8885 127.0.0.1 8882
127.0.0.1 8880 8885 127.0.0.1 8883
127.0.0.1 0 9999 127.0.0.1 7000
```

Este router default, que corresponde al router al cual se enviarán todos los paquetes que no vayan dirigidos a ningún otro router de la red, queda enlistado al final de todas las tablas de rutas. Como se revisan de forma descendente, este será el último en revisar.

5. Pruebas Mini-Internet sin TTL

5.1. Tabla de rutas mal configurada

Al introducir errores deliberados en `rutas_R2_v2.txt`, se genera un ciclo de reenvío infinito entre R1 y R2. La configuración incorrecta hace que ambos routers se referencien mutuamente como siguiente salto hacia R3, resultando en paquetes que circulan indefinidamente sin alcanzar su destino.

Comando de prueba:

```
nc -u 127.0.0.1 8881 << EOF
127.0.0.1;8883;Hola R3!
EOF
```

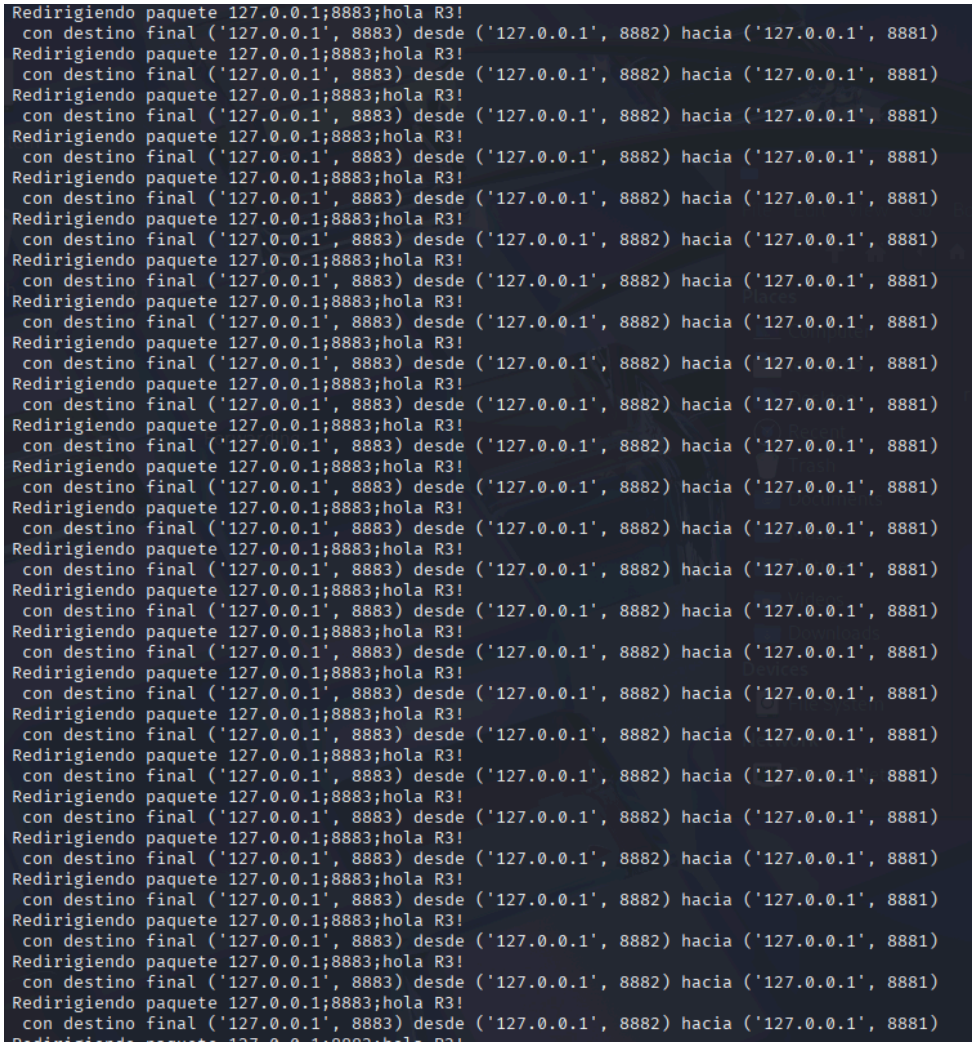


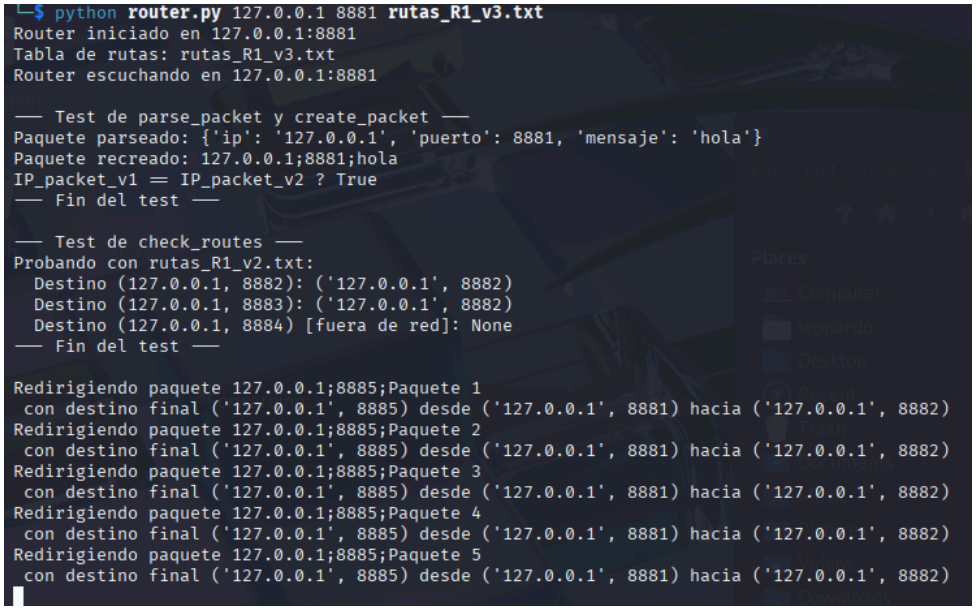
Figura 1: Loop infinito causado por tabla mal configurada

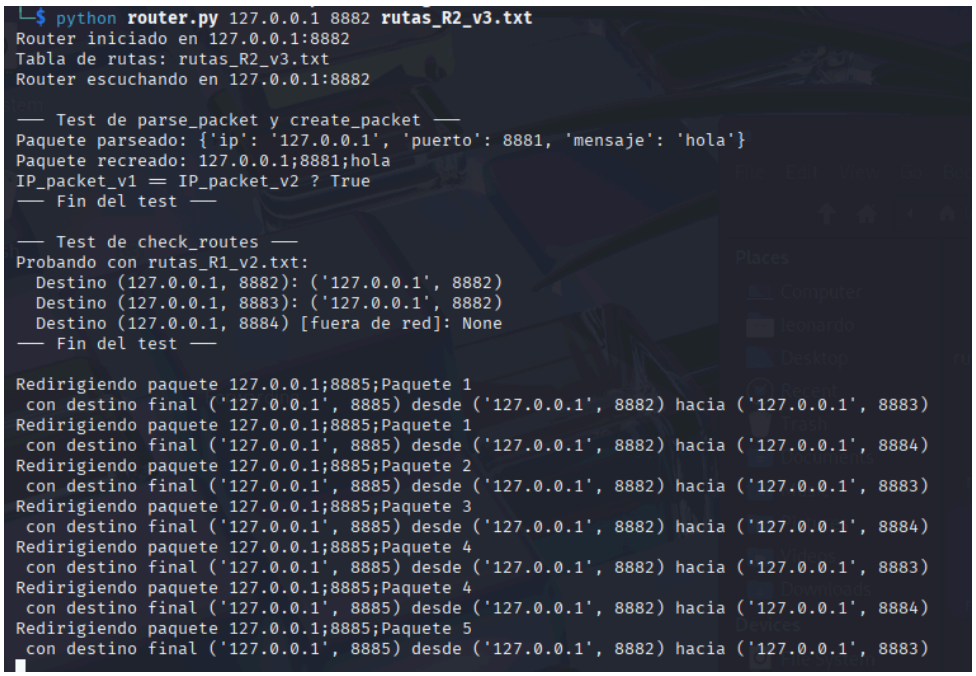
5.2. Round-robin en topología v3 (5 routers)

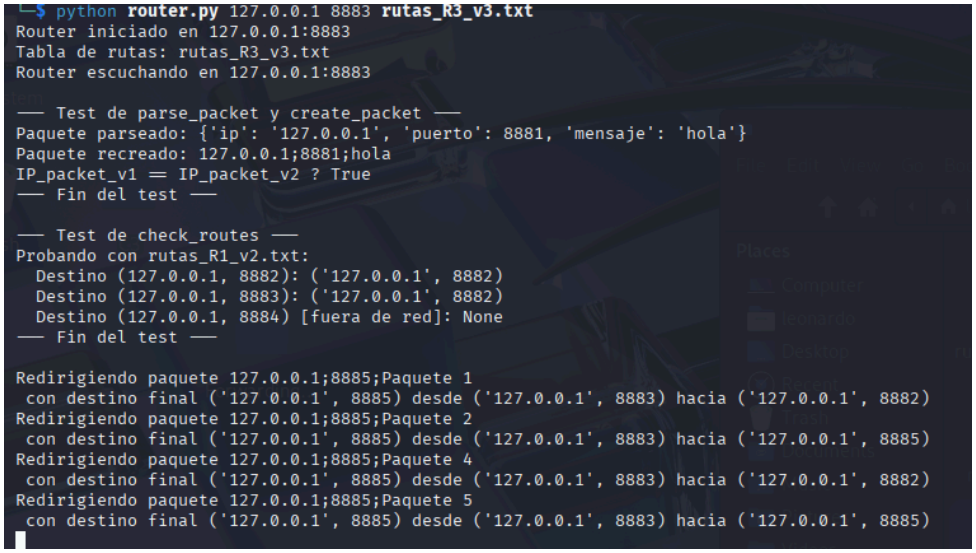
El algoritmo Round-Robin distribuye el tráfico alternando entre múltiples caminos disponibles. Al enviar paquetes consecutivos desde R1 hacia R5, se observan tres rutas diferentes que se van seleccionando cíclicamente:

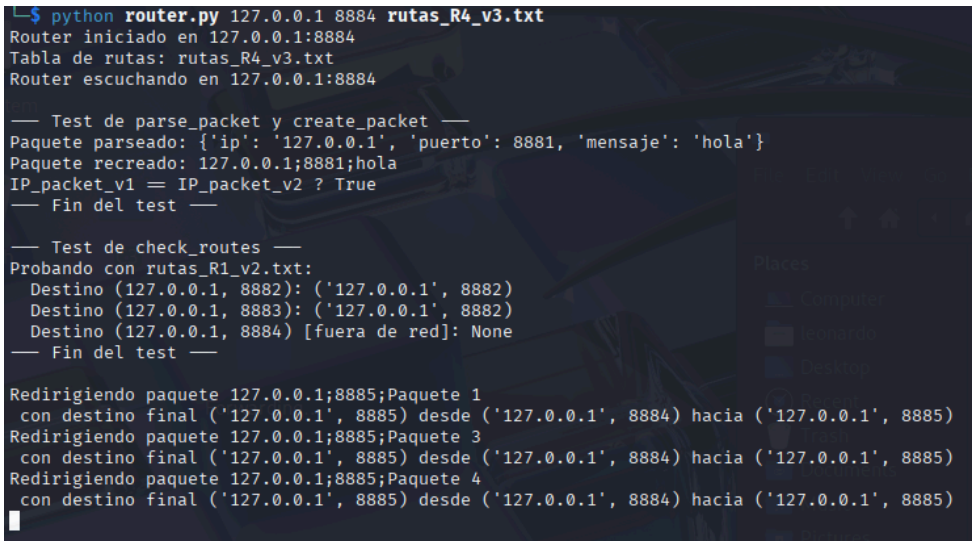
- R1 -> R2 -> R3 -> R2 -> R4 -> R5 (5 saltos)
- R1 -> R2 -> R3 -> R5 (3 saltos)
- R1 -> R2 -> R4 -> R5 (3 saltos)

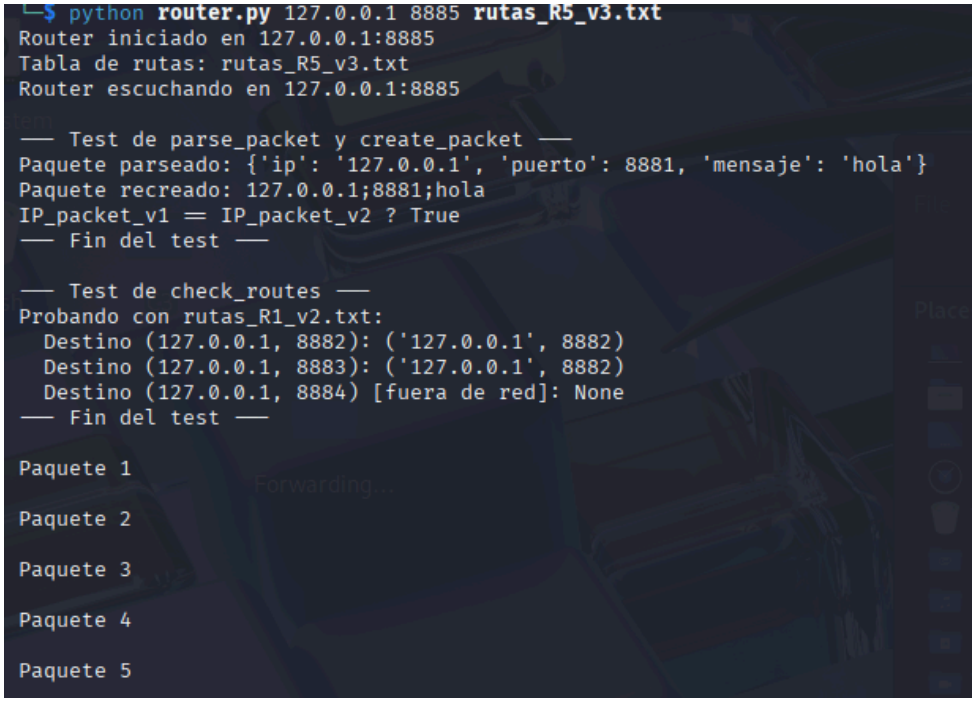
La ruta de 5 saltos aparece porque cuando R2 selecciona a R3, este último puede devolver el paquete nuevamente a R2 según su propio ciclo de round-robin, generando un camino más extenso antes de alcanzar el destino final.






└─$ python router.py 127.0.0.1 8883 rutas_R3_v3.txt
Router iniciado en 127.0.0.1:8883
Tabla de rutas: rutas_R3_v3.txt
Router escuchando en 127.0.0.1:8883
── Test de parse_packet y create_packet ──
Paquete parseado: {'ip': '127.0.0.1', 'puerto': 8881, 'mensaje': 'hola'}
Paquete recreado: 127.0.0.1;8881;hola
IP_packet_v1 == IP_packet_v2 ? True
── Fin del test ──
── Test de check_routes ──
Probando con rutas_R1_v2.txt:
Destino (127.0.0.1, 8882): ('127.0.0.1', 8882)
Destino (127.0.0.1, 8883): ('127.0.0.1', 8882)
Destino (127.0.0.1, 8884) [fuera de red]: None
── Fin del test ──
Redirigiendo paquete 127.0.0.1;8885;Paquete 1
con destino final ('127.0.0.1', 8885) desde ('127.0.0.1', 8883) hacia ('127.0.0.1', 8882)
Redirigiendo paquete 127.0.0.1;8885;Paquete 2
con destino final ('127.0.0.1', 8885) desde ('127.0.0.1', 8883) hacia ('127.0.0.1', 8885)
Redirigiendo paquete 127.0.0.1;8885;Paquete 4
con destino final ('127.0.0.1', 8885) desde ('127.0.0.1', 8883) hacia ('127.0.0.1', 8882)
Redirigiendo paquete 127.0.0.1;8885;Paquete 5
con destino final ('127.0.0.1', 8885) desde ('127.0.0.1', 8883) hacia ('127.0.0.1', 8885)


└─$ python router.py 127.0.0.1 8884 rutas_R4_v3.txt
Router iniciado en 127.0.0.1:8884
Tabla de rutas: rutas_R4_v3.txt
Router escuchando en 127.0.0.1:8884
── Test de parse_packet y create_packet ──
Paquete parseado: {'ip': '127.0.0.1', 'puerto': 8881, 'mensaje': 'hola'}
Paquete recreado: 127.0.0.1;8881;hola
IP_packet_v1 == IP_packet_v2 ? True
── Fin del test ──
── Test de check_routes ──
Probando con rutas_R1_v2.txt:
Destino (127.0.0.1, 8882): ('127.0.0.1', 8882)
Destino (127.0.0.1, 8883): ('127.0.0.1', 8882)
Destino (127.0.0.1, 8884) [fuera de red]: None
── Fin del test ──
Redirigiendo paquete 127.0.0.1;8885;Paquete 1
con destino final ('127.0.0.1', 8885) desde ('127.0.0.1', 8884) hacia ('127.0.0.1', 8885)
Redirigiendo paquete 127.0.0.1;8885;Paquete 3
con destino final ('127.0.0.1', 8885) desde ('127.0.0.1', 8884) hacia ('127.0.0.1', 8885)
Redirigiendo paquete 127.0.0.1;8885;Paquete 4
con destino final ('127.0.0.1', 8885) desde ('127.0.0.1', 8884) hacia ('127.0.0.1', 8885)



5.3. Round-robin en topología extendida (7 routers)

Utilizando los archivos de rutas Test2 (`rutas_R0_Test2.txt` hasta `rutas_R6_Test2.txt`), se realizaron 7 experimentos enviando mensajes desde R1 hacia R5. Todos los paquetes llegaron exitosamente a destino, sin embargo, la mayor cantidad de alternativas de enrutamiento incrementa significativamente el número promedio de saltos.

El comportamiento confirma la implementación correcta del algoritmo Round-Robin, aunque la eficiencia disminuye debido a trayectorias que atraviesan repetidamente los mismos routers intermedios.

6. Pruebas Mini-Internet con TTL

6.1. Loop con TTL

Repitiendo la configuración errónea anterior pero utilizando TTL=10, el comportamiento cambia drásticamente. El paquete circula entre R1 y R2 mientras el campo TTL se decrementa en cada reenvío. Al alcanzar cero, el router receptor descarta el paquete automáticamente e imprime:

```
Se recibió paquete 127.0.0.1;8883;0;mensaje con TTL 0
```

Diferencias observadas:
- Sin TTL: Ciclo indefinido, consumo ilimitado de recursos de red
- Con TTL=10: Terminación automática tras 10 reenvíos, el paquete se descarta

El mecanismo TTL funciona como salvaguarda contra loops de enrutamiento, limitando el impacto en la red aunque el paquete no alcance su destino.

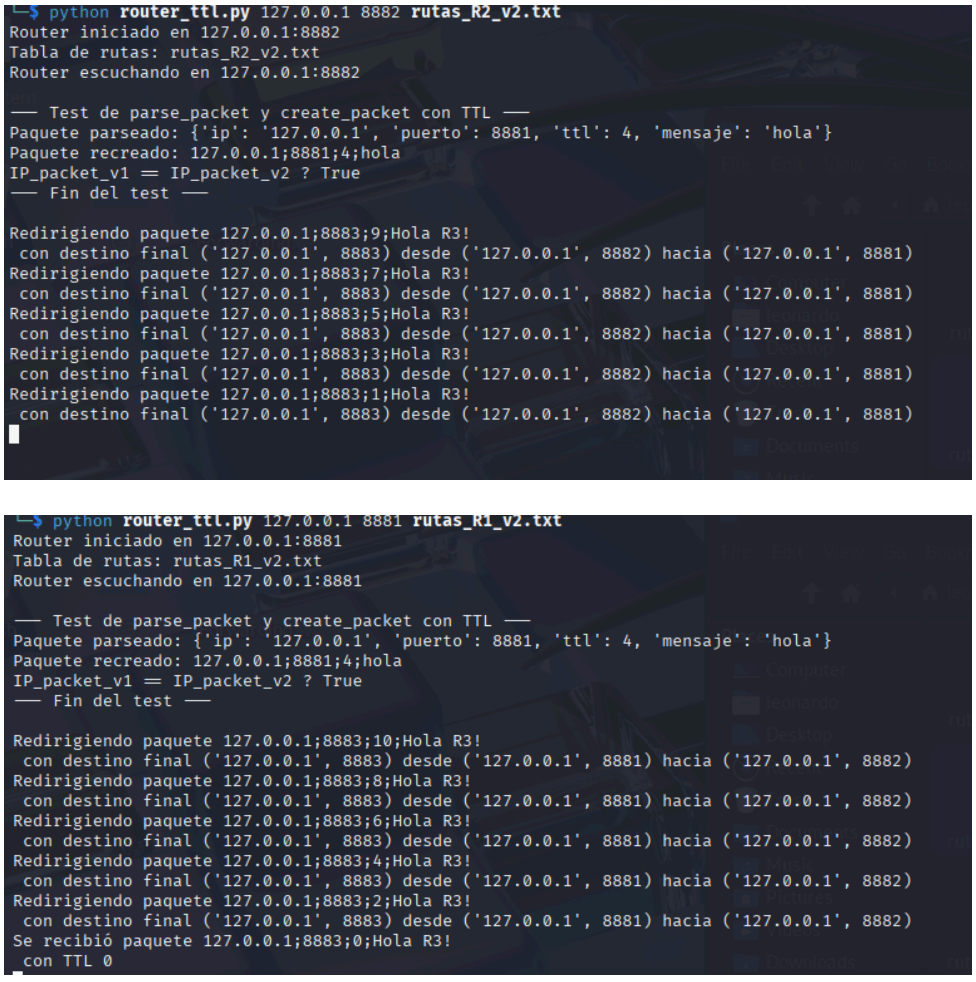


Figura 8: Loop detenido por TTL=0

6.2. Orden de paquetes al enviar archivo

El script `prueba_router.py` implementa el envío secuencial de múltiples líneas desde un archivo, encapsulando cada una con los headers IP correspondientes:

```
python3 prueba_router.py "127.0.0.1;8885;10" 127.0.0.1 8881 archivo_prueba.txt
```

Al transmitir un archivo con líneas numeradas desde R1 hacia R5 en la topología v3, se observa que **el orden de llegada no se preserva**.

Causas del desorden:
- Round-robin asigna rutas de diferentes longitudes a cada paquete
- Los paquetes por rutas cortas (3 saltos) llegan antes que los enviados previamente por rutas largas (5 saltos)
- UDP no implementa control de secuencia ni reordenamiento

Este comportamiento es consecuencia directa del balanceo de carga: paquetes enviados consecutivamente toman caminos distintos con tiempos de tránsito variables, permitiendo que paquetes posteriores adelanten a los anteriores.

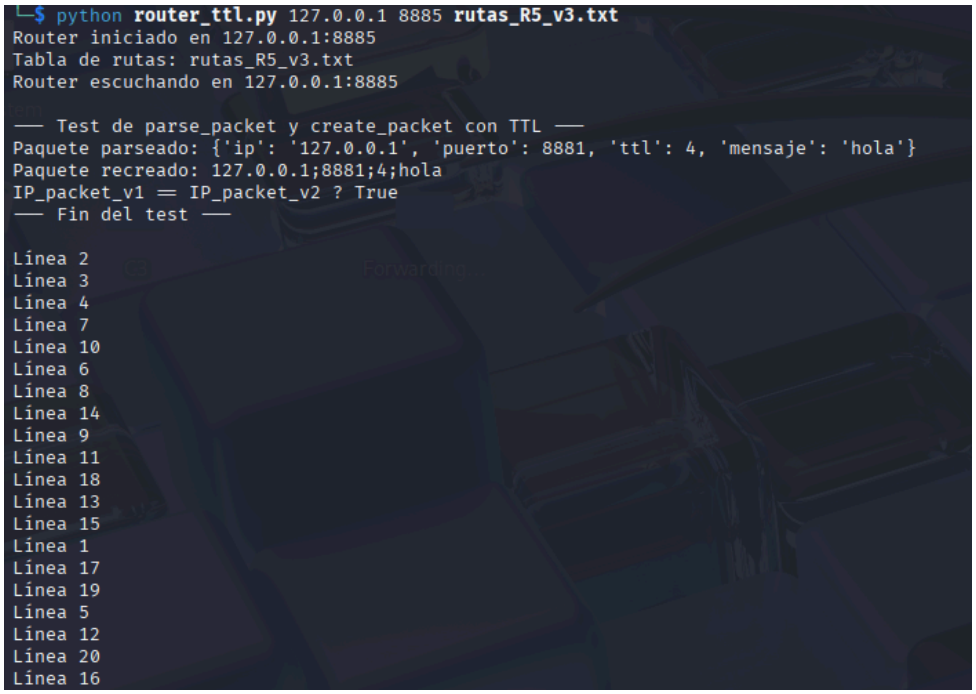


Figura 9: Paquetes que no llegan en orden