Redes de Computadoras

Curso 2024

Grupo 02

Informe - Obligatorio 02

Autores:

Mayte Carro	CI: 5.396.847-1
Alejandro Orellano	CI: 4.806.099-5
Bryan Salamone	CI: 5.608.740-6

Docente:
Jorge Visca

November 15, 2024

Contents

1 Aspectos generales

A lo largo de este laboratorio, que consistió en implementar parte del software de un router, nos enfocamos en el procesamiento, reenvío y enrutamiento de paquetes IPv4. Dada la falta de claridad en algunos requisitos y especificaciones, fue necesario tomar decisiones de diseño sobre ciertos aspectos. Además, nuestra limitada experiencia en redes y en el lenguaje C llevó ocasionalmente a "malas practicas" e introducción de errores en el proyecto. En esta sección, documentamos de manera técnica y detallada todos los aspectos relevantes, incluidas las decisiones de diseño tomadas, los problemas identificados y aquellos que no fue posible resolver.

1.1 Decisiones de diseño

La principal decisión que tomamos estuvo relacionada con el manejo de los mensajes ICMP de error. En escenarios donde es necesario responder con un ICMP de tipo 3, debemos construir manualmente el paquete, el cual está compuesto por las cabeceras Ethernet, IP e ICMP. En la cabecera IP, todos los campos relacionados con la fragmentación de paquetes fueron establecidos en cero, ya que, al estar trabajando en un entorno controlado con enlaces Ethernet, no es necesario manejar fragmentación. Sin embargo, esta configuración no sería adecuada en un entorno real, donde las MTU pueden variar entre interfaces.

Por otro lado, el campo TTL fue establecido en 64, ya que el RFC de ICMP especifica que debe utilizarse un valor lo suficientemente alto como para garantizar que el paquete llegue al destino. El valor elegido no es arbitrario, sino que corresponde al valor utilizado por el sistema operativo Linux cuando se abre un socket TCP o UDP. Esta información fue obtenida al analizar el tráfico generado por nuestra biblioteca RPC. Los demás campos fueron asignados conforme a las especificaciones del RFC de ICMP.[1]

1.2 Problemas Identificados

A lo largo del laboratorio, enfrentamos diversos errores que fueron resueltos en tiempo y forma. Sin embargo, actualmente persisten dos problemas críticos que no hemos logrado solucionar.

El primero está relacionado con nuestro protocolo de enrutamiento, basado

en OSPF, que requiere calcular las rutas de costo mínimo mediante la función run_dijkstra. Esta función presenta un bug de concurrencia, probablemente causado por una configuración incorrecta de los mutex que protegen la zona crítica. Este problema ocurre cuando hay dos instancias de la función ejecutándose en hilos separados, lo que genera conflictos entre ellas. Como resultado, en ciertos escenarios (que no hemos logrado recrear consistentemente), el programa lanza una excepción del tipo Core Dumped.

El segundo problema involucra el módulo POX, el cual ocasionalmente detiene la ejecución de un router de forma aleatoria sin generar mensajes de error en pantalla. La falta de retroalimentación dificulta la identificación de la causa raíz, y no hemos logrado reproducir este comportamiento de manera consistente para analizarlo a fondo.

1.3 Aspectos a mejorar

A pesar de los errores mencionados anteriormente, que constituyen un área de mejora, existen otros aspectos relevantes para optimizar el laboratorio. Durante el proceso de programación, observamos una cantidad significativa de código repetido, lo cual podría beneficiarse de ser encapsulado en funciones, lo que contribuiría a simplificar y hacer más legible el código. Además, somos conscientes de que el producto generado no es el más eficiente posible, ya que deben existir oportunidades de mejora que podrían resultar en tiempos de respuesta más cortos.

2 Pseudo-Código

Mnemotecnias:
rt - routing table
iface, if - interface
eth - ethernet
gw - gateway

Global :
g_topology
g_router_id

g_neighbors
OSPF DEFAULT HELLOINT
OSPF_NEIGHBOR_TIMEOUT

2.1 sr_send_icmp_error_packet

```
1: sr_send_icmp_error_packet(type, code, router, ipDst, ipPacket)
2: rt_entry = router.search_rt_entry(ipDst) {Buscar la entrada de la tabla de enrutamiento que coincida con ipDst}
3: if rt_entry exists then
4:   iface = get_interface(router, rt_entry.interface) {Determinar la interfaz de salida}
5:   arp_ip_target = (rt_entry.gw == 0) ?  ipDst :  rt_entry.gw {Definir IP de destino para ARP}
6:   cache_entry = search_arp_cache(router.cache, ipDst)
7:   if type == 3 then
8:     icmp_Packet = crear_icmp_unreachable_packet(type, code, ipPacket, ipDst, iface)
9:     if is_in_cache(cache_entry) then
10:      configure_eth_header(icmp_Packet, src_mac = iface.interface.mac, dst_mac = cache_entry.mac)
11:      enviar icmp_Packet tipo 3, code a través de la interfaz
12:    else
13:      req = queue_arp_packet(router.cache, arp_ip_target, icmp_Packet, rt_entry.interface)
14:      handle_arp_req(router, req)
15:    end if
16:    liberar icmp_Packet
17:  else if type == 11 then
18:    icmp_Packet = create_icmp_time_exceeded_packet(type, code, ipPacket, iface, ipDst)
19:    if is_in_cache(cache_entry) then
20:      configure_eth_header(icmp_Packet, src_mac = iface.interface.mac, dst_mac = cache_entry.mac)
```

```
21:       enviar icmp_Packet tipo 11, code a través de la interfaz
22:     else
23:       req = queue_arp_packet(router.cache, arp_ip_target, icmp_Packet,
          rt_entry.interface)
24:       handle_arp_req(router, req)
25:     end if
26:     liberar icmp_Packet
27:   end if
28: end if
```

2.2 sr_handle_ip_packet

```
1: sr_handle_ip_packet(router, packet, len, srcaddr, destaddr, interface_name,
   ehdr)
2: iphdr = get_ip_hdr(packet)
3: senderIp = iphdr.src
4: targetIp = iphdr.dst
5: if is_multicast_OSPF(targetIp) then
6:   iface = get_interface(router, interface_name)
7:   sr_handle_pwospf_packet(router, packet, len, iface)
8: else if not is_for_router(router, targetIp) then
9:   rt_entry = router.search_rt_entry(targetIp)
10:  if rt_entry exists then
11:    ttl = reducir_ttl(iphdr)
12:    if ttl == 0 then
13:      Enviar un icmp error 11, 0 a senderIp
14:    else
15:      iface = get_interface(router, rt_entry.interface)
16:      cache_entry = search_arp_cache(router.cache, targetIp)
17:      if is_in_cache(cache_entry) then
18:        configure_eth_header(packet, src_mac = iface.interface.mac,
          dst_mac = cache_entry.mac)
19:        Enviar packet a través de rt_entry.interface
20:      else
```

```
21:         arp_ip_target = (rt_entry.gw == 0) ?  targetIp :  rt_entry.gw

22:         req = queue_arp_packet(router.cache, arp_ip_target, packet,
            len, rt_entry.interface)
23:         handle_arp_req(router, req)
24:       end if
25:     end if
26:   else
27:     Enviar icmp error 3, 0 a senderIp
28:   end if
29: else if is_for_router(router, targetIp) then
30:   if iphdr.protocol == ICMP then
31:     icmp_hdr = get_icmp_hdr(packet)
32:     if icmp_hdr.type == 8 then
33:       rt_entry = router.search_rt_entry(senderIp)
34:       if rt_entry exists then
35:         configure_icmp_reply_packet(iphdr, icmp_hdr, targetIp, senderIp)

36:         cache_entry = search_arp_cache(router.cache, senderIp)
37:         if is_in_cache(cache_entry) then
38:           iface = get_interface(router, rt_entry.interface)
39:           configure_eth_header(packet, src_mac = iface.interface.mac,
              dst_mac = cache_entry.mac)
40:           Enviar packet a través de iface
41:         else
42:           arp_ip_target = (rt_entry.gw == 0) ?  senderIp :  rt_entry.gw

43:           req = queue_arp_packet(router.cache, arp_ip_target, packet,
              len, rt_entry.interface)
44:           handle_arp_req(router, req)
45:         end if
46:       else
47:         Enviar icmp error 3, 0 a senderIp
48:       end if
```

```
49:     end if
50:   else if iphdr.protocol == TCP OR iphdr.protocol == UDP then
51:     Enviar icmp error 3, 3 a senderIp
52:   else if iphdr.protocol == OSPF then
53:     handle_pwospf_packet(router, packet, len, get_interface(router,
        targetIp))
54:   end if
55: end if
```

2.3 check_neighbors_life

```
1: check_neighbors_life(router)
2: while true do
3:   Esperar 1 second
4:   result = check_neighbors_alive(g_neighbors)
5:   aux_iface = router.if_list
6:   while result is not null do
7:     while aux_iface is not null and aux_iface.neighbor_id ≠ result.neighbor_id
       do
8:       aux_iface = aux_iface.next
9:     end while
10:    if aux_iface is not null then
11:      aux_iface.neighbor_id = 0
12:      aux_iface.neighbor_ip = 0
13:    end if
14:    resultaux = result
15:    result = result.next
16:    free(resultaux)
17:  end while
18: end while
```

2.4 check_topology_entries_age

```
1: check_topology_entries_age(router)
2: while true do
```

```
3:   Esperar 1 segundo
4:   p = check_topology_age(g_topology)
5:   if p == 1 then
6:     dij_param = create_dij_param(router, g_router_id, g_topology)
7:     create_thread(run_dijkstra, dij_param)
8:   end if
9: end while
```

2.5 send_hellos

```
1: send_hellos(router)
2: while true do
3:   Esperar 1 segundo
4:   mutex.lock()
5:   aux_iface = router.if_list
6:   while aux_iface ≠ NULL do
7:     hello_params = create_hello_params(router, aux_iface)
8:     if aux_iface.helloint == 0 then
9:       send_hello_packet(hello_params)
10:      aux_iface.helloint = OSPF_DEFAULT_HELLOINT
11:    else
12:      aux_iface.helloint -= 1
13:    end if
14:    aux_iface = aux_iface.next
15:  end while
16:  mutex.unlock()
17: end while
18: return
```

2.6 send_hello_packet

```
1: send_hello_packet(hello_params)
2: packet_hello = reservar memoria para un paquete HELLO
3: configure_eth_hdr(packet_hello, src_mac = hello_param.interface.mac,
   dst_mac = g_ospf_multicast_mac)
```

```
configure_ip_hdr(packet_hello, src_ip = hello_param.interface.ip ,dst_ip = OSPF_AllSPFRouters)
configure_ospf_hdr(packet_hello, router_id = g_router_id)
configure_hello_hdr(packet_hello, hello_interval = OSPF_DEFAULT_HELLOINT, interface_mask = hello_param.interface.mask)
enviar packet_hello a través hello_param.interface
liberar packet_hello
```

2.7 sr_handle_pwospf_hello_packet

```
sr_handle_pwospf_hello_packet(router, packet, length, rx_if)
ip_hdr = get_ip_hdr(packet)
rx_ospfv2_hdr = get_ospfv2_hdr(packet)
rx_ospfv2_hello_hdr = get_ospfv2_hello_hdr(packet)
calculado_checksum = ospfv2_cksum(rx_ospfv2_hdr)
if rx_ospfv2_hdr.csum ≠ calculado_checksum then
    return
end if
if rx_ospfv2_hello_hdr.nmask ≠ rx_if.mask then
    return
end if
if ntohs(rx_ospfv2_hello_hdr.helloint) ≠ OSPF_DEFAULT_HELLOINT then
    return
end if
rx_if.neighbor_id = rx_ospfv2_hdr.rid
rx_if.neighbor_ip = ip_hdr.ip_src
neighbor = g_neighbors
while neighbor ≠ NULL and neighbor.neighbor_id ≠ rx_if.neighbor_id do
    neighbor = neighbor.next
end while
if neighbor == NULL then
    add_neighbor(g_neighbors, create_ospfv2_neighbor(rx_if.neighbor_id))
```

```
23:   mutex.lock()
24:   iface = router.if_list
25:   while iface ≠ NULL do
26:     lsu_param = create_lsu_param(router, iface)
27:     if iface.neighbor_id ≠ 0 then
28:       send_lsu(lsu_param)
29:     end if
30:     iface = iface.next
31:   end while
32:   mutex.unlock()
33: else
34:   neighbor.alive = OSPF_NEIGHBOR_TIMEOUT
35: end if
```

2.8 send_all_lsu

```
1: send_all_lsu(router)
2: while true do
3:   Esperar OSPF_DEFAULT_LSUINT segundos
4:   mutex.lock()
5:   aux_iface = router.if_list
6:   while aux_iface ≠ NULL do
7:     if aux_iface.neighbor_id ≠ 0 then
8:       lsu_param = create_lsu_param(router, aux_iface)
9:       send_lsu(lsu_param)
10:    end if
11:    aux_iface = aux_iface.next
12:  end while
13:  mutex.unlock()
14: end while
```

2.9 send_lsu

```
1: send_lsu(lsu_param)
2: Reservar memoria para un paquete LSU con routes_count LSA's
```

```
3: packet_lsu = Reservar memoria para un paquete lsu con routes_count lsa's
4: configure_ip_hdr(packet_lsu, ip_src = lsu_param.interface.ip,
   ip_dst = lsu_param.interface.neighbor_ip)
5: rt_entry = lsu_param.router.routing_table
6: while rt_entry ≠ NULL do
7:   if rt_entry.admin_dst == 1 or rt_entry.admin_dst == 0 then
8:     add_lsa(lsa_list, mask = rt_entry.mask,
       router_id = rt_entry.interface.neighbor_id, subnet = rt_entry.dest)
9:   end if
10:  rt_entry = rt_entry.next
11: end while
12: configure_lsu_hdr(packet_lsu, seq = g_sequence_num, num_adv = routes_count,
    lsa_list)
13: configure_ospf_hdr(packet_lsu, router_id = g_router_id)
14: cache_entry = sr_arpcache_lookup(lsu_param.router.cache, lsu_param.interface.neig

15: if cache_entry ≠ NULL then
16:   configure_eth_header(packet_lsu, lsu_param.interface.addr, cache_entry.mac)

17:   Enviar packet_lsu a través de lsu_param.interface
18:   liberar cache_entry
19: else
20:   req = sr_arpcache_queuereq(lsu_param.router.cache,
      lsu_param.interface.neighbor_ip, packet_lsu,
      lsu_length, lsu_param.interface.name)
21:   handle_arpreq(lsu_param.router, req)
22: end if
23: liberar packet_lsu
```

2.10 sr_handle_pwospf_lsu_packet

```
1: sr_handle_pwospf_lsu_packet(rx_lsu_param)
2: rx_lsu_param = obtener_rx_lsu_param()
3: next_hop_id = rx_lsu_param.rx_if.neighbor_id
```

```
4: next_hop_ip = rx_lsu_param.rx_if.neighbor_ip
5: hdr_ospf = get_ospfv2_hdr(rx_lsu_param.packet)
6: calculado_checksum = ospfv2_cksum(hdr_ospf)
7: if hdr_ospf.csum ≠ calculado_checksum then
8:   RETORNAR NULL
9: end if
10: if hdr_ospf.rid = g_router_id then
11:   RETORNAR NULL
12: end if
13: hdr_lsu = get_ospfv2_lsu_hdr(rx_lsu_param.packet)
14: source_rid = hdr_ospf.rid
15: if !check_sequence_number(g_topology, source_rid, hdr_lsu.seq) then
16:   RETORNAR
17: end if
18: number_advertisements = hdr_lsu.num_adv
19: link = get_first_link_of_lsu(rx_lsu_param.packet)
20: while number_advertisements ≠ 0 do
21:   subnet = link.subnet
22:   mask = link.mask
23:   rid = link.rid
24:   refresh_topology_entry(g_topology, hdr_ospf.rid, subnet, mask, rid,
      next_hop_ip, hdr_lsu.seq)
25:   link = get_next_link_in_lsu(link)
26:   number_advertisements = number_advertisements - 1
27: end while
28: dij_param = create_dij_param(rx_lsu_param.router, g_router_id, g_topology)

29: crear_hilo(g_dijkstra_thread, run_dijkstra, dij_param)
30: aux = rx_lsu_param.router.if_list
31: hdr_eth = get_ethernet_hdr(rx_lsu_param.packet)
32: hdr_ip = get_ip_hdr(rx_lsu_param.packet)
33: hdr_lsu.ttl = hdr_lsu.ttl - 1
34: if hdr_lsu.ttl ≤ 0 then
35:   RETORNAR NULL
```

```
36: end if
37: while aux ≠ NULL do
38:   if aux.ip ≠ rx_lsu_param.rx_if.ip and aux.neighbor_ip ≠ 0 and aux.neighbor_id
      ≠ 0 then
39:     set_eth_mac(hdr_eth, aux.addr)
40:     set_ip_packet(hdr_ip, aux.ip, aux.neighbor_ip)
41:     hdr_ip.ip_sum = ip_cksum(hdr_ip)
42:     hdr_ospf.csum = ospfv2_cksum(hdr_ospf)
43:     entry = router_arpcache_lookup(rx_lsu_param.router.cache, aux.neighbor_ip)
44:     if entry ≠ NULL then
45:       send_packet(rx_lsu_param.router, rx_lsu_param.packet, rx_lsu_param.length,
          aux.name)
46:       Liberar entry
47:     else
48:       req = router_arpcache_queuereq(rx_lsu_param.router.cache, aux.neighbor_ip,
          rx_lsu_param.packet, rx_lsu_param.length, aux.name)
49:       handle_arpreq(rx_lsu_param.router, req)
50:     end if
51:   end if
52:   aux = aux.next
53: end while
54: RETORNAR
```

3 Descripción de las Pruebas Realizadas

3.1 Análisis de enrutamiento dinámico

Una de las pruebas realizadas fue utilizar la topología de los cinco routers para ejecutar un traceroute hacia el servidor 1, cuya dirección IP es 150.150.0.2, con el objetivo de analizar el recorrido realizado para llegar al destino.

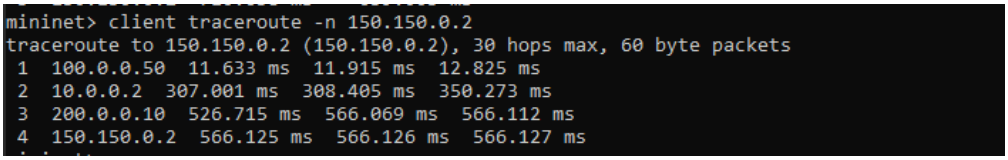

Figure 1: Traceroute al server1.

Como se observa en la figura 1, el camino tomado desde el cliente hacia el servidor 1 pasa a través de los siguientes nodos: vhost1 (100.0.0.50), vhost2 (10.0.0.2), vhost4 (200.0.0.10) y, finalmente, llega al servidor 1 (150.150.0.2). Durante esta etapa, también probamos nuestra biblioteca desarrollada en el laboratorio 1, verificando que funcionaba correctamente.

Posteriormente, decidimos simular una falla en la red apagando el nodo vhost2, para observar cómo se modifica la tabla de enrutamiento y qué nuevo camino es seleccionado para alcanzar el servidor 1. Realizamos nuevamente el traceroute y obtuvimos el siguiente resultado:

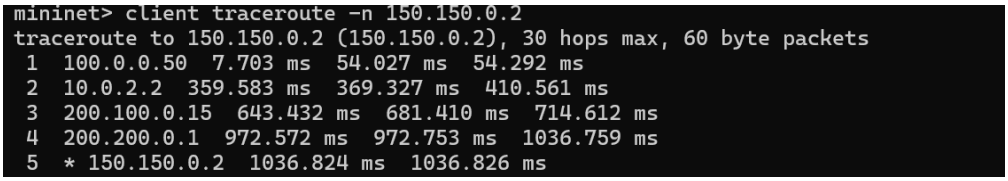

Figure 2: Traceroute al server1 sin el vhost2.

En la figura 2, se puede apreciar que la ruta cambia significativamente, pasando ahora por los nodos: vhost1 (100.0.0.50), vhost3 (10.0.2.2), vhost5 (200.100.0.12), vhost4 (200.200.0.1) y, finalmente, llegando al servidor 1 (150.150.0.2). En este caso, también probamos nuevamente la biblioteca desarrollada en el laboratorio 1, confirmando que sigue funcionando correctamente bajo esta nueva configuración. Los resultados capturados en las pruebas se encuentran incluidos en los archivos .pcap que serán entregados como parte de la documentación.

3.2 Segmentación de red y recuperación de conectividad

Otra prueba realizada consistió en dividir la red en dos. Para esto, apagamos los vhost3 y vhost4, lo que imposibilita la conectividad a los servidores. En esta

prueba, decidimos variar el objetivo y realizamos los análisis con el servidor 2, cuya dirección IP es 100.100.0.2.
En la figura 3, se observa que no es posible alcanzar el servidor 2, se confirma mediante un intento fallido de ping:

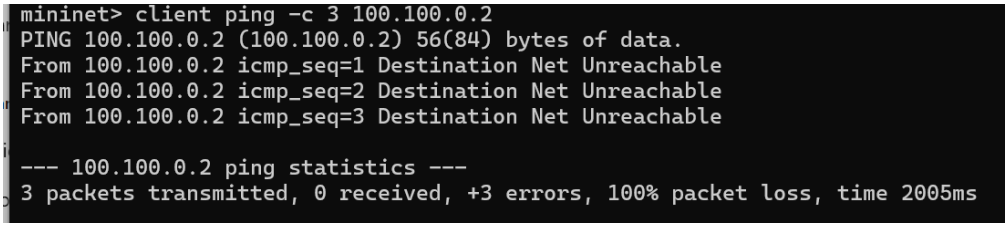

Figure 3: Ping al servidor2 sin éxito.

Posteriormente, reactivamos los nodos vhost3 y vhost4, lo que restauró la conectividad hacia el servidor 2. En la figura 4, se puede apreciar que ahora el ping al servidor 2 se realiza con éxito:

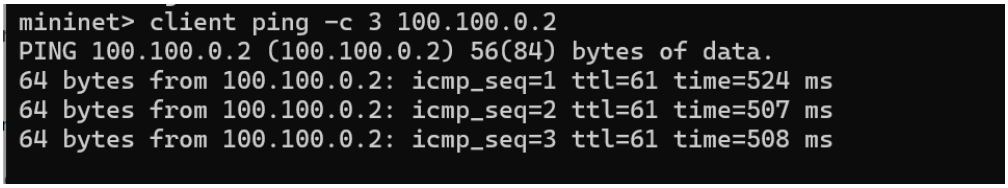

Figure 4: Ping al servidor2 con éxito.

4 Conclusiones

El laboratorio funciona como fue solicitado inicialmente, cumpliendo con todos los requisitos establecidos. Aunque existen algunos aspectos a mejorar y ciertos errores por resolver, estos no impiden el funcionamiento general del sistema. Como se muestra en la prueba 3.1, la solución es capaz de soportar todas las solicitudes de la biblioteca RPC sin generar un retardo excesivo, comportamientos inesperados o caídas del sistema. Por otro lado, la solución diseñada se comporta de forma correcta ante cambios dinámicos en la topología, tal como se demuestra en la prueba 3.1 y 3.2. Sin embargo, se reconoce que hay margen para optimizar la eficiencia, particularmente en términos de tiempo de respuesta y manejo de errores.

Se podrían implementar mejoras de código, como la eliminación de redundancias y la optimización. Estas modificaciones permitirían una mayor estabilidad y un rendimiento más ágil en entornos más complejos o de mayor carga, es decir, en entornos reales.

References

[1] Postel, J. (1981). *RFC 792: Internet Control Message Protocol*. Accesible en línea: `https://www.rfc-editor.org/info/rfc792`. *Última visita:* Noviembre 2024.