

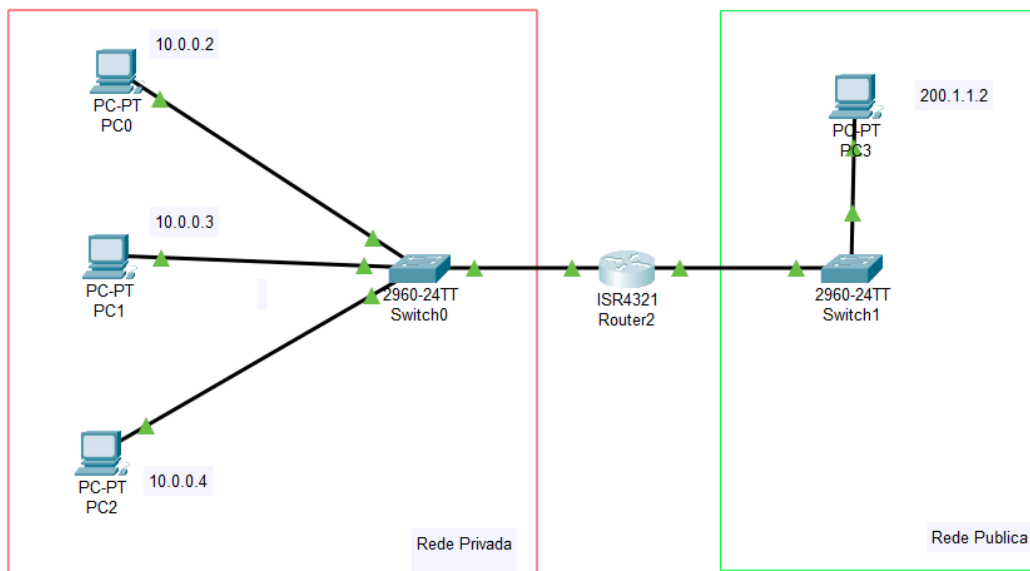


UNIVERSIDADE FEDERAL DO PIAUÍ - UFPI
CURSO: BACHARELADO EM SISTEMAS DE INFORMAÇÃO
DISCIPLINA: Redes de Computadores I
PROFESSOR: Prof. Dr. Rayner Gomes
Nome: Alisson Rodrigo Carneiro da Silva



Questão 01: Vimos que o NAT é um serviço muito importante pra isolar redes privadas da Internet e ao mesmo tempo permite que os hosts com IPs privados possam acessar a Internet. Configure o roteador para que ele realize o NAT. O roteador está ligado em duas redes, a 10.0.0.0/24 (IP Privado) e a 200.1.1.0/24 (IP Público). Sabe-se que os pacotes das redes privadas não podem ser encaminhadas para a rede pública sem o NAT, uma vez que não haverá rotas de retorno. Logo, garanta que os hosts da rede 10.0.0.0/24 consiga enviar pacotes ao Server (200.1.1.2) por meio do roteador (200.1.1.254) após o NAT realizado pelo roteador. (1 pt).

Passo 1: Topologia da rede



- **Rede Privada: 10.0.0.0/24**
 - PC0: 10.0.0.2 - Gateway padrão: 10.0.0.1
 - PC1: 10.0.0.3 - Gateway padrão: 10.0.0.1

- PC2: 10.0.0.4 - Gateway padrão: 10.0.0.1
- **Rede Pública:** 200.1.1.0/24
 - Servidor PC3: 200.1.1.2 - Gateway padrão: 200.1.1.254
 - Roteador ISR4321: 200.1.1.254
- **Equipamentos:**
 - Switch0: Interconexão da rede privada
 - Switch1: Interconexão da rede pública
 - Roteador ISR4321/K9: Responsável pelo NAT

Passo 2: Configuração das Interfaces

Interface GigabitEthernet0/0/0 (Rede Privada):

```
interface GigabitEthernet0/0/0
ip address 10.0.0.1 255.255.255.0
ip nat inside
duplex auto
speed auto
```

Interface GigabitEthernet0/0/1 (Rede Pública):

```
interface GigabitEthernet0/0/1
ip address 200.1.1.254 255.255.255.0
ip nat outside
duplex auto
speed auto
```

Passo 3: Configuração do NAT

Access List para definir tráfego a ser traduzido:

```
access-list 1 permit 10.0.0.0 0.0.0.255
```

Configuração do NAT Overload (PAT):

```
ip nat inside source list 1 interface GigabitEthernet0/0/1
overload
```

Comandos Utilizados:

```
Router> enable
Router# configure terminal
Router(config)# interface gigabitethernet0/0/0
Router(config-if)# ip address 10.0.0.1 255.255.255.0
Router(config-if)# ip nat inside
Router(config-if)# no shutdown
Router(config-if)# exit

Router(config)# interface gigabitethernet0/0/1
Router(config-if)# ip address 200.1.1.254 255.255.255.0
```

```

Router(config-if)# ip nat outside
Router(config-if)# no shutdown
Router(config-if)# exit

Router(config)# access-list 1 permit 10.0.0.0 0.0.0.255
Router(config)# ip nat inside source list 1 interface
gigabitethernet0/0/1 overload
Router(config)# exit

Router# write memory

```

Passo 4: Resultados dos Testes:

Esse teste do ping foi realizado nos 3 PCs da rede privada, abaixo está ilustrado a saída do PC1, na log da configuração NAT está exibido o ping de todos os PCs da rede privada.

Comando Utilizado: ping 200.1.1.2 para testar a comunicação.

```

C:\>ping 200.1.1.2

Pinging 200.1.1.2 with 32 bytes of data:

Reply from 200.1.1.2: bytes=32 time<1ms TTL=127
Reply from 200.1.1.2: bytes=32 time<1ms TTL=127
Reply from 200.1.1.2: bytes=32 time=1ms TTL=127
Reply from 200.1.1.2: bytes=32 time<1ms TTL=127

Ping statistics for 200.1.1.2:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 1ms, Average = 0ms

```

Comando para ver as translations realizadas no NAT: **ip nat translation**

```

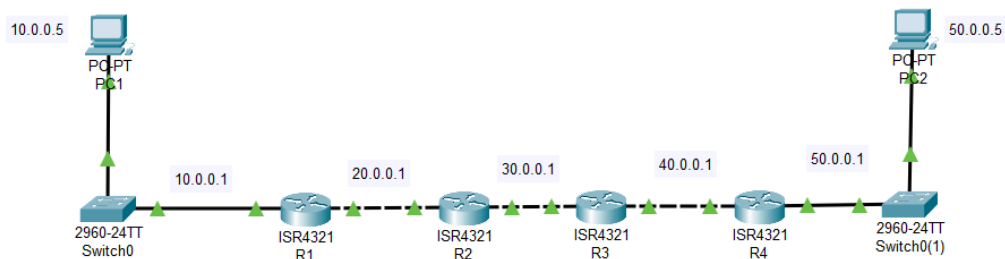
Router>show ip nat translations
Router>show ip nat translations

```

Pro	Inside global	Inside local	Outside local	Outside global
icmp	200.1.1.254:1024	10.0.0.3:10	200.1.1.2:10	200.1.1.2:1024
icmp	200.1.1.254:1025	10.0.0.4:8	200.1.1.2:8	200.1.1.2:1025
icmp	200.1.1.254:1026	10.0.0.3:11	200.1.1.2:11	200.1.1.2:1026
icmp	200.1.1.254:1027	10.0.0.4:9	200.1.1.2:9	200.1.1.2:1027
icmp	200.1.1.254:1028	10.0.0.4:10	200.1.1.2:10	200.1.1.2:1028
icmp	200.1.1.254:1029	10.0.0.4:11	200.1.1.2:11	200.1.1.2:1029
icmp	200.1.1.254:10	10.0.0.2:10	200.1.1.2:10	200.1.1.2:10
icmp	200.1.1.254:11	10.0.0.2:11	200.1.1.2:11	200.1.1.2:11
icmp	200.1.1.254:12	10.0.0.2:12	200.1.1.2:12	200.1.1.2:12
icmp	200.1.1.254:13	10.0.0.2:13	200.1.1.2:13	200.1.1.2:13
icmp	200.1.1.254:8	10.0.0.3:8	200.1.1.2:8	200.1.1.2:8
icmp	200.1.1.254:9	10.0.0.3:9	200.1.1.2:9	200.1.1.2:9

Questão 02: Seguindo as configurações das redes na figura abaixo e usando o CPT, configure adequadamente a rede e através do roteamento estático, permita que o Host 10.0.0.5 consiga “PINGAR” pacotes para o Host 50.0.0.5. Respeite os IPs e as netmasks da imagem. (1 pt).

Passo 1: Topologia da rede



PC1 (que corresponde ao Host 10.0.0.5 da questão original) conectado ao Switch0.

Switch0 conectado ao roteador R1. A interface de R1 nesta rede é 10.0.0.1.

R1 conectado a R2. A interface de R1 para R2 está na rede 20.0.0.0/x (IP 20.0.0.1 mostrado).

R2 conectado a R3. A interface de R2 para R3 está na rede 30.0.0.0/x (IP 30.0.0.1 mostrado).

R3 conectado a R4. A interface de R3 para R4 está na rede 40.0.0.0/x (IP 40.0.0.1 mostrado).

R4 conectado ao Switch0(1) (que é o Switch1 da questão original). A interface de R4 nesta rede é 50.0.0.1.

PC2 (que corresponde ao Host 50.0.0.5 da questão original) conectado ao Switch0(1).

Passo 2: Endereçamento IP Detalhado

A rede é segmentada com máscaras de sub-rede /8 (255.0.0.0):

- Rede A (PC1 e R1): 10.0.0.0/8
- Rede B (R1 e R2): 20.0.0.0/8
- Rede C (R2 e R3): 30.0.0.0/8
- Rede D (R3 e R4): 40.0.0.0/8
- Rede E (R4 e PC2): 50.0.0.0/8

- **PC1:**
 - IP: 10.0.0.5
 - Máscara: 255.0.0.0
 - Gateway: 10.0.0.1 (IP de R1 na Rede A)
- **Roteador R1:**
 - Interface para Rede A (ex: GigabitEthernet0/0/0): 10.0.0.1/8
 - Interface para Rede B (ex: GigabitEthernet0/0/1): 20.0.0.1/8
- **Roteador R2:**
 - Interface para Rede B (ex: GigabitEthernet0/0/0): 20.0.0.2/8 (Exemplo, R2 precisa de um IP nesta rede)
 - Interface para Rede C (ex: GigabitEthernet0/0/1): 30.0.0.1/8
- **Roteador R3:**
 - Interface para Rede C (ex: GigabitEthernet0/0/0): 30.0.0.2/8 (Exemplo)
 - Interface para Rede D (ex: GigabitEthernet0/0/1): 40.0.0.1/8
- **Roteador R4:**
 - Interface para Rede D (ex: GigabitEthernet0/0/0): 40.0.0.2/8 (Exemplo)
 - Interface para Rede E (ex: GigabitEthernet0/0/1): 50.0.0.1/8
- **PC2:**
 - IP: 50.0.0.5
 - Máscara: 255.0.0.0
 - Gateway: 50.0.0.1 (IP de R4 na Rede E)

Ok, aqui está o formato final para a seção "4. Configuração das Rotas Estáticas" do seu relatório da Questão 02, utilizando a abordagem mais completa (que você exemplificou) e os IPs da sua topologia.

Passo 3: Configuração Roteamento Estático

As rotas estáticas foram configuradas em cada roteador para garantir que os pacotes possam ser encaminhados entre todas as sub-redes, permitindo a comunicação entre o PC1 (10.0.0.5 na rede 10.0.0.0/8) e o PC2 (50.0.0.5 na rede 50.0.0.0/8). Isso foi realizado acessando cada roteador no Cisco Packet Tracer e adicionando as rotas necessárias. A configuração pode ser feita através da interface gráfica (aba "Config" > "Routing" > "Static").

Roteador R1:

- **Network:** 30.0.0.0 | **Mask:** 255.0.0.0 | **Next Hop:** 20.0.0.2 | Add
- **Network:** 40.0.0.0 | **Mask:** 255.0.0.0 | **Next Hop:** 20.0.0.2 | Add
- **Network:** 50.0.0.0 | **Mask:** 255.0.0.0 | **Next Hop:** 20.0.0.2 | Add

Roteador R2:

- **Network:** 10.0.0.0 | **Mask:** 255.0.0.0 | **Next Hop:** 20.0.0.1 | Add
- **Network:** 40.0.0.0 | **Mask:** 255.0.0.0 | **Next Hop:** 30.0.0.2 | Add
- **Network:** 50.0.0.0 | **Mask:** 255.0.0.0 | **Next Hop:** 30.0.0.2 | Add

Roteador R3:

- **Network:** 10.0.0.0 | **Mask:** 255.0.0.0 | **Next Hop:** 30.0.0.1 | Add
- **Network:** 20.0.0.0 | **Mask:** 255.0.0.0 | **Next Hop:** 30.0.0.1 | Add
- **Network:** 50.0.0.0 | **Mask:** 255.0.0.0 | **Next Hop:** 40.0.0.2 | Add

Roteador R4:

- **Network:** 10.0.0.0 | **Mask:** 255.0.0.0 | **Next Hop:** 40.0.0.1 | Add
- **Network:** 20.0.0.0 | **Mask:** 255.0.0.0 | **Next Hop:** 40.0.0.1 | Add
- **Network:** 30.0.0.0 | **Mask:** 255.0.0.0 | **Next Hop:** 40.0.0.1 | Add

Passo 4: Testes e Validação

A validação foi feita com um teste de ping do PC1 (10.0.0.5) para o PC2 (50.0.0.5).

Comando no PC1: C:\> ping 50.0.0.5

```
C:\>ping 50.0.0.5

Pinging 50.0.0.5 with 32 bytes of data:

Reply from 50.0.0.5: bytes=32 time<1ms TTL=124
Reply from 50.0.0.5: bytes=32 time<1ms TTL=124
Reply from 50.0.0.5: bytes=32 time<1ms TTL=124
Reply from 50.0.0.5: bytes=32 time<1ms TTL=124

Ping statistics for 50.0.0.5:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

O resultado indica que o ping foi bem-sucedido:

- Reply from 50.0.0.5: bytes=32 time<1ms TTL=124: Esta linha, repetida quatro vezes, mostra que o host de destino (50.0.0.5) respondeu a cada uma das quatro solicitações de ping enviadas.
- Packets: Sent = 4, Received = 4, Lost = 0 (0% loss): Esta estatística confirma que todos os quatro pacotes enviados foram recebidos de volta, sem nenhuma perda.

Isso significa que há comunicação bidirecional entre o host que executou o comando ping e o host com o endereço IP 50.0.0.5. Os dispositivos estão se comunicando com sucesso através da rede.

3 - Usando a infraestrutura da Questão 2 mostre o funcionamento do “Tracerouter”.

Dica: No CPT o comando tracerouter é o tracert. Mostre o funcionamento do traceroute por meio de telas salvas do CPT conjuntamente com o seu texto explicando-as. (1 pt)

Passo 1: Comando Executado e Resultado:

Para rastrear a rota entre o PC1 (10.0.0.5) e o PC2 (50.0.0.5), o seguinte comando foi executado no prompt de comando do PC1:

```
Cisco Packet Tracer PC Command Line 1.0
C:\>tracert 50.0.0.5

Tracing route to 50.0.0.5 over a maximum of 30 hops:

  1  0 ms      0 ms      0 ms      10.0.0.1
  2  *          0 ms      0 ms      20.0.0.2
  3  *          0 ms      0 ms      30.0.0.2
  4  *          0 ms      1 ms      40.0.0.2
  5  *          0 ms      0 ms      50.0.0.5

Trace complete.
```

Passo 2: Explicação;

A saída do comando tracert lista a sequência de roteadores (saltos ou "hops") que os pacotes atravessam para alcançar o destino. Cada linha numerada representa um salto na rota:

- Linha 1: 10.0.0.1
 - Este é o primeiro salto a partir do PC1 (10.0.0.5). Corresponde ao endereço IP da interface do Roteador R1 (o gateway padrão do PC1) na rede 10.0.0.0/8.
- Linha 2: 20.0.0.2
 - Este é o segundo salto. Corresponde ao endereço IP da interface do Roteador R2 que está conectada à rede 20.0.0.0/8 (a interface que recebe o pacote vindo de R1).
- Linha 3: 30.0.0.2
 - Este é o terceiro salto. Corresponde ao endereço IP da interface do Roteador R3 conectada à rede 30.0.0.0/8.
- Linha 4: 40.0.0.2
 - Este é o quarto salto. Corresponde ao endereço IP da interface do Roteador R4 conectada à rede 40.0.0.0/8.
- Linha 5: 50.0.0.5
 - Este é o destino final, o Host PC2. A resposta do próprio host de destino confirma que ele foi alcançado.

As colunas de tempo (ex: 0 ms, 2 ms) indicam o tempo de ida e volta (Round Trip Time - RTT) para cada um dos três pacotes de teste enviados para aquele salto específico. A mensagem "Trace complete." confirma que o destino foi alcançado com sucesso.

Passo 3: Conclusão

A saída do tracert demonstra que as rotas estáticas configuradas na Questão 02 direcionam os pacotes corretamente pela sequência de roteadores R1, R2, R3 e R4 até o PC2. Cada IP listado na saída é uma interface de um roteador que encaminhou o pacote, confirmando o caminho estabelecido.

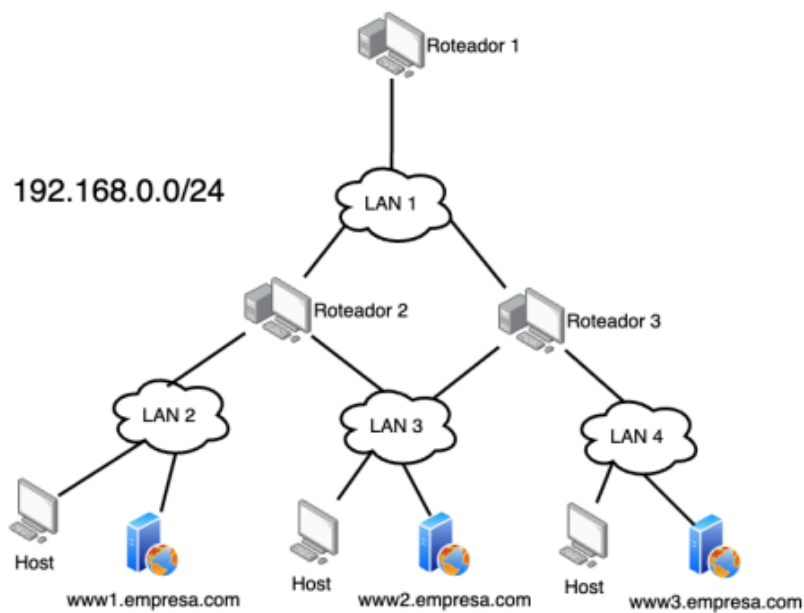
Questão 04: A Figura abaixo ilustra uma topologia com 4 subredes e 3 roteadores. As subredes LAN2, LAN3 e LAN4 possuem um host e um servidor web. Utilizando Docker implante uma rede com tais características. Garanta que todos os hosts consigam “pingar” entre si. Todas as máquinas são containers Docker, sugere-se utilizar a imagem ubuntu como SO padrão em todas. Nos servidores WEB utilize uma imagem com o Apache ou NGINX, configure uma página de boas vindas com o nome do site, SITE A, SITE B ou SITE C (Pode usar nomes mais criativos). Os sites devem estar disponíveis para os usuários de qualquer container, a checagem poderá ser feita pelo comando links (navegador no modo texto). Atribua os endereços IPs de forma a garantir a existência conceitual das subredes, veja que todas devem começar com 192.168.X.Y/24. Configure os roteadores para que os encaminhamentos dos pacotes sejam feitos (no linux isso é feito através do comando route). Nos hosts, atribua o nome dos servidores WEB manualmente através do arquivo /etc/hosts. Por fim, garanta que o usuário no host (máquina real) consiga acessar todos servidores WEB através do IP deles. (3 pts).

Objetivo:

Criar uma topologia com 4 sub-redes e 3 roteadores, onde LAN2, LAN3 e LAN4 possuem um host e um servidor web. Todos os dispositivos são containers Docker, e devem:

- se comunicar via ping;
- acessar os servidores web uns dos outros;
- exibir páginas personalizadas em servidores NGINX;
- ser acessíveis também a partir da máquina real (host).

Topologia da Rede:



- Subrede LAN1: 192.168.50.0/24
 - Router1: 192.168.50.10
 - Router2: 192.168.50.20
 - Router3: 192.168.50.30
- Subrede LAN2: 192.168.60.0/24
 - Router2: 192.168.60.10 (Default Gateway para a LAN2)
 - Host2: 192.168.60.50
 - Servidor Web SiteA: 192.168.60.100
- Subrede LAN3: 192.168.70.0/24
 - Router2: 192.168.70.10
 - Router3: 192.168.70.20
 - Host3: 192.168.70.50
 - Servidor Web SiteB: 192.168.70.100
- Subrede LAN4: 192.168.80.0/24
 - Router3: 192.168.80.10 (Default Gateway para a LAN4)

- Host4: 192.168.80.50
- Servidor Web SiteC: 192.168.80.100

Passo 1 – Criação das Redes Virtuais (Sub-redes Docker)

As redes representam as LANs, cada uma com seu próprio intervalo de IP e gateway.

```
docker network create --driver bridge --subnet=192.168.50.0/24 --gateway=192.168.50.1 lan1
```

```
docker network create --subnet=192.168.60.0/24 --gateway=192.168.60.1 lan2
```

```
docker network create --subnet=192.168.70.0/24 --gateway=192.168.70.1 lan3
```

```
docker network create --subnet=192.168.80.0/24 --gateway=192.168.80.1 lan4
```

Passo 2 – Criação dos Roteadores (Containers Ubuntu com IP Forward)

Cada roteador se conecta a duas ou mais redes. O IP forwarding é ativado para permitir roteamento.

```
docker run -dit --name router1 --privileged --net lan1 --ip 192.168.50.10 ubuntu
```

```
docker run -dit --name router2 --privileged --net lan1 --ip 192.168.50.20 ubuntu
```

```
docker network connect --ip 192.168.60.10 lan2 router2
```

```
docker network connect --ip 192.168.70.10 lan3 router2
```

```
docker run -dit --name router3 --privileged --net lan1 --ip 192.168.50.30 ubuntu
```

```
docker network connect --ip 192.168.70.20 lan3 router3
```

```
docker network connect --ip 192.168.80.10 lan4 router3
```

Passo 3 – Criação dos Hosts e Servidores Web

Cada LAN (LAN2, LAN3 e LAN4) possui:

- Um host Ubuntu (cliente)
- Um servidor NGINX com página personalizada
-

```
docker run -dit --name host2 --privileged --net lan2 --ip 192.168.60.50 ubuntu
```

```
docker run -dit --name servidor_a --net lan2 --ip 192.168.60.100 -p 8081:80 nginx
```

```
docker run -dit --name host3 --privileged --net lan3 --ip 192.168.70.50 ubuntu
```

```
docker run -dit --name servidor_b --net lan3 --ip 192.168.70.100 -p 8082:80 nginx
```

```
docker run -dit --name host4 --privileged --net lan4 --ip 192.168.80.50 ubuntu
```

```
docker run -dit --name servidor_c --net lan4 --ip 192.168.80.100 -p 8083:80 nginx
```

Passo 4 – Configuração dos Roteadores

- Instalação de ferramentas de rede
- Ativação do IP Forward
- Definição de rotas entre as LANs

```
for r in router1 router2 router3; do
```

```
    docker exec -it $r bash -c "apt update && apt install net-tools iproute2 iputils-ping -y"
```

```
    docker exec -it $r bash -c "echo 1 > /proc/sys/net/ipv4/ip_forward"
```

```
done
```

```
docker exec -it router1 bash -c "ip route add 192.168.60.0/24 via 192.168.50.20 && ip route add 192.168.70.0/24 via 192.168.50.20 && ip route add 192.168.80.0/24 via 192.168.50.30"
```

```
docker exec -it router2 bash -c "ip route add 192.168.80.0/24 via 192.168.50.30"
```

```
docker exec -it router3 bash -c "ip route add 192.168.60.0/24 via 192.168.50.20"
```

Passo 5 – Configuração dos Hosts

Atualizamos os hosts com ferramentas básicas, definimos os gateways padrão e adicionamos nomes amigáveis no arquivo /etc/hosts, permitindo acesso aos sites por nome.

Processo:

- Instalação de utilitários (ping, net-tools, navegador links)
- Configuração do gateway padrão
- Atribuição de nomes aos servidores via /etc/hosts

```
for h in host2 host3 host4; do
  docker exec -it $h bash -c "apt update && apt install net-tools iproute2 iputils-ping links -y"
done
```

```
docker exec -it host2 bash -c "ip route replace default via 192.168.60.10"
docker exec -it host3 bash -c "ip route replace default via 192.168.70.10"
docker exec -it host4 bash -c "ip route replace default via 192.168.80.10"
```

```
for h in host2 host3 host4; do
  docker exec -it $h bash -c "echo '192.168.60.100 www.sitea.com' >> /etc/hosts && echo
'192.168.70.100 www.siteb.com' >> /etc/hosts && echo '192.168.80.100 www.sitec.com' >>
/etc/hosts"
done
```

Passo 6 – Personalização das Páginas Web (NGINX)

Cada servidor recebe uma página HTML com nome e IP. Isso permite validar visualmente que cada container está servindo conteúdo próprio corretamente.

Exemplo para o servidor A:

```
docker exec -it servidor_a bash -c "apt update && apt install vim -y && cat >
/usr/share/nginx/html/index.html << 'EOF'
<!DOCTYPE html>
<html><head><title>Site A</title>
<style>body{font-family:Arial;background:linear-gradient(45deg,#667eea,#764ba2);color:whi
te;text-align:center;padding:50px;}h1{font-size:3em;}p{font-size:1.2em;}</style></head>
<body><h1>Site A</h1><p>IP: 192.168.60.100 | LAN2</p></body></html>
EOF
nginx -s reload"
```

(repetido para servidor B e C com seus IPs)

Passo 7 – Testes de Conectividade

Realizamos testes de ping entre os hosts e acessos via navegador em modo texto para validar o funcionamento dos sites.

```
docker exec -it host2 ping -c 4 192.168.70.50
```

```
$ docker exec -it host2 ping -c 4 192.168.70.50
PING 192.168.70.50 (192.168.70.50) 56(84) bytes of data.
64 bytes from 192.168.70.50: icmp_seq=1 ttl=63 time=0.238 ms
64 bytes from 192.168.70.50: icmp_seq=2 ttl=63 time=0.087 ms
64 bytes from 192.168.70.50: icmp_seq=3 ttl=63 time=0.072 ms
64 bytes from 192.168.70.50: icmp_seq=4 ttl=63 time=0.089 ms

--- 192.168.70.50 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3245ms
rtt min/avg/max/mdev = 0.072/0.121/0.238/0.067 ms
```

docker exec -it host2 ping -c 4 192.168.80.50

```
$ docker exec -it host2 ping -c 4 192.168.80.50
PING 192.168.80.50 (192.168.80.50) 56(84) bytes of data.
64 bytes from 192.168.80.50: icmp_seq=1 ttl=62 time=0.598 ms
64 bytes from 192.168.80.50: icmp_seq=2 ttl=62 time=0.088 ms
64 bytes from 192.168.80.50: icmp_seq=3 ttl=62 time=0.084 ms
64 bytes from 192.168.80.50: icmp_seq=4 ttl=62 time=0.090 ms

--- 192.168.80.50 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3264ms
rtt min/avg/max/mdev = 0.084/0.215/0.598/0.221 ms
```

docker exec -it host3 ping -c 4 192.168.80.50

```
$ docker exec -it host3 ping -c 4 192.168.80.50
PING 192.168.80.50 (192.168.80.50) 56(84) bytes of data.
64 bytes from 192.168.80.50: icmp_seq=1 ttl=63 time=0.278 ms
64 bytes from 192.168.80.50: icmp_seq=2 ttl=63 time=0.079 ms
64 bytes from 192.168.80.50: icmp_seq=3 ttl=63 time=0.090 ms
64 bytes from 192.168.80.50: icmp_seq=4 ttl=63 time=0.082 ms

--- 192.168.80.50 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3281ms
rtt min/avg/max/mdev = 0.079/0.132/0.278/0.084 ms
```

docker exec -it host2 bash

```
root@1eeb047bbab5:/# links http://www.sitea.com
root@1eeb047bbab5:/# links http://www.siteb.com
root@1eeb047bbab5:/# links http://www.sitec.com
root@1eeb047bbab5:/#
```

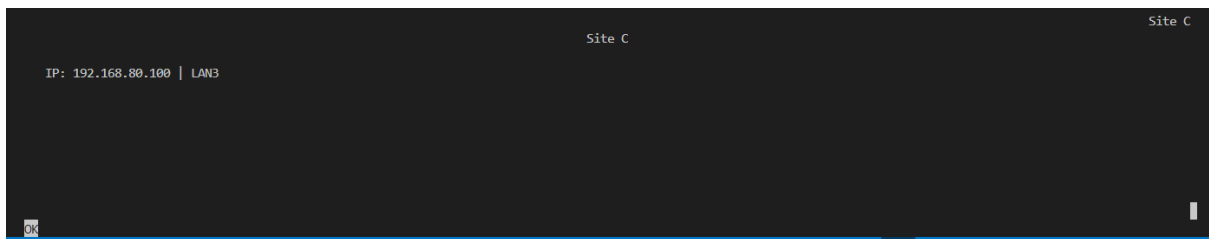
links <http://www.sitea.com>



links <http://www.siteb.com>



links <http://www.sitec.com>



(Esse passo foi e pode ser repetido nos outros HOSTs, que irá funcionar corretamente).

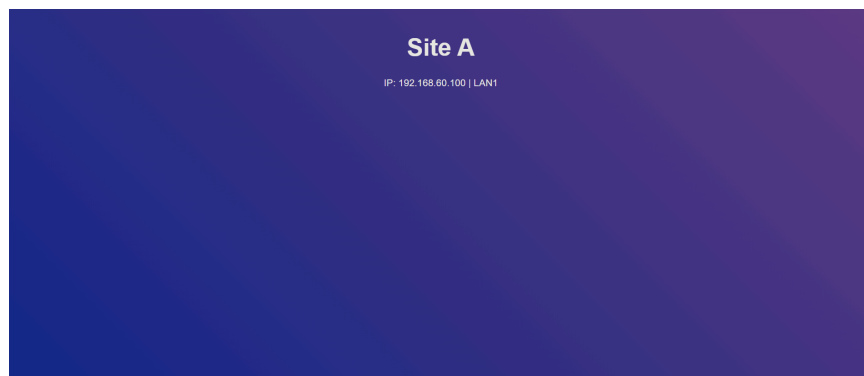
Para executar nos outros HOSTs, mude o comando:

docker exec -it host3 bash

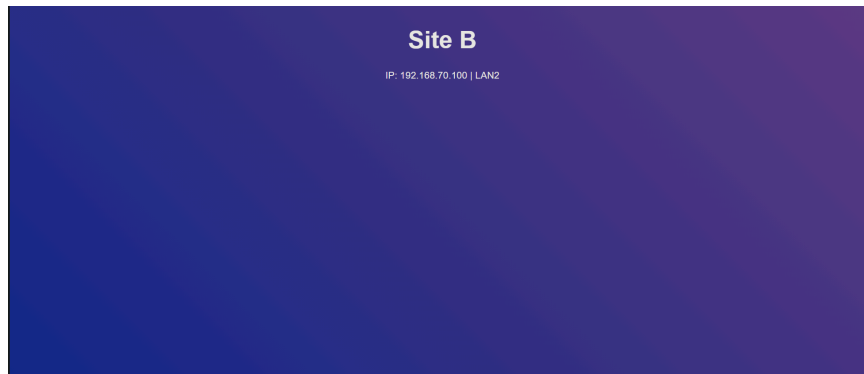
docker exec -it host4 bash

Teste no computador principal, via navegador google chrome:

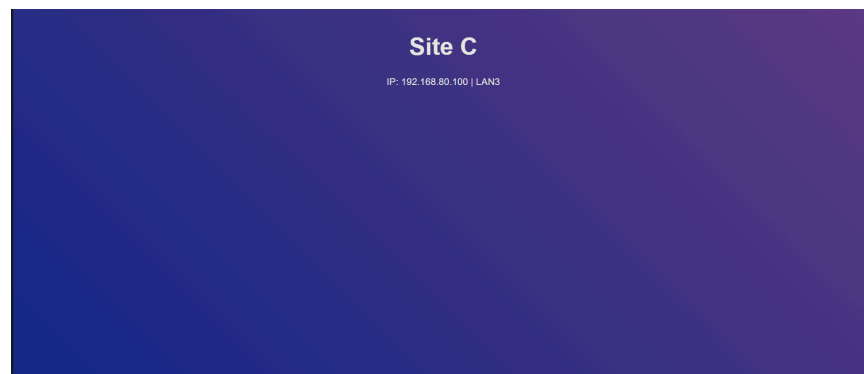
<http://localhost:8081> → Site A



<http://localhost:8082> → Site B



http://localhost:8083 → Site C



Conclusão

A simulação foi concluída com sucesso. Todos os containers conseguem se comunicar, os sites estão acessíveis via IP e nome, e o ambiente se comporta como uma rede real com sub-redes roteadas. Essa configuração pode ser expandida para testes com RIP, OSPF ou análise de pacotes, conforme solicitado nas próximas questões.

Questão 05: Refaça o roteamento estático implementando na Questão 04 pelo roteamento RIP. Demonstre através da capturação das telas a configuração e os testes. (1pt).

Objetivo: Refazer o roteamento estático da Questão 04 implementando o protocolo RIP (Routing Information Protocol) para automatizar o processo de roteamento entre as sub-redes.

1. Preparação dos Roteadores para o Roteamento Dinâmico

Após a criação dos containers dos roteadores com o Ubuntu, utilizamos o comando `apt-get` dentro de cada container para instalar o FRRouting:

`apt-get install -y frr`

1.1 Habilitação dos serviços do FRR

O FRR utiliza um arquivo chamado `/etc/frr/daemons` para habilitar os protocolos desejados. No nosso caso, habilitamos os serviços zebra (gerenciador de tabela de roteamento) e ripd (serviço que implementa o protocolo RIP):

```
sed -i 's/zebra=no/zebra=yes/' /etc/frr/daemons
```

```
sed -i 's/ripd=no/ripd=yes/' /etc/frr/daemons
```

2. Configuração do Protocolo RIP em Cada Roteador

Em cada roteador, o arquivo `/etc/frr/frr.conf` foi criado com as configurações específicas para o RIP.:

2.1 Estrutura do arquivo de configuração

O arquivo de configuração principal do FRR é o `/etc/frr/frr.conf`. Ele define o nome do roteador, as redes que participarão do protocolo e as interfaces conectadas:

Exemplo de configuração para o router2:

```
docker exec router2 bash -c "$FRR_DAEMONS_CONFIG && \
cat > /etc/frr/frr.conf << EOF
frr defaults traditional
hostname router2
log syslog informational
!
router rip
version 2
network 192.168.50.0/24
network 192.168.60.0/24
network 192.168.70.0/24
redistribute connected
!
line vty
!
EOF
service frr restart"
```

(Esse passo se repete nos outros roteadores).

- network: redes que participarão do processo RIP.
- redistribute connected: permite que o roteador compartilhe com os outros roteadores suas redes diretamente conectadas.

2.2 Arquivo `frr.conf` é criado manualmente com redirecionamento `cat` > dentro do container.

2.3 Reinicialização do serviço FRR

Após configurar os daemons e o arquivo `frr.conf`, reiniciamos o serviço para aplicar as mudanças:

```
service frr restart
```

3. Verificação do Funcionamento do RIP

3.1 Tempo de Convergência

O script aguarda **20 segundos** após reiniciar o FRR para garantir que as informações de roteamento sejam compartilhadas entre os roteadores via RIP.

3.2 Visualização das Rotas

Utiliza-se o comando `vttysh` para acessar a interface CLI do FRR e verificar o status do roteamento:

```
vttysh -c "show ip route"
```

```
vttysh -c "show ip rip status"
```

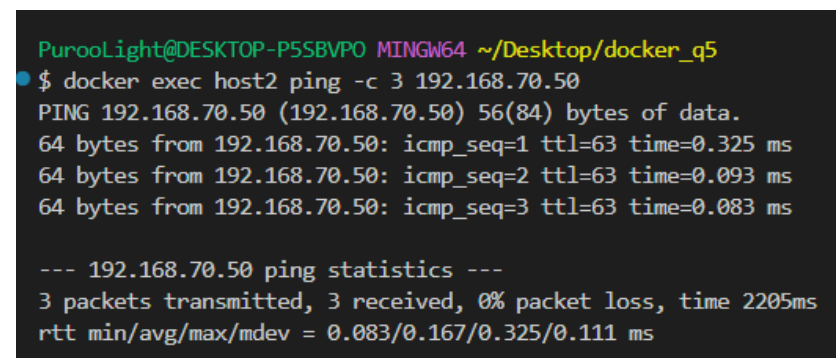
```
vttysh -c "show ip rip"
```

6. Testes e Validações

6.1 Testes de Conectividade com ping

host2 → host3:

```
docker exec host2 ping -c 3 192.168.70.50
```



```
Puroolight@DESKTOP-P5SBVPO MINGW64 ~/Desktop/docker_q5
$ docker exec host2 ping -c 3 192.168.70.50
PING 192.168.70.50 (192.168.70.50) 56(84) bytes of data.
 64 bytes from 192.168.70.50: icmp_seq=1 ttl=63 time=0.325 ms
 64 bytes from 192.168.70.50: icmp_seq=2 ttl=63 time=0.093 ms
 64 bytes from 192.168.70.50: icmp_seq=3 ttl=63 time=0.083 ms

--- 192.168.70.50 ping statistics ---
 3 packets transmitted, 3 received, 0% packet loss, time 2205ms
 rtt min/avg/max/mdev = 0.083/0.167/0.325/0.111 ms
```

host2 → host4:

```
docker exec host2 ping -c 3 192.168.80.50
```

```
PurooLight@DESKTOP-P5SBVPO MINGW64 ~/Desktop/docker_q5
• $ docker exec host2 ping -c 3 192.168.80.50
PING 192.168.80.50 (192.168.80.50) 56(84) bytes of data.
 64 bytes from 192.168.80.50: icmp_seq=1 ttl=62 time=0.250 ms
 64 bytes from 192.168.80.50: icmp_seq=2 ttl=62 time=0.106 ms
 64 bytes from 192.168.80.50: icmp_seq=3 ttl=62 time=0.096 ms

--- 192.168.80.50 ping statistics ---
 3 packets transmitted, 3 received, 0% packet loss, time 2181ms
 rtt min/avg/max/mdev = 0.096/0.150/0.250/0.070 ms
```

host3 → host4:

docker exec host3 ping -c 3 192.168.80.50

```
PurooLight@DESKTOP-P5SBVPO MINGW64 ~/Desktop/docker_q5
• $ docker exec host3 ping -c 3 192.168.80.50
PING 192.168.80.50 (192.168.80.50) 56(84) bytes of data.
 64 bytes from 192.168.80.50: icmp_seq=1 ttl=63 time=0.199 ms
 64 bytes from 192.168.80.50: icmp_seq=2 ttl=63 time=0.086 ms
 64 bytes from 192.168.80.50: icmp_seq=3 ttl=63 time=0.088 ms

--- 192.168.80.50 ping statistics ---
 3 packets transmitted, 3 received, 0% packet loss, time 2173ms
 rtt min/avg/max/mdev = 0.086/0.124/0.199/0.052 ms
```

Todos os testes de ping tiveram resposta positiva, confirmando que os roteadores trocaram rotas com sucesso via RIP.

6.2 Testes de Acesso HTTP aos Servidores Web

Os hosts utilizaram o navegador em modo texto links para acessar os servidores web:
Cada container teve o /etc/hosts configurado com os domínios:

- 192.168.60.100 www.sitea.com
- 192.168.70.100 www.siteb.com
- 192.168.80.100 www.sitec.com

docker exec host2 links -dump <http://www.sitea.com>

```
PurooLight@DESKTOP-P5SBVPO MINGW64 ~/Desktop/docker_q5
• $ docker exec host2 links -dump www.sitea.com
Site A

IP: 192.168.60.100

Rede: LAN2

Dominio: www.sitea.com

Servidor A funcionando perfeitamente!
```

`docker exec host3 links -dump http://www.siteb.com`

```
PurooLight@DESKTOP-P5SBVPO MINGW64 ~/Desktop/docker_q5
$ docker exec host3 links -dump http://www.siteb.com
Site B

IP: 192.168.70.100

Rede: LAN3

Dominio: www.siteb.com

Servidor B operacional!
```

`docker exec host4 links -dump http://www.sitec.com`

```
PurooLight@DESKTOP-P5SBVPO MINGW64 ~/Desktop/docker_q5
$ docker exec host4 links -dump http://www.sitec.com
Site C

IP: 192.168.80.100

Rede: LAN4

Dominio: www.sitec.com

Servidor C ativo e funcionando!
```

A resolução de nomes funcionou perfeitamente em todos os testes.

6.4 Análise do Protocolo RIP

A verificação do funcionamento do RIP foi feita com o comando vtysh:

Comando: `docker exec router1 vtysh -c "show ip route"`

```
PurooLight@DESKTOP-P5SBVPO MINGW64 ~/Desktop/docker_q5
$ docker exec router1 vtysh -c "show ip route"
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, E - EIGRP, N - NHRP,
       T - Table, v - VNC, V - VNC-Direct, A - Babel, F - PBR,
       f - OpenFabric,
       > - selected route, * - FIB route, q - queued, r - rejected, b - backup
       t - trapped, o - offload failure

K>* 0.0.0.0/0 [0/0] via 192.168.50.1, eth0, 00:57:30
C>* 192.168.50.0/24 is directly connected, eth0, 00:57:30
R>* 192.168.60.0/24 [120/2] via 192.168.50.20, eth0, weight 1, 00:57:28
R>* 192.168.70.0/24 [120/2] via 192.168.50.20, eth0, weight 1, 00:57:28
R>* 192.168.80.0/24 [120/2] via 192.168.50.30, eth0, weight 1, 00:57:27
```

Comando: `docker exec router2 vtysh -c "show ip rip status"`

```
PurooLight@DESKTOP-P5SBVPO MINGW64 ~/Desktop/docker_q5
$ docker exec router2 vtysh -c "show ip rip status"
Routing Protocol is "rip"
  Sending updates every 30 seconds with +/-50%, next due in 20 seconds
  Timeout after 180 seconds, garbage collect after 120 seconds
  Outgoing update filter list for all interface is not set
  Incoming update filter list for all interface is not set
  Default redistribution metric is 1
  Redistributing: connected
  Default version control: send version 2, receive version 2
    Interface          Send  Recv   Key-chain
    eth0                2     2
    eth1                2     2
    eth2                2     2
Routing for Networks:
  192.168.50.0/24
  192.168.60.0/24
  192.168.70.0/24
Routing Information Sources:
  Gateway             BadPackets  BadRoutes   Distance  Last Update
  192.168.50.30        0           0           120       00:00:14
  192.168.70.20        0           0           120       00:00:14
Distance: (default is 120)
```

Comando: `docker exec router3 vtysh -c "show ip rip"`

```
PurooLight@DESKTOP-P5SBVPO MINGW64 ~/Desktop/docker_q5
$ docker exec router3 vtysh -c "show ip rip"
Codes: R - RIP, C - connected, S - Static, O - OSPF, B - BGP
Sub-codes:
  (n) - normal, (s) - static, (d) - default, (r) - redistribute,
  (i) - interface

    Network          Next Hop          Metric From          Tag Time
C(i) 192.168.50.0/24  0.0.0.0           1 self               0
R(n) 192.168.60.0/24  192.168.50.20     2 192.168.50.20      0 02:39
C(i) 192.168.70.0/24  0.0.0.0           1 self               0
C(i) 192.168.80.0/24  0.0.0.0           1 self               0
```

As saídas confirmaram:

- Rotas aprendidas dinamicamente, marcadas com o protocolo R.
- Tabela de roteamento completa, com caminhos para todas as LANs.
- Status ativo do daemon RIP, com vizinhos identificados corretamente.

5. Conclusão

A implementação do roteamento dinâmico com o FRRouting (FRR) e o protocolo RIP foi bem-sucedida. A topologia simulada em containers Docker conseguiu:

- Estabelecer comunicação entre sub-redes automaticamente.
- Reduzir a necessidade de rotas manuais.
- Adaptar-se dinamicamente à topologia da rede.

Todos os testes de ping, HTTP e análise de rota confirmaram o sucesso da configuração.

Questão 06: Refaça o roteamento estático implementando na Questão 04 pelo roteamento OSPF. Demonstre através da capturação das telas a configuração e os testes. (1pt)

1. Objetivo

Substituir o roteamento estático (implementado na Questão 04) por um roteamento dinâmico utilizando o protocolo OSPF (Open Shortest Path First), implementado com o FRRouting (FRR), em uma topologia de múltiplas redes e roteadores simulada com containers Docker.

2. Instalação do FRR e Configuração OSPF

2.1 Instalação do FRR

Em cada roteador, foi instalado o pacote frr junto com ferramentas básicas de rede:

```
apt install -y frr iproute2 iputils-ping net-tools
```

2.2 Habilitação dos daemons zebra e ospfd

O arquivo `/etc/frr/daemons` foi editado para ativar os serviços necessários:

```
sed -i "s/zebra=no/zebra=yes/" /etc/frr/daemons
```

```
sed -i "s/ospfd=no/ospfd=yes/" /etc/frr/daemons
```

2.3 Configuração do arquivo `/etc/frr/frr.conf`

Exemplo da configuração de router2:

```
echo "Configurando OSPF no router2..."
docker exec router2 bash -c "$FRR_DAEMONS_CONFIG && \
cat > /etc/frr/frr.conf << EOF
```

```
frr defaults traditional
hostname router2
log syslog informational
!
router ospf
ospf router-id 192.168.50.20
network 192.168.50.0/24 area 0.0.0.0
network 192.168.60.0/24 area 0.0.0.0
network 192.168.70.0/24 area 0.0.0.0
!
line vty
!
EOF
service frr restart"
```

(Essa configuração deve ser realizada nos demais roteadores);

Esse bloco define:

- O Router ID (usado pelo OSPF para identificação única).
- As redes OSPF associadas às interfaces.
- A área OSPF 0.0.0.0, usada como backbone (área padrão).

3. Habilitação do IP Forwarding

Nos três roteadores:

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

5. Configuração dos Hosts e Servidores

Todos os hosts e servidores tiveram:

- Ferramentas básicas de rede instaladas
- Gateway padrão apontando para o roteador local
- Entradas no /etc/hosts para resolução dos domínios dos sites

6. Testes e Validações

6.1 Testes de Conectividade com ping

Comandos:

```
docker exec host2 ping -c 3 192.168.70.50
```

```
PurooLight@DESKTOP-P5SBVPO MINGW64 ~/Desktop/docker_q6
$ docker exec host2 ping -c 3 192.168.70.50 || echo "Falha no ping host2 -> host3"
PING 192.168.70.50 (192.168.70.50) 56(84) bytes of data.
64 bytes from 192.168.70.50: icmp_seq=1 ttl=63 time=0.141 ms
64 bytes from 192.168.70.50: icmp_seq=2 ttl=63 time=0.076 ms
64 bytes from 192.168.70.50: icmp_seq=3 ttl=63 time=0.067 ms

--- 192.168.70.50 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2208ms
rtt min/avg/max/mdev = 0.067/0.094/0.141/0.032 ms
```

docker exec host2 ping -c 3 192.168.80.50

```
PurooLight@DESKTOP-P5SBVPO MINGW64 ~/Desktop/docker_q6
$ docker exec host2 ping -c 3 192.168.80.50
PING 192.168.80.50 (192.168.80.50) 56(84) bytes of data.
64 bytes from 192.168.80.50: icmp_seq=1 ttl=62 time=0.145 ms
64 bytes from 192.168.80.50: icmp_seq=2 ttl=62 time=0.090 ms
64 bytes from 192.168.80.50: icmp_seq=3 ttl=62 time=0.071 ms

--- 192.168.80.50 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2185ms
rtt min/avg/max/mdev = 0.071/0.102/0.145/0.031 ms
```

docker exec host3 ping -c 3 192.168.80.50

```
PurooLight@DESKTOP-P5SBVPO MINGW64 ~/Desktop/docker_q6
$ docker exec host3 ping -c 3 192.168.80.50
PING 192.168.80.50 (192.168.80.50) 56(84) bytes of data.
64 bytes from 192.168.80.50: icmp_seq=1 ttl=63 time=0.119 ms
64 bytes from 192.168.80.50: icmp_seq=2 ttl=63 time=0.094 ms
64 bytes from 192.168.80.50: icmp_seq=3 ttl=63 time=0.098 ms

--- 192.168.80.50 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2191ms
rtt min/avg/max/mdev = 0.094/0.103/0.119/0.011 ms
```

6.2 Acesso HTTP aos servidores Web

Os testes foram realizados com o navegador em modo texto (links) via IP:

docker exec host2 links -dump www.sitea.com

```
PurooLight@DESKTOP-P5SBVPO MINGW64 ~/Desktop/docker_q6
$ docker exec host2 links -dump www.sitea.com
Site A

IP: 192.168.60.100 | LAN: LAN2

Roteamento: OSPF DinA-c-mico
```

`docker exec host2 links -dump www.siteb.com`

```
PurooLight@DESKTOP-P5SBVPO MINGW64 ~/Desktop/docker_q6
$ docker exec host2 links -dump www.siteb.com
Site B

IP: 192.168.70.100 | LAN: LAN3

Roteamento: OSPF DinA-c-mico
```

`docker exec host2 links -dump www.sitec.com`

```
PurooLight@DESKTOP-P5SBVPO MINGW64 ~/Desktop/docker_q6
$ docker exec host2 links -dump www.sitec.com
Site C

IP: 192.168.80.100 | LAN: LAN4

Roteamento: OSPF DinA-c-mico
```

7. Análise do Funcionamento do OSPF

7.1 Tabela de Rotas (vtysh)

Comando utilizado: `docker exec router1 vtysh -c "show ip route"`

Exemplo de saída:

```
PurooLight@DESKTOP-P5SBVPO MINGW64 ~/Desktop/docker_q6
$ docker exec router1 vtysh -c "show ip route"
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, E - EIGRP, N - NHRP,
       T - Table, v - VNC, V - VNC-Direct, A - Babel, F - PBR,
       f - OpenFabric,
       > - selected route, * - FIB route, q - queued, r - rejected, b - backup
       t - trapped, o - offload failure

K>* 0.0.0.0/0 [0/0] via 192.168.50.1, eth0, 00:20:48
O  192.168.50.0/24 [110/10] is directly connected, eth0, weight 1, 00:19:57
C>* 192.168.50.0/24 is directly connected, eth0, 00:20:48
O>* 192.168.60.0/24 [110/20] via 192.168.50.20, eth0, weight 1, 00:19:57
O>* 192.168.70.0/24 [110/20] via 192.168.50.20, eth0, weight 1, 00:19:57
   *                          via 192.168.50.30, eth0, weight 1, 00:19:57
O>* 192.168.80.0/24 [110/20] via 192.168.50.30, eth0, weight 1, 00:19:57
```

As rotas OSPF aparecem marcadas com O, indicando aprendizado dinâmico.

7.2 Vizinhos OSPF

Comando utilizado: `docker exec router2 vtysh -c "show ip ospf neighbor"`

Saída esperada:

```
PuroolLight@DESKTOP-P5SBVPO MINGW64 ~/Desktop/docker_q6
$ docker exec router2 vtysh -c "show ip ospf neighbor"
```

Neighbor ID	Pri	State	Up Time	Dead Time	Address	Interface	RXmtL	RqstL	DBsmL
192.168.50.10	1	Full/DROther	22m21s	33.016s	192.168.50.10	eth0:192.168.50.20	0	0	0
192.168.50.30	1	Full/DR	22m26s	33.614s	192.168.50.30	eth0:192.168.50.20	0	0	0
192.168.50.30	1	Full/DR	22m26s	33.615s	192.168.70.20	eth2:192.168.70.10	0	0	0

Os roteadores descobriram automaticamente seus vizinhos OSPF via rede compartilhada (LAN1), formando adjacências completas.

8. Conclusão

Com a substituição do roteamento estático pelo roteamento dinâmico via OSPF, a topologia passou a aprender rotas automaticamente, eliminando a necessidade de configurar manualmente rotas em cada roteador.

Vantagens observadas:

- Simplicidade na expansão da rede.
- Detecção automática de vizinhos.
- Aprendizado automático de rotas.
- Tolerância a falhas (em cenários com múltiplos caminhos).

Todos os testes de ping, acesso HTTP e análise do protocolo OSPF foram bem-sucedidos.

Questão 07: Monitore a rede da Questão 04 com o TCPDump ou Wireshak. Mostre e explique as requisições e respostas dos protocolos IP, ICMP e ARP submetidos nas redes: Capture apenas uma amostra de cada. Mostre a tela com a captura e depois explique os campos e seus valores dos cabeçalhos envolvidos. (2 pts)

Objetivo:

Análise de tráfego de rede utilizando a ferramenta TCPDump para capturar e examinar os protocolos IP, ICMP e ARP em uma topologia de rede simulada com Docker. O objetivo é demonstrar o funcionamento desses protocolos através da captura e análise de seus cabeçalhos.

1. Ambiente de Teste

- Ferramenta de Virtualização: Docker
- Ferramenta de Captura: TCPDump
- Topologia: 4 LANs interconectadas por roteadores
- Hosts de Teste: host2, host3, host4

2. Configuração Inicial

2.1 Instalar TCPDump nos hosts

`docker exec -it host2 bash -c "apt update && apt install tcpdump iputils-ping net-tools -y"`

`docker exec -it host3 bash -c "apt update && apt install tcpdump iputils-ping net-tools -y"`

`docker exec -it host4 bash -c "apt update && apt install tcpdump iputils-ping net-tools -y"`

```
PurooLight@DESKTOP-P5SBVPO MINGW64 ~/Desktop/docker_q4
$ docker exec -it host2 bash -c "apt update && apt install tcpdump iputils-ping net-tools -y"
docker exec -it host3 bash -c "apt update && apt install tcpdump iputils-ping net-tools -y"
docker exec -it host4 bash -c "apt update && apt install tcpdump iputils-ping net-tools -y"
```

2.2 Verificação das Interfaces

É importante realizar a verificação do endereçamento IP de cada host:

`docker exec -it host2 ip a`

```
PurooLight@DESKTOP-P5SBVPO MINGW64 ~/Desktop/docker_q4
$ docker exec -it host2 ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0@if10: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 4e:60:a7:10:25:08 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 192.168.60.50/24 brd 192.168.60.255 scope global eth0
        valid_lft forever preferred_lft forever
```

`docker exec -it host3 ip a`

```
PurooLight@DESKTOP-P5SBVPO MINGW64 ~/Desktop/docker_q4
$ docker exec -it host3 ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0@if16: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 6a:4d:17:a6:17:a7 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 192.168.70.50/24 brd 192.168.70.255 scope global eth0
        valid_lft forever preferred_lft forever
```

docker exec -it host4 ip a

```
PurooLight@DESKTOP-P5SBVPO MINGW64 ~/Desktop/docker_q4
$ docker exec -it host4 ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0@if11: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 5e:d2:d7:52:98:19 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 192.168.80.50/24 brd 192.168.80.255 scope global eth0
        valid_lft forever preferred_lft forever
```

3. Capturas e Análises

3.1 PROTOCOLO ICMP (Internet Control Message Protocol)

3.1.1 Procedimento de Captura

No **Terminal 1** (host2), foi iniciada a captura de pacotes ICMP:

docker exec -it host2 bash

tcpdump -i eth0 -v icmp

```
PurooLight@DESKTOP-P5SBVPO MINGW64 ~/Desktop/docker_q4
$ docker exec -it host2 bash
root@b18ad8a609b1:/# tcpdump -i eth0 -v icmp
tcpdump: listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
```

No **Terminal 2** (host3), foram gerados pacotes ICMP através do comando ping:

docker exec -it host3 bash

ping -c 4 192.168.60.50

```
PurooLight@DESKTOP-P5SBVPO MINGW64 ~/Desktop/docker_q4
$ docker exec -it host3 bash
root@61909d407ee8:/# ping -c 4 192.168.60.50
PING 192.168.60.50 (192.168.60.50) 56(84) bytes of data.
64 bytes from 192.168.60.50: icmp_seq=1 ttl=63 time=0.068 ms
64 bytes from 192.168.60.50: icmp_seq=2 ttl=63 time=0.067 ms
64 bytes from 192.168.60.50: icmp_seq=3 ttl=63 time=0.093 ms
64 bytes from 192.168.60.50: icmp_seq=4 ttl=63 time=0.120 ms

--- 192.168.60.50 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3365ms
rtt min/avg/max/mdev = 0.067/0.087/0.120/0.021 ms
root@61909d407ee8:/#
```

3.1.2 Resultado da Captura

```
PurooLight@DESKTOP-P5SBVPO MINGW64 ~/Desktop/docker_q4
$ docker exec -it host2 bash
root@b18ad8a609b1:/# tcpdump -i eth0 -v icmp
tcpdump: listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
15:15:34.830018 IP (tos 0x0, ttl 63, id 27083, offset 0, flags [DF], proto ICMP (1), length 84)
  192.168.70.50 > b18ad8a609b1: ICMP echo request, id 7, seq 1, length 64
15:15:34.830027 IP (tos 0x0, ttl 64, id 48497, offset 0, flags [none], proto ICMP (1), length 84)
  b18ad8a609b1 > 192.168.70.50: ICMP echo reply, id 7, seq 1, length 64
15:15:35.834143 IP (tos 0x0, ttl 63, id 27106, offset 0, flags [DF], proto ICMP (1), length 84)
  192.168.70.50 > b18ad8a609b1: ICMP echo request, id 7, seq 2, length 64
15:15:35.834155 IP (tos 0x0, ttl 64, id 48517, offset 0, flags [none], proto ICMP (1), length 84)
  b18ad8a609b1 > 192.168.70.50: ICMP echo reply, id 7, seq 2, length 64
15:15:36.874180 IP (tos 0x0, ttl 63, id 27174, offset 0, flags [DF], proto ICMP (1), length 84)
  192.168.70.50 > b18ad8a609b1: ICMP echo request, id 7, seq 3, length 64
15:15:36.874194 IP (tos 0x0, ttl 64, id 48592, offset 0, flags [none], proto ICMP (1), length 84)
  b18ad8a609b1 > 192.168.70.50: ICMP echo reply, id 7, seq 3, length 64
15:15:37.914146 IP (tos 0x0, ttl 63, id 27241, offset 0, flags [DF], proto ICMP (1), length 84)
  192.168.70.50 > b18ad8a609b1: ICMP echo request, id 7, seq 4, length 64
15:15:37.914205 IP (tos 0x0, ttl 64, id 48686, offset 0, flags [none], proto ICMP (1), length 84)
  b18ad8a609b1 > 192.168.70.50: ICMP echo reply, id 7, seq 4, length 64
█
```

A captura revelou os seguintes campos do protocolo ICMP:

3.1.3 Observações Importantes da Captura Real

A captura ICMP revelou aspectos importantes do funcionamento da rede:

- Roteamento Inter-LAN:
 - TTL=63 nos requests indica que o pacote passou por 1 roteador (64-1=63)
 - TTL=64 nos replies indica comunicação direta de volta
 - Confirma que host3 (192.168.70.50) está a 1 salto do host2
- Resolução de Nomes:
 - O TCPDump mostra "b18ad8a609b1" que é o hostname do container host2
 - Isso demonstra que o sistema está fazendo resolução reversa de DNS
- Timing de Rede:
 - Latência muito baixa (< 1ms) entre request e reply
 - Interval de ~1 segundo entre cada ping (comportamento padrão)
- Identificadores únicos:
 - Cada pacote IP tem um ID único (27083, 48497, 27106, 48517...)
 - ID ICMP permanece 7 para toda a sessão ping
 - Sequência ICMP incrementa corretamente (1, 2, 3, 4)
- Flags de Fragmentação:
 - Requests têm flag DF (Don't Fragment) - política do sistema origem
 - Replies têm flag "none" - sem restrições de fragmentação

3.2 PROTOCOLO ARP (Address Resolution Protocol)

3.2.1 Procedimento de Captura

No **Terminal 1** (host2), foi iniciada a captura de pacotes ARP:

```
docker exec -it host2 bash
```

```
tcpdump -i eth0 -v arp
```

```
Puroolight@DESKTOP-P5SBVPO MINGW64 ~/Desktop/docker_q4
$ docker exec -it host2 bash
root@b18ad8a609b1:/# ping -c 1 192.168.60.10
PING 192.168.60.10 (192.168.60.10) 56(84) bytes of data.
64 bytes from 192.168.60.10: icmp_seq=1 ttl=64 time=0.134 ms

--- 192.168.60.10 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.134/0.134/0.134/0.000 ms
root@b18ad8a609b1:/#
```

No **Terminal 2** (host2), foi forçada uma requisição ARP:

```
docker exec -it host2 bash
```

```
ip neigh flush all
```

```
ping -c 1 192.168.60.10 # Gateway da LAN2
```

3.2.2 Resultado da Captura

Campos Analisados:

```
Puroolight@DESKTOP-P5SBVPO MINGW64 ~/Desktop/docker_q4
$ docker exec -it host2 bash
root@b18ad8a609b1:/# tcpdump -i eth0 -v arp
tcpdump: listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
15:30:34.328297 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has router2.lan2 tell b18ad8a609b1, length 28
15:30:34.328311 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has b18ad8a609b1 tell router2.lan2, length 28
15:30:34.328316 ARP, Ethernet (len 6), IPv4 (len 4), Reply b18ad8a609b1 is-at 12:12:82:a3:49:04 (oui Unknown), length 28
15:30:34.328322 ARP, Ethernet (len 6), IPv4 (len 4), Reply router2.lan2 is-at 06:dd:49:09:a7:99 (oui Unknown), length 28

```

- Hardware Type: Ethernet (len 6) - Endereços MAC de 6 bytes
- Protocol Type: IPv4 (len 4) - Endereços IP de 4 bytes
- Operation: Request (who-has) e Reply (is-at)
- Sender Hardware Address:
 - host2: 12:12:82:a3:49:04
 - router2: 06:dd:49:09:a7:99

- Target Identification:
 - "router2.lan2" = gateway da LAN2 (192.168.60.10)
 - "b18ad8a609b1" = host2 (192.168.60.50)
- Length: 28 bytes (tamanho padrão do pacote ARP)

3.2.4 Observações Importantes da Captura ARP Real

A captura ARP revelou aspectos interessantes:

1. Troca Bidirecional:
 - Primeiro: host2 pergunta "quem tem router2.lan2?"
 - Simultâneo: router2 pergunta "quem tem b18ad8a609b1?"
 - Ambos respondem com seus respectivos MACs
2. Resolução de Nomes:
 - Sistema resolve "router2.lan2" para o gateway
 - "b18ad8a609b1" é o hostname do container host2
3. Timing Preciso:
 - Todas as 4 mensagens ARP ocorrem em ~0.025ms
 - Demonstra a velocidade da comunicação local
4. Endereços MAC:
 - MACs únicos para cada interface de rede
 - OUI (Organizationally Unique Identifier) aparece como "Unknown" (típico em ambientes virtualizados)

3.3 PROTOCOLO IP (Internet Protocol)

3.3.1 Procedimento de Captura

Para analisar especificamente o cabeçalho IP, foi utilizado o filtro para capturar todo o tráfego:

```
docker exec -it host2 bash
```

```
tcpdump -i eth0 -v -n
```

```
root@b18ad8a609b1:/# tcpdump -i eth0 -v -n
tcpdump: listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
^
```

Em seguida, foi gerado tráfego entre diferentes redes:

```
docker exec -it host4 bash
```

```
ping -c 2 192.168.60.50 # host4 -> host2
```

```
Puroolight@DESKTOP-P55BVPO MINGW64 ~/Desktop/docker_q4
$ docker exec -it host4 bash
root@b858358178ef:/# ping -c 2 192.168.60.50 # host4 -> host2
PING 192.168.60.50 (192.168.60.50) 56(84) bytes of data.
64 bytes from 192.168.60.50: icmp_seq=1 ttl=62 time=0.140 ms
64 bytes from 192.168.60.50: icmp_seq=2 ttl=62 time=0.081 ms

--- 192.168.60.50 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1143ms
rtt min/avg/max/mdev = 0.081/0.110/0.140/0.029 ms
root@b858358178ef:/#
```

3.3.2 Resultado da Captura

```
root@b18ad8a609b1:/# tcpdump -i eth0 -v -n
tcpdump: listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
15:36:50.878480 IP (tos 0x0, ttl 62, id 31373, offset 0, flags [DF], proto ICMP (1), length 84)
  192.168.80.50 > 192.168.60.50: ICMP echo request, id 11, seq 1, length 64
15:36:50.878493 IP (tos 0x0, ttl 64, id 55701, offset 0, flags [none], proto ICMP (1), length 84)
  192.168.60.50 > 192.168.80.50: ICMP echo reply, id 11, seq 1, length 64
15:36:51.926501 IP (tos 0x0, ttl 62, id 31377, offset 0, flags [DF], proto ICMP (1), length 84)
  192.168.80.50 > 192.168.60.50: ICMP echo request, id 11, seq 2, length 64
15:36:51.926512 IP (tos 0x0, ttl 64, id 55784, offset 0, flags [none], proto ICMP (1), length 84)
  192.168.60.50 > 192.168.80.50: ICMP echo reply, id 11, seq 2, length 64
15:36:55.926455 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 192.168.60.10 tell 192.168.60.50, length 28
15:36:55.926507 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 192.168.60.50 tell 192.168.60.10, length 28
15:36:55.926510 ARP, Ethernet (len 6), IPv4 (len 4), Reply 192.168.60.50 is-at 12:12:82:a3:49:04, length 28
15:36:55.926519 ARP, Ethernet (len 6), IPv4 (len 4), Reply 192.168.60.10 is-at 06:dd:49:09:a7:99, length 28
```

Campos Analisados:

- **Versão:** 4 (IPv4)
- **IHL:** 5 (20 bytes de cabeçalho)
- **ToS (Type of Service):** 0x0 (prioridade normal)
- **Comprimento Total:** 84 bytes
- **Identificação:** 31373, 55701, 31377, 55784 (identificadores únicos dos pacotes)
- **Flags:** DF (Don't Fragment) nos requests, none nos replies
- **TTL:** 62 nos requests (decrementado de 64 após 2 roteadores), 64 nos replies
- **Protocolo:** 1 (ICMP)
- **Endereço Origem:** 192.168.80.50 (host4)
- **Endereço Destino:** 192.168.60.50 (host2)

3.3.4 Observações da Captura IP Real

A captura demonstrou perfeitamente o roteamento inter-LAN:

1. Roteamento Multi-Hop:
 - TTL=62 confirma 2 saltos: host4 → router3 → router2 → host2
 - Caminho: LAN4 → LAN1 → LAN2
2. Tráfego Adicional:
 - Captura também mostra tráfego ARP subsequente
 - Renovação de cache ARP entre host2 e seu gateway
3. Identificadores Únicos:
 - Cada pacote tem ID único (31373, 55701, 31377, 55784)
 - ICMP ID permanece 11 para a sessão completa

6. CONCLUSÕES

6.1 Protocolo ICMP

O protocolo ICMP funcionou corretamente para diagnóstico de conectividade, demonstrando:

- Correlação Request/Reply: ID ICMP consistente (7 e 11) para cada sessão
- Sequenciamento: Incremento correto da sequência (1, 2, 3, 4)
- Impacto do Roteamento: TTL decrementando conforme o número de saltos
- Latência: Comunicação em microssegundos entre hosts em LANs diferentes

6.2 Protocolo ARP

O protocolo ARP operou adequadamente para resolução de endereços:

- Descoberta Bidirecional: Troca simultânea de informações MAC entre host e gateway
- Cache ARP: Funcionamento transparente com renovação automática
- Resolução de Nomes: Integração com sistema de nomes (router2.lan2)
- Escopo Local: Operação restrita à LAN local (não atravessa roteadores)

6.3 Protocolo IP

O protocolo IPv4 demonstrou funcionamento robusto:

- Roteamento Multi-Hop: Caminho correto através de múltiplos roteadores
- Decrementação TTL: Controle adequado de loops (63 para 1 salto, 62 para 2 saltos)
- Fragmentação: Controle via flags DF nos requests
- Identificação Única: IDs únicos para cada pacote garantindo integridade

Os resultados obtidos validam completamente o funcionamento da topologia de rede implementada e demonstram a importância de cada protocolo na comunicação inter-redes. A ferramenta TCPDump provou-se essencial para análise de tráfego e diagnóstico de conectividade em ambientes complexos de rede.

Link do repositório com arquivos fontes:

<https://github.com/Alisson-Rodrigo/Avaliacao2-Redes1.git>

Link Video do Youtube: <https://www.youtube.com/watch?v=RVm00ANhsaw>

