

Protocolo IPv4

Hugo Manuel Cunha, Marcos Daniel Teixeira da Silva, Susana Vitória Sá Silva
Marques

Universidade do Minho, Departamento de Informática, 4710-057 Braga, Portugal
Email:{a84656,a78566,a84167} @alunos.uminho.pt

1 Parte 1

1.1 Questões

Pergunta 1

1.a)

Active o wireshark ou o tcpdump no pc s1. Numa shell de s1, execute o comando traceroute -I para o endereço IP do host h5.

Resposta

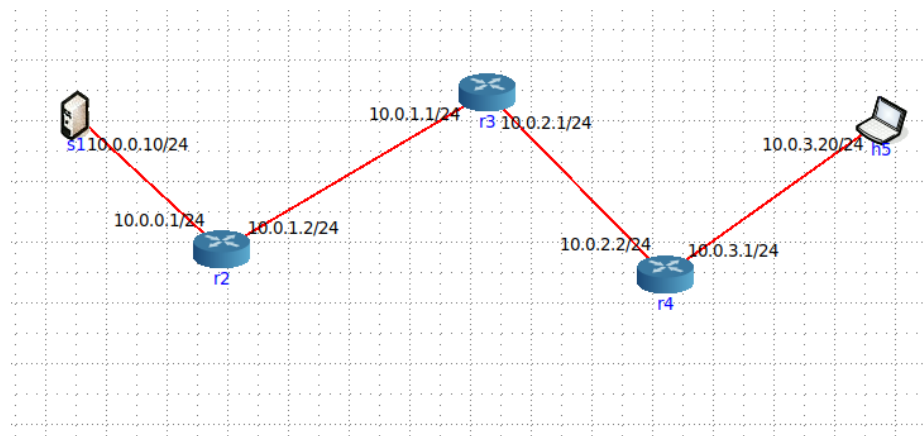


Fig. 1. Topologia Core

```

root@s1:/tmp/pycore.43631/s1.conf# traceroute -I 10.0.3.20
traceroute to 10.0.3.20 (10.0.3.20), 30 hops max, 60 byte packets
 1 10.0.0.1 (10.0.0.1) 0.068 ms 0.147 ms 0.013 ms
 2 10.0.1.1 (10.0.1.1) 0.017 ms 0.008 ms 0.007 ms
 3 10.0.2.2 (10.0.2.2) 0.017 ms 0.011 ms 0.009 ms
 4 10.0.3.20 (10.0.3.20) 0.034 ms 0.012 ms 0.011 ms
root@s1:/tmp/pycore.43631/s1.conf#

```

Fig. 2. Resultados de execução do traceroute

1.b)

Registre e analise o tráfego ICMP enviado por s1 e o tráfego ICMP recebido como resposta. Comente os resultados face ao comportamento esperado.

Resposta

| | | | | | |
|----|-------------|-----------|-----------|------|--|
| 3 | 4.630366433 | 10.0.0.10 | 10.0.3.20 | ICMP | 74 Echo (ping) request id=0x0027, seq=1/256, ttl=1 (no response found!) |
| 4 | 4.630417793 | 10.0.0.10 | 10.0.3.20 | ICMP | 102 Time-to-live exceeded (time to live exceeded in transit) |
| 5 | 4.630434216 | 10.0.0.10 | 10.0.3.20 | ICMP | 74 Echo (ping) request id=0x0027, seq=2/512, ttl=1 (no response found!) |
| 6 | 4.630448991 | 10.0.0.10 | 10.0.3.20 | ICMP | 102 Time-to-live exceeded (time to live exceeded in transit) |
| 7 | 4.630453545 | 10.0.0.10 | 10.0.3.20 | ICMP | 74 Echo (ping) request id=0x0027, seq=3/768, ttl=1 (no response found!) |
| 8 | 4.630469892 | 10.0.0.10 | 10.0.3.20 | ICMP | 102 Time-to-live exceeded (time to live exceeded in transit) |
| 9 | 4.630479432 | 10.0.0.10 | 10.0.3.20 | ICMP | 74 Echo (ping) request id=0x0027, seq=4/1024, ttl=2 (no response found!) |
| 10 | 4.630515790 | 10.0.0.10 | 10.0.3.20 | ICMP | 102 Time-to-live exceeded (time to live exceeded in transit) |
| 11 | 4.630529350 | 10.0.0.10 | 10.0.3.20 | ICMP | 74 Echo (ping) request id=0x0027, seq=5/1280, ttl=2 (no response found!) |
| 12 | 4.630540885 | 10.0.0.10 | 10.0.3.20 | ICMP | 102 Time-to-live exceeded (time to live exceeded in transit) |
| 13 | 4.630554764 | 10.0.0.10 | 10.0.3.20 | ICMP | 74 Echo (ping) request id=0x0027, seq=6/1536, ttl=2 (no response found!) |
| 14 | 4.630574223 | 10.0.0.10 | 10.0.3.20 | ICMP | 102 Time-to-live exceeded (time to live exceeded in transit) |
| 15 | 4.630590638 | 10.0.0.10 | 10.0.3.20 | ICMP | 74 Echo (ping) request id=0x0027, seq=7/1792, ttl=3 (no response found!) |
| 16 | 4.630621697 | 10.0.0.10 | 10.0.3.20 | ICMP | 102 Time-to-live exceeded (time to live exceeded in transit) |
| 17 | 4.630638908 | 10.0.0.10 | 10.0.3.20 | ICMP | 74 Echo (ping) request id=0x0027, seq=8/2048, ttl=3 (no response found!) |
| 18 | 4.630659222 | 10.0.0.10 | 10.0.3.20 | ICMP | 102 Time-to-live exceeded (time to live exceeded in transit) |
| 19 | 4.630697771 | 10.0.0.10 | 10.0.3.20 | ICMP | 74 Echo (ping) request id=0x0027, seq=9/2304, ttl=3 (no response found!) |
| 20 | 4.630702830 | 10.0.0.10 | 10.0.3.20 | ICMP | 102 Time-to-live exceeded (time to live exceeded in transit) |
| 21 | 4.630713123 | 10.0.0.10 | 10.0.3.20 | ICMP | 74 Echo (ping) request id=0x0027, seq=10/2560, ttl=4 (reply in 22) |
| 22 | 4.630779316 | 10.0.0.10 | 10.0.3.20 | ICMP | 74 Echo (ping) reply id=0x0027, seq=10/2560, ttl=61 (request in 21) |
| 23 | 4.630777286 | 10.0.0.10 | 10.0.3.20 | ICMP | 74 Echo (ping) request id=0x0027, seq=11/2816, ttl=4 (reply in 24) |
| 24 | 4.630792698 | 10.0.0.10 | 10.0.3.20 | ICMP | 74 Echo (ping) reply id=0x0027, seq=11/2816, ttl=61 (request in 23) |
| 25 | 4.630797930 | 10.0.0.10 | 10.0.3.20 | ICMP | 74 Echo (ping) request id=0x0027, seq=12/3072, ttl=4 (reply in 26) |
| 26 | 4.630811217 | 10.0.0.10 | 10.0.3.20 | ICMP | 74 Echo (ping) reply id=0x0027, seq=12/3072, ttl=61 (request in 25) |
| 27 | 4.630816376 | 10.0.0.10 | 10.0.3.20 | ICMP | 74 Echo (ping) request id=0x0027, seq=13/3328, ttl=5 (reply in 28) |
| 28 | 4.630829418 | 10.0.0.10 | 10.0.3.20 | ICMP | 74 Echo (ping) reply id=0x0027, seq=13/3328, ttl=61 (request in 27) |
| 29 | 4.630833418 | 10.0.0.10 | 10.0.3.20 | ICMP | 74 Echo (ping) request id=0x0027, seq=14/3584, ttl=5 (reply in 30) |
| 30 | 4.630845857 | 10.0.0.10 | 10.0.3.20 | ICMP | 74 Echo (ping) reply id=0x0027, seq=14/3584, ttl=61 (request in 29) |

Fig. 3. Análise tráfego wireshark

No início, foram enviados vários pacotes com TTL=1, descartados por r2. Em seguida, foram enviados pacotes com TTL=2, também descartados por r3 e foram depois enviados pacotes com TTL=3 que foram descartados por r4. No final foram enviados pacotes com TTL=4 que chegaram ao seu destino, h5. Para cada pacote descartado foi recebido um outro pacote do router que o descartou: "Time-to-live exceeded". Como resposta aos pacotes que alcançaram destino foi recebido um pacote "Echo(ping) reply".

1.c)

Qual deve ser o valor inicial mínimo do campo TTL para alcançar o destino h5? Verifique na prática que a sua resposta está correta.

Resposta

Para alcançar o destino h5, o campo TTL deve ter pelo menos o valor 4. Podemos verificar isto pela Fig.3 uma vez que é a partir dos pacotes com TTL=4 que se obtém "Echo(ping) reply", indicando-nos que estes pacotes chegaram ao destino h5.

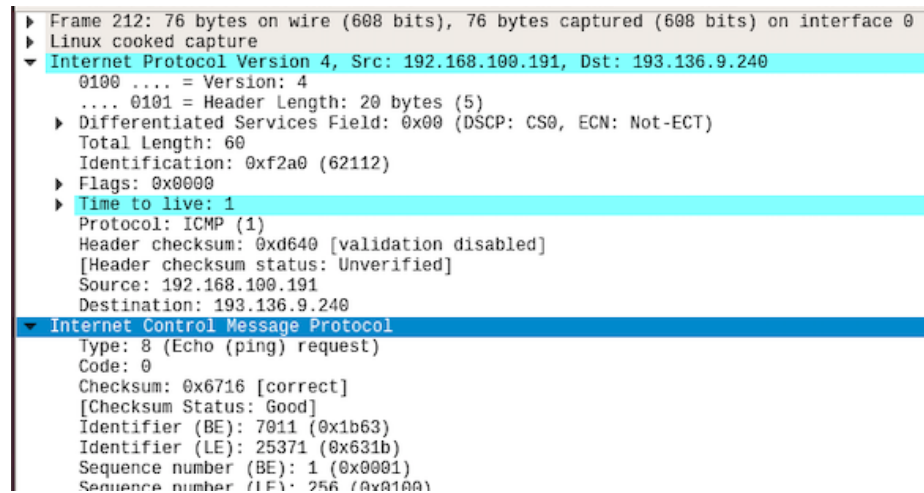
1.d)

Qual o valor médio do tempo de ida-e-volta (Round-Trip Time) obtido?

Resposta

$(0.034 \text{ ms} + 0.012 \text{ ms} + 0.011 \text{ ms}) / 3 = 0.019 \text{ ms}$

Pergunta 2



```
▶ Frame 212: 76 bytes on wire (608 bits), 76 bytes captured (608 bits) on interface 0
▶ Linux cooked capture
▼ Internet Protocol Version 4, Src: 192.168.100.191, Dst: 193.136.9.240
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 60
    Identification: 0xf2a0 (62112)
    ▶ Flags: 0x0000
    ▶ Time to live: 1
    Protocol: ICMP (1)
    Header checksum: 0xd640 [validation disabled]
    [Header checksum status: Unverified]
    Source: 192.168.100.191
    Destination: 193.136.9.240
▼ Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x6716 [correct]
    [Checksum Status: Good]
    Identifier (BE): 7011 (0x1b63)
    Identifier (LE): 25371 (0x631b)
    Sequence number (BE): 1 (0x0001)
    Sequence number (LE): 256 (0x0100)
```

Fig. 4. Cabeçalho da comunicação

2.a)

Qual é o endereço IP da interface ativa do seu computador?

Resposta

O endereço IP da interface ativa do computador é 192.168.100.191, indicado pelo campo Source.

2.b)

Qual é o valor do campo protocolo? O que identifica?

Resposta

O valor do campo protocolo é 1, identificando assim o protocolo ICMP (Internet Control Message Protocol).

2.c)

Quantos bytes tem o cabeçalho IP(v4)? Quantos bytes tem o campo de dados (payload) do datagrama? Como se calcula o tamanho do payload?

Resposta

O cabeçalho IPv4 possui 20 bytes. O campo de dados (payload) do datagrama tem 40 bytes. Este valor é calculado pela diferença entre o tamanho total do datagrama (60 bytes) e o cabeçalho IPv4, ou seja, $60 - 20 = 40$ bytes.

2.d)

O datagrama IP foi fragmentado? Justifique.

Resposta

| | | | | | |
|-----|--------------|-----------------|---------------|------|---|
| 210 | 20.974037020 | 192.168.100.191 | 193.136.9.240 | DNS | 80 Standard query request 0xabab AAAA marco.unlho.pt OPT |
| 211 | 25.974633740 | 127.0.0.1 | 127.0.0.1 | DNS | 80 Standard query response 0xabab AAAA marco.unlho.pt OPT |
| 212 | 25.975406542 | 192.168.100.191 | 193.136.9.240 | ICMP | 76 Echo (ping) request id=0x1b63, seq=2/512, ttl=1 (no response found!) |
| 213 | 25.975514608 | 192.168.100.191 | 193.136.9.240 | ICMP | 76 Echo (ping) request id=0x1b63, seq=2/512, ttl=1 (no response found!) |

▶ Frame 212: 76 bytes on wire (608 bits), 76 bytes captured (608 bits) on interface 0
▶ Linux cooked capture
▼ Internet Protocol Version 4, Src: 192.168.100.191, Dst: 193.136.9.240
0100 ... = Version: 4
... 0101 = Header Length: 20 bytes (5)
▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
Total Length: 60
Identification: 0xf2a0 (62112)
▼ Flags: 0x0000
0... .. = Reserved bit: Not set
.0... .. = Don't fragment: Not set
..0... .. = More fragments: Not set
...0 0000 0000 0000 = Fragment offset: 0
▶ Time to live: 1
Protocol: ICMP (1)
Header checksum: 0xd640 (validation disabled)
[Header checksum status: Unverified]
Source: 192.168.100.191

Fig. 5.

Não. Uma vez que não só, no indicador flags, o Fragment Offset, que nos indica a posição de um fragmento relativamente ao datagrama original, tem o valor 0, como também More Fragments, que nos indica se existem fragmentos para além do atual, se encontra a 0(Not Set). Logo, este datagrama não foi fragmentado, pois este corresponde ao datagrama original.

2.e)

Ordene os pacotes capturados de acordo com o endereço IP fonte (e.g., selecionando o cabeçalho da coluna Source), e analise a sequência de tráfego ICMP gerado a partir do endereço IP atribuído à interface da sua máquina. Para a sequência de mensagens ICMP enviadas pelo seu computador, indique que campos do cabeçalho IP variam de pacote para pacote.

Resposta

| Time | Source | Destination | Protocol | Length | Info |
|------------------|-----------------|----------------|----------|--------|---|
| 210.25.974579895 | 192.168.100.191 | 193.137.16.145 | DNS | 88 | Standard query response 0xa5e8 AAAA marco.uminho.pt OPT |
| 211.25.974633740 | 127.0.0.1 | 127.0.0.1 | DNS | 88 | Standard query response 0x0bab AAAA marco.uminho.pt OPT |
| 212.25.974505092 | 192.168.100.191 | 193.136.9.240 | ICMP | 76 | Echo (ping) request id=0x1b63, seq=2/512, ttl=1 (no response found!) |
| 213.25.975514608 | 192.168.100.191 | 193.136.9.240 | ICMP | 76 | Echo (ping) request id=0x1b63, seq=3/768, ttl=1 (no response found!) |
| 214.25.975519755 | 192.168.100.191 | 193.136.9.240 | ICMP | 76 | Echo (ping) request id=0x1b63, seq=4/1924, ttl=2 (no response found!) |
| 215.25.975524731 | 192.168.100.191 | 193.136.9.240 | ICMP | 76 | Echo (ping) request id=0x1b63, seq=5/1280, ttl=2 (no response found!) |
| 216.25.975529284 | 192.168.100.191 | 193.136.9.240 | ICMP | 76 | Echo (ping) request id=0x1b63, seq=6/1536, ttl=2 (no response found!) |
| 217.25.975533626 | 192.168.100.191 | 193.136.9.240 | ICMP | 76 | Echo (ping) request id=0x1b63, seq=7/1792, ttl=3 (reply in 233) |
| 218.25.975539797 | 192.168.100.191 | 193.136.9.240 | ICMP | 76 | Echo (ping) request id=0x1b63, seq=8/2048, ttl=3 (reply in 234) |
| 219.25.975544778 | 192.168.100.191 | 193.136.9.240 | ICMP | 76 | Echo (ping) request id=0x1b63, seq=9/2304, ttl=3 (reply in 236) |
| 220.25.975548742 | 192.168.100.191 | 193.136.9.240 | ICMP | 76 | Echo (ping) request id=0x1b63, seq=10/2560, ttl=4 (reply in 237) |
| 221.25.975555580 | 192.168.100.191 | 193.136.9.240 | ICMP | 76 | Echo (ping) request id=0x1b63, seq=11/2816, ttl=4 (reply in 239) |
| 222.25.975560680 | 192.168.100.191 | 193.136.9.240 | ICMP | 76 | Echo (ping) request id=0x1b63, seq=12/3072, ttl=4 (reply in 240) |
| 223.25.975565366 | 192.168.100.191 | 193.136.9.240 | ICMP | 76 | Echo (ping) request id=0x1b63, seq=13/3328, ttl=5 (reply in 241) |
| 224.25.975571441 | 192.168.100.191 | 193.136.9.240 | ICMP | 76 | Echo (ping) request id=0x1b63, seq=14/3584, ttl=5 (reply in 242) |
| 225.25.975580750 | 192.168.100.191 | 193.136.9.240 | ICMP | 76 | Echo (ping) request id=0x1b63, seq=15/3840, ttl=5 (reply in 243) |
| 226.25.975597678 | 192.168.100.191 | 193.136.9.240 | ICMP | 76 | Echo (ping) request id=0x1b63, seq=16/4096, ttl=5 (reply in 244) |

Fig. 6.

Os campos do cabeçalho IP que variam de pacote para pacote são o campo de identificação do datagrama, o TTL (time to live) e o Header Checksum.

2.f)

Observa algum padrão nos valores do campo de Identificação do datagrama IP e TTL?

Resposta

Sim, ambos os campos são incrementados em 1 unidade de pacote para pacote.

2.g)

Ordene o tráfego capturado por endereço destino e encontre a série de respostas ICMP TTL exceeded enviadas ao seu computador. Qual é o valor do campo TTL? Esse valor permanece constante para todas as mensagens de resposta ICMP TTL exceeded enviados ao seu host? Porquê?

Resposta

| | | | | | |
|---|--------------|-----------------|-----------------|------|--|
| 250 | 25.977526825 | 127.0.0.1 | 127.0.0.53 | DNS | 100 Standard query 0x9f7b PTR 254.19.136.193.in-addr.arpa OPT |
| 256 | 25.978397556 | 127.0.0.1 | 127.0.0.53 | DNS | 99 Standard query 0x0cd3 PTR 240.9.136.193.in-addr.arpa OPT |
| 226 | 25.97694889 | 192.168.100.254 | 192.168.100.191 | ICMP | 104 Time-to-live exceeded (Time to live exceeded in transit) |
| 229 | 25.976929683 | 192.168.100.254 | 192.168.100.191 | ICMP | 104 Time-to-live exceeded (Time to live exceeded in transit) |
| 230 | 25.976832980 | 192.168.100.254 | 192.168.100.191 | ICMP | 104 Time-to-live exceeded (Time to live exceeded in transit) |
| 233 | 25.976869566 | 193.136.9.240 | 192.168.100.191 | ICMP | 76 Echo (ping) reply id=0x1b63, seq=7/1792, ttl=62 (request in 218) |
| 234 | 25.976912274 | 193.136.9.240 | 192.168.100.191 | ICMP | 76 Echo (ping) reply id=0x1b63, seq=8/2848, ttl=62 (request in 219) |
| 236 | 25.976924832 | 193.136.9.240 | 192.168.100.191 | ICMP | 76 Echo (ping) reply id=0x1b63, seq=9/2304, ttl=62 (request in 220) |
| 237 | 25.976927335 | 193.136.9.240 | 192.168.100.191 | ICMP | 76 Echo (ping) reply id=0x1b63, seq=10/2560, ttl=62 (request in 221) |
| 238 | 25.976931388 | 193.136.9.240 | 192.168.100.191 | ICMP | 72 Time-to-live exceeded (Time to live exceeded in transit) |
| 239 | 25.976938148 | 193.136.9.240 | 192.168.100.191 | ICMP | 76 Echo (ping) reply id=0x1b63, seq=11/2816, ttl=62 (request in 222) |
| 240 | 25.976942180 | 193.136.9.240 | 192.168.100.191 | ICMP | 76 Echo (ping) reply id=0x1b63, seq=12/3072, ttl=62 (request in 223) |
| 244 | 25.976948314 | 193.136.9.240 | 192.168.100.191 | ICMP | 76 Echo (ping) reply id=0x1b63, seq=13/3328, ttl=62 (request in 224) |
| ▶ Frame 230: 104 bytes on wire (832 bits), 104 bytes captured (832 bits) on interface 0 | | | | | |
| Linux cooked capture | | | | | |
| Internet Protocol Version 4, Src: 192.168.100.254, Dst: 192.168.100.191 | | | | | |
| 0100 = Version: 4 | | | | | |
| 0101 = Header Length: 20 bytes (5) | | | | | |
| ▶ Differentiated Services Field: 0xc0 (DSCP: CS6, ECN: Not-ECT) | | | | | |
| Total Length: 88 | | | | | |
| Identification: 0x59eb (23019) | | | | | |
| ▶ Flags: 0x0000 | | | | | |
| Time to live: 64 | | | | | |
| Protocol: ICMP (1) | | | | | |
| Header checksum: 0xd4eb [validation disabled] | | | | | |
| [Header checksum status: Unverified] | | | | | |
| Source: 192.168.100.254 | | | | | |
| Destination: 192.168.100.191 | | | | | |
| ▶ Internet Control Message Protocol | | | | | |

Fig. 7.

O valor do campo TTL é 64 e este não permanece constante, sendo decrementado em 1. Aconteceu routing loop, isto é, o pacote foi enviado para trás e para a frente entre os pontos de loop até que o TTL atingisse o valor zero. É nesta altura que o router envia a mensagem “TTL exceeded”.

Pergunta 3

3.a)

Localize a primeira mensagem ICMP. Porque é que houve necessidade de fragmentar o pacote inicial?

Resposta

| | | | | | |
|---|--------------|-----------------|-----------------|------|---|
| 284 | 25.644159879 | 192.168.100.254 | 192.168.100.191 | DNS | 142 Standard query response 0xc7b6 AAAA marco.uminho.pt SOA dns.uminho.pt OPT |
| 285 | 25.644376574 | 127.0.0.53 | 127.0.0.1 | DNS | 104 Standard query response 0x5edf A marco.uminho.pt A 193.136.9.240 OPT |
| 286 | 25.644498727 | 127.0.0.53 | 127.0.0.1 | DNS | 88 Standard query response 0x06ec AAAA marco.uminho.pt OPT |
| 287 | 25.644514531 | 192.168.100.191 | 193.136.9.240 | IPV4 | 1516 Fragmented IP protocol (proto=ICMP 1, off=1480, ID=3d10) [Reassembled in #289] |
| 288 | 25.644723133 | 192.168.100.191 | 193.136.9.240 | IPV4 | 1516 Fragmented IP protocol (proto=ICMP 1, off=1480, ID=3d10) [Reassembled in #289] |
| 289 | 25.644724589 | 192.168.100.191 | 193.136.9.240 | ICMP | 1261 Echo (ping) request id=0x1e18, seq=1/256, ttl=1 (no response found!) |
| 290 | 25.644741924 | 192.168.100.191 | 193.136.9.240 | IPV4 | 1516 Fragmented IP protocol (proto=ICMP 1, off=1480, ID=3d10) [Reassembled in #292] |
| 291 | 25.644736837 | 192.168.100.191 | 193.136.9.240 | IPV4 | 1516 Fragmented IP protocol (proto=ICMP 1, off=1480, ID=3d10) [Reassembled in #292] |
| 292 | 25.644737892 | 192.168.100.191 | 193.136.9.240 | ICMP | 1261 Echo (ping) request id=0x1e18, seq=2/512, ttl=1 (no response found!) |
| 293 | 25.644741445 | 192.168.100.191 | 193.136.9.240 | IPV4 | 1516 Fragmented IP protocol (proto=ICMP 1, off=1480, ID=3d10) [Reassembled in #295] |
| 294 | 25.644746824 | 192.168.100.191 | 193.136.9.240 | IPV4 | 1516 Fragmented IP protocol (proto=ICMP 1, off=1480, ID=3d10) [Reassembled in #295] |
| 295 | 25.644747873 | 192.168.100.191 | 193.136.9.240 | ICMP | 1261 Echo (ping) request id=0x1e18, seq=3/768, ttl=1 (no response found!) |
| 296 | 25.644749308 | 192.168.100.191 | 193.136.9.240 | IPV4 | 1516 Fragmented IP protocol (proto=ICMP 1, off=1480, ID=3d10) [Reassembled in #298] |
| 297 | 25.645518821 | 192.168.100.191 | 193.136.9.240 | IPV4 | 1516 Fragmented IP protocol (proto=ICMP 1, off=1480, ID=3d10) [Reassembled in #298] |
| 298 | 25.645524463 | 192.168.100.191 | 193.136.9.240 | ICMP | 1261 Echo (ping) request id=0x1e18, seq=4/1024, ttl=2 (no response found!) |
| 299 | 25.645526419 | 192.168.100.191 | 193.136.9.240 | IPV4 | 1516 Fragmented IP protocol (proto=ICMP 1, off=1480, ID=3d10) [Reassembled in #301] |
| ▶ Frame 287: 1516 bytes on wire (12128 bits), 1516 bytes captured (12128 bits) on interface 0 | | | | | |
| Linux cooked capture | | | | | |
| Internet Protocol Version 4, Src: 192.168.100.191, Dst: 193.136.9.240 | | | | | |
| 0100 = Version: 4 | | | | | |
| 0101 = Header Length: 20 bytes (5) | | | | | |
| ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT) | | | | | |
| Total Length: 1500 | | | | | |

Fig. 8.

Mensagem 287. Surgiu a necessidade de fragmentação do datagrama, pois o tamanho do mesmo(1516) era superior ao suportado pela rede (Total Length: 1500 bytes).

3.b)

Imprima o primeiro fragmento do datagrama IP segmentado. Que informação no cabeçalho indica que o datagrama foi fragmentado? Que informação no cabeçalho IP indica que se trata do primeiro fragmento? Qual é o tamanho deste datagrama IP?

Resposta

```
▶ Frame 287: 1516 bytes on wire (12128 bits), 1516 bytes captured (12128 bits) on interface 0
▶ Linux cooked capture
▼ Internet Protocol Version 4, Src: 192.168.100.191, Dst: 193.136.9.240
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 1500
    Identification: 0x3d19 (15641)
    ▼ Flags: 0x2000, More fragments
      0... .. = Reserved bit: Not set
      .0.. .. = Don't fragment: Not set
      ..1. .... = More fragments: Set
      ...0 0000 0000 0000 = Fragment offset: 0
    ▶ Time to live: 1
    Protocol: ICMP (1)
```

Fig. 9.

A informação no cabeçalho que nos indica se o datagrama foi fragmentado é a flag More Fragments que, estando a 1(Set), nos indica que existem mais fragmentos. A flag Fragment Offset, que nos informa acerca da posição de um fragmento em relação ao datagrama original, está a 0, ou seja, este é o primeiro fragmento. O tamanho do datagrama é de 1480 bytes (1500- 20 bytes).

3.c)

Imprima o segundo fragmento do datagrama IP original. Que informação do cabeçalho IP indica que não se trata do 1º fragmento? Há mais fragmentos? O que nos permite afirmar isso?

Resposta

```
▶ Frame 288: 1516 bytes on wire (12128 bits), 1516 bytes captured (12128 bits) on interface 0
▶ Linux cooked capture
▼ Internet Protocol Version 4, Src: 192.168.100.191, Dst: 193.136.9.240
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 1500
    Identification: 0x3d19 (15641)
    ▼ Flags: 0x20b9, More fragments
      0... .. = Reserved bit: Not set
      .0... .. = Don't fragment: Not set
      ..1... .. = More fragments: Set
      ...0 0000 1011 1001 = Fragment offset: 185
    ▶ Time to live: 1
    Protocol: ICMP (1)
```

Fig. 10.

Neste caso, a Fragment Offset está a 185, ou seja, este não se trata do primeiro fragmento uma vez que, nesse caso, a flag estaria com valor 0. Sabemos que existem mais fragmentos, pois a flag More Fragments encontra-se com valor 1.

3.d)

Quantos fragmentos foram criados a partir do datagrama original? Como se detecta o último fragmento correspondente ao datagrama original?

Resposta

Foram criados 3 fragmentos a partir do datagrama original. O último fragmento é detetado através da flag More Fragments. Quando esta se encontra a 0(Not Set), podemos concluir que o fragmento em causa é o último e não existem mais para além do mesmo.

3.e)

Indique, resumindo, os campos que mudam no cabeçalho IP entre os diferentes fragmentos, e explique a forma como essa informação permite reconstruir o datagrama original.

Resposta

Os campos que variam no cabeçalho IP dos diferentes fragmentos são as flags Fragment Offset e More Fragments. Fragment Offset indica a posição de um fragmento em relação ao datagrama original e More Fragments alerta para a existência ou não de mais fragmentos (valor 1 em caso afirmativo e 0 caso contrário). Desta forma, é possível a reconstrução do datagrama original ao sabermos a anterior posição de cada fragmento no mesmo. Ordena-se os fragmentos por ordem crescente do Fragment Offset até que o bit de More Fragments seja 0, isto é, estejamos no último fragmento.

2 Parte 2

2.1 Questões

Pergunta 1

1.a)

Indique que endereços IP e máscaras de rede foram atribuídos pelo CORE a cada equipamento. Para simplificar, pode incluir uma imagem que ilustre de forma clara a topologia definida e o endereçamento usado.

Resposta

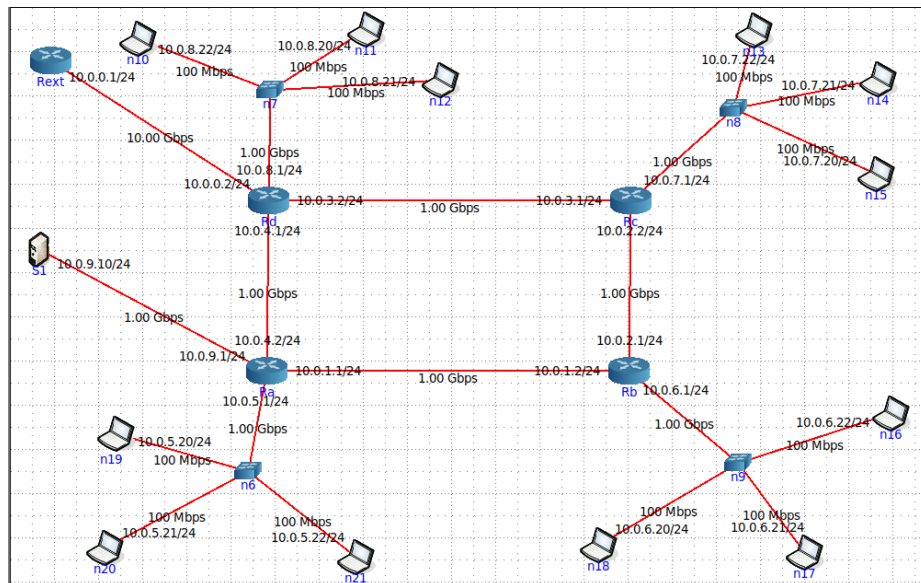


Fig. 11. Topologia Core

Utilizando o laptop n19 com o endereço IP de 10.0.5.20/24 , conseguimos concluir que a máscara terá um total de 24 bytes, ou seja a máscara de rede será 255.255.255.0

1.b)

Trata-se de endereços públicos ou privados? Porquê?

Resposta

Tratam-se de endereços privados porque utilizam como prefixo um dos blocos reservados a endereços privados na norma RFC 1918 pela IANA ("10.0.0.0 - 10.255.255.255(10/8 prefix)").

1.c)

Por que razão não é atribuído um endereço IP aos switches?

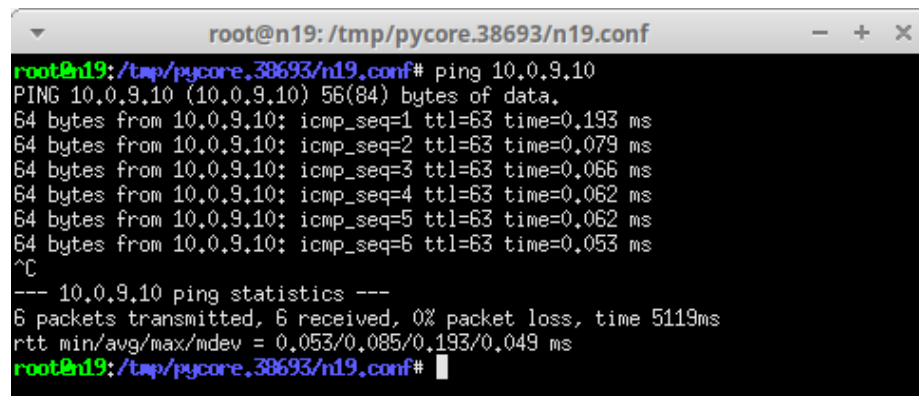
Resposta

Os switches não recebem qualquer IP porque estes operam numa camada abaixo (Layer 2 do TCP/IP) enquanto que o IPv4 opera na layer 3.

1.d)

Usando o comando ping certifique-se que existe conectividade IP entre os laptops dos vários departamentos e o servidor do departamento A (basta certificar-se da conectividade de um laptop por departamento).

Resposta

A terminal window titled 'root@n19: /tmp/pycore.38693/n19.conf' displays the output of a 'ping 10.0.9.10' command. The output shows six successful pings with decreasing response times. A Ctrl-C (^C) is pressed, followed by a summary of the ping statistics. The statistics indicate 6 packets transmitted and received with 0% packet loss and a total time of 5119ms. The round-trip time (rtt) statistics are also shown.

```
root@n19: /tmp/pycore.38693/n19.conf# ping 10.0.9.10
PING 10.0.9.10 (10.0.9.10) 56(84) bytes of data.
64 bytes from 10.0.9.10: icmp_seq=1 ttl=63 time=0.193 ms
64 bytes from 10.0.9.10: icmp_seq=2 ttl=63 time=0.079 ms
64 bytes from 10.0.9.10: icmp_seq=3 ttl=63 time=0.066 ms
64 bytes from 10.0.9.10: icmp_seq=4 ttl=63 time=0.062 ms
64 bytes from 10.0.9.10: icmp_seq=5 ttl=63 time=0.062 ms
64 bytes from 10.0.9.10: icmp_seq=6 ttl=63 time=0.053 ms
^C
--- 10.0.9.10 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5119ms
rtt min/avg/max/mdev = 0.053/0.085/0.193/0.049 ms
root@n19: /tmp/pycore.38693/n19.conf#
```

Fig. 12. Ping do laptop n19(do departamento A) para S1 (servidor do departamento A).

Através das figuras acima, podemos concluir que existe conectividade IP entre os laptops dos vários departamentos e o servidor do departamento A.

```
root@n18: /tmp/pycore.38693/n18.conf
root@n18: /tmp/pycore.38693/n18.conf# ping 10.0.9.10
PING 10.0.9.10 (10.0.9.10) 56(84) bytes of data.
64 bytes from 10.0.9.10: icmp_seq=1 ttl=62 time=0.101 ms
64 bytes from 10.0.9.10: icmp_seq=2 ttl=62 time=0.079 ms
64 bytes from 10.0.9.10: icmp_seq=3 ttl=62 time=0.068 ms
64 bytes from 10.0.9.10: icmp_seq=4 ttl=62 time=0.137 ms
^C
--- 10.0.9.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3067ms
rtt min/avg/max/mdev = 0.068/0.096/0.137/0.027 ms
root@n18: /tmp/pycore.38693/n18.conf#
```

Fig. 13. Ping do laptop n18(do departamento B) para S1 (servidor do departamento A).

```
root@n13: /tmp/pycore.38693/n13.conf
root@n13: /tmp/pycore.38693/n13.conf# ping 10.0.9.10
PING 10.0.9.10 (10.0.9.10) 56(84) bytes of data.
64 bytes from 10.0.9.10: icmp_seq=1 ttl=61 time=0.106 ms
64 bytes from 10.0.9.10: icmp_seq=2 ttl=61 time=0.174 ms
64 bytes from 10.0.9.10: icmp_seq=3 ttl=61 time=0.082 ms
64 bytes from 10.0.9.10: icmp_seq=4 ttl=61 time=0.079 ms
^C
--- 10.0.9.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3054ms
rtt min/avg/max/mdev = 0.079/0.110/0.174/0.038 ms
root@n13: /tmp/pycore.38693/n13.conf#
```

Fig. 14. Ping do laptop n13(do departamento C) para S1 (servidor do departamento A).

```
root@n10: /tmp/pycore.38693/n10.conf
root@n10: /tmp/pycore.38693/n10.conf# ping 10.0.9.10
PING 10.0.9.10 (10.0.9.10) 56(84) bytes of data.
64 bytes from 10.0.9.10: icmp_seq=1 ttl=62 time=0.065 ms
64 bytes from 10.0.9.10: icmp_seq=2 ttl=62 time=0.069 ms
64 bytes from 10.0.9.10: icmp_seq=3 ttl=62 time=0.066 ms
64 bytes from 10.0.9.10: icmp_seq=4 ttl=62 time=0.069 ms
64 bytes from 10.0.9.10: icmp_seq=5 ttl=62 time=0.073 ms
64 bytes from 10.0.9.10: icmp_seq=6 ttl=62 time=0.081 ms
^C
--- 10.0.9.10 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5113ms
rtt min/avg/max/mdev = 0.065/0.070/0.081/0.009 ms
root@n10: /tmp/pycore.38693/n10.conf#
```

Fig. 15. Ping do laptop n10(do departamento D) para S1 (servidor do departamento A).

1.e)

Verifique se existe conectividade IP do router de acesso Rext para o servidor S1.

Resposta

```
root@Rext: /tmp/pycore.38693/Rext.conf
root@Rext: /tmp/pycore.38693/Rext.conf# ping 10.0.9.10
PING 10.0.9.10 (10.0.9.10) 56(84) bytes of data.
64 bytes from 10.0.9.10: icmp_seq=1 ttl=62 time=0.066 ms
64 bytes from 10.0.9.10: icmp_seq=2 ttl=62 time=0.090 ms
64 bytes from 10.0.9.10: icmp_seq=3 ttl=62 time=0.073 ms
64 bytes from 10.0.9.10: icmp_seq=4 ttl=62 time=0.078 ms
64 bytes from 10.0.9.10: icmp_seq=5 ttl=62 time=0.107 ms
64 bytes from 10.0.9.10: icmp_seq=6 ttl=62 time=0.079 ms
64 bytes from 10.0.9.10: icmp_seq=7 ttl=62 time=0.069 ms
^C
--- 10.0.9.10 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6125ms
rtt min/avg/max/mdev = 0.066/0.080/0.107/0.014 ms
root@Rext: /tmp/pycore.38693/Rext.conf#
```

Fig. 16. Ping para S1 a partir de Rext.

Através da figura podemos concluir que existe conectividade IP do router de acesso Rext para o servidor S1.

Pergunta 2

Para o router e um laptop do departamento B:

2.a)

Execute o comando `netstat -rn` por forma a poder consultar a tabela de encaminhamento unicast (IPv4). Inclua no seu relatório as tabelas de encaminhamento obtidas; interprete as várias entradas de cada tabela. Se necessário, consulte o manual respetivo (`man netstat`).

Resposta

```
CC-Min/avg/max/mdev = 0.000/0.003/0.119/0.021 ms
root@Rb:/tmp/pycore.46865/Rb.conf# netstat -rn
Kernel IP routing table
Destination Gateway Genmask Flags MSS Window irtt Iface
10.0.0.0 10.0.1.1 255.255.255.0 UG 0 0 0 eth0
10.0.1.0 0.0.0.0 255.255.255.0 U 0 0 0 eth0
10.0.2.0 0.0.0.0 255.255.255.0 U 0 0 0 eth1
10.0.3.0 10.0.2.2 255.255.255.0 UG 0 0 0 eth1
10.0.4.0 10.0.1.1 255.255.255.0 UG 0 0 0 eth0
10.0.5.0 10.0.1.1 255.255.255.0 UG 0 0 0 eth0
10.0.6.0 0.0.0.0 255.255.255.0 U 0 0 0 eth2
10.0.7.0 10.0.2.2 255.255.255.0 UG 0 0 0 eth1
10.0.8.0 10.0.1.1 255.255.255.0 UG 0 0 0 eth0
10.0.9.0 10.0.1.1 255.255.255.0 UG 0 0 0 eth0
root@Rb:/tmp/pycore.46865/Rb.conf#
```

Fig. 17. Tabela de encaminhamento do router do departamento B (Rb).

Pela tabela do router Rb podemos verificar as redes cujos endereços são 10.0.1.0, 10.0.2.0 e 10.0.6.0 estão ligadas directamente ao router Rb de modo que não necessitam de um Gateway (0.0.0.0). O redirecionamento é único devido ao switch. Os datagramas cujo destino é 10.0.0.0, 10.0.4.0 10.0.5.0, 10.0.8.0, 10.0.9.0 são direcionados para a interface do endereço ("Gateway") 10.0.1.1 que pertence ao router Ra, saindo pela interface local "Iface". Os datagramas com destino 10.0.3.0 e 10.0.7.0 são direcionados para o Gateway 10.0.2.2 que pertence ao router Rc. Como estamos a utilizar Classless, é obrigatório mencionar a máscara que será sempre 24 (255.255.255.0). As flags verificam se a route é válida e dependem do Gateway estar definido ou não (U- não está definido; UG- está definido).

```

cc min/avg/max/mdev = 0.000/0.003/0.113/0.021 ms
root@Rb:/tmp/pycore.46865/Rb.conf# netstat -rn
Kernel IP routing table
Destination      Gateway         Genmask         Flags   MSS Window  irtt Iface
10.0.0.0          10.0.1.1        255.255.255.0   UG        0 0          0 eth0
10.0.1.0          0.0.0.0         255.255.255.0   U          0 0          0 eth0
10.0.2.0          0.0.0.0         255.255.255.0   U          0 0          0 eth1
10.0.3.0          10.0.2.2        255.255.255.0   UG        0 0          0 eth1
10.0.4.0          10.0.1.1        255.255.255.0   UG        0 0          0 eth0
10.0.5.0          10.0.1.1        255.255.255.0   UG        0 0          0 eth0
10.0.6.0          0.0.0.0         255.255.255.0   U          0 0          0 eth2
10.0.7.0          10.0.2.2        255.255.255.0   UG        0 0          0 eth1
10.0.8.0          10.0.1.1        255.255.255.0   UG        0 0          0 eth0
10.0.9.0          10.0.1.1        255.255.255.0   UG        0 0          0 eth0
root@Rb:/tmp/pycore.46865/Rb.conf#

```

Fig. 18. Tabela de encaminhamento do laptop n18 do departamento B.

Pela tabela do laptop n18 podemos verificar que os pacotes cujo destino é 10.0.6.0 são encaminhados diretamente para o seu destino através do switch, enquanto todos os outros (default:0.0.0.0) são encaminhados, por defeito, para o Gateway do router Rb (10.0.6.1).

2.b)

Diga, justificando, se está a ser usado encaminhamento estático ou dinâmico (sugestão: analise que processos estão a correr em cada sistema).

Resposta

```

root@Rb:/tmp/pycore.46865/Rb.conf# ps -A
PID TTY          TIME CMD
  1 ?            00:00:00 vncd
 56 ?            00:00:00 zebra
 62 ?            00:00:00 ospf6d
 66 ?            00:00:00 ospfd
 83 pts/2        00:00:00 bash
 98 pts/2        00:00:00 ps

```

Fig. 19.

Pela tabela podemos verificar que o Router Rb está a correr um processo de ospf o que significa que está constantemente a atualizar a sua tabela de endereços logo o encaminhamento é dinâmico.

```
root@n18: /tmp/pycore.46865/n18.conf
root@n18:/tmp/pycore.46865/n18.conf# ps -A
  PID TTY          TIME CMD
    1 ?            00:00:00 vncd
   48 pts/4        00:00:00 bash
   56 pts/4        00:00:00 ps
root@n18:/tmp/pycore.46865/n18.conf#
```

Fig. 20.

Pelo contrário o laptop n18 não tem nenhum processo a correr relativamente ao encaminhamento logo o mesmo é estático.

2.c)

Admita que, por questões administrativas, a rota por defeito (0.0.0.0 ou default) deve ser retirada definitivamente da tabela de encaminhamento do servidor S1 localizado no departamento A. Use o comando route delete para o efeito. Que implicação tem esta medida para os utilizadores da empresa que acedem ao servidor? Justifique.

Resposta

```
root@S1:/tmp/pycore.46865/S1.conf# route del -net 0.0.0.0
root@S1:/tmp/pycore.46865/S1.conf# netstat -rn
Kernel IP routing table
Destination Gateway Genmask Flags MSS Window irtt Iface
10.0.9.0 0.0.0.0 255.255.255.0 U 0 0 0 eth0
```

Fig. 21.

```
root@S1:/tmp/pycore.46865/S1.conf# route del -net 0.0.0.0
root@S1:/tmp/pycore.46865/S1.conf# ping 10.0.7.22
connect: Network is unreachable
```

Fig. 22.

```

root@n13:/tmp/pycore.46865/n13.conf# ping 10.0.9.10
PING 10.0.9.10 (10.0.9.10) 56(84) bytes of data.
^C
--- 10.0.9.10 ping statistics ---
17 packets transmitted, 0 received, 100% packet loss, time 16375ms

```

Fig. 23.

Este servidor consegue agora apenas enviar pacotes cujo destino é o Router Ra, o que implica que os utilizadores que tenham enviado mensagens para ele não consigam receber uma resposta.

2.d)

Adicione as rotas estáticas necessárias para restaurar a conectividade para o servidor S1 por forma a contornar a restrição imposta na alínea c). Utilize para o efeito o comando `route add` e registre os comandos que usou.

Resposta

```

root@S1:/tmp/pycore.46865/S1.conf# route delete 0.0.0.0
root@S1:/tmp/pycore.46865/S1.conf# netstat -rn
Kernel IP routing table
Destination    Gateway         Genmask         Flags   MSS Window  irtt Iface
10.0.9.0        0.0.0.0         255.255.255.0   U        0 0          0 eth0
root@S1:/tmp/pycore.46865/S1.conf# route add default gw 10.0.9.1 eth0
root@S1:/tmp/pycore.46865/S1.conf# netstat -rn
Kernel IP routing table
Destination    Gateway         Genmask         Flags   MSS Window  irtt Iface
0.0.0.0         10.0.9.1        0.0.0.0         UG        0 0          0 eth0
10.0.9.0        0.0.0.0         255.255.255.0   U        0 0          0 eth0

```

Fig. 24.

2.e)

Teste a nova política de encaminhamento garantindo que o servidor está novamente acessível utilizando para o efeito o comando `ping`. Registe a nova tabela de encaminhamento do servidor.

Resposta

```
root@S1:/tmp/pycore.46865/S1.conf# ping 10.0.7.22
PING 10.0.7.22 (10.0.7.22) 56(84) bytes of data:
64 bytes from 10.0.7.22: icmp_seq=1 ttl=61 time=0.203 ms
64 bytes from 10.0.7.22: icmp_seq=2 ttl=61 time=0.083 ms
64 bytes from 10.0.7.22: icmp_seq=3 ttl=61 time=0.083 ms
64 bytes from 10.0.7.22: icmp_seq=4 ttl=61 time=0.091 ms
64 bytes from 10.0.7.22: icmp_seq=5 ttl=61 time=0.084 ms
```

Fig. 25.

```
root@S1:/tmp/pycore.46865/S1.conf# route add default gw 10.0.9.1 eth0
root@S1:/tmp/pycore.46865/S1.conf# netstat -rn
Kernel IP routing table
Destination    Gateway         Genmask         Flags   MSS Window  irtt Iface
0.0.0.0        10.0.9.1        0.0.0.0         UG      0 0        0 eth0
10.0.9.0       0.0.0.0         255.255.255.0   U       0 0        0 eth0
```

Fig. 26. Tabela de encaminhamento do servidor (S1).

Pergunta 3

3.1)

Considere que dispõe apenas do endereço de rede IP 172.yyx.32.0/20, em que “yy” são os dígitos correspondendo ao seu número de grupo (Gyy) e “x” é o dígito correspondente ao seu turno prático (PLx). Defina um novo esquema de endereçamento para as redes dos departamentos (mantendo a rede de acesso e core inalteradas) e atribua endereços às interfaces dos vários sistemas envolvidos. Deve justificar as opções usadas.

Resposta

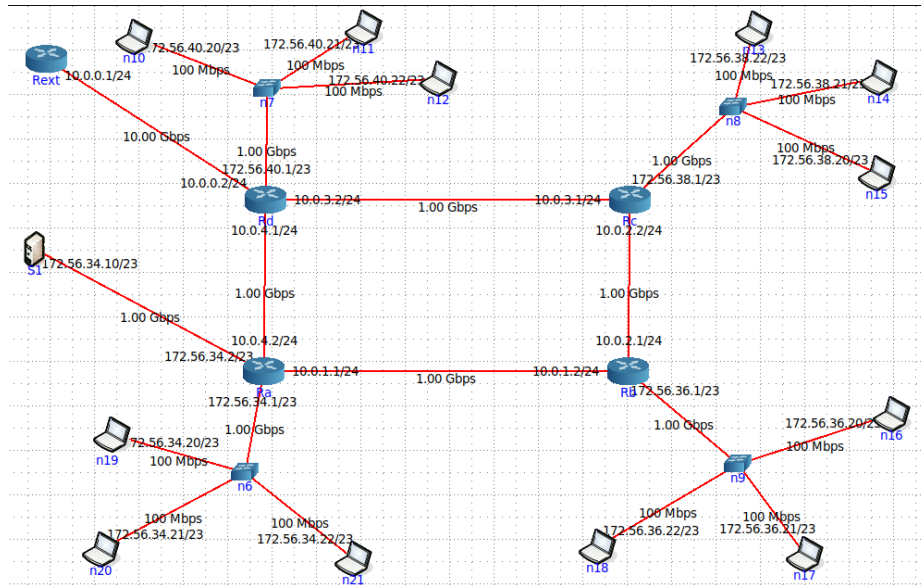


Fig. 27. Topologia Core

O nosso turno é o pl6 e o nosso grupo é o 05. Assim temos direito aos endereços 172.056.032.0/20 que dividimos em quatro subredes usando 3 bits($2^3 - 2 = 6$) :

172.056.034.0/23

172.056.36.0/23

172.056.38.0/23

172.056.40.0/23

Existem 2 subredes que ficarão para uso futuro:

172.056.42.0/23

172.056.44.0/23

Para as interfaces usamos os endereços: 1 a 9 para router; 10 a 19 para os servidores; e os restantes para os laptops.

3.2)

Qual a máscara de rede que usou (em notação decimal)? Quantos interfaces IP pode interligar em cada departamento? Justifique.

Resposta

A máscara que foi usada é 23. Como temos 9 bits de interface podemos interligar 510 interfaces IP em cada departamento ($2^9 - 2$).

3.3)

Garanta e verifique que a conectividade IP entre as várias redes locais da organização MIEI-RC é mantida. Explique como procedeu.

Resposta

```
root@n15:/tmp/pycore.36807/n15.conf# ping 10.0.9.10
PING 10.0.9.10 (10.0.9.10) 56(84) bytes of data.
64 bytes from 10.0.9.10: icmp_seq=1 ttl=61 time=0.092 ms
64 bytes from 10.0.9.10: icmp_seq=2 ttl=61 time=0.119 ms
64 bytes from 10.0.9.10: icmp_seq=3 ttl=61 time=0.103 ms
64 bytes from 10.0.9.10: icmp_seq=4 ttl=61 time=0.115 ms
64 bytes from 10.0.9.10: icmp_seq=5 ttl=61 time=0.102 ms
```

Fig. 28. Ping do laptop n15 para o servidor S1

```
root@n15:/tmp/pycore.36807/n15.conf# ping 172.56.34.21
PING 172.56.34.21 (172.56.34.21) 56(84) bytes of data.
64 bytes from 172.56.34.21: icmp_seq=1 ttl=61 time=0.094 ms
64 bytes from 172.56.34.21: icmp_seq=2 ttl=61 time=0.126 ms
64 bytes from 172.56.34.21: icmp_seq=3 ttl=61 time=0.104 ms
64 bytes from 172.56.34.21: icmp_seq=4 ttl=61 time=0.120 ms
64 bytes from 172.56.34.21: icmp_seq=5 ttl=61 time=0.103 ms
```

Fig. 29. Ping do laptop n15 para o laptop n20

Pelas figuras acima podemos verificar que a conectividade foi mantida (existiu troca de pacotes aquando dos pings).

3 Conclusão

Neste trabalho, conseguimos adquirir ferramentas de aprendizagem para uma melhor compreensão de como o IP (Internet Protocol) funciona e se relaciona com a rede em questão.

Começamos por analisar a estrutura do ipv4 como um datagrama onde vimos o conteúdo do seu Header e aprendemos a analisar o seu tamanho, a fragmentação deste, o TTL(time-to-live), que servem para assegurar rotas, garantir que o pacote chegue ao destino e enviar corretamente a mensagem independentemente do tamanho dos dados.

De seguida percebemos a importância do ipv4 das interfaces dos equipamentos com o routing, as tabelas de encaminhamento e as subredes.

As tabelas de encaminhamento serviram para analisarmos o tráfego de pacotes podendo ajustar as suas entradas para casos alternativos.

No final a divisão em subredes mostrou a sua utilidade na organização e gestão dos endereços, sacrificando espaço de endereçamento.

Assim, este trabalho serviu de complemento às aulas teóricas ajudando a consolidar a matéria aprendida em todo o capítulo do protocolo IP.