

Projeto Laboratorial

REDES DE COMPUTADORES II

Mestrado Integrado em Engenharia de Telecomunicações e Informática
Grupo 6:



Leandro Alves
A82157



J. Eduardo Santos
A82350

UNIVERSIDADE DO MINHO | ANO LETIVO 2018/2019



Índice

Índice	2
Índice de Tabelas	2
1 Introdução	3
2 Topologia da rede	3
3 Funcionalidades e procedimentos	3
3.1 Tabela de encaminhamento	4
3.2 Tabela de endereços MAC	4
3.3 Transmissão de pacotes.....	4
4 Testes e exemplos:.....	5
5 Conclusões.....	9
6 Referências	9

Índice de Tabelas

Figura 2.1 – Topologia da rede no CORE.	3
Figura 4.1 – Comando <i>ping</i> usado em PCA (n7) para PCB (n12).	5
Figura 4.2 – Estrutura dos pacotes Ethernet gerados no <i>request</i> do <i>ping</i>	5
Figura 4.3 – Pacotes visualizados no Wireshark da interface eth0 do <i>host</i> (n1).	6
Figura 4.4 – Pacotes visualizados no Wireshark da interface eth1 do <i>host</i>	7
Figura 4.5 – Comando <i>ping</i> usado no PC n6 para o endereço 8.8.8.8.	8
Figura 4.6 – Pacotes visualizados no Wireshark da interface eth1 do <i>router</i>	8

1 Introdução

Este projeto foi desenvolvido no âmbito da Unidade Curricular de Redes de Computadores II com o objetivo de se desenvolver um *router* capaz de receber pacotes IPv4 e de os encaminhar de acordo com as regras contidas numa tabela de encaminhamento.

2 Topologia da rede

Foi desenvolvida em ambiente CORE uma topologia, demonstrada na Figura 2.1, com quatro redes de área local (LANs) do tipo Ethernet, interligadas através de um *host* que inclui quatro interfaces de rede ligadas a computadores, e uma interface ligada à internet através de um *router*.

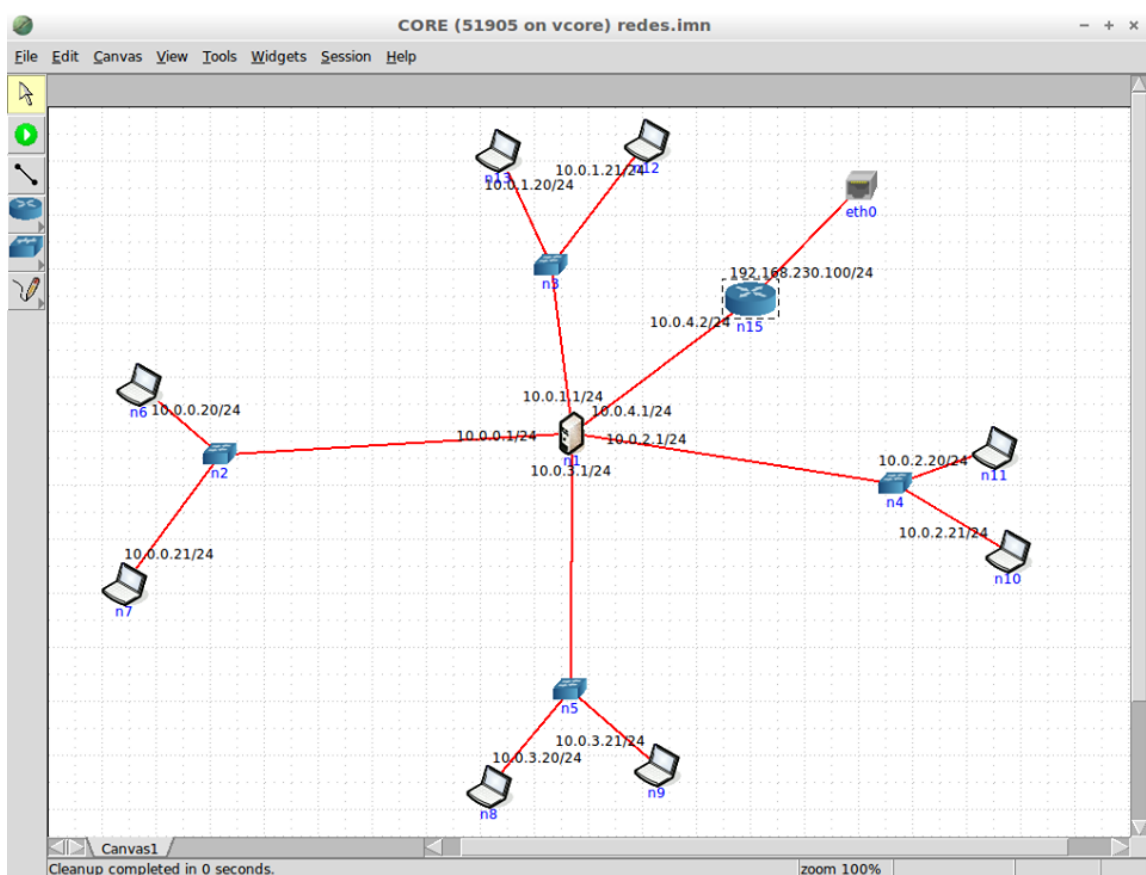


Figura 2.1 – Topologia da rede no CORE.

3 Funcionalidades e procedimentos

As funcionalidades deste *router* incluem reencaminhar pacotes Ethernet entre computadores independentemente das interfaces a que estão ligados.

O programa foi desenvolvido em linguagem Java com o apoio da biblioteca pcap4j^[1].

3.1 Tabela de encaminhamento

Após a execução do programa que irá simular o *router*, será lida uma tabela de encaminhamento que pode ser gerada de duas formas:

- Manualmente – através da leitura dum ficheiro que contem uma linha por interface, e tem como colunas os respetivos endereços de rede, de interface e de máscara, e o nome dessa interface (“eth<n>”).
- Automaticamente – através da listagem de todas as interfaces Ethernet (“eth<n>”) detetadas durante a execução do CORE.

3.2 Tabela de endereços MAC

Os endereços MAC que correspondem aos endereços IP de cada computador são guardados *hardcoded* numa tabela, visto que a biblioteca usada necessitava de tramas Ethernet como pacotes transmitidos, o que necessita dos endereços MAC de origem e destino^[2].

3.3 Transmissão de pacotes

Para a receção de pacotes, é criada uma *thread* com um *socket* por interface para que cada uma esteja à escuta de pacotes.

Quando um computador quer comunicar com outro computador (por exemplo, através do comando *ping* <IP_{destino}>), é gerado um pacote Ethernet que é enviado para a interface a que se encontra ligado. Se esse pacote contiver um pacote IPv4, este é aceite pela interface e é-lhe lido o endereço IP de destino e através da tabela de endereços MAC, é obtido o endereço MAC de destino.

Para se saber para que interface é que se deve redirecionar o pacote, esta é procurada através da tabela de encaminhamento, comparando o endereço IP destino com os endereços de rede, e através da respetiva máscara, é descoberta a rede a que o endereço IP destino pertence, assim como a respetiva interface de saída.

Se não houver nenhuma correspondência entre o IP de destino e cada endereço de rede, assume-se que esse IP não pertence à rede local e, portanto, deve ser redirecionado para a interface ligada à Internet (eth4).

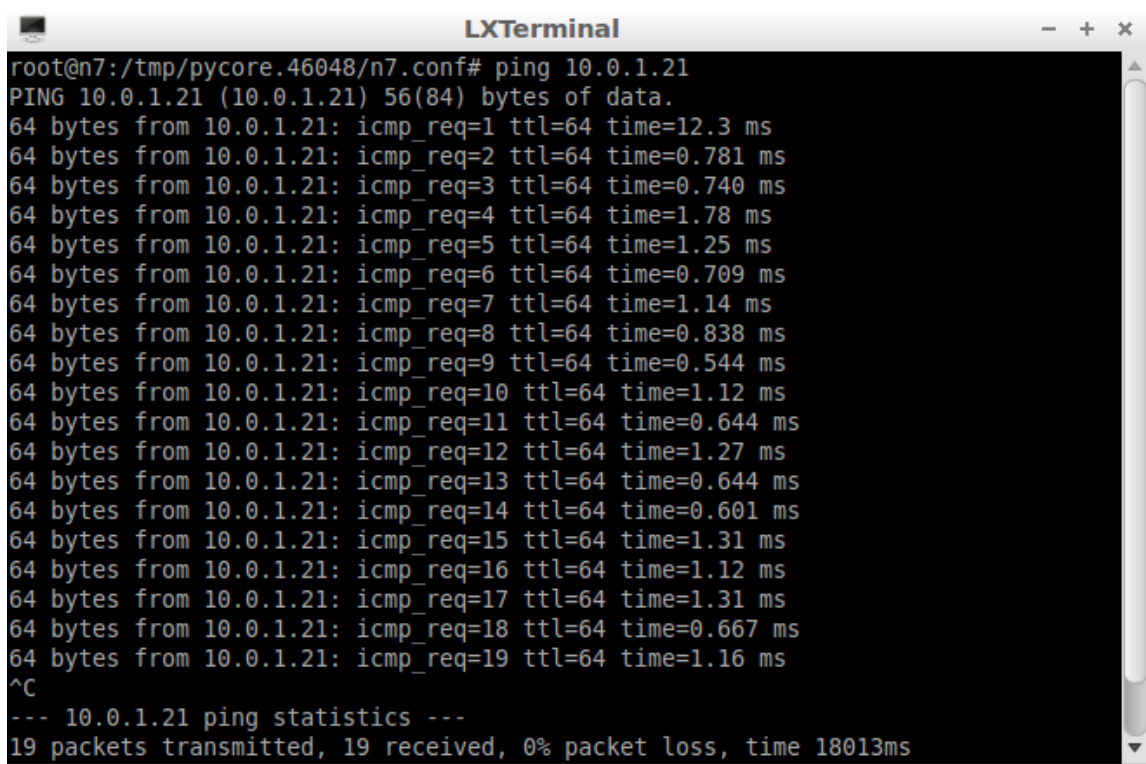
Após ser descoberta a interface de saída, o pacote IPv4 (que contem o IP de destino) contido no pacote Ethernet é retirado do mesmo e é criado um novo pacote Ethernet, sendo-lhe atribuído o mesmo pacote IPv4, o endereço MAC da própria interface como MAC de origem, e o endereço MAC relativo ao IP destino como MAC de destino. Este pacote é construído e enviado para o *socket* indexado pelo identificador da respetiva interface que reencaminha esse pacote para o computador destinado.

O encaminhamento de pacotes para a Internet necessitou de um *router* ligado ao *host* com NAT configurado na Firewall.

4 Testes e exemplos:

Sendo PCA o computador n7 ($IP_{PCA} = 10.0.0.21/24$, $MAC_{PCA} = 00:00:00:aa:00:01$) e PCB o computador n12 ($IP_{PCB} = 10.0.1.21/24$, $MAC_{PCB} = 00:00:00:aa:00:04$):

O comando *ping* na *Shell window* > *bash* de PCA para o endereço IP_{PCB} , é demonstrado na Figura 4.1.



```
root@n7:/tmp/pycore.46048/n7.conf# ping 10.0.1.21
PING 10.0.1.21 (10.0.1.21) 56(84) bytes of data.
64 bytes from 10.0.1.21: icmp_req=1 ttl=64 time=12.3 ms
64 bytes from 10.0.1.21: icmp_req=2 ttl=64 time=0.781 ms
64 bytes from 10.0.1.21: icmp_req=3 ttl=64 time=0.740 ms
64 bytes from 10.0.1.21: icmp_req=4 ttl=64 time=1.78 ms
64 bytes from 10.0.1.21: icmp_req=5 ttl=64 time=1.25 ms
64 bytes from 10.0.1.21: icmp_req=6 ttl=64 time=0.709 ms
64 bytes from 10.0.1.21: icmp_req=7 ttl=64 time=1.14 ms
64 bytes from 10.0.1.21: icmp_req=8 ttl=64 time=0.838 ms
64 bytes from 10.0.1.21: icmp_req=9 ttl=64 time=0.544 ms
64 bytes from 10.0.1.21: icmp_req=10 ttl=64 time=1.12 ms
64 bytes from 10.0.1.21: icmp_req=11 ttl=64 time=0.644 ms
64 bytes from 10.0.1.21: icmp_req=12 ttl=64 time=1.27 ms
64 bytes from 10.0.1.21: icmp_req=13 ttl=64 time=0.644 ms
64 bytes from 10.0.1.21: icmp_req=14 ttl=64 time=0.601 ms
64 bytes from 10.0.1.21: icmp_req=15 ttl=64 time=1.31 ms
64 bytes from 10.0.1.21: icmp_req=16 ttl=64 time=1.12 ms
64 bytes from 10.0.1.21: icmp_req=17 ttl=64 time=1.31 ms
64 bytes from 10.0.1.21: icmp_req=18 ttl=64 time=0.667 ms
64 bytes from 10.0.1.21: icmp_req=19 ttl=64 time=1.16 ms
^C
--- 10.0.1.21 ping statistics ---
19 packets transmitted, 19 received, 0% packet loss, time 18013ms
```

Figura 4.1 – Comando *ping* usado em PCA (n7) para PCB (n12).

Na Figura 4.2 é demonstrada a estrutura dos pacotes Ethernet gerados no *request* de PCA para PCB.

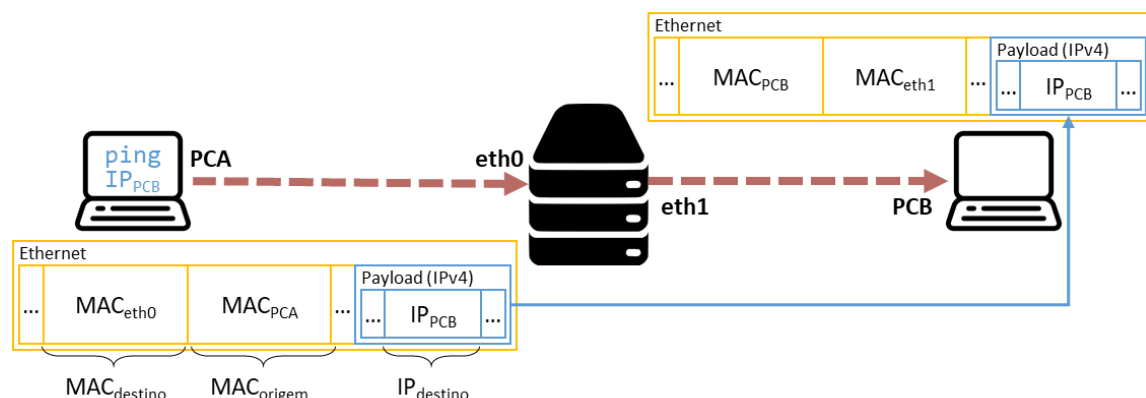


Figura 4.2 – Estrutura dos pacotes Ethernet gerados no *request* do *ping*.

A visualização dos pacotes gerados no Wireshark nas interfaces envolvidas está demonstrada na Figura 4.3 e na Figura 4.4. É de notar os endereços das interfaces ($IP_{host.eth0} = 10.0.0.1/24$, $MAC_{host.eth0} = 00:00:00:aa:00:02$, $IP_{host.eth1} = 10.0.1.1/24$, $MAC_{host.eth1} = 00:00:00:aa:00:05$).

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	00:00:00 aa:00:01	Broadcast	ARP	42	Who has 10.0.0.1? Tell 10.0.0.21
2	0.000039	00:00:00 aa:00:02	00:00:00 aa:00:01	ARP	42	10.0.0.1 is at 00:00:00:aa:00:02
3	0.000050	10.0.0.21	10.0.1.21	ICMP	98	Echo (ping) request id=0x001a, seq=1/256, ttl
4	0.012250	10.0.1.21	10.0.0.21	ICMP	98	Echo (ping) reply id=0x001a, seq=1/256, ttl
5	1.001830	10.0.0.21	10.0.1.21	ICMP	98	Echo (ping) request id=0x001a, seq=2/512, ttl
6	1.002556	10.0.1.21	10.0.0.21	ICMP	98	Echo (ping) reply id=0x001a, seq=2/512, ttl
7	2.000896	10.0.0.21	10.0.1.21	ICMP	98	Echo (ping) request id=0x001a, seq=3/768, ttl
8	2.001587	10.0.1.21	10.0.0.21	ICMP	98	Echo (ping) reply id=0x001a, seq=3/768, ttl
9	3.000430	10.0.0.21	10.0.1.21	ICMP	98	Echo (ping) request id=0x001a, seq=4/1024, tt
10	3.002178	10.0.1.21	10.0.0.21	ICMP	98	Echo (ping) reply id=0x001a, seq=4/1024, tt
11	4.003215	10.0.0.21	10.0.1.21	ICMP	98	Echo (ping) request id=0x001a, seq=5/1280, tt
12	4.004429	10.0.1.21	10.0.0.21	ICMP	98	Echo (ping) reply id=0x001a, seq=5/1280, tt
13	5.005052	10.0.0.21	10.0.1.21	ICMP	98	Echo (ping) request id=0x001a, seq=6/1536, tt
14	5.005728	10.0.1.21	10.0.0.21	ICMP	98	Echo (ping) reply id=0x001a, seq=6/1536, tt
15	6.004454	10.0.0.21	10.0.1.21	ICMP	98	Echo (ping) request id=0x001a, seq=7/1792, tt
16	6.005565	10.0.1.21	10.0.0.21	ICMP	98	Echo (ping) reply id=0x001a, seq=7/1792, tt
17	7.005850	10.0.0.21	10.0.1.21	ICMP	98	Echo (ping) request id=0x001a, seq=8/2048, tt
18	7.006642	10.0.1.21	10.0.0.21	ICMP	98	Echo (ping) reply id=0x001a, seq=8/2048, tt
19	8.004822	10.0.0.21	10.0.1.21	ICMP	98	Echo (ping) request id=0x001a, seq=9/2304, tt

Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits)

Ethernet II, Src: 00:00:00 aa:00:01 (00:00:00:aa:00:01), Dst: Broadcast (ff:ff:ff:ff:ff:ff)

Address Resolution Protocol (request)

```

0000  ff ff ff ff ff ff 00 00 00 aa 00 01 08 06 00 01  .....
0010  08 00 06 04 00 01 00 00 00 aa 00 01 0a 00 00 15  .....
0020  00 00 00 00 00 00 0a 00 00 01  .....

```

Figura 4.3 – Pacotes visualizados no Wireshark da interface eth0 do *host* (n1).

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.0.0.21	10.0.1.21	ICMP	98	Echo (ping) request id=0x001a, seq=1/256, ttl=64
2	0.000073	00:00:00_aa:00:04	Broadcast	ARP	42	Who has 10.0.1.1? Tell 10.0.1.21
3	0.000085	00:00:00_aa:00:05	00:00:00_aa:00:04	ARP	42	10.0.1.1 is at 00:00:00:aa:00:05
4	0.000092	10.0.1.21	10.0.0.21	ICMP	98	Echo (ping) reply id=0x001a, seq=1/256, ttl=64
5	0.991696	10.0.0.21	10.0.1.21	ICMP	98	Echo (ping) request id=0x001a, seq=2/512, ttl=64
6	0.991714	10.0.1.21	10.0.0.21	ICMP	98	Echo (ping) reply id=0x001a, seq=2/512, ttl=64
7	1.990832	10.0.0.21	10.0.1.21	ICMP	98	Echo (ping) request id=0x001a, seq=3/768, ttl=64
8	1.990872	10.0.1.21	10.0.0.21	ICMP	98	Echo (ping) reply id=0x001a, seq=3/768, ttl=64
9	2.991242	10.0.0.21	10.0.1.21	ICMP	98	Echo (ping) request id=0x001a, seq=4/1024, ttl=64
10	2.991308	10.0.1.21	10.0.0.21	ICMP	98	Echo (ping) reply id=0x001a, seq=4/1024, ttl=64
11	3.993485	10.0.0.21	10.0.1.21	ICMP	98	Echo (ping) request id=0x001a, seq=5/1280, ttl=64
12	3.993615	10.0.1.21	10.0.0.21	ICMP	98	Echo (ping) reply id=0x001a, seq=5/1280, ttl=64
13	4.994972	10.0.0.21	10.0.1.21	ICMP	98	Echo (ping) request id=0x001a, seq=6/1536, ttl=64
14	4.995029	10.0.1.21	10.0.0.21	ICMP	98	Echo (ping) reply id=0x001a, seq=6/1536, ttl=64
15	5.994682	10.0.0.21	10.0.1.21	ICMP	98	Echo (ping) request id=0x001a, seq=7/1792, ttl=64
16	5.994756	10.0.1.21	10.0.0.21	ICMP	98	Echo (ping) reply id=0x001a, seq=7/1792, ttl=64
17	6.995801	10.0.0.21	10.0.1.21	ICMP	98	Echo (ping) request id=0x001a, seq=8/2048, ttl=64
18	6.995822	10.0.1.21	10.0.0.21	ICMP	98	Echo (ping) reply id=0x001a, seq=8/2048, ttl=64
19	7.994673	10.0.0.21	10.0.1.21	ICMP	98	Echo (ping) request id=0x001a, seq=9/2304, ttl=64

Frame 19: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface n1.eth1.83
 Ethernet II, Src: 00:00:00_aa:00:04 (00:00:00:aa:00:04), Dst: 00:00:00_aa:00:05 (00:00:00:aa:00:05)
 Internet Protocol Version 4, Src: 10.0.1.21 (10.0.1.21), Dst: 10.0.0.21 (10.0.0.21)
 Internet Control Message Protocol

0000 00 00 00 aa 00 05 00 00 00 aa 00 04 08 00 45 00E.
 0010 00 54 d4 c8 00 00 40 01 90 b7 0a 00 01 15 0a 00 .T....@.
 0020 00 15 00 00 96 89 00 1a 00 0d f6 79 e5 5c 9e 75y..u
 0030 04 00 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15
 0040 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25
 0050 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35
 0060 36 37 38 39 3a 3b 3c 3d 3e 3f 40 41 42 43 44 45
 0070 46 47 48 49 4a 4b 4c 4d 4e 4f 50 51 52 53 54 55
 0080 56 57 58 59 5a 5b 5c 5d 5e 5f 60 61 62 63 64 65
 0090 66 67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75
 00a0 76 77 78 79 7a 7b 7c 7d 7e 7f 80 81 82 83 84 85
 00b0 86 87 88 89 8a 8b 8c 8d 8e 8f 90 91 92 93 94 95
 00c0 96 97 98 99 9a 9b 9c 9d 9e 9f a0 a1 a2 a3 a4 a5
 00d0 a6 a7 a8 a9 aa ab ac ad ae af b0 b1 b2 b3 b4 b5
 00e0 b6 b7 b8 b9 ba bb bc bd be bf c0 c1 c2 c3 c4 c5
 00f0 c6 c7 c8 c9 ca cb cc cd ce cf d0 d1 d2 d3 d4 d5
 0100 d6 d7 d8 d9 da db dc dd de df e0 e1 e2 e3 e4 e5
 0110 e6 e7 e8 e9 ea eb ec ed ee ef f0 f1 f2 f3 f4 f5
 0120 f6 f7 f8 f9 fa fb fc fd fe ff

Figura 4.4 – Pacotes visualizados no Wireshark da interface eth1 do *host*.

Como se pode verificar, no *request* a origem é IP_{PCA} e o destino é IP_{PCB} , e no *reply* o sentido é o oposto.

Se for usado o comando *ping* para um endereço IP fora da rede, por exemplo 8.8.8.8, demonstrado na Figura 4.5, o *request* será reencaminhado para o *router* ($n15$, $IP_{router.eth1} = 192.168.230.100/24$, $MAC_{router.eth1} = 00:00:00:aa:00:0d$) através da interface eth4 do *host* ($IP_{host.eth4} = 10.0.4.1/24$, $MAC_{host.eth4} = 00:00:00:aa:00:0c$).


```

root@n6:/tmp/pycore.51904/n6.conf# ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_req=1 ttl=127 time=23.1 ms
64 bytes from 8.8.8.8: icmp_req=2 ttl=127 time=23.6 ms
64 bytes from 8.8.8.8: icmp_req=3 ttl=127 time=22.9 ms
64 bytes from 8.8.8.8: icmp_req=4 ttl=127 time=23.1 ms
64 bytes from 8.8.8.8: icmp_req=5 ttl=127 time=20.1 ms
64 bytes from 8.8.8.8: icmp_req=6 ttl=127 time=29.5 ms
64 bytes from 8.8.8.8: icmp_req=7 ttl=127 time=22.2 ms
64 bytes from 8.8.8.8: icmp_req=8 ttl=127 time=23.5 ms
64 bytes from 8.8.8.8: icmp_req=9 ttl=127 time=28.5 ms
64 bytes from 8.8.8.8: icmp_req=10 ttl=127 time=30.5 ms
64 bytes from 8.8.8.8: icmp_req=11 ttl=127 time=30.8 ms
64 bytes from 8.8.8.8: icmp_req=12 ttl=127 time=27.9 ms
64 bytes from 8.8.8.8: icmp_req=13 ttl=127 time=22.3 ms
64 bytes from 8.8.8.8: icmp_req=14 ttl=127 time=23.7 ms
64 bytes from 8.8.8.8: icmp_req=15 ttl=127 time=22.2 ms
64 bytes from 8.8.8.8: icmp_req=16 ttl=127 time=22.1 ms
64 bytes from 8.8.8.8: icmp_req=17 ttl=127 time=20.0 ms
^C
--- 8.8.8.8 ping statistics ---
17 packets transmitted, 17 received, 0% packet loss, time 16052ms
rtt min/avg/max/mdev = 20.078/24.515/30.835/3.405 ms
root@n6:/tmp/pycore.51904/n6.conf#

```

Figura 4.5 – Comando *ping* usado no PC n6 para o endereço 8.8.8.8.

Os pacotes visualizados no Wireshark do *router* estão apresentados na Figura 4.6.

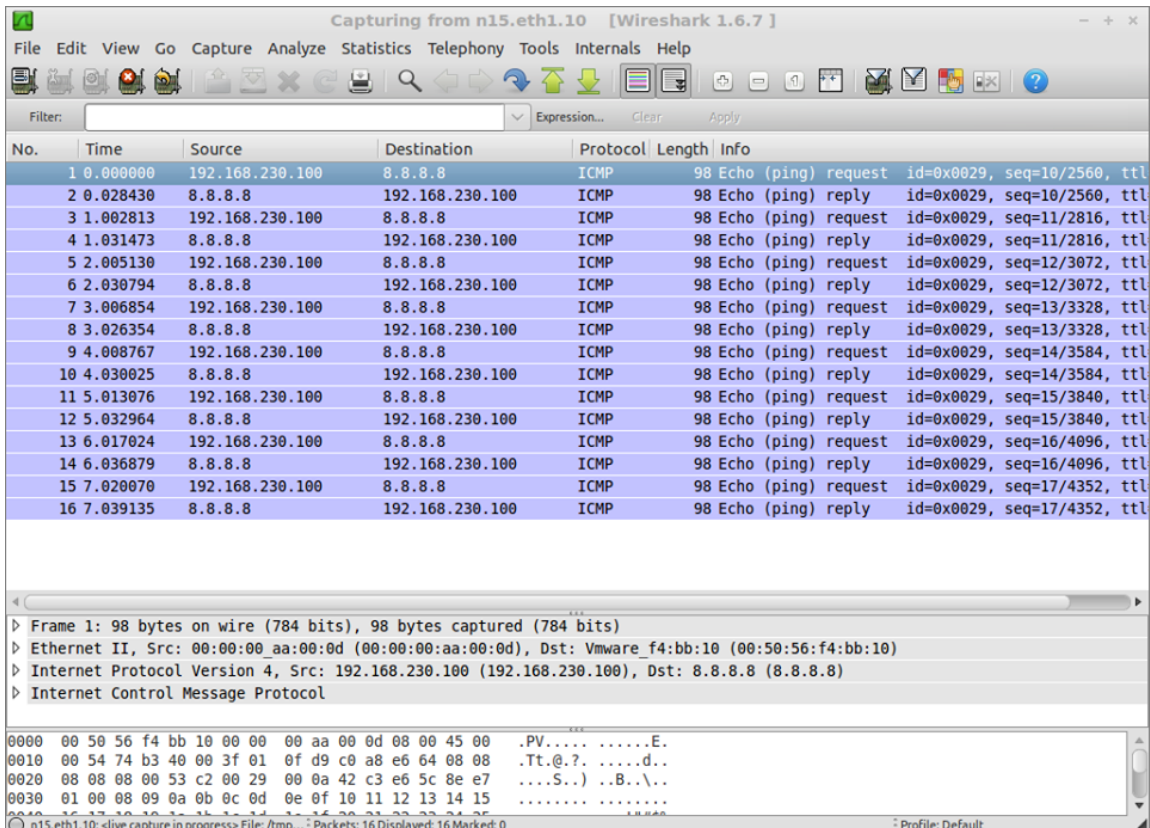


Figura 4.6 – Pacotes visualizados no Wireshark da interface eth1 do *router*.

5 Conclusões

Este projeto ajudou-nos a adquirir competências relacionadas com o processo de encaminhamento de pacotes em redes IP (IPv4 e Ethernet) e conhecimentos de novas bibliotecas de Java. Também foi possível por em prática conhecimentos de outras Unidades Curriculares, nomeadamente Sistemas Distribuídos, acerca da implementação de *threads* e *sockets* num sistema cliente/servidor (*host/computador*).

Durante o desenvolvimento deste projeto, surgiram algumas dificuldades, nomeadamente no suporte das bibliotecas e na disponibilidade de *raw sockets* nas mesmas. Contudo, com a ajuda de outros grupos, achamos que todas as dificuldades foram ultrapassadas.

6 Referências

- [1] Yamada, K. (2013). *kaitoy/pcap4j*. [online] GitHub. Disponível em: <https://github.com/kaitoy/pcap4j> [Acedido a 20/04/2019].
- [2] Yamada, K. (2019). *Can't set dlt Raw IP in handle · Issue #221 · kaitoy/pcap4j*. [online] GitHub. Disponível em: <https://github.com/kaitoy/pcap4j/issues/221?fbclid=IwAR0-jmcXODY0UFpkgNBwKT2SNG1aGT0dozmlxsGN-nacS93yVsUA-bqB8RTQ> [Acedido a 04/05/2019].