

Universidade do Minho

Relatório - Redes de Computadores

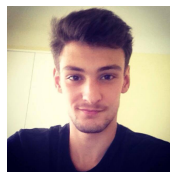
MIEI - 3º ANO - 1º SEMESTRE
UNIVERSIDADE DO MINHO

CAMADA DE LIGAÇÃO LÓGICA: ETHERNET E PROTOCOLO ARP

GRUPO 58



Dinis Peixoto
A75353



Ricardo Pereira
A74185

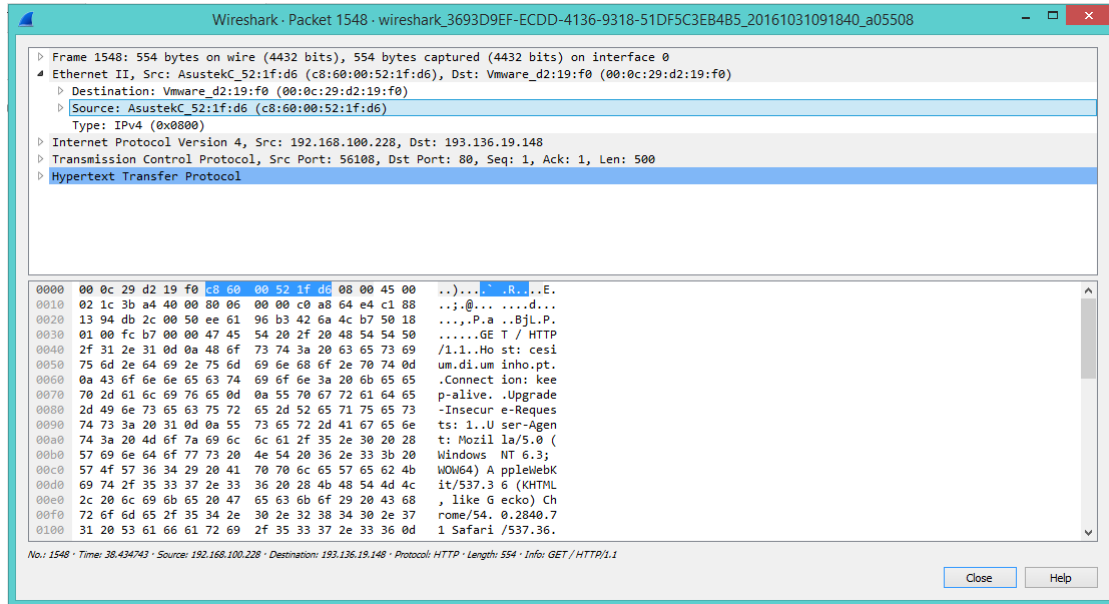


Marcelo Lima
A75210

16 de Novembro de 2017

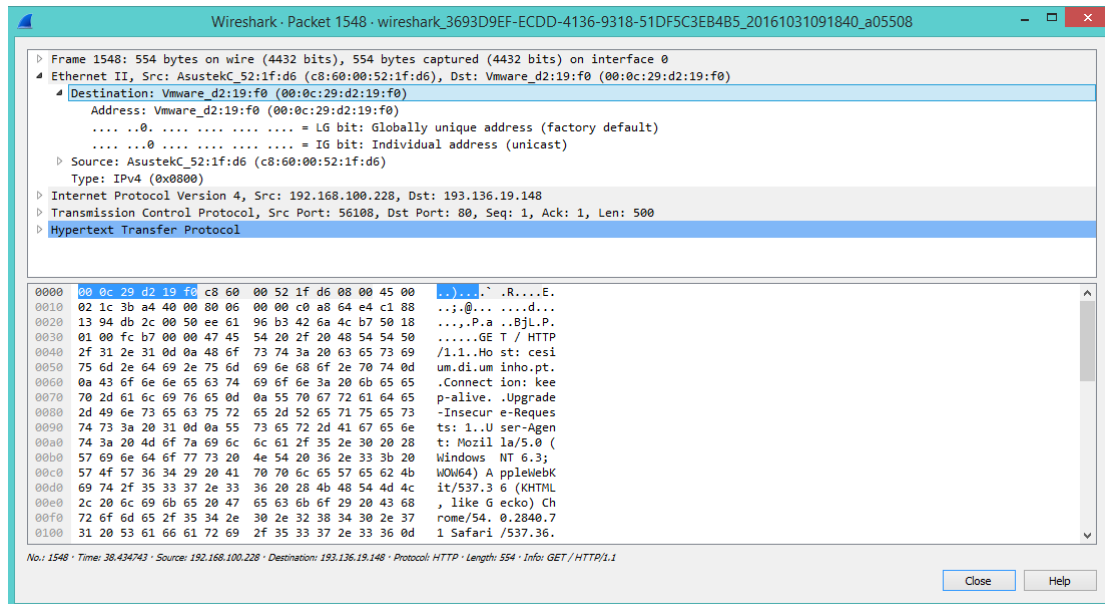
Parte I

1. Qual é o endereço MAC da interface ativa do seu computador?



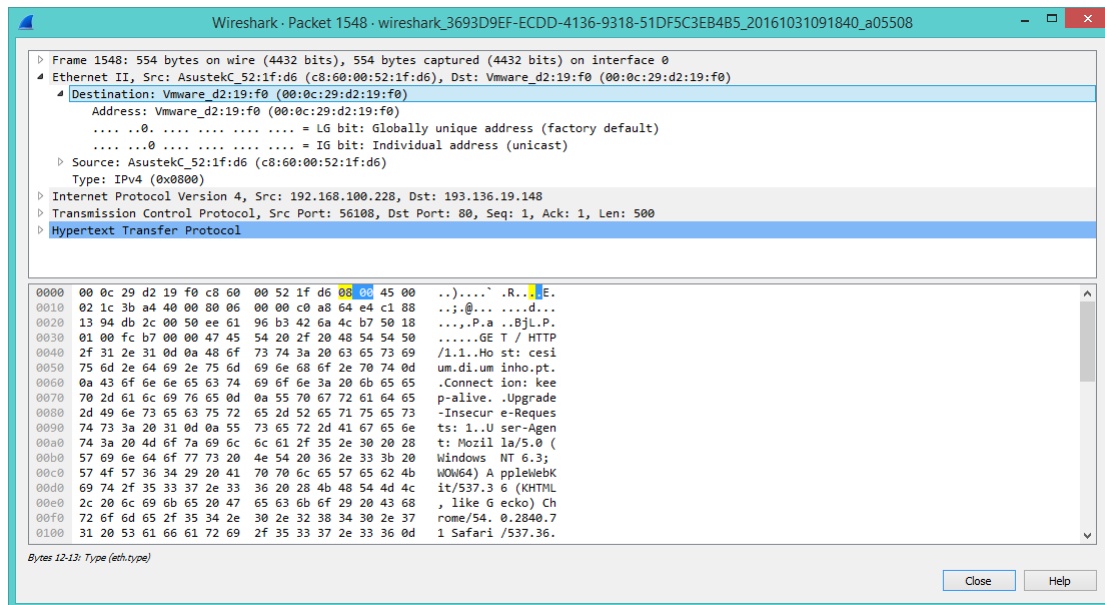
Como podemos verificar pelo *printscreen* o endereço MAC da interface ativa do computador utilizado é *c8:60:00:52:1f:d6*.

2. Qual é o endereço MAC destino da trama? A que sistema é destinada essa trama, será o endereço Ethernet do servidor `http` para `cesium.di.uminho.pt`? Justifique.



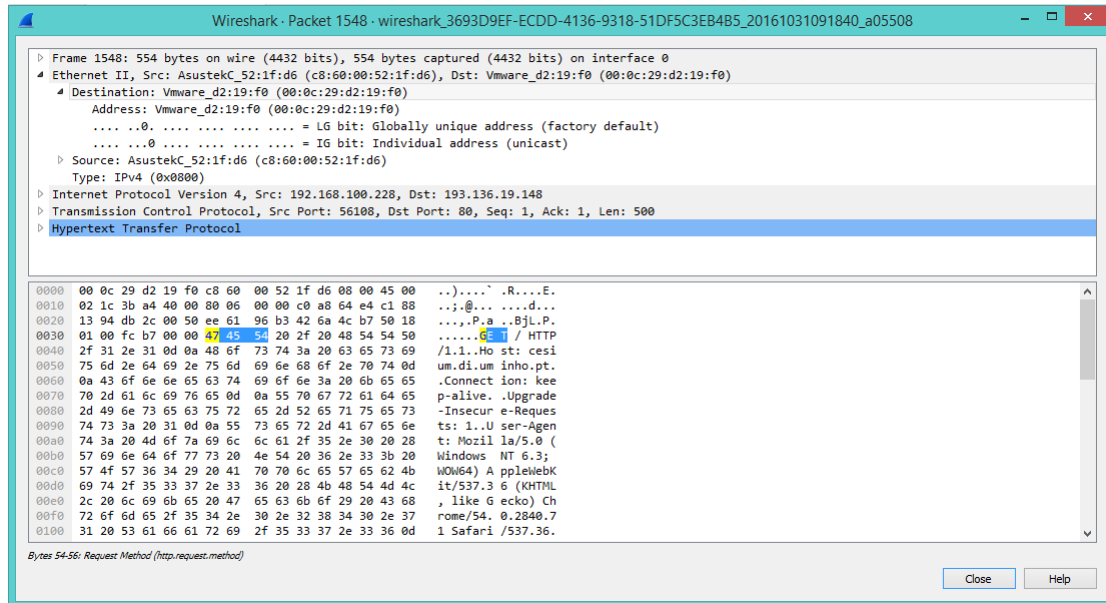
O endereço é `00:0c:29:d2:19:f0`. Este não corresponde ao endereço do servidor mas sim do router que faz a ligação com ele.

3. Qual o valor hexadecimal do campo *Type* da trama Ethernet? O que significa?



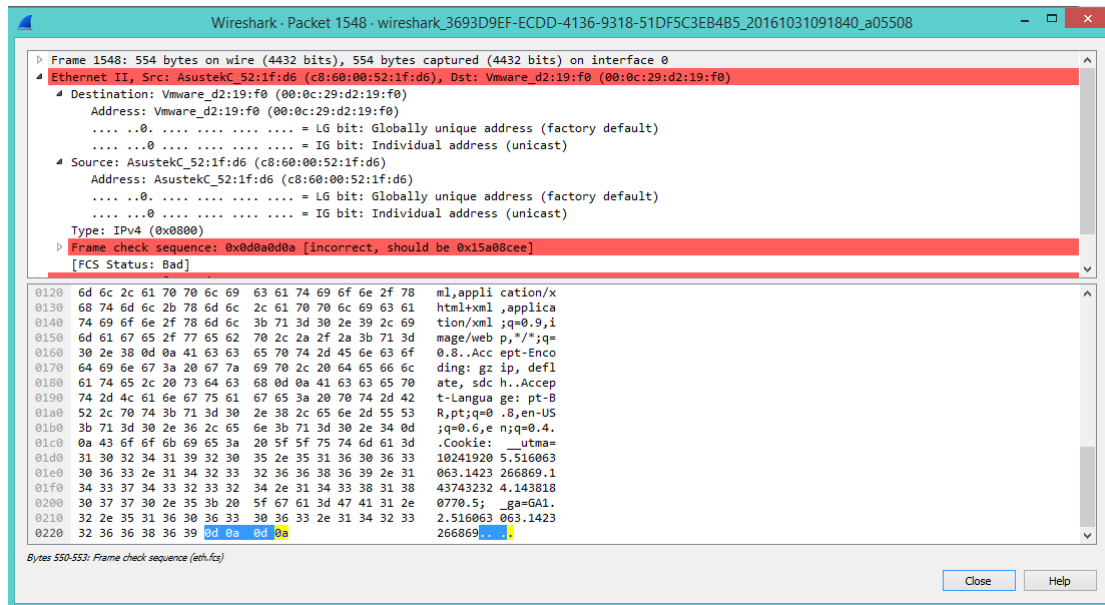
Tal como podemos verificar pelo *printscreen* o valor hexadecimal é *0x0800*, o que significa que o campo de dados são pacotes IPv4.

4. Quantos bytes são usados desde o início da trama até ao caractere ASCII "G" do método HTTP GET? Calcule e indique, em percentagem, a sobrecarga (*overhead*) introduzida pela pilha protocolar no envio do HTTP GET.



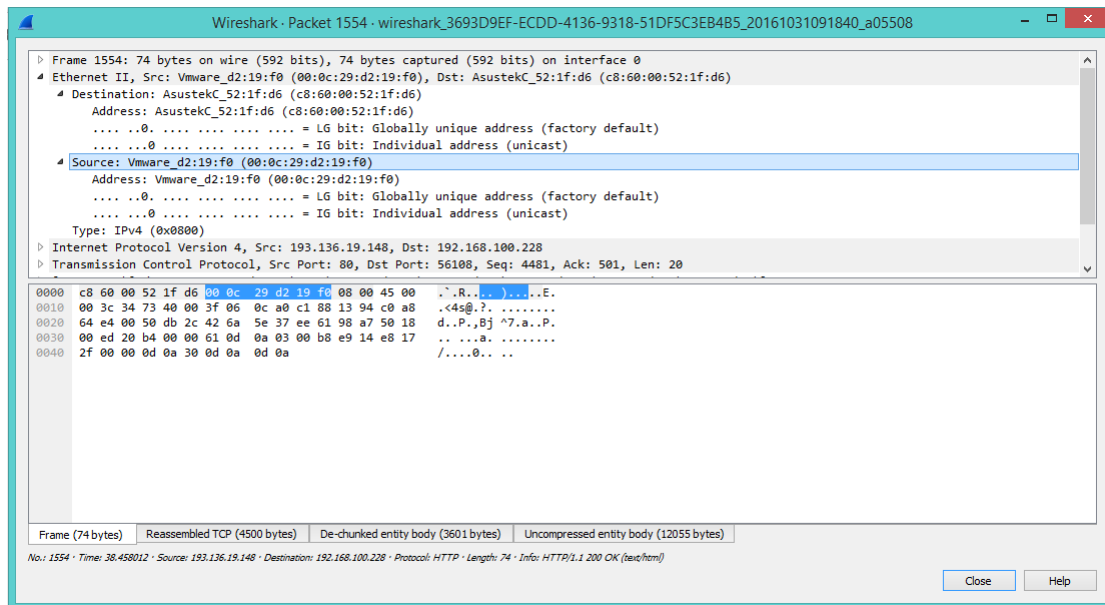
Desde o início da trama até ao caractere "G" do método HTTP GET são usados 54 bytes. A sobrecarga é: $54/553 = 9,76 \%$.

5. Em ligações com fios pouco susceptíveis a erros, nem sempre as NICs geram o código de detecção de erros. Verifique se o campo FCS está a ser utilizado. Aceda à opção Edit/Preferences/Protocols/Ethernet e indique que é assumido o uso do campo FCS. Verifique qual o valor hexadecimal desse campo na trama capturada. Que conclui? Reponha a configuração original.



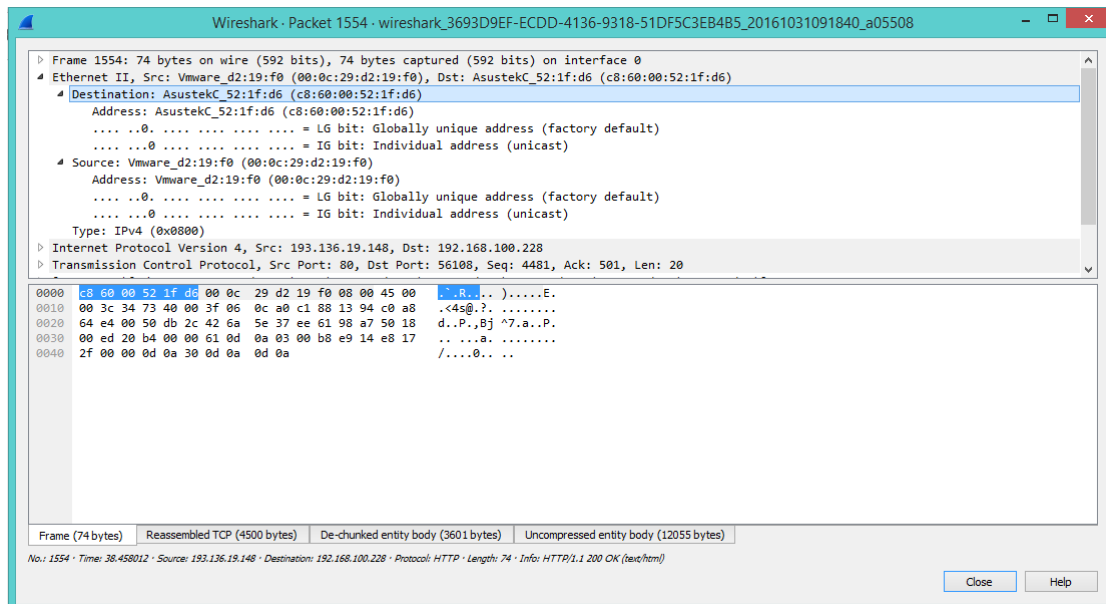
O valor do campo FCS na trama capturada é *0x0d0a0d0a*. Como na versão original não gerou os bytes do campo FCS, a trama estava correta, no entanto ao adicionar a opção para obrigar a aparecer este campo, este indica que tem erro, isso acontece porque o *Wireshark* interpretou a parte final como FCS, logo considera a trama errada quando na verdade está correta.

6. Qual é o endereço Ethernet da fonte? A que sistema de rede corresponde? Justifique.



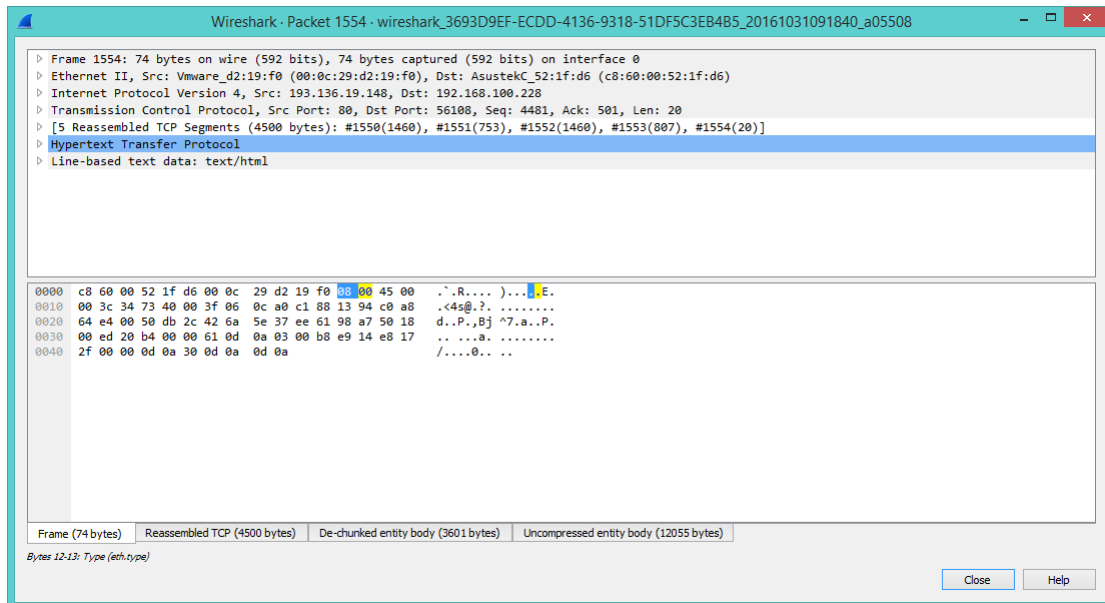
O endereço Ethernet da fonte é `00:0c:29:d2:19:f0` e corresponde ao router da rede local à qual estamos ligados.

7. Qual é o endereço MAC do destino? A que sistema corresponde?



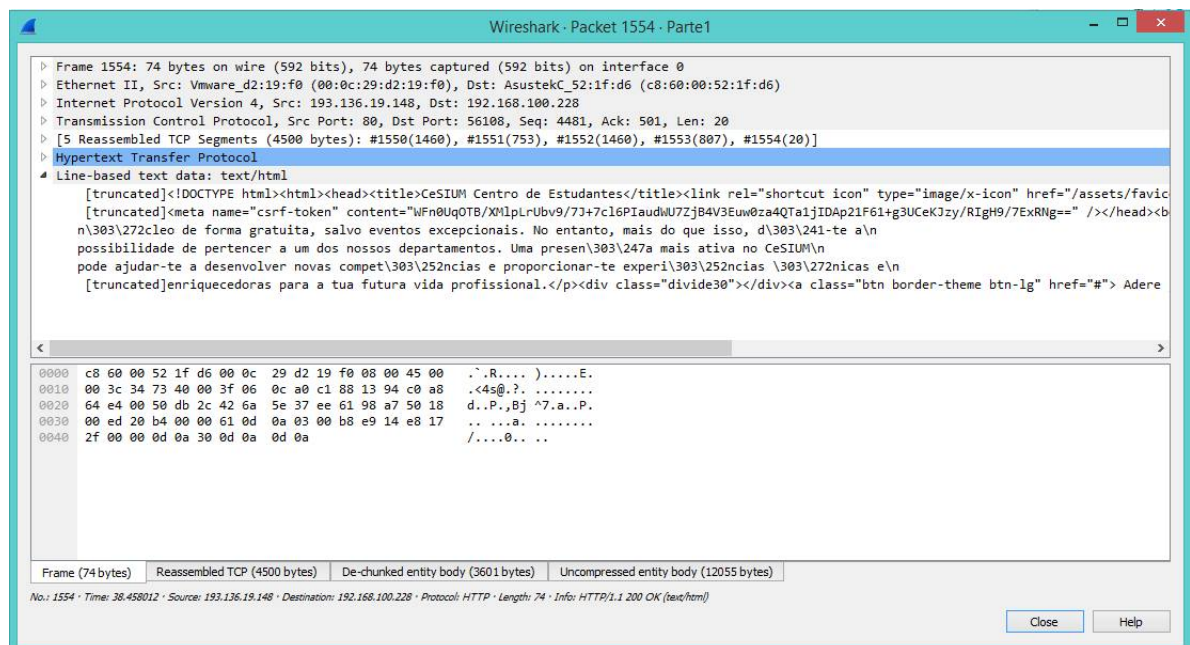
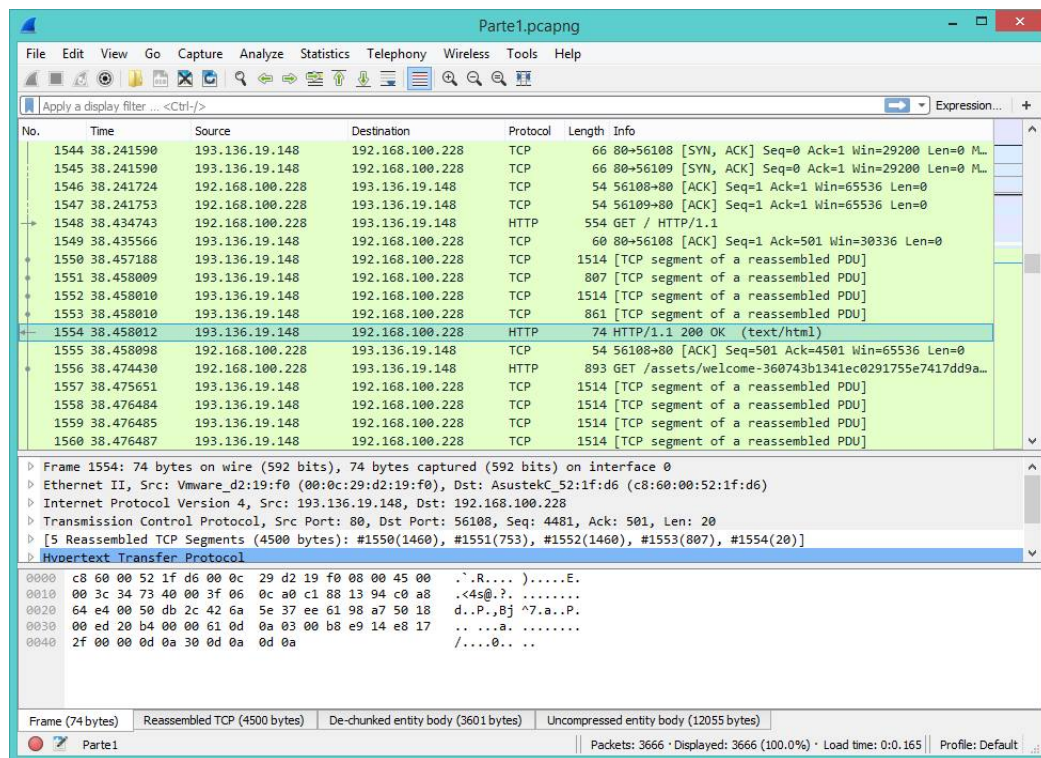
O endereço MAC destino é *c8:60:00:52:1f:d6* , que corresponde ao da nossa máquina.

8. Qual é o valor hexadecimal do campo tipo (*Type*)



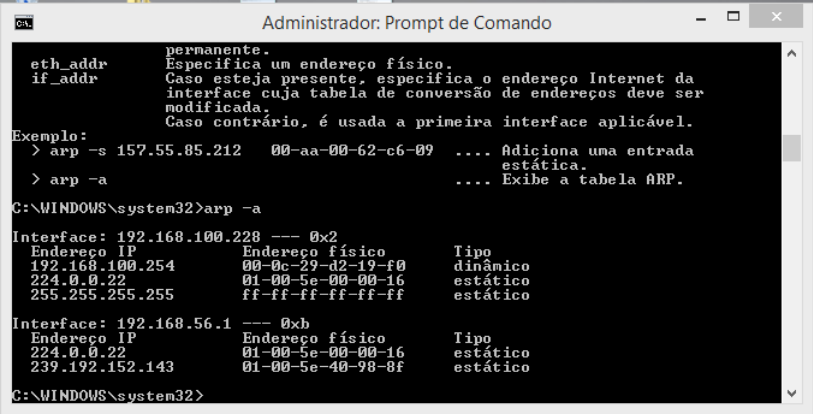
O campo tipo tem o valor *0x0800*.

9. Que tipo de resposta foi enviada pelo servidor?



É uma resposta do tipo *OK* e envia o HTML correspondente.

10. Observe o conteúdo da tabela ARP. Diga o que significa cada uma das colunas?



```
eth_addr      permanente.
if_addr       Especifica um endereço físico.
               Caso esteja presente, especifica o endereço Internet da
               interface cuja tabela de conversão de endereços deve ser
               modificada.
               Caso contrário, é usada a primeira interface aplicável.
Exemplo:
> arp -s 157.55.85.212 00-aa-00-62-c6-09 .... Adiciona uma entrada
               estática.
> arp -a      .... Exibe a tabela ARP.

C:\WINDOWS\system32>arp -a

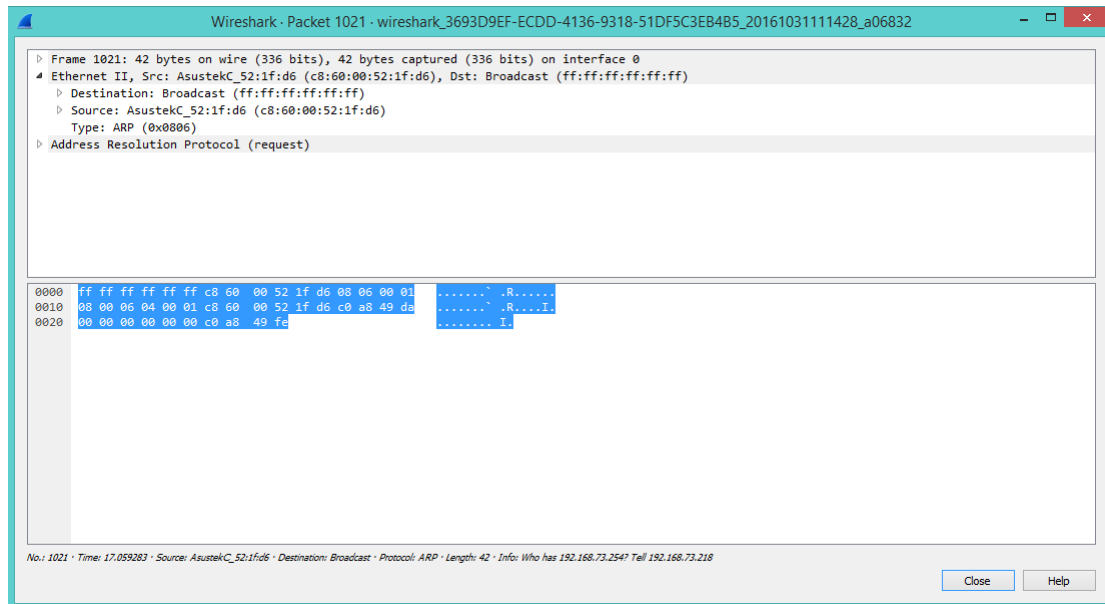
Interface: 192.168.100.228 --- 0x2
Endereço IP      Endereço físico      Tipo
192.168.100.254   00-0c-29-d2-19-f0   dinâmico
224.0.0.22        01-00-5e-00-00-16   estático
255.255.255.255   ff-ff-ff-ff-ff-ff   estático

Interface: 192.168.56.1 --- 0xb
Endereço IP      Endereço físico      Tipo
224.0.0.22        01-00-5e-00-00-16   estático
239.192.152.143   01-00-5e-48-98-8f   estático

C:\WINDOWS\system32>
```

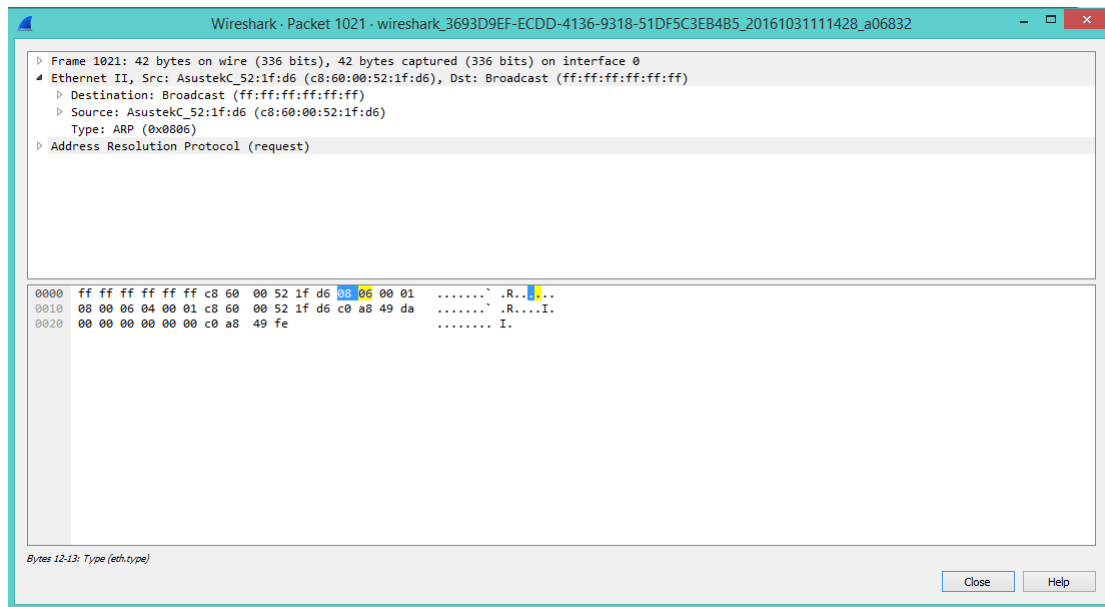
A primeira coluna apresenta o endereço IP, enquanto que a segunda apresenta o endereço MAC. Por fim, a terceira coluna apresenta o tipo. O comando *arp* exibe a tabela de conversão que mapeia o endereço IP para o endereço MAC dos sistemas que se comunicaram recentemente.

11. Qual é o valor hexadecimal dos endereços origem e destino na trama Ethernet que contém a mensagem com o pedido ARP (*ARP Request*)? Como interpreta e justifica o endereço destino usado?



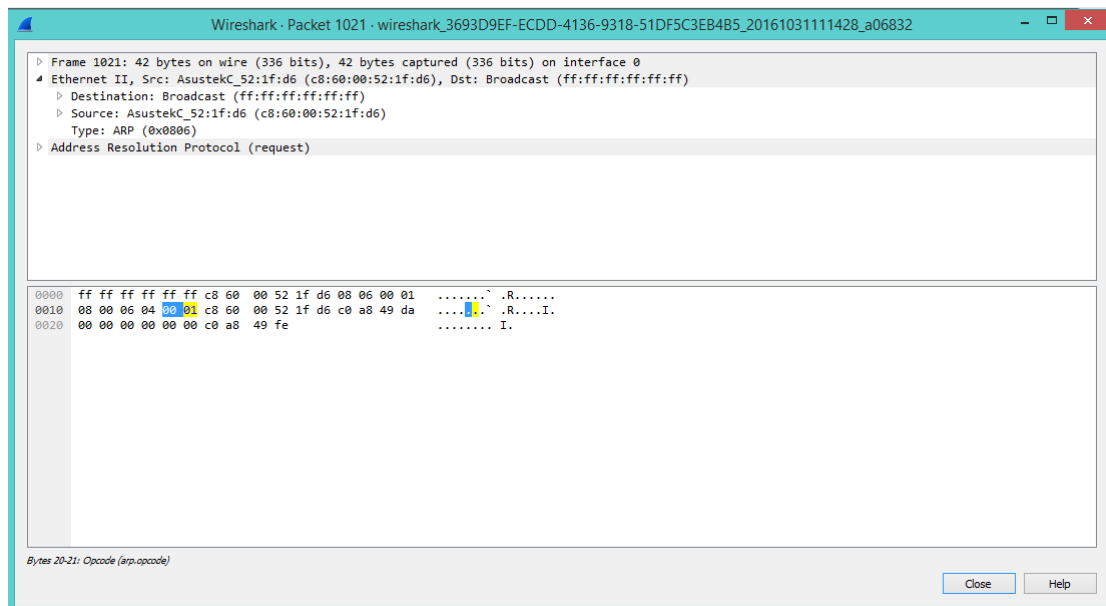
O endereço de origem é *c8:60:00:52:1f:d6* e o endereço de destino é *ff:ff:ff:ff:ff:ff*. Este endereço de destino implica que todos os endereços ligados à rede recebem o pedido, no entanto só o endereço pretendido é que responde com o endereço físico.

12. Qual o valor hexadecimal do campo tipo da trama Ethernet? O que indica?



O valor hexadecimal do campo tipo da trama é *0x0806*, indicando assim que o campo de dados pertence ao ARP.

13. Qual o valor do campo ARP *opcode*? O que especifica?



O valor do *opcode* é *0x0001*. O *opcode* determina se é um pedido ou uma resposta a um pedido, neste caso indica que é um pedido.

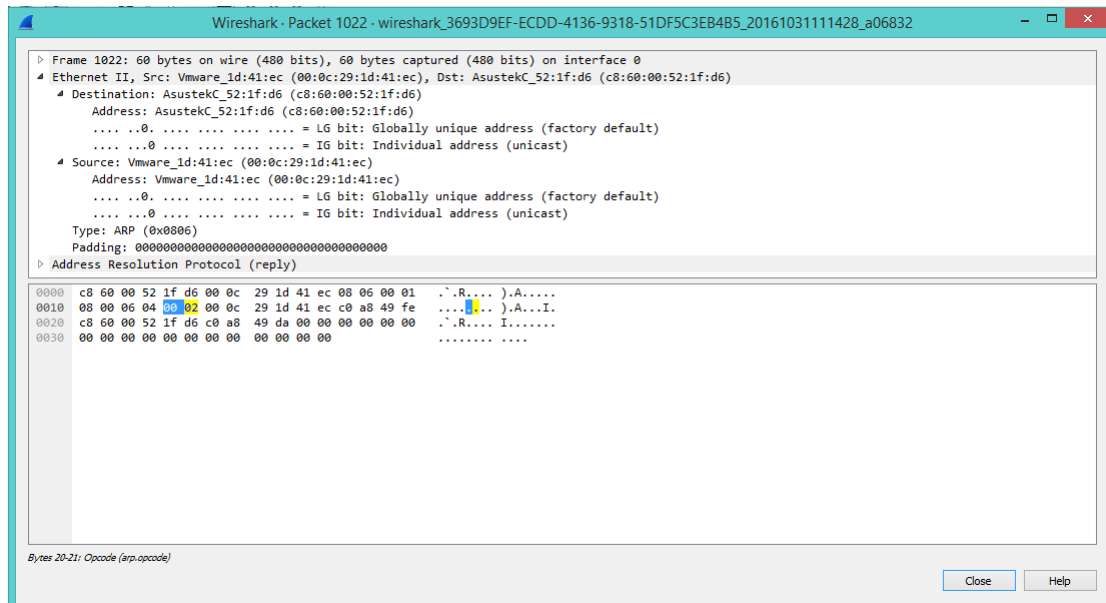
14. A mensagem ARP contém o endereço IP de origem? Que tipo de pergunta é feita?

1018	16.412349	AsustekC_52:1f:d6	Vmware_1d:41:ec	0x0800	54	IPv4
1019	16.414475	AsustekC_52:1f:d6	Vmware_1d:41:ec	0x0800	100	IPv4
1020	16.472179	Vmware_1d:41:ec	AsustekC_52:1f:d6	0x0800	60	IPv4
1021	17.059283	AsustekC_52:1f:d6	Broadcast	ARP	42	Who has 192.168.73.254? Tell 192.168.73.218
1022	17.060146	Vmware_1d:41:ec	AsustekC_52:1f:d6	ARP	60	192.168.73.254 is at 00:0c:29:1d:41:ec
1023	18.000953	CiscoInc_73:97:10	Spanning-tree-(for-...	STP	60	Conf. Root = 4096/731/00:0a:8a:97:74:80 Cost = 306
1024	18.349503	AsustekC_52:1f:d6	Vmware_1d:41:ec	0x0800	82	IPv4
.....

Não, só contém o endereço físico. A mensagem pede o endereço físico correspondente aquele endereço lógico.

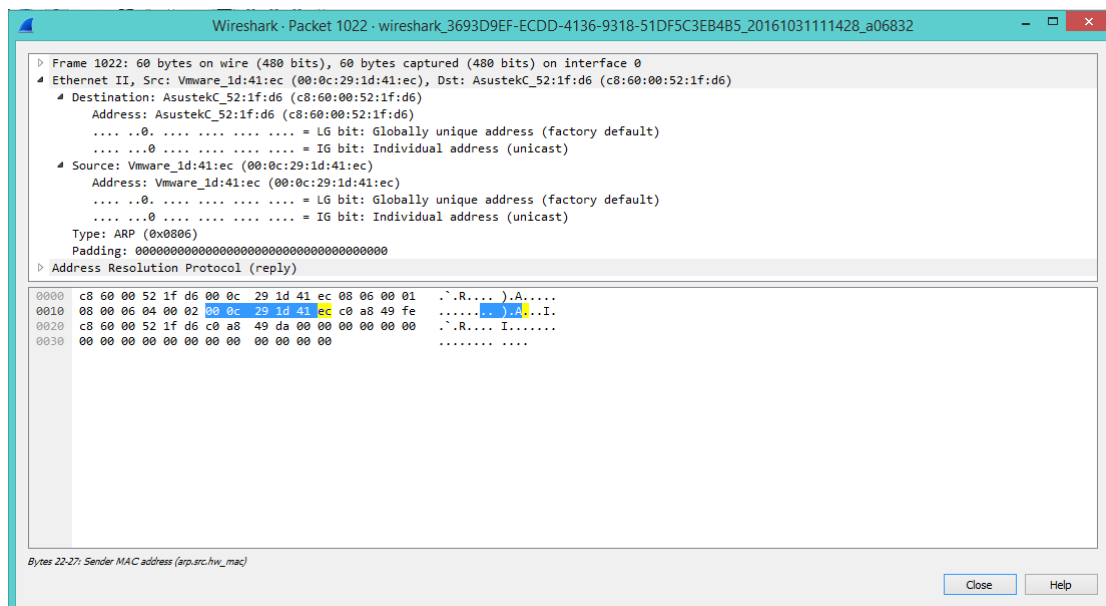
15. Localize a mensagem ARP que é a resposta ao pedido ARP efetuado.

a. Qual o valor do campo ARP opcode? O que especifica?



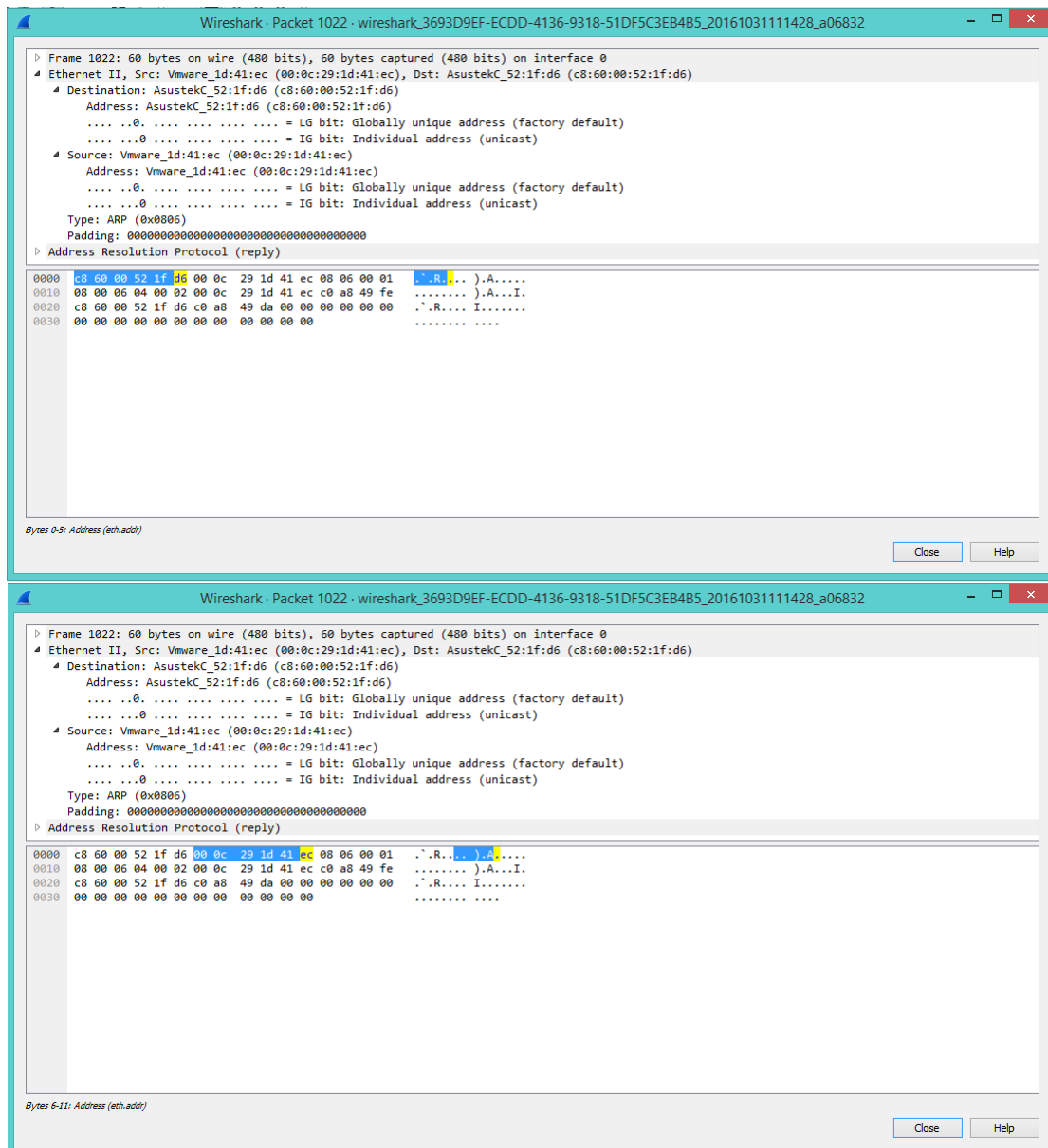
O opcode é *0x0002*, que significa que é a resposta ao pedido.

b. Em que posição da mensagem ARP está a resposta ao pedido ARP?



Está na posição 22-27 bytes, e corresponde a *00:0c:29:1d:41:ec*.

16. Quais são os valores hexadecimais para os endereços origem e destino da trama que contém a resposta ARP? Que conclui?



Concluimos que o destino é igual à mensagem enviada.

17. Com o auxílio do comando *ifconfig* obtenha os endereços Ethernet das interfaces dos diversos routers.

```
root@n1:/tmp/pycore.55388/n1.conf# ifconfig
eth0    Link encap:Ethernet  HWaddr 00:00:00:aa:00:00
        inet addr:10.0.0.1  Bcast:0.0.0.0  Mask:255.255.255.0
        inet6 addr: fe80::200:fff:feaa:0/64 Scope:Link
        inet6 addr: 2001::1/64 Scope:Global
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:90 errors:0 dropped:0 overruns:0 frame:0
        TX packets:55 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:1302 (13.0 KB)  TX bytes:5066 (5.0 KB)

lo      Link encap:Local Loopback
        inet addr:127.0.0.1  Mask:255.0.0.0
        inet6 addr: ::1/128 Scope:Host
        UP LOOPBACK RUNNING  MTU:16436  Metric:1
        RX packets:0 errors:0 dropped:0 overruns:0 frame:0
        TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:0
        RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

root@n1:/tmp/pycore.55388/n1.conf#
```

```
root@n2:/tmp/pycore.55388/n2.conf# ifconfig
eth0    Link encap:Ethernet  HWaddr 00:00:00:aa:00:01
        inet addr:10.0.0.2  Bcast:0.0.0.0  Mask:255.255.255.0
        inet6 addr: fe80::200:fff:feaa:1/64 Scope:Link
        inet6 addr: 2001::2/64 Scope:Global
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:91 errors:0 dropped:0 overruns:0 frame:0
        TX packets:64 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:12822 (12.8 KB)  TX bytes:6172 (6.1 KB)

eth1    Link encap:Ethernet  HWaddr 00:00:00:aa:00:02
        inet addr:10.0.0.11 Bcast:0.0.0.0  Mask:255.255.255.0
        inet6 addr: fe80::200:fff:feaa:2/64 Scope:Link
        inet6 addr: 2001::11/64 Scope:Global
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:90 errors:0 dropped:0 overruns:0 frame:0
        TX packets:62 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:12876 (12.8 KB)  TX bytes:6060 (6.0 KB)

lo      Link encap:Local Loopback
        inet addr:127.0.0.1  Mask:255.0.0.0
        inet6 addr: ::1/128 Scope:Host
        UP LOOPBACK RUNNING  MTU:16436  Metric:1
        RX packets:4 errors:0 dropped:0 overruns:0 frame:0
        TX packets:4 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:0
        RX bytes:324 (324.0 B)  TX bytes:324 (324.0 B)

root@n2:/tmp/pycore.55388/n2.conf#
```

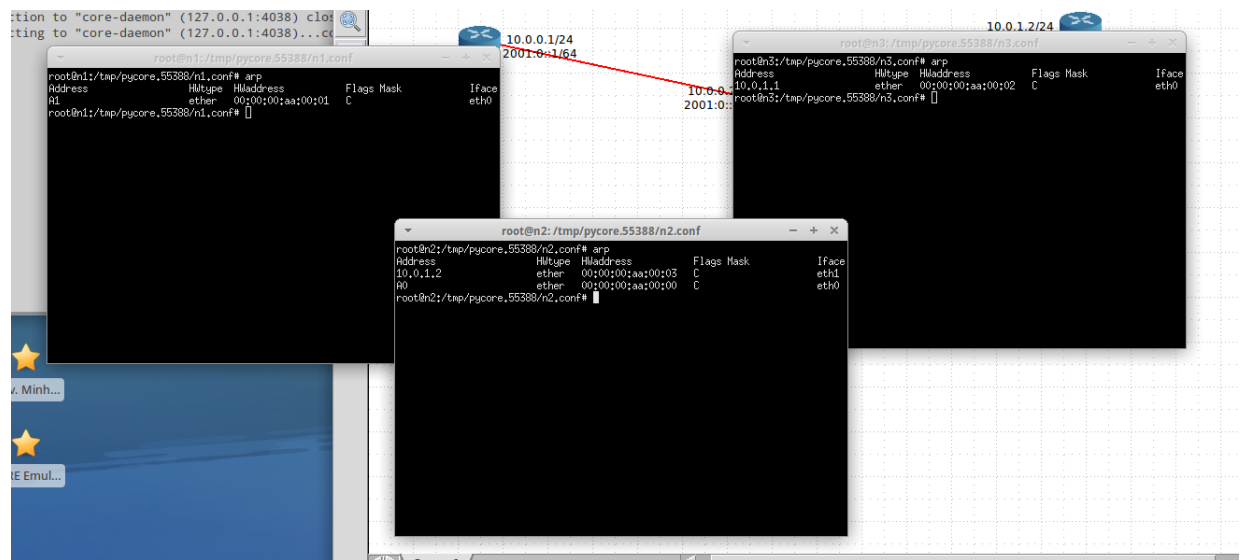
```
root@n3:/tmp/pycore.55388/n3.conf# ifconfig
eth0    Link encap:Ethernet  HWaddr 00:00:00:aa:00:03
        inet addr:10.0.0.3  Bcast:0.0.0.0  Mask:255.255.255.0
        inet6 addr: fe80::200:fff:feaa:3/64 Scope:Link
        inet6 addr: 2001::3/64 Scope:Global
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:86 errors:0 dropped:0 overruns:0 frame:0
        TX packets:53 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:12760 (12.7 KB)  TX bytes:5010 (5.0 KB)

lo      Link encap:Local Loopback
        inet addr:127.0.0.1  Mask:255.0.0.0
        inet6 addr: ::1/128 Scope:Host
        UP LOOPBACK RUNNING  MTU:16436  Metric:1
        RX packets:4 errors:0 dropped:0 overruns:0 frame:0
        TX packets:4 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:0
        RX bytes:324 (324.0 B)  TX bytes:324 (324.0 B)

root@n3:/tmp/pycore.55388/n3.conf#
```

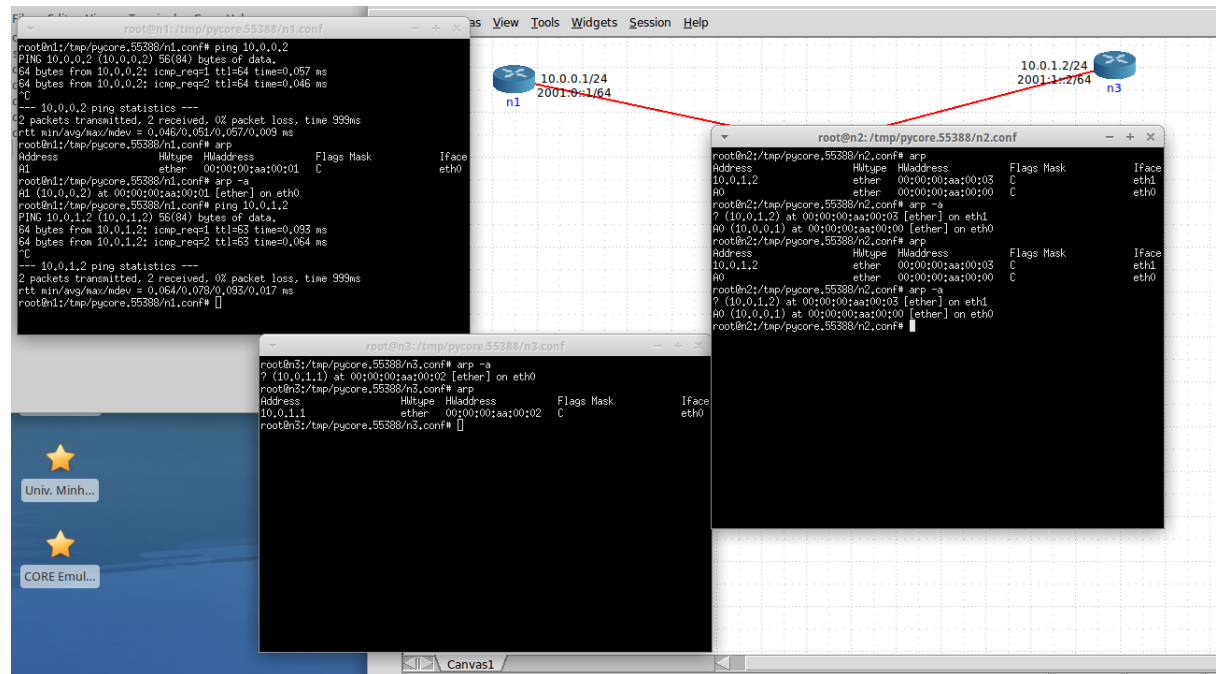
Com o comando *ifconfig* obtemos os endereços mostrados no *printscreen*.

18. Usando o comando *arp* obtenha as caches *arp* dos diversos sistemas.



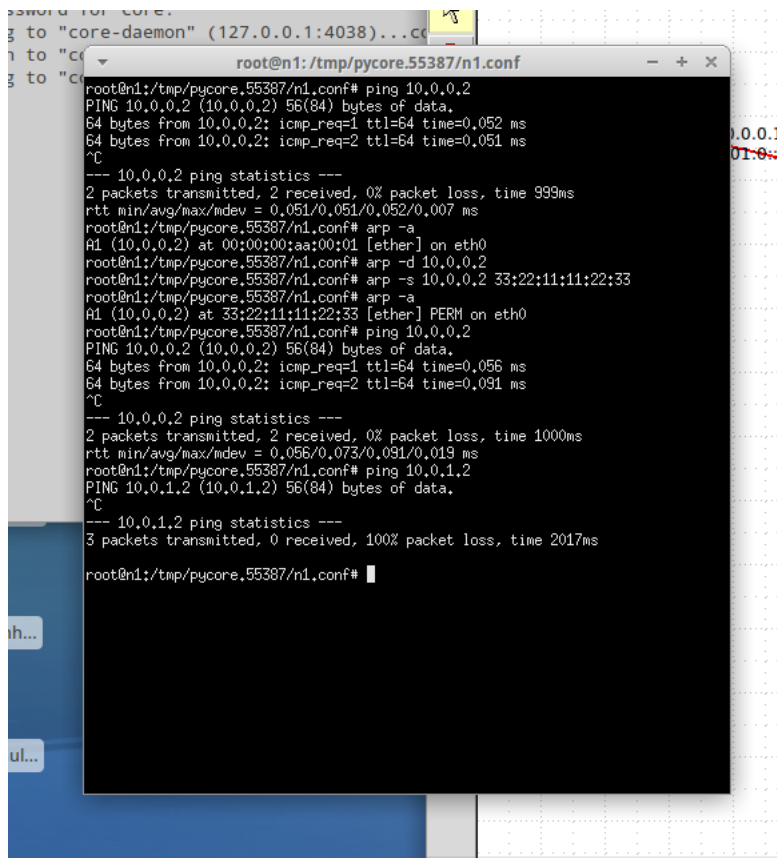
Usando o comando *arp* obtivemos as caches apresentadas no *printscreen* anterior.

19. Faça *ping* de n1 para n2. Que modificações observa nas caches ARP desses sistemas? Faça *ping* de n1 para n3. Consulte as caches ARP. Que conclui?



No fim de fazer ping do nodo n1 para o nodo n2, as caches ARP desses sistemas mantiveram-se inalteráveis, como é visível nos *printscreens* anteriores. O nodo n1, manteve o registo do endereço IP e MAC, do nodo n2. O nodo n2, manteve o registo dos endereços IP e MAC, dos nodos n1 e n3. Por fim, o nodo n3 manteve o registo do endereço IP e MAC, do nodo n2.

20. Em n1 remova a entrada correspondente a n2. Coloque uma nova entrada para n2 com o endereço Ethernet inexistente. O que acontece?

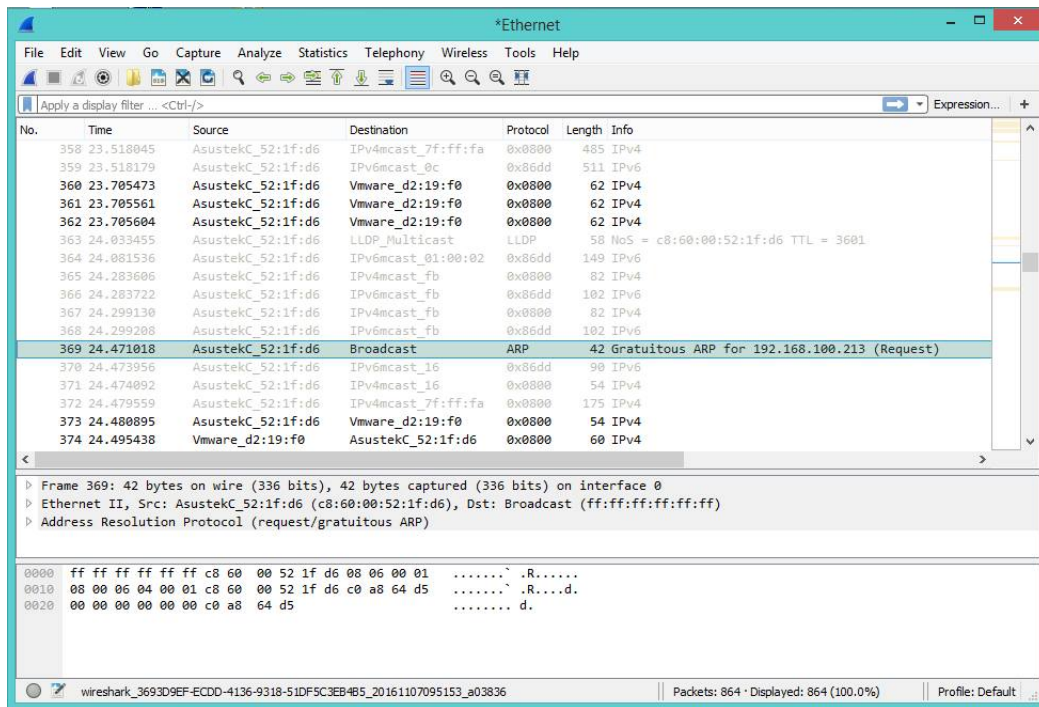


```
root@n1:/tmp/pycore.55387/n1.conf# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_req=1 ttl=64 time=0.052 ms
64 bytes from 10.0.0.2: icmp_req=2 ttl=64 time=0.051 ms
^C
--- 10.0.0.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.051/0.051/0.052/0.007 ms
root@n1:/tmp/pycore.55387/n1.conf# arp -a
A1 (10.0.0.2) at 00:00:00:aa:00:01 [ether] on eth0
root@n1:/tmp/pycore.55387/n1.conf# arp -d 10.0.0.2
root@n1:/tmp/pycore.55387/n1.conf# arp -s 10.0.0.2 33:22:11:11:22:33
root@n1:/tmp/pycore.55387/n1.conf# arp -a
A1 (10.0.0.2) at 33:22:11:11:22:33 [ether] PERM on eth0
root@n1:/tmp/pycore.55387/n1.conf# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_req=1 ttl=64 time=0.056 ms
64 bytes from 10.0.0.2: icmp_req=2 ttl=64 time=0.031 ms
^C
--- 10.0.0.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 0.056/0.073/0.091/0.019 ms
root@n1:/tmp/pycore.55387/n1.conf# ping 10.0.1.2
PING 10.0.1.2 (10.0.1.2) 56(84) bytes of data.
^C
--- 10.0.1.2 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2017ms
root@n1:/tmp/pycore.55387/n1.conf#
```

Como é visível, inicialmente com os endereços correctos, é possível fazer ping do nodo n1 para o nodo n2, sem problema. No entanto, quando alterámos o endereço MAC na tabela arp do nodo n1, neste caso para *33:22:11:11:22:33*, o que deu origem a outra situação. Ao fim de alterar o endereço MAC, ainda era possível fazer ping do nodo n1 para o n2, o que não era esperado. A razão para tal, deve-se ao facto do sistema ser virtual e guardar o endereço original, mesmo depois de a alterarmos. No entanto, quando tentamos fazer ping para o nodo n3, já não é possível a conexão, necessita de aceder primeiro ao nodo n2, e como os endereços MAC são diferentes, não deixa proseguir para o nodo n3. Desta forma verificamos que com os endereços MAC diferentes, não é possível haver conexão.

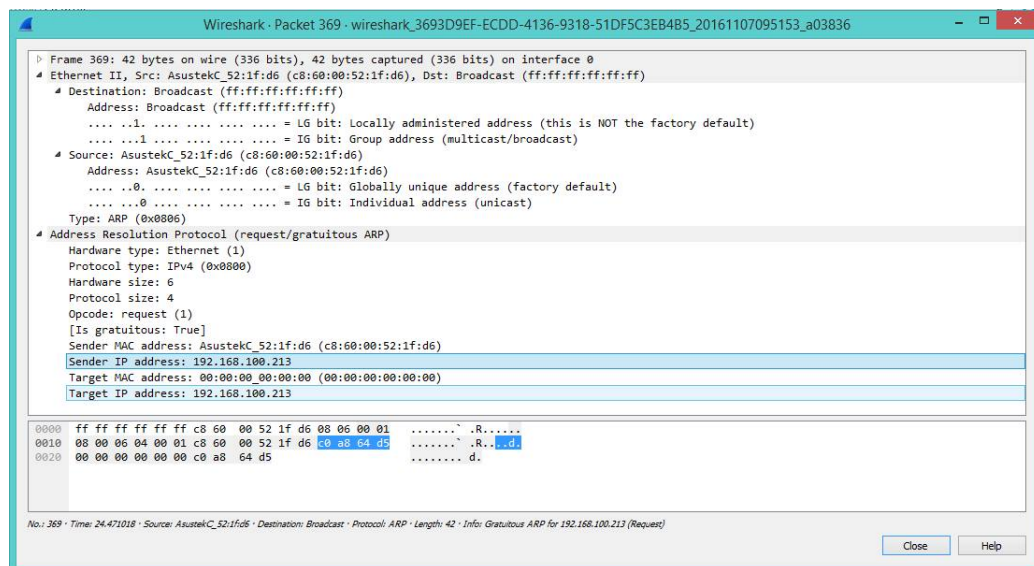
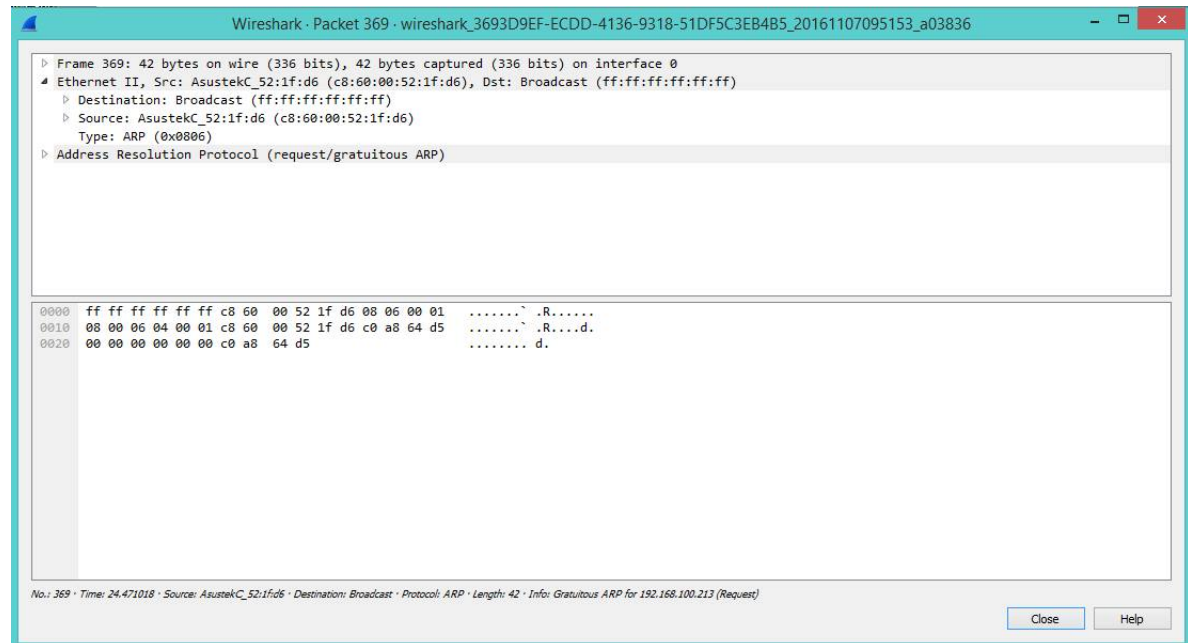
Parte II

1. Identifique um pacote de pedido ARP gratuito originado pelo seu sistema. Verifique quantos pacotes ARP gratuito foram enviados e com que intervalo temporal?



Apenas foi enviado um pacote com ARP gratuito.

2. Analise o conteúdo de um pedido ARP gratuito e identifique em que se distingue dos restantes pedidos ARP. Registo a trama Ethernet correspondente. Qual o resultado esperado face ao pedido ARP gratuito enviado?



O conteúdo do pedido ARP gratuito distingue-se dos restantes pedidos ARP pelo facto do IP origem ser igual ao IP destino. Como o objectivo do pedido ARP gratuito é verificar se existe alguém com o mesmo endereço IP da fonte, a resposta que é esperada obter é não obter resposta, pois isso significa que não existe ninguém com o seu IP, não havendo assim conflitos.

3. Faça *ping* de n1 para n2. Verifique com a opção *tcpdump* como flui o tráfego nas diversas interfaces dos vários dispositivos. Que conclui?

```

root@n2:/tmp/pycore.55392/n2.conf
~v~
^C11:35:42,801369 ARP, Request who-has A9 tell 10.0.0.20, length 28
11:35:42,801403 ARP, Reply A9 is-at 00:00:00:aa:00:01 (oui Ethernet), length 28
11:35:42,801418 IP 10.0.0.20 > A9: ICMP echo request, id 72, seq 1, length 64
11:35:42,801427 IP A9 > 10.0.0.20: ICMP echo reply, id 72, seq 1, length 64
11:35:43,803617 IP 10.0.0.20 > A9: ICMP echo request, id 72, seq 2, length 64
11:35:43,803634 IP A9 > 10.0.0.20: ICMP echo reply, id 72, seq 2, length 64
11:35:44,802935 IP 10.0.0.20 > A9: ICMP echo request, id 72, seq 3, length 64
11:35:44,802966 IP A9 > 10.0.0.20: ICMP echo reply, id 72, seq 3, length 64
11:35:45,802607 IP 10.0.0.20 > A9: ICMP echo request, id 72, seq 4, length 64
11:35:45,802631 IP A9 > 10.0.0.20: ICMP echo reply, id 72, seq 4, length 64
11:35:46,802598 IP 10.0.0.20 > A9: ICMP echo request, id 72, seq 5, length 64
11:35:46,802583 IP A9 > 10.0.0.20: ICMP echo reply, id 72, seq 5, length 64
11:35:47,802499 IP 10.0.0.20 > A9: ICMP echo request, id 72, seq 6, length 64
11:35:47,802523 IP A9 > 10.0.0.20: ICMP echo reply, id 72, seq 6, length 64
11:35:47,811228 ARP, Request who-has 10.0.0.20 tell A9, length 28
11:35:47,811295 ARP, Reply 10.0.0.20 is-at 00:00:00:aa:00:00 (oui Ethernet), len
gth 28

16 packets captured
16 packets received by filter
0 packets dropped by kernel
root@n2:/tmp/pycore.55392/n2.conf#

root@n1:/tmp/pycore.55392/n1.conf
root@n1:/tmp/pycore.55392/n1.conf# ping 10.0.0.10
PING 10.0.0.10 (10.0.0.10) 56(84) bytes of data:
64 bytes from 10.0.0.10: icmp_req=1 ttl=64 time=0.097 ms
64 bytes from 10.0.0.10: icmp_req=2 ttl=64 time=0.070 ms
64 bytes from 10.0.0.10: icmp_req=3 ttl=64 time=0.085 ms
64 bytes from 10.0.0.10: icmp_req=4 ttl=64 time=0.076 ms
64 bytes from 10.0.0.10: icmp_req=5 ttl=64 time=0.077 ms
64 bytes from 10.0.0.10: icmp_req=6 ttl=64 time=0.076 ms
^C
--- 10.0.0.10 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5001ms
rtt min/avg/max/mdev = 0.070/0.080/0.097/0.010 ms
root@n1:/tmp/pycore.55392/n1.conf#

root@n3:/tmp/pycore.55392/n3.conf
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
^C11:35:42,801367 ARP, Request who-has A9 tell 10.0.0.20, length 28
11:35:42,801417 IP 10.0.0.20 > A9: ICMP echo request, id 72, seq 1, length 64
11:35:42,801431 IP A9 > 10.0.0.20: ICMP echo reply, id 72, seq 1, length 64
11:35:43,803614 IP 10.0.0.20 > A9: ICMP echo request, id 72, seq 2, length 64
11:35:43,803640 IP A9 > 10.0.0.20: ICMP echo reply, id 72, seq 2, length 64
11:35:44,802932 IP 10.0.0.20 > A9: ICMP echo request, id 72, seq 3, length 64
11:35:44,802973 IP A9 > 10.0.0.20: ICMP echo reply, id 72, seq 3, length 64
11:35:45,802604 IP 10.0.0.20 > A9: ICMP echo request, id 72, seq 4, length 64
11:35:45,802637 IP A9 > 10.0.0.20: ICMP echo reply, id 72, seq 4, length 64
11:35:46,802596 IP 10.0.0.20 > A9: ICMP echo request, id 72, seq 5, length 64
11:35:46,802599 IP A9 > 10.0.0.20: ICMP echo reply, id 72, seq 5, length 64
11:35:47,802497 IP 10.0.0.20 > A9: ICMP echo request, id 72, seq 6, length 64
11:35:47,802523 IP A9 > 10.0.0.20: ICMP echo reply, id 72, seq 6, length 64
11:35:47,811272 ARP, Request who-has 10.0.0.20 tell A9, length 28
11:35:47,811294 ARP, Reply 10.0.0.20 is-at 00:00:00:aa:00:00 (oui Ethernet), len
gth 28

16 packets captured
16 packets received by filter
0 packets dropped by kernel
root@n3:/tmp/pycore.55392/n3.conf#

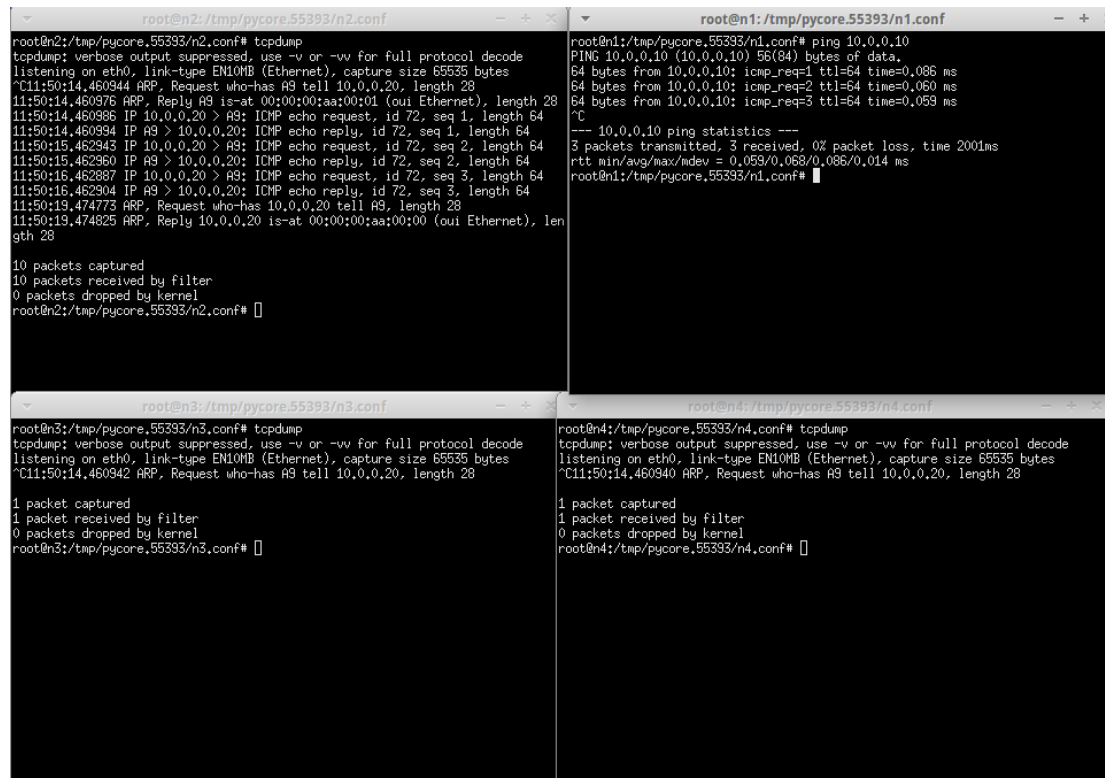
root@n4:/tmp/pycore.55392/n4.conf
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
^C11:35:42,801365 ARP, Request who-has A9 tell 10.0.0.20, length 28
11:35:42,801416 IP 10.0.0.20 > A9: ICMP echo request, id 72, seq 1, length 64
11:35:42,801430 IP A9 > 10.0.0.20: ICMP echo reply, id 72, seq 1, length 64
11:35:43,803610 IP 10.0.0.20 > A9: ICMP echo request, id 72, seq 2, length 64
11:35:43,803638 IP A9 > 10.0.0.20: ICMP echo reply, id 72, seq 2, length 64
11:35:44,802929 IP 10.0.0.20 > A9: ICMP echo request, id 72, seq 3, length 64
11:35:44,802971 IP A9 > 10.0.0.20: ICMP echo reply, id 72, seq 3, length 64
11:35:45,802601 IP 10.0.0.20 > A9: ICMP echo request, id 72, seq 4, length 64
11:35:45,802636 IP A9 > 10.0.0.20: ICMP echo reply, id 72, seq 4, length 64
11:35:46,802552 IP 10.0.0.20 > A9: ICMP echo request, id 72, seq 5, length 64
11:35:46,802587 IP A9 > 10.0.0.20: ICMP echo reply, id 72, seq 5, length 64
11:35:47,802493 IP 10.0.0.20 > A9: ICMP echo request, id 72, seq 6, length 64
11:35:47,802528 IP A9 > 10.0.0.20: ICMP echo reply, id 72, seq 6, length 64
11:35:47,811269 ARP, Request who-has 10.0.0.20 tell A9, length 28
11:35:47,811293 ARP, Reply 10.0.0.20 is-at 00:00:00:aa:00:00 (oui Ethernet), len
gth 28

16 packets captured
16 packets received by filter
0 packets dropped by kernel
root@n4:/tmp/pycore.55392/n4.conf#

```

Quando é feito ping do laptop n1 para o host n2, a mensagem é transmitida do n1, para o hub. Os hubs são dispositivos de interligação que operam a nível físico, isto é, repetem o sinal que chega através de uma porta de entrada para todas as outras portas. Assim, quando o hub recebe a mensagem do laptop n1, este repete para os hosts n2, n3 e n4, sendo por essa razão, visível com o comando *tcpdump*, que todos os hosts receberam varias mensagens (request e reply), no entanto a mensagem era apenas para o host n2.

4. Na topologia de rede substitua o *hub* por um *switch*. Repita os procedimentos que realizou na pergunta anterior. Comente os resultados obtidos quanto à utilização de *hubs* e *switches* no contexto de controlar ou dividir domínios de colisão. Documente as suas observações e conclusões com base no tráfego observado/capturado.



```
root@n2:/tmp/pycore.55393/n2.conf# tcpdump
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
^C11:50:14.460944 ARP, Request who-has A9 tell 10.0.0.20, length 28
11:50:14.460976 ARP, Reply A9 is-at 00:00:00:aa:00:01 (oui Ethernet), length 28
11:50:14.460986 IP 10.0.0.20 > A9: ICMP echo request, id 72, seq 1, length 64
11:50:14.460994 IP A9 > 10.0.0.20: ICMP echo reply, id 72, seq 1, length 64
11:50:15.462943 IP 10.0.0.20 > A9: ICMP echo request, id 72, seq 2, length 64
11:50:15.462960 IP A9 > 10.0.0.20: ICMP echo reply, id 72, seq 2, length 64
11:50:16.462887 IP 10.0.0.20 > A9: ICMP echo request, id 72, seq 3, length 64
11:50:16.462904 IP A9 > 10.0.0.20: ICMP echo reply, id 72, seq 3, length 64
11:50:19.474773 ARP, Request who-has 10.0.0.20 tell A9, length 28
11:50:19.474825 ARP, Reply 10.0.0.20 is-at 00:00:00:aa:00:00 (oui Ethernet), length 28

10 packets captured
10 packets received by filter
0 packets dropped by kernel
root@n2:/tmp/pycore.55393/n2.conf# []

root@n1:/tmp/pycore.55393/n1.conf# ping 10.0.0.10
PING 10.0.0.10 (10.0.0.10) 56(84) bytes of data:
64 bytes from 10.0.0.10: icmp_req=1 ttl=64 time=0.086 ms
64 bytes from 10.0.0.10: icmp_req=2 ttl=64 time=0.060 ms
64 bytes from 10.0.0.10: icmp_req=3 ttl=64 time=0.059 ms
^C
--- 10.0.0.10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2001ms
rtt min/avg/max/mdev = 0.059/0.068/0.086/0.014 ms
root@n1:/tmp/pycore.55393/n1.conf# []

root@n3:/tmp/pycore.55393/n3.conf# tcpdump
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
^C11:50:14.460942 ARP, Request who-has A9 tell 10.0.0.20, length 28

1 packet captured
1 packet received by filter
0 packets dropped by kernel
root@n3:/tmp/pycore.55393/n3.conf# []

root@n4:/tmp/pycore.55393/n4.conf# tcpdump
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
^C11:50:14.460940 ARP, Request who-has A9 tell 10.0.0.20, length 28

1 packet captured
1 packet received by filter
0 packets dropped by kernel
root@n4:/tmp/pycore.55393/n4.conf# []
```

Com a substituição do hub por um switch, os resultados obtidos são diferentes. Quando o laptop n1 fz ping para o host n2, a mensagem é transmitida para o switch. No entanto, o switch em vez de repetir o sinal como o hub, recebe a mensagem e envia para o host pretendido. Desta forma, quando verificamos os resultados do comando `tcpdump`, não existe actividade nos hosts n3 e n4, apenas possuem uma captura de um pacote ARP, que corresponde ao pacote arp-broadcast, tendo a função de fazer requisição ao switch. Isto ocorre porque, quando uma trama chega a um switch e não consegue comutar com base na tabela de comutação, o switch difunde a mensagem através de todas as suas interfaces. Com base nos resultados obtidos dos hubs e switches no contexto de controlar ou dividir domínios de colisão, podemos afirmar, que como os hubs repetem a mensagem por todos os nodos ligados a ele, e ao ter apenas um canal de comunicação, a probabilidade de colisão é muito grande. Consequentemente, é necessário o reenvio das mensagens, que torna a rede lenta. No entanto, os switches tem como objectivo eliminar as colisões. Os switches ao limitar o envio das mensagens apenas para o destino pretendido, permitem assim não partilhar as mensagens pelos outros nodos. Desta forma, ao ter várias portas, para cada interface, garante vários domínios de colisão, diminuindo a probabilidade de colisão.

Conclusão

Este relatório tem como objetivo apresentar as respostas às questões das fichas apresentadas nas aulas práticas. À medida que íamos resolvendo as questões propostas, ganhamos conhecimentos à cerca dos temas abordados, como Detecção e Correção de Erros, Protocolos de Acesso de Controlo de Ligação, Endereços MAC, Address Resolution Protocol (ARP), Ethernet, Interligação de Redes Locais. Para fortalecer esse conhecimentos, a captura e análise a tramas de Ethernet, através do Wireshark, foi importante, pois deu-nos uma visão mais realista das tramas e da sua constituição. Além disto, ofereceu-nos uma melhor percepção do mecanismo de envio e relação dos vários endereços existentes, e a suas funções e importância. Para o melhor entendimento dos mecanismos de mapeamento entre os endereços de rede e os endereços de uma tecnologia de ligação de dados, o estudo do protocolo ARP é essencial. Aliás, o manuseamento do core, numa máquina virtual, proporcionou uma melhor noção do seu funcionamento, e consequências no uso de certos equipamentos de ligação.