

Redes de Computadores

Ensaio - 3º Trabalho Prático | Ethernet e protocolo ARP

Paulo Caldas (a79089), Pedro Henrique (a77377), Vitor Peixoto (a79175)

Universidade do Minho, Departamento de Informática, 4710-057 Braga, Portugal
e-mail: {a79089, a77377, a79175}@alunos.uminho.pt

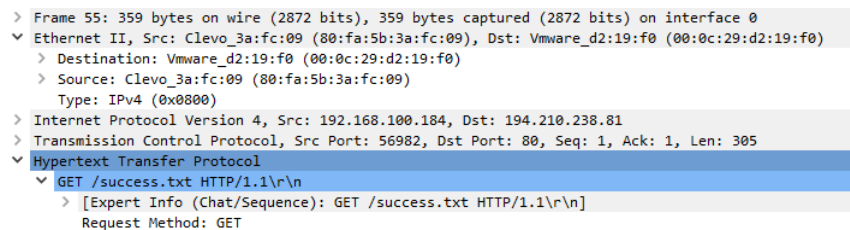
1 Parte I

Captura e análise de Tramas Ethernet

Começamos por, como o enunciado indica, limpar a cache do *browser* a utilizar, ativar o Wireshark e aceder ao URL `http://miei.di.uminho.pt`. Depois, paramos a captura do Wireshark e selecionamos as mensagens de HTTP GET e HTTP *response*.

1. Anote os endereços MAC de origem e de destino da trama capturada.

Dada a seguinte análise da trama:



```
> Frame 55: 359 bytes on wire (2872 bits), 359 bytes captured (2872 bits) on interface 0
▼ Ethernet II, Src: Clevo_3a:fc:09 (80:fa:5b:3a:fc:09), Dst: Vmware_d2:19:f0 (00:0c:29:d2:19:f0)
  > Destination: Vmware_d2:19:f0 (00:0c:29:d2:19:f0)
  > Source: Clevo_3a:fc:09 (80:fa:5b:3a:fc:09)
  Type: IPv4 (0x0800)
> Internet Protocol Version 4, Src: 192.168.100.184, Dst: 194.210.238.81
> Transmission Control Protocol, Src Port: 56982, Dst Port: 80, Seq: 1, Ack: 1, Len: 305
▼ Hypertext Transfer Protocol
  > GET /success.txt HTTP/1.1\r\n
    > [Expert Info (Chat/Sequence): GET /success.txt HTTP/1.1\r\n]
      Request Method: GET
```

Figura 1. Informação da trama da mensagem HTTP GET

É então possível concluir que o endereço MAC de origem é `80:fa:5b:3a:fc:09` e o de destino é `00:0c:29:d2:19:f0`.

2. Identifique a que sistemas se referem. Justifique.

O sistema a que o endereço MAC de origem se refere é o da NIC (*Network Interface Card*) do computador em que foi feito este trabalho e o de destino corresponderá ao endereço MAC do *router* da rede local, que encaminhará o tráfego para o servidor da página com o URL `miei.di.uminho.pt`. Estas observações advêm de sabermos que a cada momento é apenas possível conhecer os endereços MAC de interfaces na mesma subrede em que o interface está, tendo estes endereços portanto âmbito exclusivamente local.

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

Partindo da informação da mesma trama mostrada na Figura 1, podemos identificar que o valor do campo *Type* é `0x0800`, que identifica o protocolo IPv4. Este campo serve para identificar o protocolo encapsulado no campo de dados da trama selecionada.

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

0000	00 0c 29 d2 19 f0 80 fa 5b 3a fc 09 08 00 45 00	..).....[:....E.
0010	01 59 72 05 40 00 80 06 00 00 c0 a8 64 b8 c2 d2	.Yr.@... ..d...
0020	ee 51 de 96 00 50 55 ee 61 0d e4 19 16 02 50 18	.Q...PU. a.....P.
0030	00 ff d7 d0 00 00 47 45 54 20 2f 73 75 63 63 65GET/succe
0040	73 73 2e 74 78 74 20 48 54 54 50 2f 31 2e 31 0d	ss.txt H TTP/1.1.
0050	0a 48 6f 73 74 3a 20 64 65 74 65 63 74 70 6f 72	.Host: detectpor
0060	74 61 6c 2e 66 69 72 65 66 6f 78 2e 63 6f 6d 0d	tal.fire fox.com.
0070	0a 55 73 65 72 2d 41 67 65 6e 74 3a 20 4d 6f 7a	.User-Ag ent: Moz
0080	69 6c 6c 61 2f 35 2e 30 20 28 57 69 6e 64 6f 77	illa/5.0 (Window
0090	73 20 4e 54 20 36 2e 31 3b 20 57 69 6e 36 34 3b	s NT 6.1 ; Win64;
00a0	20 78 36 34 3b 20 72 76 3a 35 32 2e 30 29 20 47	x64; rv :52.0) G
00b0	65 63 6b 6f 2f 32 30 31 30 30 31 30 31 20 46 69	ecko/201 00101 Fi
00c0	72 65 66 6f 78 2f 35 32 2e 30 0d 0a 41 63 63 65	refox/52 .0..Acce
00d0	70 74 3a 20 2a 2f 2a 0d 0a 41 63 63 65 70 74 2d	pt: /*. .Accept-
00e0	4c 61 6e 67 75 61 67 65 3a 20 65 6e 2d 47 42 2c	Language : en-GB,
00f0	65 6e 3b 71 3d 30 2e 35 0d 0a 41 63 63 65 70 74	en;q=0.5 ..Accept
0100	2d 45 6e 63 6f 64 69 6e 67 3a 20 67 7a 69 70 2c	-Encodin g: gzip,
0110	20 64 65 66 6c 61 74 65 0d 0a 43 61 63 68 65 2d	deflate ..Cache-
0120	43 6f 6e 74 72 6f 6c 3a 20 6e 6f 2d 63 61 63 68	Control: no-cach
0130	65 0d 0a 50 72 61 67 6d 61 3a 20 6e 6f 2d 63 61	e..Pragm a: no-ca
0140	63 68 65 0d 0a 44 4e 54 3a 20 31 0d 0a 43 6f 6e	che..DNT : 1..Con
0150	6e 65 63 74 69 6f 6e 3a 20 6b 65 65 70 2d 61 6c	nection: keep-al
0160	69 76 65 0d 0a 0d 0a	ive....

Figura 2. Conteúdo da trama e posição do carácter 'G' do método HTTP GET

Analisando o conteúdo da trama, identificamos que o carácter 'G' corresponde ao byte 55. Assim sendo, são usados 54 bytes desde o início da trama até ao carácter ASCII 'G' do método HTTP GET. Sabendo que o tamanho total do pacote é 359 bytes (Figura 1), a sobrecarga da pilha protocolar é de:

$$\frac{54}{359} \times 100 \approx 15.04\%$$

5. Em ligações com fios pouco suscetíveis a erros, nem sempre as NICs geram o código de deteção de erros. Através de visualização direta de uma trama capturada, verifique se o campo FCS está visível i.e., se 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.

Indicando que é assumido o uso de FCS, podemos verificar que o Wireshark apresenta agora a existência de vários problemas, como pacotes que estão agora malformados, outros cuja *Ethernet Frame Check Sequence* está incorreta, entre outros.

Como é possível verificar, o campo relativo à *Frame Check Sequence* apresenta o valor 0x77ea9494, quando deveria tomar o valor 0x87fd41ab para ser correto.

Com tudo isto, podemos afirmar que o campo FCS (*Frame Check Sequence*) não está em uso.

A seguir responda às seguintes perguntas, baseado no conteúdo da trama Ethernet que contém o primeiro byte da resposta HTTP.

Selecionamos então a trama da resposta HTTP com informação HTTP/1.1 200 OK. A informação relevante às próximas perguntas está disponível na seguinte figura, correspondente à trama selecionada:

```
> Frame 56: 438 bytes on wire (3504 bits), 438 bytes captured (3504 bits) on interface 0
▼ Ethernet II, Src: Vmware_d2:19:f0 (00:0c:29:d2:19:f0), Dst: Clevo_3a:fc:09 (80:fa:5b:3a:fc:09)
  > Destination: Clevo_3a:fc:09 (80:fa:5b:3a:fc:09)
  > Source: Vmware_d2:19:f0 (00:0c:29:d2:19:f0)
  Type: IPv4 (0x0800)
> Internet Protocol Version 4, Src: 194.210.238.81, Dst: 192.168.100.184
> Transmission Control Protocol, Src Port: 80, Dst Port: 56982, Seq: 1, Ack: 306, Len: 384
▼ Hypertext Transfer Protocol
  > HTTP/1.1 200 OK\r\n
```

Figura 6. Informação da trama de resposta HTTP

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, correspondente ao endereço MAC do *router* da rede local. Isto acontece porque um dos serviços prestados pelo nível 2 da camada protocolar é a transferência de dados entre nodos adjacentes na mesma rede local. Sendo o *router* o nodo intermediário que entrega a trama de resposta do servidor ao computador em que foi feito este trabalho, será seu o endereço MAC de origem da mensagem de resposta.

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

O endereço MAC do destino é 80:fa:5b:3a:fc:09 e o sistema a que corresponde é o computador onde foi feito este trabalho. Podemos então observar que os endereços MAC de origem e destino das mensagens HTTP GET e HTTP OK são inversos entre si.

8. Atendendo ao conceito de desencapsulamento protocolar, identifica os vários protocolos contidos na trama recebida.

Na figura 4 são observáveis os vários protocolos contidos na trama capturada. Estes são: Ethernet II, Internet Protocol version 4 (IPv4) e Transmission Control Protocol (TCP). Finalmente, o nível de aplicação contém HyperText Transfer Protocol (HTTP).

Protocolo ARP

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

O resultado de executar `arp` é o seguinte:

```
> arp -n
```

Address	Hwtype	Hwaddress	Flags	Mask	Iface
192.168.1.176	ether	c8:02:10:87:97:d8	C		wlp8s0
192.168.1.1	ether	04:bd:70:a9:8e:95	C		wlp8s0

Figura 7. Output do comando `arp`

Uma tabela ARP possui informação que mapeia um endereço IP para o respetivo endereço MAC da NIC em que a interface se encontra. Assim sendo, na tabela, a coluna **Address** regista os endereços IP, **Hwtype** identifica o tipo de protocolo de rede associado (por exemplo, Ethernet), **Hwaddress** identifica o endereço físico do dispositivo (também conhecido como o *MAC address*), **Flags** dá informação relevante sobre a entrada na tabela (por exemplo, a *flag 'C'* indica que a entrada na tabela é do tipo "completa", i.e. possui todas as colunas preenchidas sobre a entrada em questão). Adicionalmente, a coluna **Iface** identifica o interface utilizado pelo dispositivo (p.e., `wlp8s0` corresponde à *wireless interface*).

Seguindo para a fase seguinte do exercício, acedemos a <http://cesium.di.uminho.pt> e com base no resultado da captura do Wireshark, analisamos a seguinte trama relativa a um pedido ARP:

```
> Frame 2709: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0
▼ Ethernet II, Src: Clevo_3a:fc:09 (80:fa:5b:3a:fc:09), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
  > Destination: Broadcast (ff:ff:ff:ff:ff:ff)
  > Source: Clevo_3a:fc:09 (80:fa:5b:3a:fc:09)
    Type: ARP (0x0806)
▼ Address Resolution Protocol (request)
  Hardware type: Ethernet (1)
  Protocol type: IPv4 (0x0800)
  Hardware size: 6
  Protocol size: 4
  Opcode: request (1)
  Sender MAC address: Clevo_3a:fc:09 (80:fa:5b:3a:fc:09)
  Sender IP address: 192.168.100.184
  Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
  Target IP address: 192.168.100.254
```

Figura 8. Conteúdo da trama relativa ao ARP *request*

10. 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?

Vemos que o endereço de origem é `80:fa:5b:3a:fc:09` e o endereço de destino é o endereço reservado `ff:ff:ff:ff:ff:ff`. Para justificar o endereço de destino é necessário perceber o contexto de uma mensagem do tipo *ARP request*. Nestas um *host* procura o endereço MAC associado a um determinado endereço IP de uma interface. Como não conhece o nodo da rede local com a interface com tal endereço IP, envia o pedido ARP em difusão, para que este pedido chegue a todos os nodos da rede local. É então enviado para o endereço reservado para esse fim (`ff:ff:ff:ff:ff:ff` - endereço reservado para *broadcast*). Após este envio, espera obter resposta proveniente do endereço IP da interface cujo endereço MAC associado procurava, sendo esse o endereço físico da origem do *ARP reply*.

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

O campo *Type* tem o valor `0x0806` e indica o tipo de dados encapsulado pelo cabeçalho Ethernet, que neste caso se refere ao ARP (*Address Resolution Protocol*).

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

O campo *ARP opcode* tem valor 1, que identifica operações de pedido de endereço, ou seja, a trama em análise corresponde a um *ARP request*.

13. Identifique que tipo de endereços estão contidos na mensagem ARP? Que conclui?

Há 2 tipos de endereços contidos na mensagem ARP: endereços MAC (endereços físicos), relativos aos nodos de origem e destino da mensagem ARP, e endereços IP (endereços lógicos), relativos às interfaces de origem e destino da mensagem ARP. No caso da trama em análise, correspondente a um *ARP request*, os endereços MAC e IP da origem são obviamente conhecidos e identificam a *Network Interface Card* e a interface responsáveis pelo envio deste pedido (aqui são `80:fa:5b:3a:fc:09` e `192.168.100.184` respetivamente). Relativamente ao destino, como com este pedido pretendemos conhecer o endereço físico associado a um determinado IP, apenas esse IP (neste caso `192.168.100.254`) estará definido, tomando o endereço MAC de destino o valor reservado de `00:00:00:00:00:00`.

14. Explícite que tipo de pedido ou pergunta é feita pelo host de origem?

Uma mensagem do tipo ARP *request* pretende determinar o endereço MAC (físico) associado a um dado endereço IP (lógico) conhecido, de modo a que seja possível a transferência de dados entre nodos da rede local. Assim, não sendo conhecido o endereço MAC associado a essa interface, é enviado em difusão um pedido ARP, que uma vez chegado à interface de destino pretendida, espera originar uma resposta. Essa resposta, um ARP *reply*, terá como endereço MAC de origem o endereço MAC inicialmente pretendido, como endereço IP de origem o endereço IP que originalmente era conhecido e como endereços MAC e IP de destino os associados à mensagem que teria enviado o ARP *request*.

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

```
> Frame 2710: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
▼ Ethernet II, Src: Vmware_d2:19:f0 (00:0c:29:d2:19:f0), Dst: Clevo_3a:fc:09 (80:fa:5b:3a:fc:09)
  > Destination: Clevo_3a:fc:09 (80:fa:5b:3a:fc:09)
  > Source: Vmware_d2:19:f0 (00:0c:29:d2:19:f0)
    Type: ARP (0x0806)
    Padding: 00000000000000000000000000000000
  ▼ Address Resolution Protocol (reply)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: reply (2)
    Sender MAC address: Vmware_d2:19:f0 (00:0c:29:d2:19:f0)
    Sender IP address: 192.168.100.254
    Target MAC address: Clevo_3a:fc:09 (80:fa:5b:3a:fc:09)
    Target IP address: 192.168.100.184
```

Figura 9. Conteúdo da trama relativa ao ARP *reply*

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

O campo ARP *opcode* tem valor 2, que identifica operações de resposta a pedidos de endereço, ou seja, a trama em análise corresponde a um ARP *reply*.

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

A resposta está no campo **Sender MAC address**. O dispositivo de destino desta mensagem foi o que enviou o pedido ARP e saberá agora, através da mensagem de resposta, o endereço físico associado à interface de rede que procurava (neste caso é 00:0c:29:d2:19:f0).

ARP numa topologia CORE

A topologia CORE inicial é a seguinte:

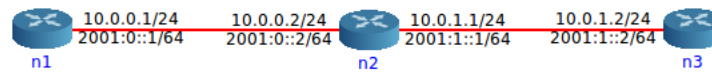


Figura 10. Topologia CORE inicial utilizada

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

Abrindo um terminal em cada um dos diversos *routers* e executando o comando `ifconfig`, verificamos que os *routers* n1 e n3 possuem uma interface Ethernet e uma interface para Local Loopback, enquanto que o *router* n2 possui duas interfaces Ethernet e uma interface para Local Loopback. As interfaces Ethernet referem-se às ligações entre nodos que foram criadas e são visíveis na imagem (por exemplo, o *router* n1 está conectado ao *router* n2, pelo que ambos possuem um interface Ethernet que permite tal conectividade). As interfaces para Local Loopback são especiais e existem para que a máquina associada possa falar consigo mesma, sendo útil para casos de diagnóstico de problemas de rede, obtenção de informação útil para alguns protocolos e utilização de serviços de rede que corram na própria máquina.

Seguem-se as imagens do output do comando `ifconfig` nos vários nodos:

```
root@n1 /tmp/pycore.41367/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:ff:feaa:0/64 Scope:Link
          inet6 addr: 2001::1/64 Scope:Global
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:58 errors:0 dropped:0 overruns:0 frame:0
          TX packets:14 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:7639 (7.6 KB)  TX bytes:1452 (1.4 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:65536  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:1
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

Figura 11. Output de `ifconfig` do nodo 1

```

root@n2 /tmp/pycore.41367/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:ff:feaa:1/64 Scope:Link
          inet6 addr: 2001::2/64 Scope:Global
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:70 errors:0 dropped:0 overruns:0 frame:0
          TX packets:18 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:9034 (9.0 KB)  TX bytes:1804 (1.8 KB)

eth1      Link encap:Ethernet  HWaddr 00:00:00:aa:00:02
          inet addr:10.0.1.1  Bcast:0.0.0.0  Mask:255.255.255.0
          inet6 addr: fe80::200:ff:feaa:2/64 Scope:Link
          inet6 addr: 2001:1::1/64 Scope:Global
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:71 errors:0 dropped:0 overruns:0 frame:0
          TX packets:17 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:9148 (9.1 KB)  TX bytes:1730 (1.7 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:65536  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:1
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

```

Figura 12. Output de ifconfig do nodo 2

```

root@n3 /tmp/pycore.41367/n3.conf # ifconfig
eth0      Link encap:Ethernet  HWaddr 00:00:00:aa:00:03
          inet addr:10.0.1.2  Bcast:0.0.0.0  Mask:255.255.255.0
          inet6 addr: 2001:1::2/64 Scope:Global
          inet6 addr: fe80::200:ff:feaa:3/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:106 errors:0 dropped:0 overruns:0 frame:0
          TX packets:49 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:13264 (13.2 KB)  TX bytes:5102 (5.1 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:65536  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:1
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

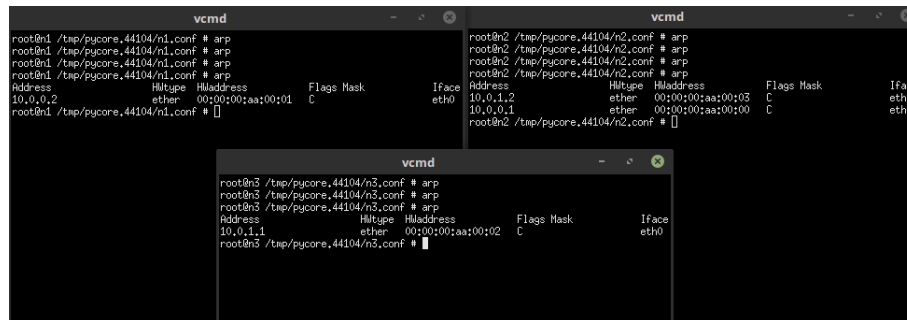
```

Figura 13. Output de ifconfig do nodo 3

Como é possível verificar nas imagens, o endereço Ethernet da interface **eth0** do *router 1* é 00:00:00:aa:00:00, das interfaces **eth0** e **eth1** do *router 2* são, respetivamente, 00:00:00:aa:00:01 e 00:00:00:aa:00:02 e da interface **eth0** do *router 3* é 00:00:00:aa:00:03.

17. Usando o comando `arp` obtenha as caches arp dos diversos sistemas.

Executando o comando `arp` no terminal de cada um dos vários *routers*, verificamos que todas as caches ARP estão inicialmente vazias. Todavia, após algum tempo de espera (no caso da resolução relativa a este relatório durou cerca de um minuto) as ligações estabilizam-se e as caches ARP são atualizadas. O resultante é o seguinte:



The image shows three terminal windows, each titled 'vcmd', displaying the output of the 'arp' command on different routers. The top-left window is for router n1, the top-right for n2, and the bottom-center for n3. Each window shows a table with columns: Address, Hltype, Hladdress, Flags, Mask, and Iface. Router n1 has one entry for 10.0.0.2 on eth0. Router n2 has two entries for 10.0.1.2 and 10.0.0.1 on eth0. Router n3 has one entry for 10.0.1.1 on eth0.

Router	Address	Hltype	Hladdress	Flags	Mask	Iface
n1	10.0.0.2	ether	00:00:00:aa:00:01	C		eth0
n2	10.0.1.2	ether	00:00:00:aa:00:03	C		eth0
n2	10.0.0.1	ether	00:00:00:aa:00:00	C		eth0
n3	10.0.1.1	ether	00:00:00:aa:00:02	C		eth0

Figura 14. Tabelas ARP dos vários *routers*

Podemos verificar que cada nodo possui agora informação na sua cache ARP sobre os nodos adjacentes a si (n1 tem informação de n2, n2 de n1 e n3, e n3 de n2), confirmando assim que o protocolo ARP tem um âmbito de operação restrito à rede local.

18. 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?

Começamos por executar o comando `ping` entre os nodos n1 e n2:

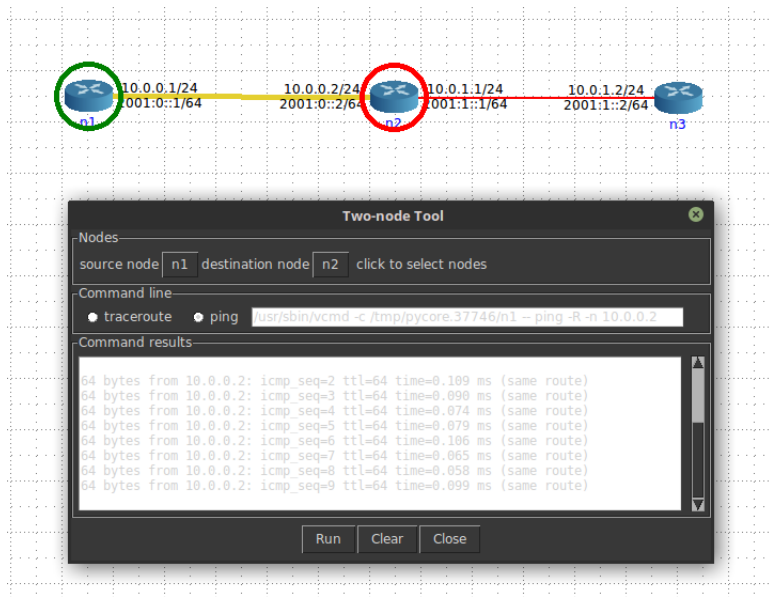


Figura 15. Execução do comando `ping` de n1 para n2

Verificamos que as caches ARP permanecem inalteradas.

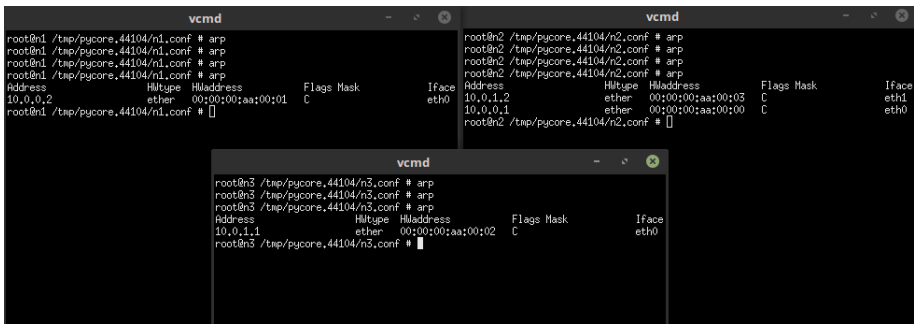


Figura 16. Caches ARP após a execução de `ping` de n1 para n2

O nodo n1 sabe qual é o endereço físico e IP do nodo n2, pelo que não há necessidade de atualizar a tabela.

Executando agora *ping* entre n1 e n3:

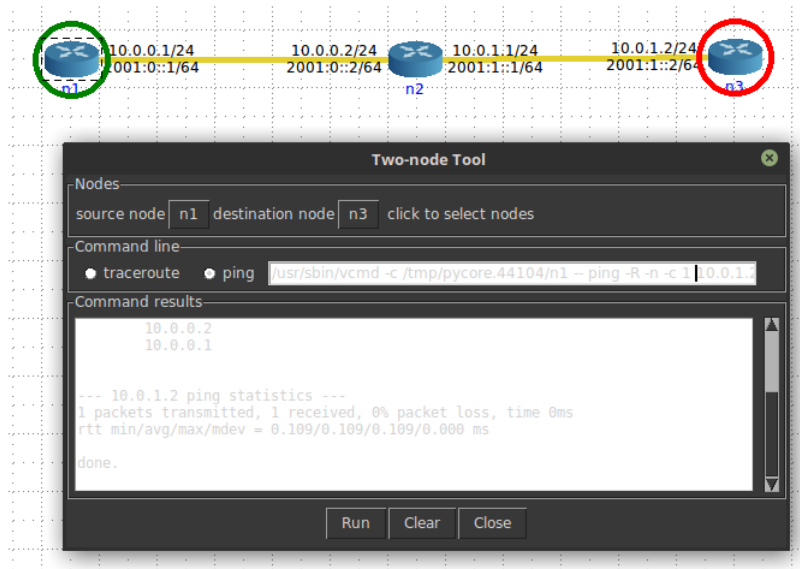


Figura 17. Execução do comando *ping* de n1 para n3

As caches ARP resultantes são as seguintes:

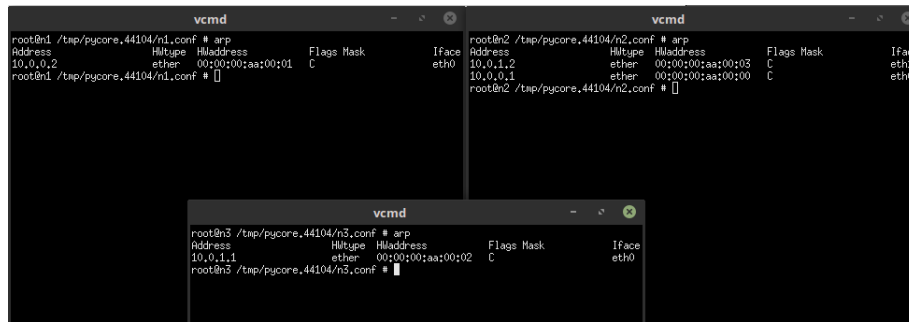


Figura 18. Caches ARP após a execução de *ping* de n1 para n3

Ou seja, o segundo `ping` executado, de n1 para n3, em nada alterou as caches ARP dos vários nodos. Isto será porque para a execução do `ping` entre n1 e n3, o nodo n2 é adjacente a ambos, ou seja, encontra-se em redes locais com os *routers* n1 e n3, pelo que todos os nodos possuem toda a informação necessária relativa a endereços físicos para transferência de dados na rede local. Neste caso, o nodo n1 pode enviar a trama para o nodo n2, que a poderá enviar para o nodo n3, possibilitando a transferência do tráfego gerado pelo comando `ping`. O nodo n1 saberá que pode comunicar com n3 através de n2 graças à sua tabela de *routing*, tópico discutido no trabalho prático anterior. Tudo isto confirma que o âmbito de operação do protocolo ARP é restrito à rede local.

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

Num terminal de n1 é executado:

```
arp -d 10.0.0.2
```

O endereço IP especificado da entrada a ser removida é do *router* n2. Como resultado, a cache ARP do *router* n1 é:

<i>Address</i>	<i>HWtype</i>	<i>HWaddress</i>	<i>FlagMask</i>	<i>Iface</i>
10.0.0.2		(incomplete)		eth0

É então adicionada uma nova entrada com:

```
arp -s 10.0.0.2 12:34:56:aa:78:90
```

Presume-se agora portanto que será impossível para o nodo n1 enviar tramas para o nodo n2, pois o endereço físico guardado na cache ARP é incorreto.

A confirmação poderá ser feita recorrendo ao comando `ping`, a partir de um terminal de n1:

```
ping 10.0.0.2
```

Isto resulta numa *packet loss* de 100%. Isto será porque, apesar de o tráfego gerado pelo `ping` ter como destino um IP existente na cache ARP do dispositivo, o endereço físico não está associado a nenhuma interface na rede, pelo que se torna impossível enviar a trama.

Seja reestruturada a ligação com o comando:

```
arp -s 10.0.0.2 00:00:00:aa:00:01
```

Assim a ligação é recuperada, pelo que a execução do comando `ping` com destino ao endereço IP 10.0.0.2 resulta numa *packet loss* de 0%.

Adicione agora um *switch* (n4) à rede e ligue o *router* n1, e os *hosts* n5 e n6 a esse *switch*.

A topologia a ser analisada de seguida será portanto a seguinte:

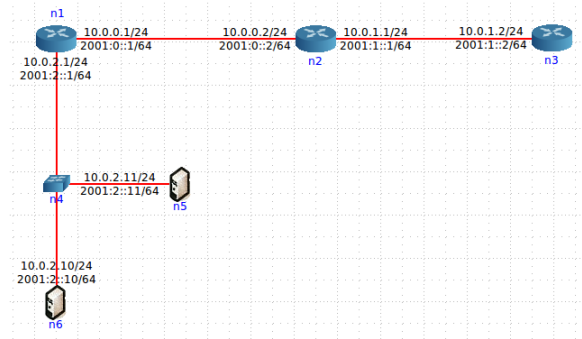


Figura 19. Nova topologia com a adição do *switch* n4 e dos *hosts* n5 e n6

20. Faça ping de n6 para n5. Sem consultar a tabela ARP anote a entrada que, em sua opinião, é criada na tabela ARP de n6. Verifique, justificando, se a sua interpretação sobre a operação da rede Ethernet e protocolo ARP estava correta.

Começamos por, num terminal de n6, executar o comando `ping 10.0.2.11`. A entrada prevista na tabela ARP do *host* n6 deverá mapear o endereço IP de destino do comando `ping` ao correspondente endereço MAC do *host* n5. Com recurso ao comando `ifconfig`, sabe-se que o endereço MAC de n5 é 00:00:00:aa:00:06, e portanto, deverá estar associado ao IP 10.0.2.11 na tabela ARP do *host* n6. Querendo o nodo n6 enviar tráfego gerado pelo comando `ping` para n5 e não existindo ainda mapeamento entre endereços MAC e IP que permitam transferência de dados na rede local, este envia um ARP *request* em *broadcast*, com destino ao endereço IP 10.0.2.11 (endereço IP de n5), para determinar o endereço MAC a ele associado. Por ter sido enviado em difusão, este pedido é comutado pelo *switch* n4, que o faz chegar ao *router* n1, para além do *host* n5. Uma vez recebido o pedido, o *host* n5 enviará a resposta com endereços físico e lógico de destino correspondentes ao *host* n6 e endereços físico e lógico a si relativos, servindo-se dos serviços de comutação oferecidos pelo *switch* na rede local, que pela sua tabela de comutação sabe a porta a que está ligado o *host* n6 devido à comunicação prévia. Após a receção do ARP *reply*, o *host* n6 poderá então atualizar a sua tabela ARP e enviar o tráfego do comando `ping` pela rede local, servindo-se do *switch* n4.

Evidências deste fluxo de tráfego encontram-se nos resultados do comando *tcpdump*, executado sobre cada uma das interfaces envolvidas neste exercício. São as seguintes:

```
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
^C11:08:28.405453 IP6 fe80::200:ff:feaa:4 > ff02::5: OSPFv3, Hello, length 36
11:08:28.433533 IP 10.0.2.1 > 224.0.0.5: OSPFv2, Hello, length 44
11:08:37.998795 ARP, Request who-has 10.0.2.11 tell 10.0.2.10, length 28
11:08:37.998848 ARP, Reply 10.0.2.11 is-at 00:00:00:aa:00:06 (oui Ethernet), length 28
11:08:37.998851 IP 10.0.2.10 > 10.0.2.11: ICMP echo request, id 50, seq 1, length 64
11:08:37.998863 IP 10.0.2.11 > 10.0.2.10: ICMP echo reply, id 50, seq 1, length 64
11:08:38.435876 IP 10.0.2.1 > 224.0.0.5: OSPFv2, Hello, length 44
11:08:38.446328 IP6 fe80::200:ff:feaa:4 > ff02::5: OSPFv3, Hello, length 36
11:08:38.999649 IP 10.0.2.10 > 10.0.2.11: ICMP echo request, id 50, seq 2, length 64
11:08:38.999683 IP 10.0.2.11 > 10.0.2.10: ICMP echo reply, id 50, seq 2, length 64
11:08:39.998646 IP 10.0.2.10 > 10.0.2.11: ICMP echo request, id 50, seq 3, length 64
11:08:39.998686 IP 10.0.2.11 > 10.0.2.10: ICMP echo reply, id 50, seq 3, length 64
11:08:41.000670 IP 10.0.2.10 > 10.0.2.11: ICMP echo request, id 50, seq 4, length 64
11:08:41.000694 IP 10.0.2.11 > 10.0.2.10: ICMP echo reply, id 50, seq 4, length 64
11:08:42.000866 IP 10.0.2.10 > 10.0.2.11: ICMP echo request, id 50, seq 5, length 64
11:08:42.000910 IP 10.0.2.11 > 10.0.2.10: ICMP echo reply, id 50, seq 5, length 64
11:08:43.031063 ARP, Request who-has 10.0.2.10 tell 10.0.2.11, length 28
11:08:43.031073 ARP, Reply 10.0.2.10 is-at 00:00:00:aa:00:05 (oui Ethernet), length 28
11:08:43.031184 IP 10.0.2.10 > 10.0.2.11: ICMP echo request, id 50, seq 6, length 64
11:08:43.031221 IP 10.0.2.11 > 10.0.2.10: ICMP echo reply, id 50, seq 6, length 64
11:08:48.445695 IP 10.0.2.1 > 224.0.0.5: OSPFv2, Hello, length 44
11:08:48.461082 IP6 fe80::200:ff:feaa:4 > ff02::5: OSPFv3, Hello, length 36
11:08:58.430113 IP6 fe80::200:ff:feaa:4 > ff02::5: OSPFv3, Hello, length 36
11:08:58.451961 IP 10.0.2.1 > 224.0.0.5: OSPFv2, Hello, length 44

24 packets captured
24 packets received by filter
0 packets dropped by kernel
root@n6:/tmp/pycore.45603/n6.conf#
```

Figura 20. Resultado do *tcpdump* no *host* n6


```

tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
^C11:08:28.405451 IP6 fe80::200:ff:feaa:4 > ff02::5: OSPFv3, Hello, length 36
11:08:28.433532 IP 10.0.2.1 > 224.0.0.5: OSPFv2, Hello, length 44
11:08:37.998822 ARP, Request who-has 10.0.2.11 tell 10.0.2.10, length 28
11:08:37.998840 ARP, Reply 10.0.2.11 is-at 00:00:00:aa:00:06 (oui Ethernet), length 28
11:08:37.998855 IP 10.0.2.10 > 10.0.2.11: ICMP echo request, id 50, seq 1, length 64
11:08:37.998861 IP 10.0.2.11 > 10.0.2.10: ICMP echo reply, id 50, seq 1, length 64
11:08:38.435874 IP 10.0.2.1 > 224.0.0.5: OSPFv2, Hello, length 44
11:08:38.446327 IP6 fe80::200:ff:feaa:4 > ff02::5: OSPFv3, Hello, length 36
11:08:38.999666 IP 10.0.2.10 > 10.0.2.11: ICMP echo request, id 50, seq 2, length 64
11:08:38.999679 IP 10.0.2.11 > 10.0.2.10: ICMP echo reply, id 50, seq 2, length 64
11:08:39.998676 IP 10.0.2.10 > 10.0.2.11: ICMP echo request, id 50, seq 3, length 64
11:08:39.998684 IP 10.0.2.11 > 10.0.2.10: ICMP echo reply, id 50, seq 3, length 64
11:08:41.000683 IP 10.0.2.10 > 10.0.2.11: ICMP echo request, id 50, seq 4, length 64
11:08:41.000690 IP 10.0.2.11 > 10.0.2.10: ICMP echo reply, id 50, seq 4, length 64
11:08:42.000898 IP 10.0.2.10 > 10.0.2.11: ICMP echo request, id 50, seq 5, length 64
11:08:42.000907 IP 10.0.2.11 > 10.0.2.10: ICMP echo reply, id 50, seq 5, length 64
11:08:43.030995 ARP, Request who-has 10.0.2.10 tell 10.0.2.11, length 28
11:08:43.031078 ARP, Reply 10.0.2.10 is-at 00:00:00:aa:00:05 (oui Ethernet), length 28
11:08:43.031193 IP 10.0.2.10 > 10.0.2.11: ICMP echo request, id 50, seq 6, length 64
11:08:43.031216 IP 10.0.2.11 > 10.0.2.10: ICMP echo reply, id 50, seq 6, length 64
11:08:48.445634 IP 10.0.2.1 > 224.0.0.5: OSPFv2, Hello, length 44
11:08:48.461080 IP6 fe80::200:ff:feaa:4 > ff02::5: OSPFv3, Hello, length 36

22 packets captured
22 packets received by filter
0 packets dropped by kernel
root@n5:/tmp/pycore.45603/n5.conf# █

```

Figura 21. Resultado do tcpdump no *host* n5

```

root@n1:/tmp/pycore.45603/n1.conf# tcpdump -i eth1
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 65535 bytes
^C11:07:38.307331 IP6 fe80::200:ff:feaa:4 > ff02::5: OSPFv3, Hello, length 36
11:07:38.398924 IP 10.0.2.1 > 224.0.0.5: OSPFv2, Hello, length 44
11:07:48.345722 IP6 fe80::200:ff:feaa:4 > ff02::5: OSPFv3, Hello, length 36
11:07:48.399196 IP 10.0.2.1 > 224.0.0.5: OSPFv2, Hello, length 44
11:07:58.336084 IP6 fe80::200:ff:feaa:4 > ff02::5: OSPFv3, Hello, length 36
11:07:58.413597 IP 10.0.2.1 > 224.0.0.5: OSPFv2, Hello, length 44
11:08:08.342395 IP6 fe80::200:ff:feaa:4 > ff02::5: OSPFv3, Hello, length 36
11:08:08.415953 IP 10.0.2.1 > 224.0.0.5: OSPFv2, Hello, length 44
11:08:18.384831 IP6 fe80::200:ff:feaa:4 > ff02::5: OSPFv3, Hello, length 36
11:08:18.431236 IP 10.0.2.1 > 224.0.0.5: OSPFv2, Hello, length 44
11:08:28.405445 IP6 fe80::200:ff:feaa:4 > ff02::5: OSPFv3, Hello, length 36
11:08:28.433525 IP 10.0.2.1 > 224.0.0.5: OSPFv2, Hello, length 44
11:08:37.998824 ARP, Request who-has 10.0.2.11 tell 10.0.2.10, length 28
11:08:38.435868 IP 10.0.2.1 > 224.0.0.5: OSPFv2, Hello, length 44
11:08:38.446304 IP6 fe80::200:ff:feaa:4 > ff02::5: OSPFv3, Hello, length 36
11:08:48.445685 IP 10.0.2.1 > 224.0.0.5: OSPFv2, Hello, length 44
11:08:48.461065 IP6 fe80::200:ff:feaa:4 > ff02::5: OSPFv3, Hello, length 36

17 packets captured
17 packets received by filter
0 packets dropped by kernel
root@n1:/tmp/pycore.45603/n1.conf# █

```

Figura 22. Resultado do tcpdump no *router* n1

2 Parte II

ARP gratuito

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

Registamos apenas o envio de um pacote ARP gratuito com origem no sistema onde foi feito este trabalho. Assim sendo, não foi possível registar o intervalo de tempo entre pacotes de ARP gratuito enviados por esse computador.

47	18.687270	Clevo_3a:fc:09	Broadcast	ARP	42 Who has 192.168.100.254? Tell 192.168.100.184
48	18.702428	Clevo_3a:fc:09	Broadcast	ARP	42 Who has 192.168.100.184? Tell 0.0.0.0
54	19.202132	Clevo_3a:fc:09	Broadcast	ARP	42 Who has 192.168.100.254? Tell 192.168.100.184
66	19.693344	Clevo_3a:fc:09	Broadcast	ARP	42 Who has 192.168.100.184? Tell 0.0.0.0
140	20.706261	Clevo_3a:fc:09	Broadcast	ARP	42 Who has 192.168.100.184? Tell 0.0.0.0
149	21.694163	Clevo_3a:fc:09	Broadcast	ARP	42 Gratuitous ARP for 192.168.100.184 (Request)
194	26.705078	Clevo_3a:fc:09	Vmware_d2:19:f0	ARP	42 192.168.100.184 is at 80:fa:5b:3a:fc:09
219	29.385502	Clevo_3a:fc:09	Broadcast	ARP	42 Who has 192.168.100.254? Tell 192.168.100.184
341	47.242369	Clevo_3a:fc:09	Broadcast	ARP	42 Who has 192.168.100.254? Tell 192.168.100.184
506	53.159559	Clevo_3a:fc:09	Vmware_d2:19:f0	ARP	42 192.168.100.184 is at 80:fa:5b:3a:fc:09

Figura 23. O único pacote de pedido ARP gratuito originado pelo sistema

Face a isto, utilizando outro computador, seguimos o mesmo procedimento e obtivemos o seguinte:

727	51.193001	Apple_0b:85:d3	Vmware_d2:19:f0	ARP	42 Who has 192.168.100.254? Tell 192.168.100.165
730	51.193258	Vmware_d2:19:f0	Apple_0b:85:d3	ARP	60 192.168.100.254 is at 00:0c:29:d2:19:f0
731	51.193470	Apple_0b:85:d3	Broadcast	ARP	42 Gratuitous ARP for 192.168.100.165 (Request)
734	51.262315	Apple_0b:85:d3	Broadcast	ARP	42 Who has 192.168.100.254? Tell 192.168.100.165
735	51.262780	Vmware_d2:19:f0	Apple_0b:85:d3	ARP	60 192.168.100.254 is at 00:0c:29:d2:19:f0
744	51.866252	Vmware_d2:19:f0	Broadcast	ARP	60 Who has 192.168.100.159? Tell 192.168.100.254
746	52.526703	Apple_0b:85:d3	Broadcast	ARP	42 Gratuitous ARP for 192.168.100.165 (Request)
747	52.730134	Apple_0b:85:d3	Broadcast	ARP	42 Who has 192.168.100.254? Tell 192.168.100.165

Figura 24. Outro sistema com mais pedidos ARP gratuitos gerados

São agora observáveis 2 pedidos ARP gratuito originários do sistema que utilizamos para este exercício. O intervalo temporal entre eles é de $52.526703 - 51.193470 = 1.333233$ s.

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

Analisando o conteúdo do pedido ARP gratuito registado no primeiro caso da resposta ao exercício anterior, é possível verificar que o endereço IP de origem e de destino são iguais e a distinção entre os pedidos ARP gratuitos e os restantes pedidos ARP passará por aí. Ambos os tipos de pedidos ARP são enviados em *broadcast*. No entanto, nos pedidos ARP gratuitos, o endereço IP de origem do pedido é igual ao de destino, enquanto que para os restantes o IP de destino será aquele cujo endereço MAC da NIC onde a interface se encontra pretendemos determinar e o de origem será o IP do *host* que pretende determinar o endereço MAC. Os *gratuitous ARP requests* serão úteis por diversas razões. Estas são:

- determinação de conflitos de endereços IP na rede local, pois quando uma máquina recebe um pedido ARP com um IP de origem igual ao de uma das suas interfaces, descobre que há um conflito de endereços IP;
- atualização de tabelas ARP de outros nodos, como por exemplo no caso de quando uma dada interface com um certo endereço IP se liga à rede, sendo agora possível determinar o endereço MAC da NIC com essa tal interface sem que outros nodos necessitem de enviar um pedido ARP;
- permitem informar um *switch* sobre a existência de um dado endereço MAC ligado a uma das suas portas, para que assim este saiba encaminhar tráfego para esse dado endereço MAC pela porta correta.

A trama correspondente a este pedido ARP gratuito apresenta-se a seguir:

```

▼ Ethernet II, Src: Clevo_3a:fc:09 (80:fa:5b:3a:fc:09), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
  > Destination: Broadcast (ff:ff:ff:ff:ff:ff)
  > Source: Clevo_3a:fc:09 (80:fa:5b:3a:fc:09)
    Type: ARP (0x0806)
▼ Address Resolution Protocol (request/gratuitous ARP)
  Hardware type: Ethernet (1)
  Protocol type: IPv4 (0x0800)
  Hardware size: 6
  Protocol size: 4
  Opcode: request (1)
  [Is gratuitous: True]
  Sender MAC address: Clevo_3a:fc:09 (80:fa:5b:3a:fc:09)
  Sender IP address: 192.168.100.184
  Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
  Target IP address: 192.168.100.184

```

Figura 25. Trama Ethernet correspondente ao pedido ARP gratuito analisado

Os resultados esperados do envio do pedido ARP gratuito são a atualização de tabelas ARP por parte de outros dispositivos na rede, uma possível alteração da tabela de um *switch*, que passa agora a saber a porta a que está ligada um dispositivo com um determinado endereço MAC ou até mesmo a mudança de endereço IP de um dos *hosts* da rede local, no caso de colisão de endereços IP.

Domínios de colisão

A rede criada, com um *hub* repetidor a ligar os diferentes hosts, é representada pela seguinte topologia:

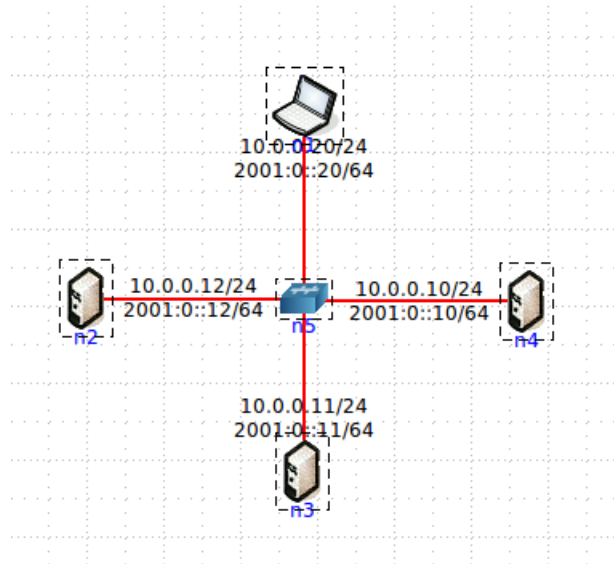


Figura 26. Rede com 4 *hosts* ligados a um *hub* repetidor

1. Faça ping de n1 para n4. Verifique com a opção `tcpdump` como flui o tráfego nas diversas interfaces dos vários dispositivos. Que conclui?

Após a execução do comando `ping`, com origem no *host* n1 e com destino ao *host* n4, é possível verificar que os resultados do comando `tcpdump` em cada um dos *hosts* da rede são iguais entre si. Isto acontece porque todos eles estão ligados a um *hub* repetidor, que fará *broadcast* do tráfego que lhe chega, fazendo com que todas as mensagens enviadas e recebidas por qualquer *host* sejam detetáveis por outros dispositivos nesta mesma rede. Novamente, apesar de não se encontrarem diretamente ligados, os dispositivos desta rede encontram-se ligados a portas de um *hub* que os interliga. Dadas as características deste equipamento, é possível afirmar que os dispositivos desta rede partilham um meio de comunicação a eles comum, e por isso, estão no mesmo domínio de colisão.

```
vcmd - + *
h 64
10:55:53.966403 IP 10.0.0.10 > 10.0.0.20: ICMP echo reply, id 39, seq 5, l
64
10:55:54.949541 ARP, Request who-has 10.0.0.20 tell 10.0.0.10, length 28
10:55:54.949553 ARP, Reply 10.0.0.20 is-at 00:00:00:aa:00:00 (oui Etherne
gth 28
10:55:54.965389 IP 10.0.0.20 > 10.0.0.10: ICMP echo request, id 39, seq 6
h 64
10:55:54.965420 IP 10.0.0.10 > 10.0.0.20: ICMP echo reply, id 39, seq 6,
64
10:55:55.964377 IP 10.0.0.20 > 10.0.0.10: ICMP echo request, id 39, seq 7
h 64
10:55:55.964435 IP 10.0.0.10 > 10.0.0.20: ICMP echo reply, id 39, seq 7,
64
10:55:56.969836 IP 10.0.0.20 > 10.0.0.10: ICMP echo request, id 39, seq 8
h 64
10:55:56.969884 IP 10.0.0.10 > 10.0.0.20: ICMP echo reply, id 39, seq 8,
64
20 packets captured
20 packets received by filter
0 packets dropped by kernel
root@n1:/tmp/pycore.36022/n1.conf# []

vcmd - + *
h 64
10:55:53.966402 IP 10.0.0.10 > 10.0.0.20: ICMP echo reply, id 39, seq 5, l
64
10:55:54.949539 ARP, Request who-has 10.0.0.20 tell 10.0.0.10, length 28
10:55:54.949557 ARP, Reply 10.0.0.20 is-at 00:00:00:aa:00:00 (oui Ethernet
gth 28
10:55:54.965403 IP 10.0.0.20 > 10.0.0.10: ICMP echo request, id 39, seq 6,
h 64
10:55:54.965418 IP 10.0.0.10 > 10.0.0.20: ICMP echo reply, id 39, seq 6, l
64
10:55:55.964396 IP 10.0.0.20 > 10.0.0.10: ICMP echo request, id 39, seq 7,
h 64
10:55:55.964433 IP 10.0.0.10 > 10.0.0.20: ICMP echo reply, id 39, seq 7, l
64
10:55:56.969851 IP 10.0.0.20 > 10.0.0.10: ICMP echo request, id 39, seq 8,
h 64
10:55:56.969883 IP 10.0.0.10 > 10.0.0.20: ICMP echo reply, id 39, seq 8, l
64
20 packets captured
20 packets received by filter
0 packets dropped by kernel
root@n3:/tmp/pycore.36022/n3.conf# []

vcmd - + *
h 64
10:55:53.966398 IP 10.0.0.10 > 10.0.0.20: ICMP echo reply, id 39, seq 5, l
64
10:55:54.949518 ARP, Request who-has 10.0.0.20 tell 10.0.0.10, length 28
10:55:54.949558 ARP, Reply 10.0.0.20 is-at 00:00:00:aa:00:00 (oui Ethernet
gth 28
10:55:54.965405 IP 10.0.0.20 > 10.0.0.10: ICMP echo request, id 39, seq 6,
h 64
10:55:54.965415 IP 10.0.0.10 > 10.0.0.20: ICMP echo reply, id 39, seq 6, l
64
10:55:55.964398 IP 10.0.0.20 > 10.0.0.10: ICMP echo request, id 39, seq 7,
h 64
10:55:55.964428 IP 10.0.0.10 > 10.0.0.20: ICMP echo reply, id 39, seq 7, l
64
10:55:56.969852 IP 10.0.0.20 > 10.0.0.10: ICMP echo request, id 39, seq 8,
h 64
10:55:56.969879 IP 10.0.0.10 > 10.0.0.20: ICMP echo reply, id 39, seq 8, l
64
20 packets captured
20 packets received by filter
0 packets dropped by kernel
root@n4:/tmp/pycore.36022/n4.conf# []
```

Figura 27. Resultados do tcpdump em cada *host* da rede (ligados a um *hub*)

2. 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.

Após executado o comando `ping` na rede com *hosts* interligados por um *switch*, com origem no *host* n1 e destino ao *host* n4, verificaram-se os seguintes resultados após a execução do comando `tcpdump` num terminal de cada um dos *hosts* da rede:

```
vcmd                                     vcmd
12:23:35,234517 IP 10.0.0.20 > 10.0.0.11: ICMP echo request, id 39, seq 5, length 64
12:23:35,234562 IP 10.0.0.11 > 10.0.0.20: ICMP echo reply, id 39, seq 5, length 64
12:23:36,217373 ARP, Request who-has 10.0.0.20 tell 10.0.0.11, length 28
12:23:36,217373 ARP, Reply 10.0.0.20 is-at 00:00:00:aa:00:01 (oui Ethernet), length 28
12:23:36,248364 IP 10.0.0.20 > 10.0.0.11: ICMP echo request, id 39, seq 6, length 64
12:23:36,248385 IP 10.0.0.11 > 10.0.0.20: ICMP echo reply, id 39, seq 6, length 64
12:23:37,255661 IP 10.0.0.20 > 10.0.0.11: ICMP echo request, id 39, seq 7, length 64
12:23:37,255681 IP 10.0.0.11 > 10.0.0.20: ICMP echo reply, id 39, seq 7, length 64
12:23:38,262833 IP 10.0.0.20 > 10.0.0.11: ICMP echo request, id 39, seq 8, length 64
12:23:38,262856 IP 10.0.0.11 > 10.0.0.20: ICMP echo reply, id 39, seq 8, length 64
20 packets captured
20 packets received by filter
0 packets dropped by kernel
root@n1:/tmp/pycore_36022/n1.conf# []

vcmd                                     vcmd
root@n2:/tmp/pycore_36022/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
^C12:23:31,203451 ARP, Request who-has 10.0.0.11 tell 10.0.0.20, length 28
1 packet captured
1 packet received by filter
0 packets dropped by kernel
root@n2:/tmp/pycore_36022/n2.conf# []

vcmd                                     vcmd
root@n3:/tmp/pycore_36022/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
^C12:23:31,203448 ARP, Request who-has 10.0.0.11 tell 10.0.0.20, length 28
1 packet captured
1 packet received by filter
0 packets dropped by kernel
root@n3:/tmp/pycore_36022/n3.conf# []

vcmd                                     vcmd
12:23:35,234537 IP 10.0.0.20 > 10.0.0.11: ICMP echo request, id 39, seq 5, length 64
12:23:35,234558 IP 10.0.0.11 > 10.0.0.20: ICMP echo reply, id 39, seq 5, length 64
12:23:36,217357 ARP, Request who-has 10.0.0.20 tell 10.0.0.11, length 28
12:23:36,217381 ARP, Reply 10.0.0.20 is-at 00:00:00:aa:00:01 (oui Ethernet), length 28
12:23:36,248376 IP 10.0.0.20 > 10.0.0.11: ICMP echo request, id 39, seq 6, length 64
12:23:36,248382 IP 10.0.0.11 > 10.0.0.20: ICMP echo reply, id 39, seq 6, length 64
12:23:37,255672 IP 10.0.0.20 > 10.0.0.11: ICMP echo request, id 39, seq 7, length 64
12:23:37,255678 IP 10.0.0.11 > 10.0.0.20: ICMP echo reply, id 39, seq 7, length 64
12:23:38,262845 IP 10.0.0.20 > 10.0.0.11: ICMP echo request, id 39, seq 8, length 64
12:23:38,262852 IP 10.0.0.11 > 10.0.0.20: ICMP echo reply, id 39, seq 8, length 64
20 packets captured
20 packets received by filter
0 packets dropped by kernel
root@n4:/tmp/pycore_36022/n4.conf# []
```

Figura 28. Resultados do `tcpdump` em cada *host* da rede (ligados a um *switch*)

Como é possível verificar pelo tráfego observado, os dispositivos deixaram de partilhar o mesmo domínio de colisão graças à ação do *switch*. Assim, tal como referido no enunciado deste trabalho prático, cada porta do *switch* passa a constituir um domínio de colisão se a comunicação for *half-duplex*, ou seja, a comunicação entre 2 dispositivos ligados está limitada a ser unidirecional em cada instante, ou os domínios de colisão passam a deixar de existir caso a comunicação seja *full-duplex*, ou seja, a comunicação entre 2 dispositivos ligados pode ser bidirecional, o que significa que poderão emitir mensagens entre si em simultâneo. Com isto, é fácil explicar os resultados explícitos do `tcpdump`. O tráfego gerado pelo comando `ping`, executado no *host* n1, será encaminhado pelo *switch* para a porta que o liga ao *host* n4, não sendo assim propagado pelas outras portas para os meios de comunicação entre o *switch* e os restantes *hosts*. Dessa forma, os *hosts* não envolvidos na comunicação não recebem tráfego a ela relativo, sendo este apenas visível para os *hosts* que nela participam.

3 Conclusões

Como este trabalho prático foi possível ver aplicados diversos conceitos e tecnologias que atuam na camada protocolar de ligação lógica, como cabeçalhos Ethernet, a utilização de endereços MAC (*Medium Access Control*), áreas de aplicação do protocolo ARP (*Address Resolution Protocol*), as suas tabelas e os pedidos e respostas que permitem fazer o mapeamento entre endereços MAC (físicos) e IP (lógicos), assim como o envio de tramas correspondentes a ARP gratuito, que tanto podem ser *requests* ou *replies*. Para além disto, o relatório apresenta a resolução de exercícios sobre domínios de colisão, que surgem em redes locais onde vários dispositivos partilham um meio de comunicação. São também abordados possíveis métodos de controlo, divisão e eliminação de domínios de colisão.

Foi útil perceber o funcionamento das redes de comunicação a este nível, uma vez que várias noções aqui abordadas permitem resolver vários problemas que podem eventualmente surgir. Para além disto, a camada de ligação de dados providencia vários serviços dos quais as camadas superiores podem usufruir, o que também poderá servir para melhor compreender o funcionamento global de redes de comunicação, assim como outros níveis protocolares e a existência de serviços por eles prestados, que em certos casos se poderão revelar redundantes.