v.1.4.4 - 2019

Practical Exercises in Computer Networks

IP forwarding between directly connected stations, ARP and ICMP (WIP)

Internet is composed of a large number of interconnected networks, the most pervasive of them is the switched Ethernet LAN. We will use a switched Ethernet as a testbed for studying basic LAN communications. Specifically, we are interested in the mapping from logical, IP addresses to physical, MAC addresses. Our final goal consists of advancing our intuition about Ethernets and particularly IP before we actually delve into it in sections 3.2 and 3.3 of the textbook.

The communication between two LAN stations

In this section we study the communication between two computers that belong to the same IP network, specifically, the simple switched LAN of fig. 1. Let's start of with this simple switched LAN which will serve for illustrating the basic concepts involved in communicating two computers connected in the same network. The technical data included in the network diagram of that figure (IP addresses, MAC addresses, names, etc.) may not be the same at use on the specific lab network on which we will resolve the exercises included in this practical, please be advised of this fact. In the exercises section we will check the knowledge gained by using a real lab network and the Wireshark network analyzer.

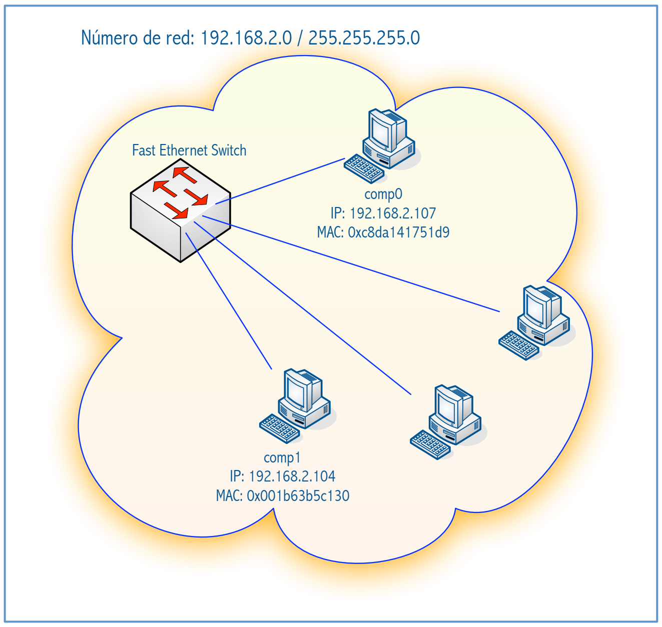


Fig. 1. Switched Lan for illustrating the communication between comp0 and comp1

The introduction to textbook chapter 3 explains the functions of a Lan Switch like that on fig. 1, now, simply recall that the switch builds a LAN that is functionally equivalent to an Ethernet, though much faster, more flexible and more secure. For the purpose of communicating comp0 and comp1, we will assume that the switch has already learned the MAC addresses of comp0 and comp1, then, the communication between them is possible at layer 2, however, this communication at layer 2 is not very practical from the standpoint of the final users and computer applications. In previous practicals we introduced how to program at the layer-2 (Datalink) with the Libpcap library which is essential in certain types of network programming which inherently work athe layer two, for example, the Wireshark network analyzer uses this library for receiving layer-2 traffic. Nevertheless, essential as it might be, this type of network programming is not by far the most used since distributed applications need to be written at much higher layers of abstraction where distributed systems transparencias may be implemented in a more straightforward fashion.

Certainly, most applications are written so that they use the *transport interface at layer 4,* more specifically, their network entry point is layer 5 where the service interface offers applications an end-to-end process-to-process communication abstraction known as Sockets (UDP and TCP). This end-to-end abstraction forces us to identify stations (hosts) by their *internetwork address or IP address*, that is the reason why we are IP-numbering our example network and each of its stations:

- IP **Network number** in DDN (**Dot Decimal Notation**): This is the 32-bit number that identifies and locates *a specific* network in the Internet. In this case the network number is: 192.168.1.0.
- **IP address mask**: This 32-bit number starts with a block of contiguous all-ones bits. When the mask *is applied* to an IP, its network number may be obtained. In this case the mask is: 255.255.255.0 which means that the first 24 bits of any in-range IP address constitute the network number.
- Stations receive the following example **IP addresses**: 192.168.1.104 and 192.168.1.107

To further emphasize the idea that the scope of physical, MAC addresses is the LAN, we state that, from a pragmatic standpoint, we may not communicate two computers if we only know their NICs MAC addresses, but, *we can communicate them only if we know their valid, logical, IP addresses*. In this case, in which we are only considering communication within a LAN, then we postulate the same requirement:

For communicating comp0 and comp1 we will have to use their valid IP addresses

Nevertheless, actual communications will proceed within the LAN by encapsulating each involved IP packet within a layer-2 frame. This latter frame has to contain the MAC destination and source addresses.

Valid station IP addresses can be assigned statically to each NIC, normally though, they will be provided by a DHCP server or a DHCP relay accessible in the considered LAN. We will describe DHCP in a later LAB exercise, for the time being, we will assume each station has obtained a valid IP address (192.168.1.4 and 192.168.1.7 in fig. 1).

Now that comp0 and comp1 have their IP addresses assigned, comp0 needs to communicate with comp1, then comp0 proceeds to determine whether comp1's IP, 192.168.1.4 belongs to its same network, to that end, comp0 applies its NIC's network mask to its IP address and obtains 192.168.1.0[1]. Since 192.168.1.4 belongs to comp0's network, comp0 knows that the station that owns that IP address is directly connected

[1] We will explain how to determine whether an IP address belongs to an IP network number in textbook section 3.2, for the time being we will simply mention that the operation that yields the Network Number of an IP address is:
IP & NetMask = 192.168.1.7 & 255.255.255.0 = 192.168.1.0

and that therefore there is no need to convey it to any router but simply encapsulate it into an Ethernet frame and transmit it over the LAN:

1. Encapsulate the IP packet we want to send to comp1 into a layer-2 frame (Ethernet). This frame will have comp0's MAC as source MAC and comp1's MAC address as destination address (See Fig. 1):

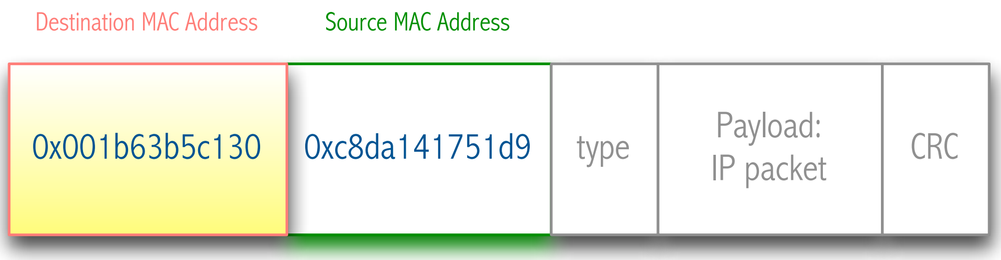


Fig. 1. IP packet encapsulated into an Ethernet frame

2. Send the resulting frame onto the network interface connected to the switch

Now we should ask ourselves how come comp0 is able to find out comp1 NIC's MAC address? As you can see, that MAC is required to *fabricate* the frame mentioned above, thus, how can comp0 find it out? We could try to devise a bunch of different mechanisms to accomplish the handing of comp1's MAC to comp0, but there is a standardized way to do that that is based upon a simple protocol known as ARP (Address Resolution Protocol): Comp0 uses its ARP software module to find out comp1's MAC, actually, in step 1 above, comp0 sends an ARP request for the MAC address of 192.168.1.4 before actually sending out the resulting frame.

In accordance with textbook section 3.2.6, where ARP is explained, in these exercises we assume an Ethernet LAN infrastructure where each NIC has a unique 48-bit MAC address and also has an IPv4 internetwork address which, probably, has been fetched by way of DHCP and which also belongs to the same IPv4 *network number*. Ethernets are broadcast networks that require the use of an *ARP mechanism* that translates logical addresses into physical addresses. In the networking world of today, the most common technology in use in *the last mile*[2] takes some form of Ethernet or Ethernet-like, *e.g.* 802.11N, thus, IPv4 ARP is an essential protocol for the health of the network.

Let's sketch now the ARP messages exchanged over time that will ultimately guarantee that comp0 obtains comp1's MAC address:

1. Comp0 sends an *ARP request* packet to the LAN's broadcast address containing the IP address of comp1. This packet is received by every host belonging to the broadcast domain, the response will be provided by comp1, though.

2. Comp1 responds with an *ARP reply* packet encapsulated in an Ethernet frame which destination address is the unicast address of comp0, this packet contains the MAC address of comp1, thus, when it is received by comp0, the ARP module at comp0 will record the newly discovered association in the local ARP table of comp0. New entries will last for about 20 minutes, after that time, it will be

[2] The user's Access network to Internet is usually referred to as *the last mile*. Today the network technology at use in the last mile is Ethernet, Wi-Fi or some form of 3G+.

erased, thereby causing a fresh copy to be obtained via ARP again and guaranteeing the coherence of the table in case intervening changes in stations NICs occur.

An example of ARP functioning

Let's illustrate ARP messages with a simple, actual switched Ethernet with two stations, let's name them station0 (192.168.1.101) and station1 (192.168.1.109). You will practice the same commands and processes that we develop in this example with a similar switched Ethernet in the laboratory, to that purpose, we first describe the equipment you will use.

Lab exercises related to layers 2 and 3 use network equipment that is used in homes and offices such as the Cisco E2000, this equipment includes a 4-port switched Ethernet and a separate Ethernet for Internet access. On the 4-port Ethernet the router offers us a DHCP server that can provide IPv4 addresses to the stations connected to it. Using a properly configured DHCP server is handy even in simple networks like that in this example. We will extend the 4-port switched LAN by using one more simple Ethernet switch so that the resulting LAN better represents real networks in which the switch provides increased access connectivity. Actually, you can perfectly reproduce this same example at your home network by using a single router.

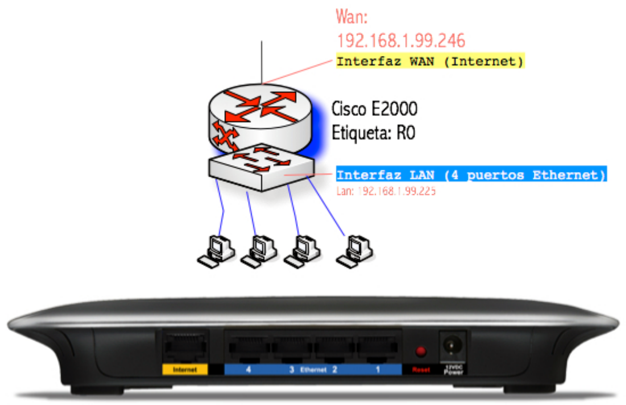


Fig. 2. A Cisco SOHO router

The Cisco E2000 and other models belonging to the same product family contain one IP router and a 4-port switch, the following figure illustrates the structure of the equipment:

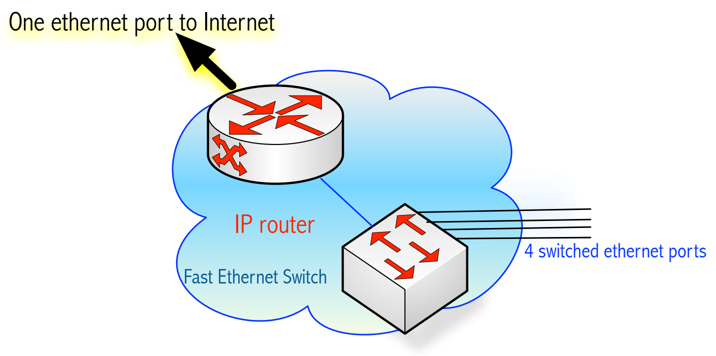
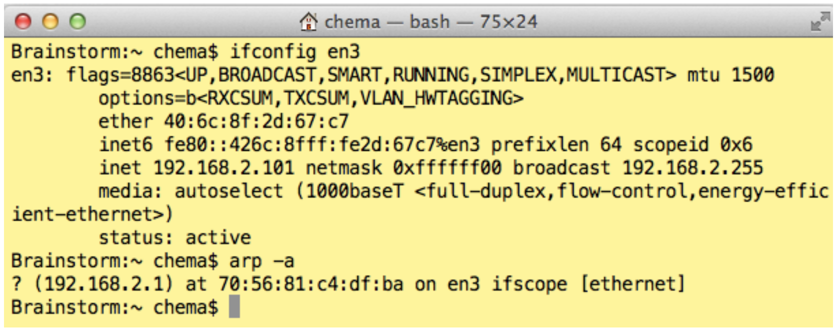


Fig. 3. A more abstract view of the Cisco SOHO router to be used in this exercise

The stations used in this illustration boot an Apple OS-X Unix system, we will explain the Unix commands that allow us to administrate the ARP tables of each NIC. Similar commands are available in other operating systems. The central command is arp.

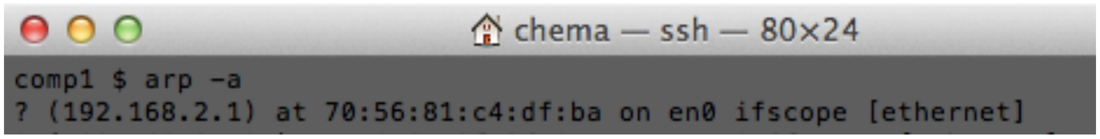
The arp command. This command offers us several flags and options to check the contents of the ARP cache, add entries into it, delete entries, etc. First we will determine the contents of the ARP cache:



```
Brainstorm:~ chema$ ifconfig en3
en3: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        options=b<RXCSUM,TXCSUM,VLAN_HWTAGGING>
        ether 40:6c:8f:2d:67:c7
        inet6 fe80::426c:8fff:fe2d:67c7%en3 prefixlen 64 scopeid 0x6
        inet 192.168.2.101 netmask 0xffffff00 broadcast 192.168.2.255
        media: autoselect (1000baseT <full-duplex,flow-control,energy-effic
ient-ethernet>)
        status: active
Brainstorm:~ chema$ arp -a
? (192.168.2.1) at 70:56:81:c4:df:ba on en3 ifscope [ethernet]
Brainstorm:~ chema$
```

Fig. 4. Contents of comp0's ARP cache

Only one entry is present so far: 192.168.2.1 and its MAC. This entry belongs to ethernet network interface `en3`. Now, we will check comp1's ARP cache:

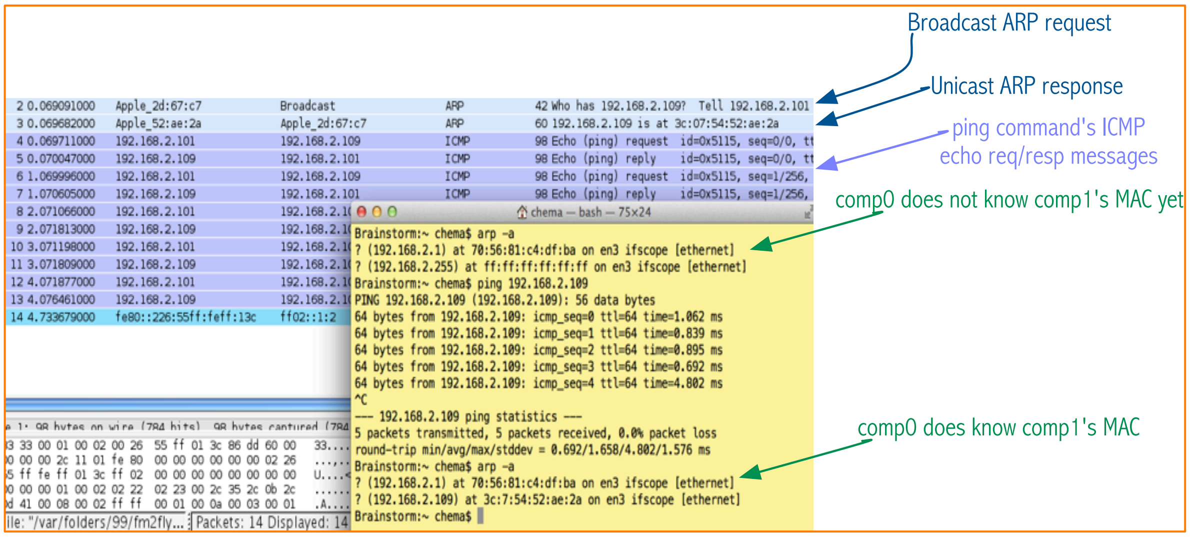


```
comp1 $ arp -a
? (192.168.2.1) at 70:56:81:c4:df:ba on en0 ifscope [ethernet]
```

Fig. 5. Contents of comp1's ARP cache

comp1 (192.168.2.109) likewise has an entry for 192.168.2.1, the default router for this network which is associated to Ethernet interface `en0` (BEWARE: Our current network prefix in Lab B6 is 192.168.1.0/24).

Now, we will contact comp1 from comp0 by using ping but, ping packets are encapsulated into Ethernet frames that must contain the destination address of the connected station, in our case, the destination address of comp1, which we do not know so far. When we submit the `ping` command, the IP stack will find out comp1's MAC by issuing an ARP REQUEST message over the network and wait for the ARP response from the target host, therefore, we will start Wirsehark to check this and then confirm the updating of the ARP cache of comp0:


Broadcast ARP request
Unicast ARP response
ping command's ICMP echo req/resp messages
comp0 does not know comp1's MAC yet
comp0 does know comp1's MAC

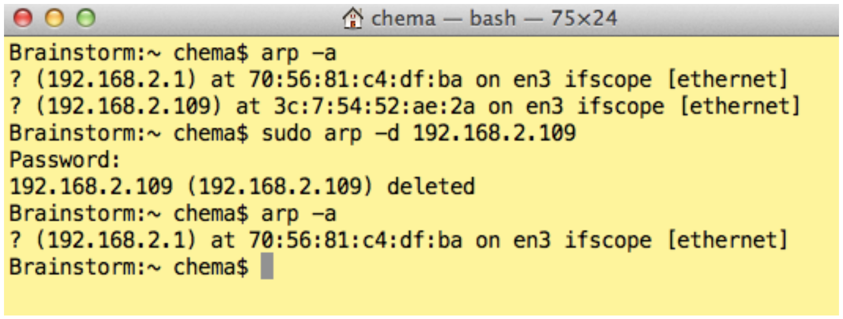
```
 2 0.069091000   Apple_2d:67:c7           Broadcast        ARP    42 Who has 192.168.2.109?  Tell 192.168.2.101
 3 0.069682000   Apple_52:ae:2a           Apple_2d:67:c7   ARP    60 192.168.2.109 is at 3c:07:54:52:ae:2a
 4 0.069711000   192.168.2.101            192.168.2.109    ICMP   98 Echo (ping) request  id=0x5115, seq=0/0, tt
 5 0.070047000   192.168.2.109            192.168.2.101    ICMP   98 Echo (ping) reply    id=0x5115, seq=0/0, tt
 6 1.069996000   192.168.2.101            192.168.2.109    ICMP   98 Echo (ping) request  id=0x5115, seq=1/256,
 7 1.070605000   192.168.2.109            192.168.2.101    ICMP   98 Echo (ping) reply    id=0x5115, seq=1/256,
 8 2.071066000   192.168.2.101            192.168.2.1(
 9 2.071813000   192.168.2.109            192.168.2.1(
10 3.071198000   192.168.2.101            192.168.2.1(
11 3.071809000   192.168.2.109            192.168.2.1(
12 4.071877000   192.168.2.101            192.168.2.1(
13 4.076461000   192.168.2.109            192.168.2.1(
14 4.733679000   fe80::226:55ff:feff:13c  ff02::1:2

e 1: 98 bytes on wire (784 bits), 98 bytes captured (784
3 33 00 01 00 02 00 26  55 ff 01 3c 86 dd 60 00   33....
0 00 00 2c 11 01 fe 80  00 00 00 00 00 00 02 26   ...,..
5 ff fe ff 01 3c ff 02  00 00 00 00 00 00 00 00   U....<
0 00 00 01 00 02 02 22  02 23 00 2c 35 2c 0b 2c   ......
d 41 00 08 00 02 ff ff  00 01 00 0a 00 03 00 01   .A....
ile: "/var/folders/99/fm2fly... Packets: 14 Displayed: 14
```

```
chema — bash — 75×24
Brainstorm:~ chema$ arp -a
? (192.168.2.1) at 70:56:81:c4:df:ba on en3 ifscope [ethernet]
? (192.168.2.255) at ff:ff:ff:ff:ff:ff on en3 ifscope [ethernet]
Brainstorm:~ chema$ ping 192.168.2.109
PING 192.168.2.109 (192.168.2.109): 56 data bytes
64 bytes from 192.168.2.109: icmp_seq=0 ttl=64 time=1.062 ms
64 bytes from 192.168.2.109: icmp_seq=1 ttl=64 time=0.839 ms
64 bytes from 192.168.2.109: icmp_seq=2 ttl=64 time=0.895 ms
64 bytes from 192.168.2.109: icmp_seq=3 ttl=64 time=0.692 ms
64 bytes from 192.168.2.109: icmp_seq=4 ttl=64 time=4.802 ms
^C
--- 192.168.2.109 ping statistics ---
5 packets transmitted, 5 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.692/1.658/4.802/1.576 ms
Brainstorm:~ chema$ arp -a
? (192.168.2.1) at 70:56:81:c4:df:ba on en3 ifscope [ethernet]
? (192.168.2.109) at 3c:7:54:52:ae:2a on en3 ifscope [ethernet]
Brainstorm:~ chema$
```

Fig. 6. Comp0 finds out comp1's MAC by issuing an ARP request message to the broadcast address

The arp command offers us other convenient command line options which, for example, allow you to delete one entry in the table (–d option) by specifying the IP address which association you want to delete, see the following example:



```
chema — bash — 75×24
Brainstorm:~ chema$ arp -a
? (192.168.2.1) at 70:56:81:c4:df:ba on en3 ifscope [ethernet]
? (192.168.2.109) at 3c:7:54:52:ae:2a on en3 ifscope [ethernet]
Brainstorm:~ chema$ sudo arp -d 192.168.2.109
Password:
192.168.2.109 (192.168.2.109) deleted
Brainstorm:~ chema$ arp -a
? (192.168.2.1) at 70:56:81:c4:df:ba on en3 ifscope [ethernet]
Brainstorm:~ chema$
```

Fig. 7. arp command *delete option*

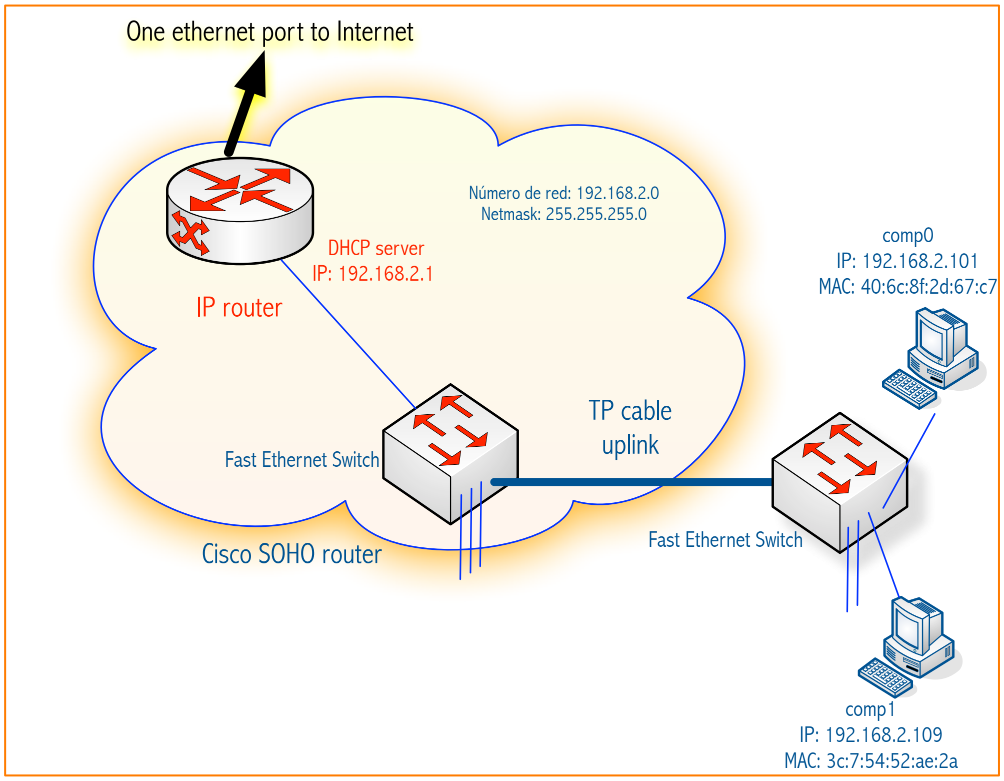


Fig. 8. The network used in the ARP request/response example

When a host arps for an IP address, it records the new entry in its arp table when it receives the arp response and the target host also records the IP/MAC of the requesting host, this is a normal functionality included in the protocol. Entries are aged after a typical period of 15 seconds, which age can be longer, depending on the operating system and network software stack version. The rest of network hosts, even though they *may* have seen the arp request containing the IP/MAC of the requester, they will not record those new entries that were not requested by them. However, if a host observes an arp transaction for an entry already in its arp table, it will reset its age timer to zero.

IP routers are more aggressive in their MAC address recording policy and will themselves arp for hosts for which they have snooped arp requests coming from other network hosts, that way, when eventually the router receives their potential traffic it will have already resolved the necessary IP/MACs. Recall that an IP router is the equipment that communicates several networks by using the IP protocol.

The arp protocol offers one more convenient functionality that serves to add consistency to the IP address assignments offered by DHCP servers, its name is Gratuitous ARP, it consists in a station *ARPing for itself*. Normally, no response should be elicited by this ARP request which means that no other node has obtained an IP address equal to that of the requesting host: this is the correct situation, however, in the process of dynamically assigning a host an IP address by a DHCP server, another host might have acquired that IP address by error which is clearly an undesirable situation. Gratuitous ARP does not correct the duplicate IP assignment problem, but warns the stations the problem has occurred and that corrective action on the network administrator is expected. Gratuitous ARP is normally generated by a station right after it has successfully received an IP address from its DHCP server.

Exercises for Lab Book (2019)

Exercise 1. Using the Lab B6's network, which is similar to the example of Fig. 8 above, check that the explanation about ARP holds. (Notice that the current internal IP Prefix address used in the Lab's network is 192.168.1.0/24 and not 192.168.2.0/24 as it appears on the network diagrams included in the present practice).

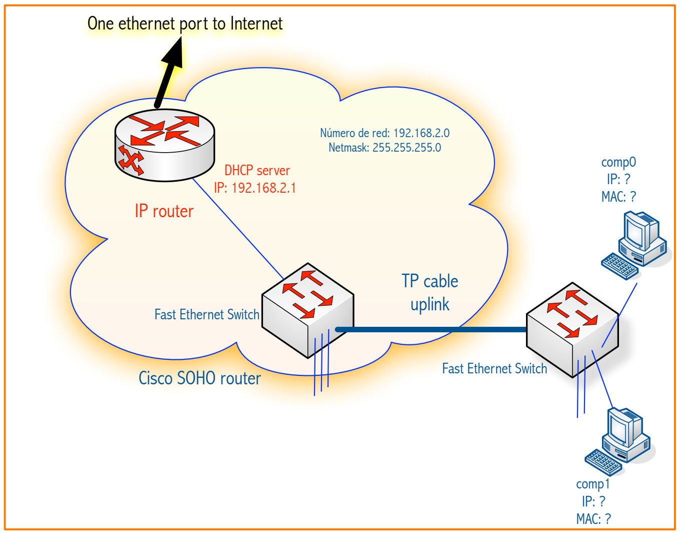


Fig. 9. The testbed network for checking ARP protocol messages

1. Power up the router and the additional switches, if any. Reset the router by pressing the reset button for 5 seconds while turning it on, from then on, it will present an IP 192.168.1.1 on the Ethernet interface.
2. Connect two stations to your switch, both must be configured to obtain their IP addresses from a DHCP server. Once booted, test their IP's by submitting an ifconfig command. Take note of their IPs, use the network diagram in Fig. 9 for taking your notes and comments.
3. Now, start Wireshark on the station that is going to initiate the communications (comp0)
4. From comp0 send a ping to comp1's IP
5. Stop the packet capture and try to identify the frames involved in the process explained above which we will briefly sketch here:
    a. Comp0 wants to find out comp'1 MAC by sending an ARQ request to the broadcast address
    b. Comp1 responds by sending its ARP response to comp0 in a unicast ARP response frame
    c. Pings to comp1 proceed normally, stop them after a few of them are displayed
    d. Check that comp0 ARP cache, now contains an entry for comp1 and, conversely, check that comp1 ARP cache now contains an entry for comp0 (In both cases, issue an arp –a to obtain a listing of the arp entries currently available in the arp cache)

In the next LAB exercise we will study the same process explained here but applied to the situation in which comp0 has to communicate with a node that is located in a network other than its own, in the internetwork.

Exercise 2. Check the Wireshark screen dump above and confirm that ping command protocol messages contain ICMP echo request and ICMP response messages. Use the textbook to consult the different types of ICMP messages.

Exercise 3. Use Wireshark to determine the encapsulation hierarchy of ICMP messages above, explain that encapsulation hierarchy, *i.e.,* an Ethernet frame contains an ICMP message:

a. What protocol number is used in the Ethernet frame's type field? It must be 0x0800 which means IP, *i.e.,* the Ethernet frame contains an IP packet. Wireshark display filter "icmp" may be useful.
b. Now, what payload is being carried into the IP packet? Check that the IP packet's protocol field contains 1 which means *ICMP.* Explain the steps that ultimately deliver the frame's payload to the IP stack's ICMP module.

Exercise 4. Employ the strategy given above to determine the encapsulation hierarchy of ARP messages above, explain that encapsulation hierarchy:

a. An Ethernet frame contains an ARP message, what protocol number is used in its type field? Explain the steps that ultimately deliver the frame's payload to the IP stack's ARP module. Wireshark display filter "arp" may be useful.

b. If you want to force the arp module to request again a certain mapping, delete the target entry first by issuing the following *example* arp command:

```
$ arp –d 192.168.1.119
```

   That command will delete the mapping IP:MAC corresponding to host 192.168.1.119; the next time that IP host is used, the arp module will send a new ARP request for 192.168.1.119 and you will be able to capture it from Wireshark.

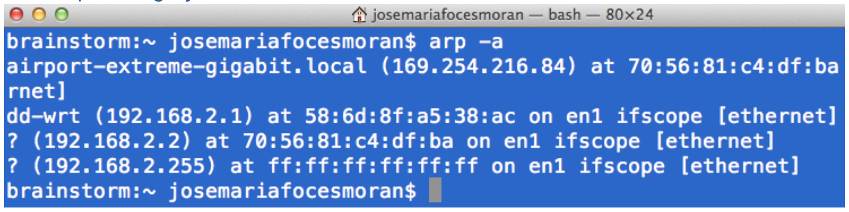
c. Study the contents of the arp request packet:
   i. What are the "Hardware type" and "Protocol type" and their sizes? Do the sizes make sense according to your current networking knowledge?
   ii. What are the "Sender IP and MAC address" and the "Target IP and MAC address"?
   iii. The Ethernet frame containing this arp request contains some broadcast address? Can you explain why?

d. Study the contents of the arp reply packet and check that all makes sense.

Exercise 5. Assuming that your prototype network is up and running (Fig. 9), list the entries of your arp module, issue the command mentioned above (If necessary, add /usr/sbin to your shell's PATH environment variable): `$ arp –a`, then, explain each of the resulting entries, particularly, we want to find out the purpose of the following entries:

a. Entry that maps an IP to MAC address ff:ff:ff:ff:ff:ff, what IP is that? What is its purpose? For example, observe the arp entry for 192.168.2.255 in the following screen dump from the

OS-X operating system:



```
brainstorm:~ josemariafocesmoran$ arp -a
airport-extreme-gigabit.local (169.254.216.84) at 70:56:81:c4:df:ba
rnet]
dd-wrt (192.168.2.1) at 58:6d:8f:a5:38:ac on en1 ifscope [ethernet]
? (192.168.2.2) at 70:56:81:c4:df:ba on en1 ifscope [ethernet]
? (192.168.2.255) at ff:ff:ff:ff:ff:ff on en1 ifscope [ethernet]
brainstorm:~ josemariafocesmoran$
```

b. What do you think will happen if you send a ping to the broadcast IP address of your LAN? On the lab network, find your IP and netmask and then issue a ping command to the broadcast address:

   $ ping 192.168.1.255

   Your host must receive ICMP echo replies from all hosts in your LAN and will record their IP/MAC combinations into their arp table. Check this in Linux:

   $ arp –v

c. Write down in your notes the MAC address (Layer 2) corresponding to your LAN broadcast IP (Layer 3), begin by $ ifconfig eth0, then observe the IP address and netmask, then, think what the MAC corresponding to the broadcast IP will be.

d. In the lab network, study the entry for IP 192.168.1.1 (The network's default router) which is not a strictly *normal station*, can you say what it is? What is its purpose? Can we state that this entry, at layer-2 behaves exactly alike any other station MAC?

Exercise 6: Gratuitous arp. This exercise is based on a network similar to that in fig. 9. Assume in this exercise that comp0 and comp1 are assigned the following valid static IPs (These IP's will be valid in the laboratory's network, so you can play with them): 192.168.1.100/24 and 192.168.1.120/24, these IPs are *assigned at boot time,* i.e., they are not dynamically assigned by using DHCP. Once the systems have booted up successfully, assume that comp0 sends a ping to comp1, then, work the following questions:

a. Discuss the validity of the following statement: *Since both stations set their IP's statically (No DHCP), the switch knows neither comp0's MAC nor comp1's, then, the ping command will not be able to elicit an ICMP echo response from comp1.*

b. Now assume that both stations get their IPs from a DHCP server and that they both have been booted up successfully. Now we connect a third station to the switch and make sure this last station is running Wireshark and capturing on the relevant network interface, then we ask whether some of comp0 or comp1 sends a Gratuitous ARP: can we watch this in this system running Wireshark? Explain with detail and consider what Ethernet equipment could be of help in this situation so that we accomplish the proposed task: watch *all* the DHCP messages involved. in this case, you may find it useful to use the following WIreshark display filter: *arp.isgratuitous*.

Exercise 7: ARP Return function. This ARP function consists in the target of an ARP transaction recording the requester's ARP association into its ARP table. Check it by issuing a ping against a host which IP/MAC association is unknown to your host (The requester) yet, then check that your host has finally recorded the new arp entry and that the target has also recorded an entry for the requester's IP/MAC. If necessary, review the above explanation (Pg. 7 of this document)on arp's return function.

Exercise 8: Aging arp entries.

a. Devise a procedure to determine how much time your host takes to delete a newly recorded ARP entry.
b. Assume that host A has an arp entry for host B, what happens to that entry when any other host arps for B? You may want to consult the textbook section on ARP or read further on pg. 7, then, once you know what is supposed to happen, try to figure out how to prove it practically.

Exercise 9: IP routers record arp entries more *aggressively* than normal hosts.

a. As explained above (Pg. 7), IP routers snoop arp activity on the neighboring LAN, every time they see a new arp resolution, they will arp for it too and make a new table record. Think how you would observe this behavior by loging into protocol.unileon.es. Explain your experiment design with detail, then carry it out and document the obtained results and conclusions.

Exercise 10: Sending arp requests with libpcap.

a. Program an arp requester program in C with libpcap which accepts the target host's IP address from the command line; the program should capture the unicast arp response packet and print it out.