Universidad de León
Bachelor Degree on Computer Science and Engineering
Course on Computer Networks

Weekly Homework no. 7 (WH7-Theory)

Weekly Homework (WH7-Theory)

Exercises

1. The *raw sockets* that we have been using the past weeks provide access to the services offered by the *datalink layer*, in our case, Ethernet. Those sockets, in Linux, are created with the socket() syscall by passing it a protocol family of PF_PACKET; the type of socket can be selected among SOCK_RAW and SOCK_DGRAM.

    a. What's the technical name for the service interface that provides access to layer 2, the IP layer? Is that *raw sockets,* also?

    PF_INET/SOCK_RAW

    b. Type the C code that invokes the socket() syscall that will create a socket that accesses the IP layer interface.

    ```
    int mySock = socket(PF_INET, SOCK_RAW, myProtoNumber);
    ```

2. You are to build a new implementation of the *Longest Prefix Matching* algorithm.

    a. Which layer's interface would you have to access to receive the IP packets that your algorithm is to forward? In other words, what specific type of socket would you need to create?

    PF_INET/SOCK_RAW with IP header included so that my program can do the look up of the Destination IP in the Forwarding Table

3. In WH5-Practice you modified a program that sent one message 'Andra tutto bene' to another host located in the same network. In the present lesson, we are studying what is necessary for a host to send a message like the latter to a host located in a separate network.

    a. Can we do what we did in WH5-Practice and simply encapsulate that message, *directly* in an Ethernet frame and have it sent to the network's default router, somehow? Explain why this is not the way to proceed.

The IP router cannot forward *any* payload but IP packets; the message encapsulated in the sent Ethernet frame and indexed with Ethertype 0x07ff, for example, cannot be forwarded by the router.

b. What would you have to do in this case about communication across different networks?

An IP packet should be sent containing the correct SRC and DST IP addresses, Protocol field, etc. and containing the intended payload. This IP packet should be encapsulated in an Ethernet frame along with an Ethertype of 0x0800 and sent to the default router; the router's interface MAC is obtained with ARP.

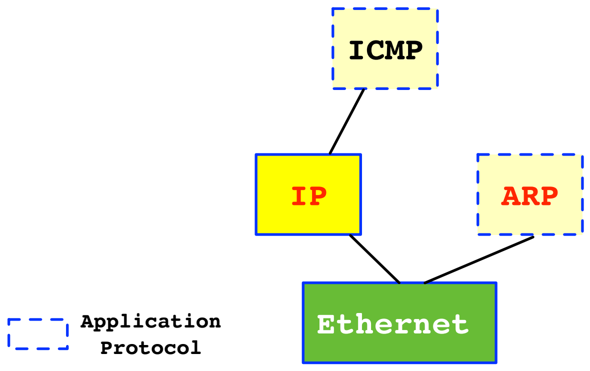
4. A router that receives an IP packet proceeds to forward it by providing the packet's destination IP address to the *Longest Prefix Matching* algorithm. The destination IP is the key used by the router for making its best approximation to the real location of the host in Internet. The host's IP address is used for *locating* that host in Internet. Yet, when the packet has to leave the router for its next hop[1], a transmission onto a router's link must be done which will involve the MAC address of the router's selected NIC as source and the next router's receiving NIC's. That is why the typical Internet protocol stack needs a *protocol* that maps an IP address to its MAC address.

a. What is the name of the protocol that implements that IP-to-MAC mapping function in the Internet Architecture?

ARP

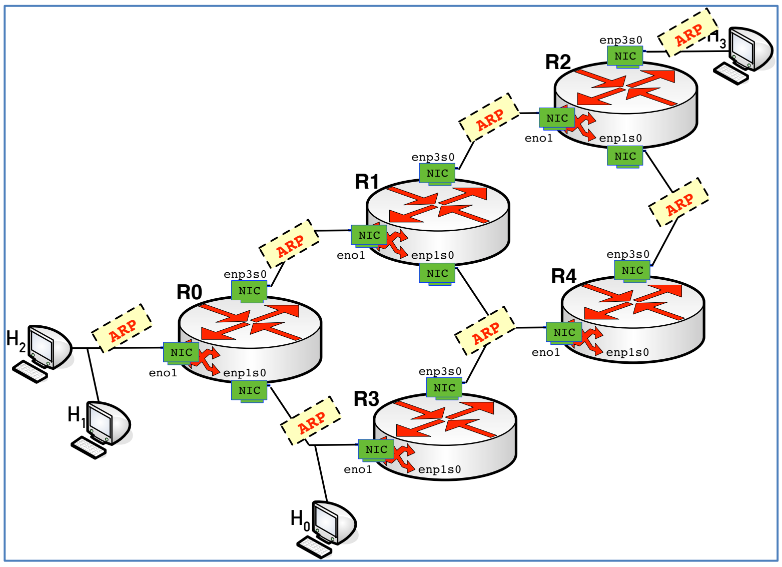
b. Which layer should ARP reside on according to the essential function that it provides? Depict the resulting protocol stack and contrast that protocol stack to that of the ICMP protocol.

ARP is an Application Protocol ancillary to IP per IETF's Host Requirements RFC



[1] Consider the literal sense of *to hop* in English

c. Put a "arp" label on all the networks in Fig. 1 where ARP resolution is needed for the correct functioning of the Internet Architecture in the internetwork.



5. In this exercise, like we did above, you are to write a new implementation of the ARP protocol, in user space in Linux. Outline the essential technical considerations necessary for a programmer to implement such ARP as a *normal program* (In Linux, the ARP protocol is implemented in kernel space in a subsystem known as *the neighboring subsystem*).

- Netlink socket to discover our MAC addresses
- Datalink raw socket to receive ARP packets (Ethertype = 0x0806)
```
int sock = socket(PF_PACKET, SOCK_DGRAM, htons(0x0806) );
```
- Analyze every ARP request for one of our MAC addresses and send response

6. Consider the internetwork in Fig. 1 for the following questions; assume that all the ARP tables are empty at the time:

a. H_1 sends a packet to H_2. How many arp resolutions are required before the packet is sent?

One, if successful

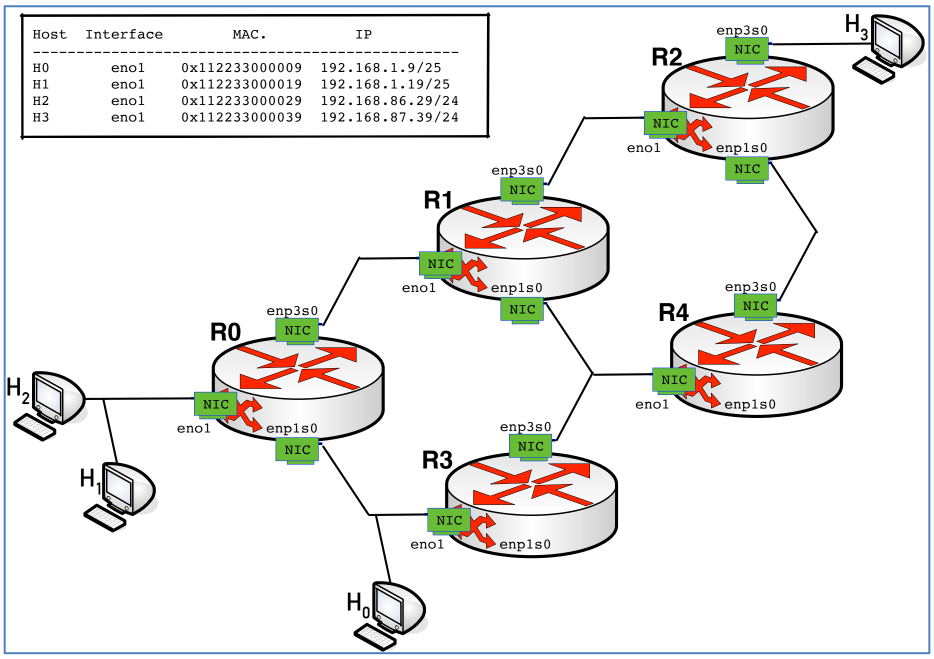
b. H_0 is directly connected to R_0, since H_2 is also connected to R_0, how many ARPs are necessary before H_0 sends a packet to H_2?

H_0 should ARP for R_0's IP address; Router R_0 should arp for H_2's IP before reencapsulating the forwarded IP packet sent by H_0

c. Right after H_0 sends a packet to H_2, H_0 sends a packet to H_3 over the path H_0 – R_0 -R_1 – R_2 – H_3.

d. Devise a method for having H_2 receive the MAC address of H_3. Does that have any usefulness?

- H_2 could communicate with an IP-based server program running in H_3 which would provide us the MAC.



```
Host  Interface         MAC.            IP
-------------------------------------------------
H0        eno1     0x112233000009  192.168.1.9/25
H1        eno1     0x112233000019  192.168.1.19/25
H2        eno1     0x112233000029  192.168.86.29/24
H3        eno1     0x112233000039  192.168.87.39/24
```

Figure 1. All links in this internetwork are Ethernets

7. Search the Internet for the RFC to ARP protocol and respond to the following questions:

a. What's the number and title of the RFC?

Search the Internet for "rfc arp"

b. Skim the whole document and sketch the basic ARP Request/Response process in general, then describe each Ethernet frame involved when H_0 resolves H_1 with ARP (Fig. 1). Include the relevant MAC addresses which appear on the diagram.

Skim the RFC

c. Continuing with the question in the preceding section, H_0 receives the MAC of H_1 successfully and shortly afterwards uses it in an IP communication with H_1. Later, a user logged in H_1 starts a tcpdump trace with the following command:

```
[internal 29] $ tcpdump -i eno1 -xe -vvv arp
```

The user observes the following ARP Request/Response:

```
Request
DMAC 0x112233000019
SMAC 0x112233000009
Ethertype 0x0806
Who has 192.168.1.19? Tell 192.168.1.9[2]

Response
DMAC 0x112233000009
SMAC 0x112233000019
Ethertype 0x0806
192.168.1.19 is at 0x112233000019
```

Does this transaction fully comply with ARP protocol?

Yes, this ARP transaction is fully compatible with ARP protocol. It's meaning can be understood by carefully studying the relevant ARP RFC.

8. How many bits are employed for representing an IPv4 address?

32 bits

9. Calculate how many distinct IPv4 addresses are possible?

2^{32}

10. What's the technical name to the notation most commonly used for the exchange of IP addresses?

DDN (Decimal Dot Notation)

11. What are the first and the last IPv4 addresses? Represent them using the notation mentioned in the preceding section and, also as unsigned integers in binary and in hexadecimal.

0.0.0.0 $\equiv 0_{10} \equiv$ 0x0

255.255.255.255 $\equiv 4294967295_{10} \equiv$ 0xffffffff

12. What encoding applies to IP addresses: unsigned integer, signed integer (2's complement) or maybe IEEE-854?

unsigned integer; 32 bits of precision

[2] This message is a *friendly* form of the ARP message carried in the ARP packet used by Wireshark and tcpdump. What the ARP packet actually carries is an encoded form of that message that saves space and is more suitbale to be interpreted by the receving ARP.

13. IP addresses, in contrast to MAC addresses, do have structure. The 32 bits that constitute an IP address are composed of two blocks. Starting with the MSb[3], which is conventionally located on the leftmost position when written down, the ensuing bits, from left to right, constitute a block of bits known as the network number. Next to the last bit comes the first bit of the Host number which extends all the way through the LSb. IP address 192.168.1.123 has its network number and its host number. The boundary between both blocks of bits is indicated by appending a small integer to the IP address that specifies the number of bits comprising the first block, the network number. In the preceding IP address, a notation like **192.168.1.123/24** indicates that the network number is comprised of the 24 most significant bits, leaving the least 32-24 = 8 significant bits for the host part. The network number can be calculated by *zeroing* the host part:

Set the lower 8 bits of 192.168.1.123 to zero = 192.168.1.000;

Network number = 192.168.1.0

a. **Solved.** Given prefix number 192.168.1.123/23, calculate the network number.

   In this case, we are to set zeros on the lower 9 bits and let the network part untouched yielding the following result in this case: 192.168.0.0

b. **Solved.** An alternate procedure to *compute* the network part consists of observing that we are setting zeros on the 9 bits from the host part and letting the remaining bits from the network part untouched. The bitwise AND operation (In the C language it is represented by operator &) applied to the IP address lets us *selectively* set a zero on some bits and let the others untouched. For example, the following operation is to set the odd bits of a data byte to zero and letting the others unaffected. We have to *design* a constant byte consistent with the bits that we want to be set to zero and which ones are left the same as in the original byte:

```
              7654 3210
Data byte:    1101 1001
Constant:   & 0101 0101
----------------------
              0101 0001
```

   Check that the odd bits in the result are all 0's and the the even bits are the same as the original bits included in the data.

[3] MSb means *Most Significant bit*

c. **Solved.** Back to prefix 192.168.1.123/23, this time we should wish to calculate the network number by setting the lower bits to zero by using the preceding technique involving the & operator and a *well thought* constant. By the way, the constant has a technical name in the IP addressing field: **network mask or netmask**. Familiar? Yes! We saw that when working the first practice of this course. Time to set the lower 9 bits to 0 by using netmasks.

```
Given network prefix: 192.168.1.123/23

IP address (Dec):    192      168      1        123
IP address (Bin):    11000000 10101000 00000001 01111011
Netmask for /23:   & 11111111 11111111 11111110 00000000
-------------------------------------------------------
Network number:      11000000 10101000 00000000 00000000
```

Check that the lower 9 bits of the result are all 0's and that the remaining upper bits all have the same values as in the given network prefix. The constant /23, which represents the length of the upper block of bits 1 in the netmask, is known as the CIDR Prefix.

d. **To do.** Calculate the network number corresponding to DNS name paloalto.unileon.es which IP prefix is 193.146.101.46/20.

```
  193.146.101.046
& 255.255.240.000
-----------------
  193.146.096.000
```

If necessary, apply the bit-wise & to 101_{10} and 240_{10} in binary.

e. **Solved.** What is the size of the block of contiguous IP addresses represented by the preceding prefix? Recall that IPv4 addresses are comprised of 32 bits. Since the CIDR prefix is 23 in this case, the number of bits remaining for being assigned to the hosts in that network is: 32 -23 = 9 bits. With 9 bits we have 2^9 IP addresses available or $2^9 = 512$. The first of those IP addresses is the network number ~~(192.168.0.0)~~ 193.146.96.0 which was calculated above.

193.146.096.000/20

First address: $193.146.096.000_{DDN}$
Last address = First address + (Size – 1)
Size – 1 = $(2^{32-20} - 1) = (2^{12} - 1) = 4095_{10}$

4095_{10} = … $0000\ 1111\ 1111\ 1111_2$ = $000.000.015.255_{DDN}$

```
Last address = 193.146.096.000 + 4095
             = 193.146.096.000
             + 000.000.015.255DDN
--------------------------------
               193.146.111.255DDN
```

To learn a little more, we can do the same calculation by using masks. The last address in an IP block is obtained by applying NOT MASK to any IP address contained in the block with the bit-wise OR operation:

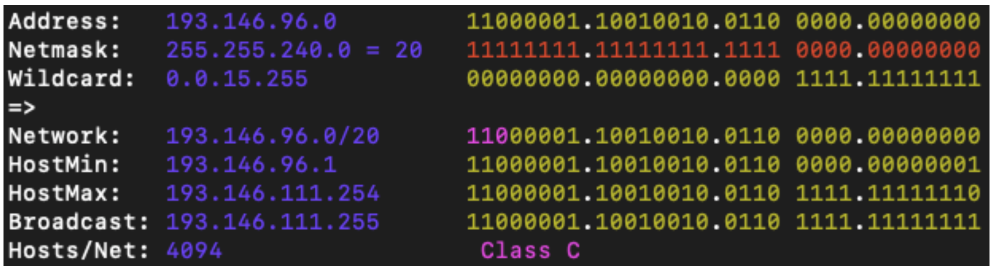
```
not(mask) = not(255.255.240.0DDN) = 000.000.015.255DDN

  193.146.101.046DDN any address from the IP block
| 000.000.015.255DDN
-------------------
  193.146.111.255DDN
```

```
An IP block's last address is reserved in every
network for the broadcast address.
```

Finally, we recommend that you check your calculations with Linux ipcalc:

```
networks@protocol:~$ ipcalc 193.146.96.0/20
```



```
Address:   193.146.96.0         11000001.10010010.0110 0000.00000000
Netmask:   255.255.240.0 = 20   11111111.11111111.1111 0000.00000000
Wildcard:  0.0.15.255           00000000.00000000.0000 1111.11111111
=>
Network:   193.146.96.0/20      11000001.10010010.0110 0000.00000000
HostMin:   193.146.96.1         11000001.10010010.0110 0000.00000001
HostMax:   193.146.111.254      11000001.10010010.0110 1111.11111110
Broadcast: 193.146.111.255      11000001.10010010.0110 1111.11111111
Hosts/Net: 4094                  Class C
```

f. **To do.** We already have calculated the first address from the block of IP addresses represented by the prefix, now, could you calculate the last address in the IP block? Hint: IP addresses are integers, consequently you can add an integer to an IP address and obtain an IP address!

g. **To do.** Repeat the calculation done in section c. of this question with the help of the ipcalc Linux utility program. If you don't have installed in your Linux system, have it installed using this command:

```
[Your prompt!] $ sudo apt-get install ipcalc
```

If the installation is successful, use the great ipcalc utility:

```
[Your prompt!] $ ipcalc 192.168.1.123/23
```

```
Address:   192.168.1.123        11000000.10101000.0000000 1.01111011
Netmask:   255.255.254.0 = 23   11111111.11111111.1111111 0.00000000
Wildcard:  0.0.1.255            00000000.00000000.0000000 1.11111111
=>
Network:   192.168.0.0/23       11000000.10101000.0000000 0.00000000
HostMin:   192.168.0.1          11000000.10101000.0000000 0.00000001
HostMax:   192.168.1.254        11000000.10101000.0000000 1.11111110
Broadcast: 192.168.1.255        11000000.10101000.0000000 1.11111111
Hosts/Net: 510                   Class C, Private Internet
```

Finished, all is there. We'll discuss this example briefly in the next lecture.