author: Riccardo Macoratti <r.macoratti@gmx.co.uk>
site: http://www.linuxvar.it
date: 29 febbraio 2016
revision: 007
description: paper for Linux Presentation Day (April 30, 2016)

Indice

Requisiti di base

- Sapere cos'è una rete
- Protocollo **ISO/OSI**
- Un minimo di conoscenza delle periferiche e dispositivi hardware e software impiegati (RJ45, protocollo Ethernet, router, switch, gateway, IP, MAC Address, servizi canonici come HTTP, DNS, MAIL, FTP...)

Utility usate

- `ip` (`ifconfig`)
  - man page
  - tutorial
- `ss` (`netstat`)
  - man page
  - tutorial
- `traceroute`
  - man page
- `tcpdump`
  - man page
  - tutorial
- `ping`
  - man page
  - tutorial
- `drill` (`dig`)
  - man page
  - tutorial
- `nmap` (`zenmap`)
  - man page
  - tutorial
- `wireshark`

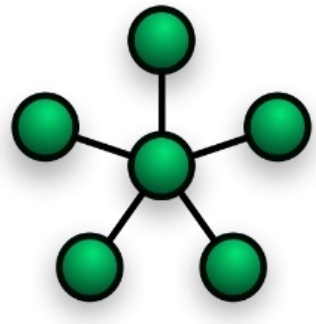
- tutorial

Un po' di teoria

Definizione di rete

Ripassiamo ora la definizione di una rete di calcolatori:

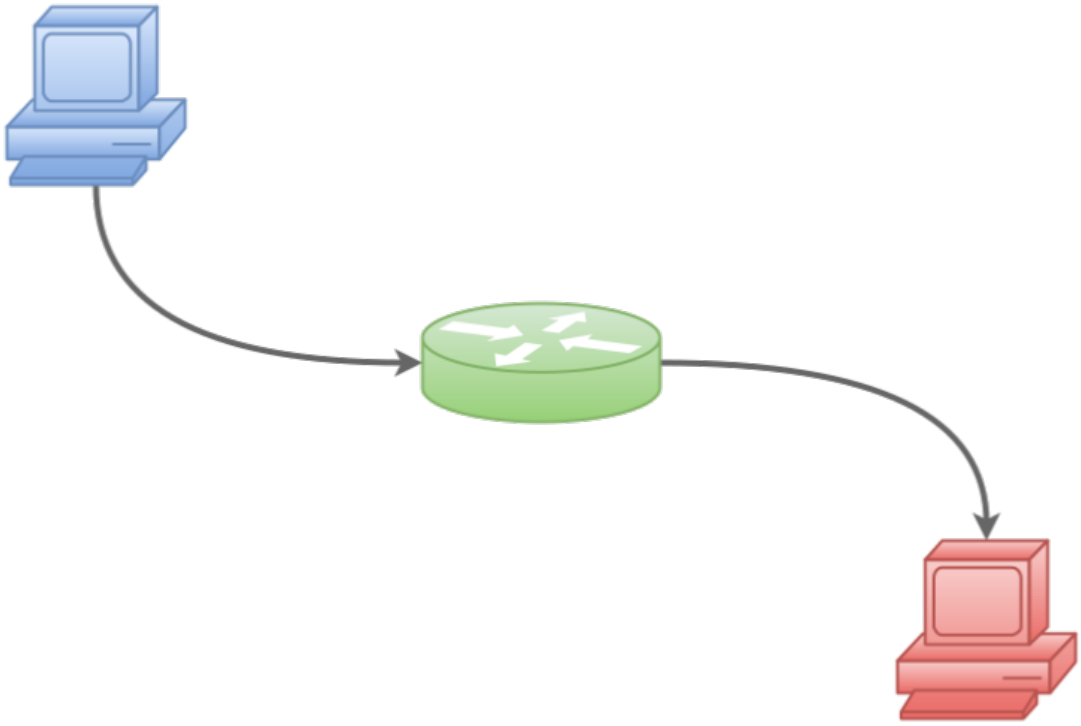
> "una rete di calcolatori è un'insieme di nodi di elaborazione totalmente autonomi tra loro, connessi mediante un opportuno sistema di comunicazione, ed in grado di interagire mediante scambio di messaggi al fine di condividere le risorse messe a disposizione da ciascon nodo"



Uno schema per visualizzare cos'è una rete. (Wikipedia)

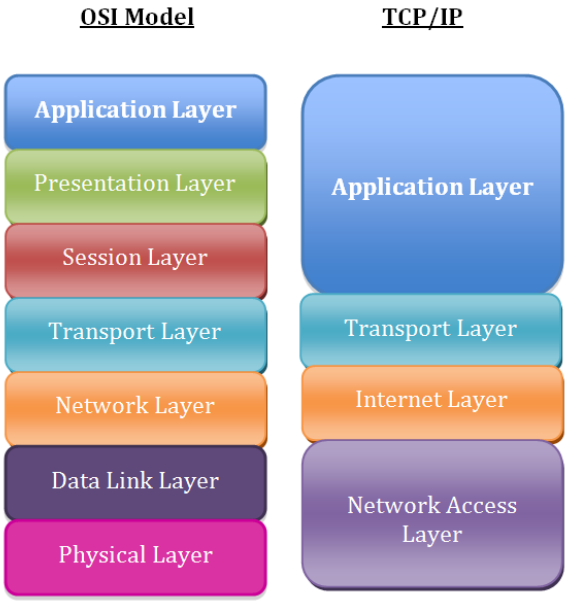
Com'è fisicamente fatta una rete

Come ci si deve immaginare una rete?



Due calcolatori collegati da una rete. Al centro ci può essere uno switch, un router oppure (con particolari accorgimenti) anche nulla!

Modello ISO/OSI



Il diagramma ISO/OSI, che mostra l'incapsulamento dei vari protocolli, accanto ad una specifica per il protocollo TCP. (Wikipedia)

Un po' di pratica

`ip` (`ifconfig`)

`ip` è la nuova versione del comando `ifconfig` che era presente nelle versioni più datate dei sistemi operativi Linux. Questo comando serve per compiere un gran numero di azioni sulle interfacce di rete [1]. Le interfacce di rete sono tante quante sono le schede di rete (ossia quelle periferiche che ci connettono ad un'altra macchina o ad Internet), più una detta di *loopback*, che è associata alla nostra macchina e rappresenta una sorta di specchio (tutto quello che le inviamo ci viene restituito nella stessa maniera.

> [1] Un'interfaccia di rete è il corrispettivo della scheda di rete, analizzata dal punto di vista del sistema operativo.

Esempi

```
# mostrare tutte le interfacce di rete
$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 30:52:cb:83:fc:07 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.143/24 brd 192.168.1.255 scope global eth0
       valid_lft forever preferred_lft forever
```

```
# aggiungere un cammino statico verso indirizzi ip multicast
$ ip route add 224.0.0.0/4 dev eth0
```

`ss` (`netstat`)

`ss` è la nuova versione del comando `netstat`, che era presente nelle vecchie versioni dei sistemi Linux. Questo comando serve per analizzare le connessioni in entrata e in uscita nella nostra macchina. Riporta una lista di connessioni aperte riguardanti tutti i protocolli, anche quelli interni al sistema operativo. Tramite delle impostazioni è possibile filtrare la lista per mostrare solo le reali connessioni verso altre macchine (spesso nei protocolli TCP e UDP).

Esempi

```
# mostrare tutte le connessioni in atto
$ ss
Netid State      Recv-Q Send-Q     Local Address:Port       Peer Address:Port
u_str ESTAB      0      0          * 25096                  * 25098
u_str ESTAB      0      0          * 26081                  * 26082
u_str ESTAB      0      0          /run/user/1000/bus 25117 * 25116
u_str ESTAB      0      0          * 24099                  * 24100
u_str ESTAB      0      0          /run/user/1000/bus 25999 * 23949
# [...]
```

```
# mostrare solo le connessioni che usano il protocollo IPv4
$ ss -4
Netid State      Recv-Q Send-Q  Local Address:Port    Peer Address:Port
tcp   CLOSE-WAIT 1      0       159.149.248.41:51912  52.21.197.186:https
tcp   ESTAB      0      0       159.149.248.41:48320  108.160.163.110:https
tcp   LAST-ACK   1      1       159.149.248.41:22622  54.174.160.243:https
tcp   CLOSE-WAIT 32     0       159.149.248.41:41624  54.192.1.110:https
tcp   CLOSE-WAIT 32     0       159.149.248.41:36258  108.160.172.193:https
tcp   ESTAB      0      0       159.149.248.41:47954  74.125.71.120:http
tcp   CLOSE-WAIT 32     0       159.149.248.41:64838  108.160.172.204:https
tcp   ESTAB      0      0       159.149.248.41:18266  216.58.212.66:https
```

```
# mostrare solo le connessioni che usano il protocollo IPv4, il processo
# associato e sia in entrata sia in uscita
$ ss -nap4
Netid State      Recv-Q Send-Q  Local Address:Port    Peer Address:Port
udp   UNCONN     0      0       *:68                  *:*
tcp   LISTEN     0      128     127.0.0.1:17603       *:*                 users:(("dropbox",pid=10535,fd=65))
tcp   LISTEN     0      128     *:17500               *:*                 users:(("dropbox",pid=10535,fd=59))
tcp   LISTEN     0      128     127.0.0.1:17600       *:*                 users:(("dropbox",pid=10535,fd=61))
tcp   CLOSE-WAIT 32     0       159.149.248.41:40642  54.192.1.110:443    users:(("dropbox",pid=10535,fd=37))
tcp   ESTAB      0      0       159.149.248.41:48320  108.160.163.110:443 users:(("dropbox",pid=10535,fd=66))
tcp   CLOSE-WAIT 32     0       159.149.248.41:36258  108.160.172.193:443 users:(("dropbox",pid=10535,fd=38))
tcp   ESTAB      0      0       159.149.248.41:18258  216.58.212.66:443   users:(("firefox",pid=10544,fd=61))
tcp   CLOSE-WAIT 32     0       159.149.248.41:40646  54.192.1.110:443    users:(("dropbox",pid=10535,fd=58))
tcp   CLOSE-WAIT 32     0       159.149.248.41:64838  108.160.172.204:443 users:(("dropbox",pid=10535,fd=23))
tcp   CLOSE-WAIT 1      0       159.149.248.41:47780  54.209.0.118:443    users:(("dropbox",pid=10535,fd=57))
```

`traceroute`

`traceroute`, come dice il nome stesso, è una utility che permette di visualizzare il percorso di un pacchetto all'interno di una rete IP. Ogni cambio di direzione rappresenta un **router** [2], definito in quest'ambito *hop* (salto). Tramite questo programma si può avere anche un'analisi dei ritardi di trasmissione del pacchetto, per esempio per capire qual è il "collo di bottiglia" della rete in oggetto.

> [2] Un **router** è un particolare tipo di elaboratore che ha il compito di instradare un pacchetto verso la destinazione desiderata, che è contenuta nei metadati del pacchetto.

Esempi

```
# mostra gli hop per raggiungere Google Italia
$ traceroute -4 -q 1 www.google.it
traceroute to www.google.it (172.217.16.3), 30 hops max, 60 byte packets
 1  gateway (192.168.1.1)  1.093 ms
 2  151.7.202.72 (151.7.202.72)  10.896 ms
 3  151.7.80.88 (151.7.80.88)  11.220 ms
 4  151.7.80.10 (151.7.80.10)  14.466 ms
 5  151.6.5.230 (151.6.5.230)  17.189 ms
 6  MISG-B01-Ge10-1.wind.it (151.6.2.54)  19.940 ms
 7  151.6.0.30 (151.6.0.30)  23.128 ms
 8  209.85.241.94 (209.85.241.94)  25.001 ms
 9  64.233.174.245 (64.233.174.245)  40.682 ms
10  mil02s06-in-f3.1e100.net (172.217.16.3)  32.592 ms
```

`tcpdump` e `ping`

`ping` (Packet INternet Groper) è una utility che invia pacchetti ICMP [3] verso un altro dispositivo in rete, misurandone il Round Trip Time [4] e l'effettivo ritorno del pacchetto.

`tcpdump` è l'utensile più basico per effettuare analisi di rete: cattura e mostra i pacchetti TCP/IP in transito in rete. Può analizzare sia quelli in entrata che quelli in uscita, ma spesso viene usato solo per i primi. Si adatta bene ad essere usato in combinazione con `ping`.

> [3] ICMP (Internet Control Message Protocol) è un particolare tipo di protocollo che permette di inviare messaggi di controllo verso altri dispositivi in rete.

> [4] Il Round Trip Time (RTT) è il tempo che un pacchetto impiega per arrivare a destinazione e tornare indietro.

Esempi

Prendiamo una caso base: capire se due macchine riescono a comunicare (in maniera soddisfacente) tra loro in una rete sconosciuta. Poniamo che le due macchine siano fisicamente molto lontane.

Le due macchine dovranno scambiarsi dei pacchetti e ci sarà bisogno di una conferma di ricezione. Riportiamo l'esempio sulla macchina che abbiamo a disposizione utilizzando l'interfaccia di *loopback*.

Iniziamo `ping`-ando la nostra stessa macchina.

```
$ ping localhost
PING localhost.localdomain (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost.localdomain (127.0.0.1): icmp_seq=1 ttl=64 time=0.040 ms
64 bytes from localhost.localdomain (127.0.0.1): icmp_seq=2 ttl=64 time=0.027 ms
64 bytes from localhost.localdomain (127.0.0.1): icmp_seq=3 ttl=64 time=0.050 ms
64 bytes from localhost.localdomain (127.0.0.1): icmp_seq=4 ttl=64 time=0.054 ms
64 bytes from localhost.localdomain (127.0.0.1): icmp_seq=5 ttl=64 time=0.051 ms
# [...]
```

Ora impostiamo `tcdump` per captare pacchetti ICMP (solo numero 8 che corrisponde ai pacchetti inviati da `ping` attraverso l'interfaccia di *loopback*.

```
$ tcpdump -nni lo -e icmp[icmptype]==8
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on lo, link-type EN10MB (Ethernet), capture size 262144 bytes
12:47:35.295384 00:00:00:00:00:00 > 00:00:00:00:00:00, ethertype IPv4 (0x0800), length 98: 127.0.0.1 > 127.0.0.1:
ICMP echo request, id 17854, seq 1, length 64
12:47:36.295543 00:00:00:00:00:00 > 00:00:00:00:00:00, ethertype IPv4 (0x0800), length 98: 127.0.0.1 > 127.0.0.1:
ICMP echo request, id 17854, seq 2, length 64
12:47:37.295540 00:00:00:00:00:00 > 00:00:00:00:00:00, ethertype IPv4 (0x0800), length 98: 127.0.0.1 > 127.0.0.1:
ICMP echo request, id 17854, seq 3, length 64
12:47:38.295560 00:00:00:00:00:00 > 00:00:00:00:00:00, ethertype IPv4 (0x0800), length 98: 127.0.0.1 > 127.0.0.1:
ICMP echo request, id 17854, seq 4, length 64
12:47:39.295532 00:00:00:00:00:00 > 00:00:00:00:00:00, ethertype IPv4 (0x0800), length 98: 127.0.0.1 > 127.0.0.1:
ICMP echo request, id 17854, seq 5, length 64
```

`drill` (`dig`)

`drill` è la versione più recente del comando `dig` che si trovava nelle versioni più vecchie dei sistemi unix-like. Questo comando permette di richiedere ad un server DNS (molto spesso quello associato alla nostra connessione ad Internet) ogni sorta di informazioni. Cosa significa chiedere informazioni al DNS? Significa risolvere quello che si chiama nome di dominio, del tipo *www.example.com*, in un valore numero, cioè l'indirizzo IP [5].

[5] Un indirizzo IP è una 4-upla di quattro numeri che spaziano da 0 a 255, quindi 2^8. Ogni numero è rappresentabile con 8 bit, perciò un byte. Un indirizzo IP "pesa" quindi 4 byte.

Esempi

```
# vogliamo sapere l'IP dell'host www.google.it (record di tipo A)
$ drill www.google.it
;; ->>HEADER<<- opcode: QUERY, rcode: NOERROR, id: 9773
;; flags: qr rd ra ; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0
;; QUESTION SECTION:
;; www.google.it. IN  A

;; ANSWER SECTION:
www.google.it.  13  IN  A 216.58.212.99

;; AUTHORITY SECTION:

;; ADDITIONAL SECTION:

;; Query time: 97 msec
;; SERVER: 172.28.172.1
;; WHEN: Mon Mar  7 17:37:18 2016
;; MSG SIZE  rcvd: 47
```

```
# vogliamo sapere tutto il percorso della stessa richiesta DNS verso
# i vari server
$ drill -T www.google.it
it. 172800  IN  NS  m.dns.it.
it. 172800  IN  NS  dns.nic.it.
it. 172800  IN  NS  nameserver.cnr.it.
it. 172800  IN  NS  a.dns.it.
it. 172800  IN  NS  s.dns.it.
it. 172800  IN  NS  r.dns.it.
```

```
google.it.  10800 IN  NS  ns3.google.com.
google.it.  10800 IN  NS  ns4.google.com.
google.it.  10800 IN  NS  ns1.google.com.
google.it.  10800 IN  NS  ns2.google.com.
google.it.  10800 IN  NS  ns3.google.com.
google.it.  10800 IN  NS  ns4.google.com.
google.it.  10800 IN  NS  ns1.google.com.
google.it.  10800 IN  NS  ns2.google.com.
ns3.google.com.google.it. 10800 IN  NS  ns3.google.com.
google.it.  10800 IN  NS  ns4.google.com.
google.it.  10800 IN  NS  ns1.google.com.
google.it.  10800 IN  NS  ns2.google.com.
ns4.google.com.google.it. 10800 IN  NS  ns3.google.com.
google.it.  10800 IN  NS  ns4.google.com.
google.it.  10800 IN  NS  ns1.google.com.
google.it.  10800 IN  NS  ns2.google.com.
ns1.google.com.google.it. 10800 IN  NS  ns3.google.com.
google.it.  10800 IN  NS  ns4.google.com.
google.it.  10800 IN  NS  ns1.google.com.
google.it.  10800 IN  NS  ns2.google.com.
ns2.google.com.www.google.it. 300 IN  A 216.58.212.67
```

`nmap` e Zenmap

`nmap` è il coltellino svizzero delle utility di analisi di rete in ambiente Linux. Può scansionare una rete locale o remota e, per ogni dispositivo incontrato, può effettuare vari test sulle porte software aperte o il riconoscimento del sistema operativo. Nasce come utility a riga di comando.

Per facilitare l'uso di un'applicazione complessa come `nmap`, gli stessi sviluppatori hanno deciso di dotarla di una semplice, ma efficace, interfaccia grafica, creando Zenmap. Tra le feature aggiunte viene annoverato una mappa grafica dei dispositivi analizzati fino al momento della visualizzazione.

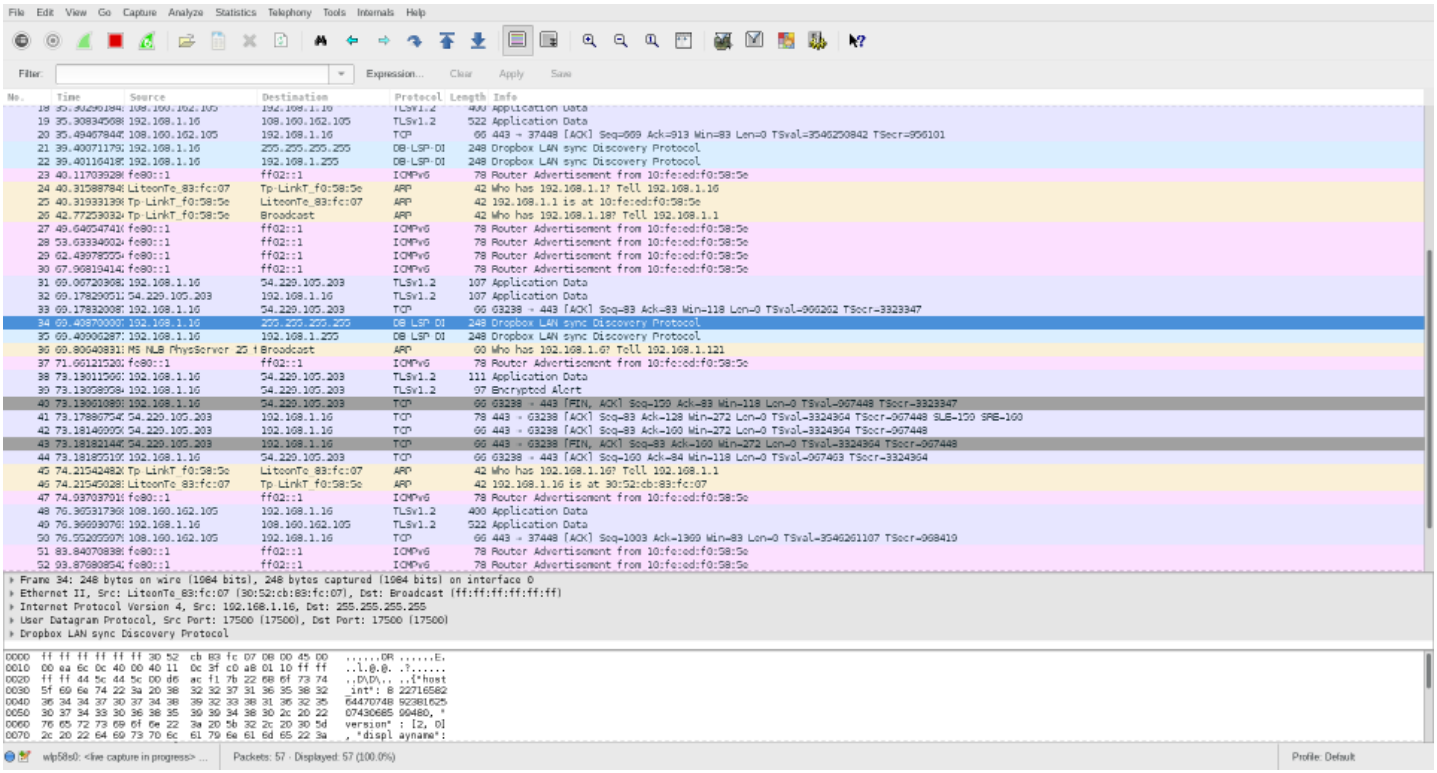
Esempi

```
# scansiona tutte le porte TCP sulla classe C (255 host) della sottorete
# specificata, cercando di rilevarne l'os (e ignorando il blocco ping)
$ sudo nmap -sS -Pn 192.168.1.0/24

Starting Nmap 7.01 ( https://nmap.org ) at 2016-03-01 00:58 CET
Nmap scan report for 192.168.1.1
Host is up (0.0039s latency).
Not shown: 994 closed ports
PORT      STATE SERVICE
23/tcp    open  telnet
80/tcp    open  http
139/tcp   open  netbios-ssn
445/tcp   open  microsoft-ds
1900/tcp  open  upnp
49152/tcp open  unknown
MAC Address: 10:FE:ED:F0:58:5E (Tp-link Technologies)

Nmap scan report for 192.168.1.3
Host is up (0.016s latency).
Not shown: 992 closed ports
PORT     STATE SERVICE
21/tcp   open  ftp
22/tcp   open  ssh
80/tcp   open  http
1755/tcp open  wms
4444/tcp open  krb524
6666/tcp open  irc
6881/tcp open  bittorrent-tracker
9999/tcp open  abyss
MAC Address: 00:01:E3:0F:37:CA (Siemens AG)

Nmap scan report for 192.168.1.4
Host is up (0.012s latency).
Not shown: 999 closed ports
PORT   STATE SERVICE
80/tcp open  http
MAC Address: 84:1B:5E:4A:06:CC (Netgear)
```

```
Nmap scan report for 192.168.1.5
Host is up (0.011s latency).
Not shown: 998 closed ports
PORT   STATE SERVICE
53/tcp open  domain
80/tcp open  http
MAC Address: 00:01:E3:0F:37:CA (Siemens AG)

Nmap scan report for 192.168.1.6
Host is up (0.011s latency).
Not shown: 996 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
80/tcp   open  http
111/tcp  open  rpcbind
9090/tcp open  zeus-admin
MAC Address: 00:01:E3:0F:37:CA (Siemens AG)

Nmap scan report for 192.168.1.11
Host is up (0.013s latency).
All 1000 scanned ports on 192.168.1.11 are closed
MAC Address: 98:4B:4A:00:17:69 (Arris Group)

Nmap scan report for 192.168.1.18
Host is up (0.0065s latency).
All 1000 scanned ports on 192.168.1.18 are closed
MAC Address: 20:02:AF:BC:B3:D2 (Murata Manufacturing)

Nmap scan report for 192.168.1.121
Host is up (0.022s latency).
Not shown: 998 filtered ports
PORT      STATE SERVICE
1234/tcp  open  hotline
49153/tcp open  unknown
MAC Address: 02:19:FB:50:C7:D8 (Unknown)

Nmap scan report for 192.168.1.16
Host is up (0.0000090s latency).
All 1000 scanned ports on 192.168.1.16 are closed

Nmap done: 256 IP addresses (9 hosts up) scanned in 37.48 seconds
```

```
# ora provare ad inserire lo stesso comando all'interno di Zenmap

# una volta finita la procedura di scan, visualizzare la tab 'topologia' e
# osservare l'utile schema prodotto (consiglio: attiva il fish-eye)
```

Wireshark



L'interfaccia grafica di Wireshark, utile complemento alle sue funzionalità.

Finiamo parlando di Wireshark, l'analizzatore di traffico open source più diffuso al mondo, ormai quasi uno standard.

Cosa fa?

Cattura ogni singolo pacchetto transitante per la rete della quale la nostra macchina fa parte e lo analizza riportandoci particolari informazioni.

Perchè?

Poniamo, esempio, di avere a che fare con una rete che risulta molto intasata e doverne capire la ragione. Iniziamo analizzando i pacchetti in transito e osserviamo che un host all'interno della rete invia una quantità sospetta di pacchetti di un certo tipo. Spegnendolo risolveremmo il problema, ma più efficientemente possiamo provare a capire il tipo di pacchetti inviati e dedurre la ragione di una tale quantità. In questa maniera possiamo agire sul software che invia questi pacchetti e farlo smettere.

Poniamo, altro esempio, di dover analizzare le prestazioni di un sotware di streaming video. Sarebbe utile considerare la grandezza e il numero di pacchetti presenti in rete, ma abbiamo bisogno di analizzarli e filtrare solo quelli appartenenti al software di streaming.

In che modo?

Viene attivata la **modalità promiscua** [6] dell'interfaccia di rete.

[6] La modalità promisqua ordina alla scheda di rete di considerare tutti i pacchetti che transitano sulla rete, contrariamente a quanto accade in una situazione normale, nella quale solo i pacchetti destinati al nostro indirizzo vengono presi in considerazione.

Quando non si ha una macchina Linux a portata di mano

Capitano spesso situazioni in cui bisogna testare l'effettivo funzionamento di una rete e non si ha a disposizione un elaboratore esterno alla rete stessa, oppure non se ne ha disponibile uno con un ambiente Linux consono. Vediamo cosa è possibile fare in questi casi.

Se dobbiamo effettuare dei test generici quali ricerca del nostro IP, *whois*, *reverse whois*, *speedtest* oppure operare con i server DNS, esistono dei comodi tool online (e per giunta gratuitamente utilizzabili).

- Domain Tools

  Questo servizio web permette di effettuare svariati test online: *whois*, *reverse whois*, *reverse IP, NS, MX lookup*, ricercare il proprio IP e operare coi DNS.

- you get signal

  Quest'altro servizio invece offre diversi tipi di ricerche, anche più avanzate: check delle porte software aperte verso Internet, ricerca del proprio IP, geolocazione di una rete, IP, numero di telefono, *traceroute* visuale, *reverse mail lookup* e *whois*.

- DSLReports

  Un servizio non ottenibile localmente (e quindi utilizzabile esclusivamente via web), ma estremamente utile in alcuni casi è il test della velocità della connessione che si ha sotto mano al momento. Questo sito offre esattamente questo servizio.