

# Strumenti di analisi della rete

Valentina Ceoletta, Marco De Bona, Federico De Meo, Davide Quaglia

## 1. Introduzione agli analizzatori di rete

Esistono strumenti SW che consentono di analizzare i pacchetti che arrivano alla propria interfaccia di rete:

- **tcpdump** ([http://www.tcpdump.org/tcpdump\\_man.html](http://www.tcpdump.org/tcpdump_man.html)) storico tool in linea di comando per Linux
- **wireshark** (<http://www.wireshark.org/>) nuovo tool con interfaccia grafica disponibile per Linux, Windows e Mac
- **ethereal** (<ftp://gnu.cs.pu.edu.tw/network/Ethereal/>) versione in linea di comando di Wireshark
- **windump** (<http://www.winpcap.org/windump>) per Windows.

Questi tool di analisi si basano tutti sulla libreria C chiamata **libpcap**, reperibile al sito <http://www.tcpdump.org> (tcpdump è lo sniffer per eccellenza che sfrutta la libpcap).

Le principali funzioni di questa libreria sono la possibilità di cercare e trovare interfacce di rete, gestire potenti filtri di cattura, analizzare ciascun pacchetto, gestire errori e statistiche di cattura.

### 1.1 Scaricamento e installazione

I tool si possono scaricare liberamente dalle rispettive pagine web o probabilmente sono presenti già nell'installazione della propria distribuzione Linux.

**ATTENZIONE:** per poter utilizzare le funzionalità di cattura di questi tool in ambiente Linux bisogna essere autenticati come utente *root* o aver installato il tool con *setuid* a root. In ogni caso questi tool possono essere utilizzati per analizzare catture precedentemente effettuate da utente root e salvate su file.

In questa esercitazione non cattureremo del traffico dalla rete ma analizzeremo del traffico precedentemente catturato.

### 1.2 Alcuni concetti sullo *sniffing*

**Sniffing in reti non-switched:** In questo tipo di reti ethernet il mezzo trasmissivo è condiviso e quindi tutte le schede di rete dei computer nella rete locale ricevono tutti i pacchetti, anche quelli destinati ad altri, selezionando i propri a seconda dell'indirizzo **MAC** (indirizzo hardware specifico della scheda di rete). Lo sniffing in questo caso consiste nell'impostare sull'interfaccia di rete la cosiddetta **modalità promiscua**, che disattiva il “filtro hardware” basato sul MAC permettendo al sistema l'ascolto di tutto il traffico passante sul cavo. Esempio di rete non switched è la **rete WiFi**.

**Sniffing in reti ethernet switched:** In questo caso l'apparato centrale della rete, definito switch, si preoccupa, dopo un breve transitorio, di inoltrare su ciascuna porta solo il traffico destinato ai dispositivi collegati a quella porta; ciascuna interfaccia di rete riceve, quindi solo i pacchetti destinati al proprio indirizzo, i pacchetti multicast e quelli broadcast. L'impostazione della modalità promiscua è quindi insufficiente per poter intercettare il traffico in una rete gestita da switch. Un metodo per poter ricevere tutto il traffico dallo switch da una porta qualunque è il **MAC flooding**. Tale tecnica consiste nell'inviare ad uno switch pacchetti appositamente costruiti per riempire la *CAM table* dello switch di indirizzi MAC fittizi. Questo attacco costringe lo switch ad entrare in

una condizione detta di *fail open* che lo fa comportare come un hub, inviando così gli stessi dati a tutti gli apparati ad esso collegati.

## 2. Utilizzo di Wireshark

Alcune caratteristiche:

- i dati possono essere acquisiti direttamente dall'interfaccia di rete oppure possono essere letti su un file di cattura precedente
- i dati possono essere catturati dal vivo da reti Ethernet, WiFi, ADSL, ecc.
- i dati di rete catturati possono essere esplorati nelle loro parti tramite un'interfaccia grafica
- i filtri di visualizzazione possono essere usati per colorare o visualizzare selettivamente le informazioni sommarie sui pacchetti
- i protocolli di comunicazione possono essere scomposti, in quanto wireshark riesce a “comprendere” la struttura dei diversi protocolli di rete, quindi è in grado di visualizzare incapsulamenti e campi singoli, ed di interpretare il loro significato
- è possibile studiare le statistiche di una connessione TCP e di estrarne il contenuto.

Per lanciare l'applicazione, digitare su riga di comando:

```
$ wireshark
```

oppure, essendo wireshark un'applicazione grafica, basta lanciarlo cliccando sull'apposita icona:



**NB:** Si ricordi che è necessario essere l'utente *root* se si intende fare una cattura diretta dalle interfacce di rete, ma in questa esercitazione useremo un file con delle catture già fatte, quindi non si necessitano i privilegi dell'utente *root*.

### 2.1 Cattura dei pacchetti

Per avviare la cattura dei pacchetti è necessario specificare da quale interfaccia si vuole effettuare la cattura. Per fare ciò aprire il menu *Capture/Interfaces*, la finestra che si apre chiederà quale interfaccia di rete utilizzare per la cattura (Figura 1).

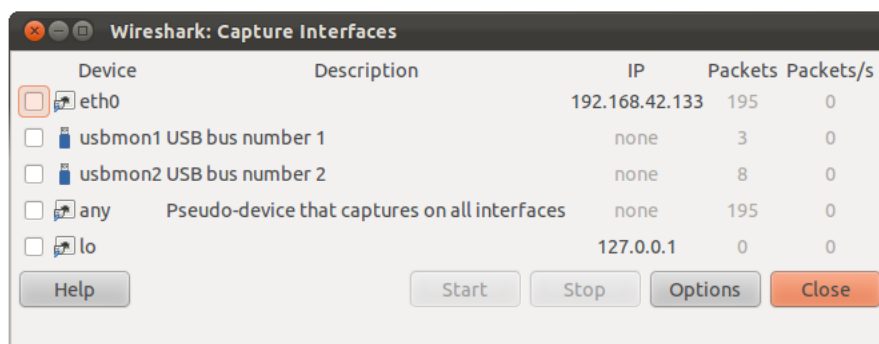


Figura 1 Interfacce di cattura

Solitamente, in linux, la prima interfaccia di rete ha come nome *eth0* mentre in Windows ha il nome del produttore della scheda.

Una volta individuata l'interfaccia, premendo il tasto *Options* è possibile dare diverse impostazioni, tra le quali anche i filtri da utilizzare per la cattura (Figura 2).

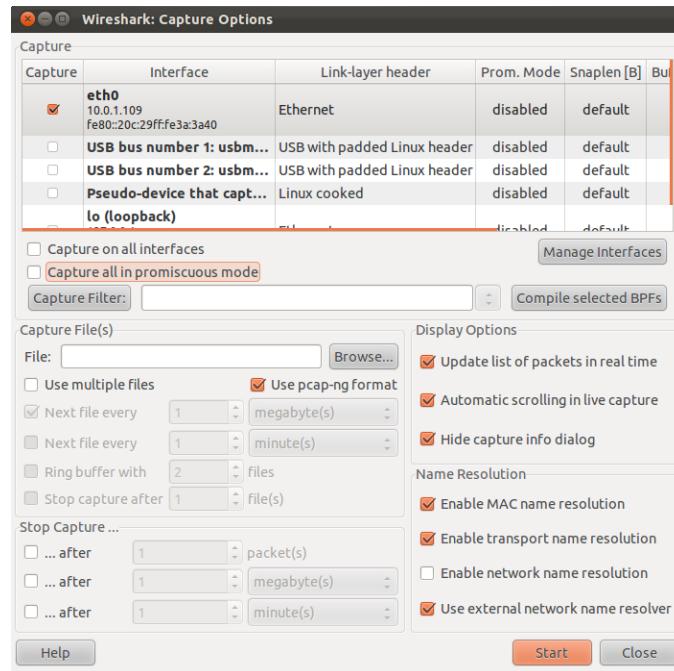


Figura 2 Opzioni di cattura

Premendo Start si avvia la cattura e viene visualizzata una finestra che mostra le statistiche di cattura. Per terminare la cattura (se non si è impostato un limite in pacchetti, secondi o byte) si usa il comando Stop dal menu Capture.

## 2.2 Applicazione dei filtri nella cattura

E' possibile limitare la cattura ai soli pacchetti che rispettano certi requisiti impostando un filtro di cattura nella finestra di Capture Options precedentemente vista, prendendo sul tasto Capture Filter (Figura 3).

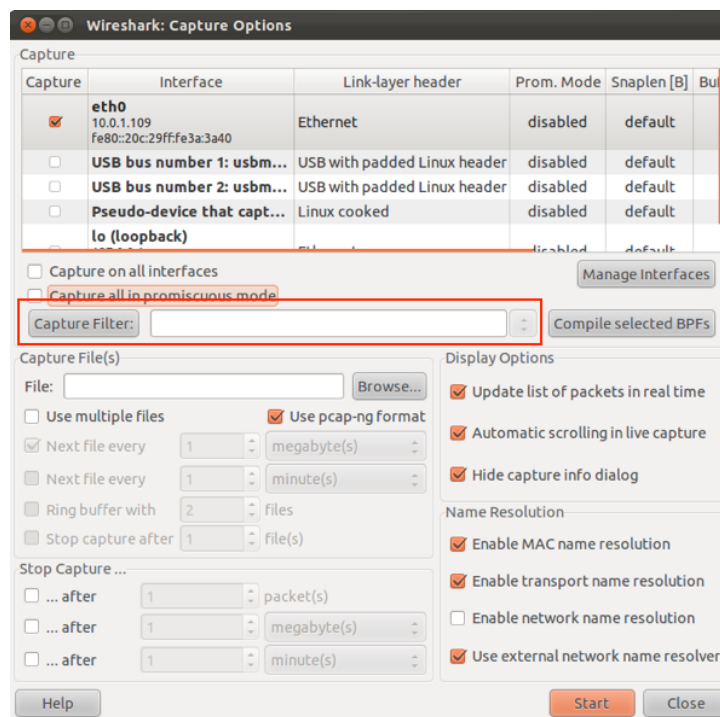


Figura 3 Filtri di cattura

Nella sezione apposita è possibile scrivere l'espressione del filtro (trovate la sintassi nell'appendice

A) oppure selezionare un filtro preesistente cliccando sull'apposito bottone e al limite modificarlo (Figura 4).

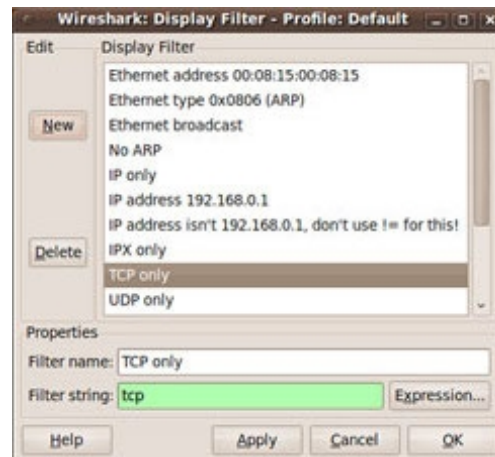


Figura 4 Filtri di cattura già pronti

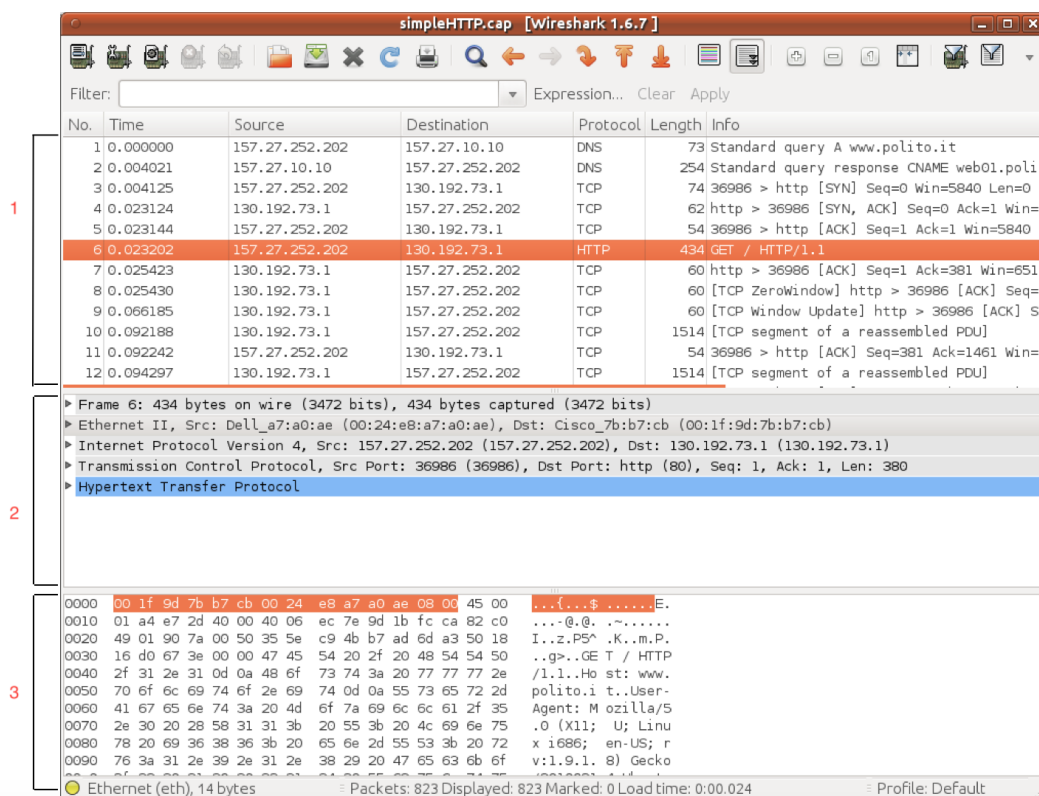
I filtri di cattura programmano la scheda di rete al fine di catturare solo determinati pacchetti; tali filtri si usano quando la quantità di pacchetti che passano sul tratto di rete osservato è tale che se tutti i pacchetti venissero passati alla CPU le sue prestazioni sarebbero compromesse.

Al termine della cattura la finestra principale di wireshark mostra i dati catturati.

## 2.3 Finestra principale

La finestra è divisa in tre parti, nell'ordine:

1. Tabella dei pacchetti catturati;
2. Vista sulla incapsulazione dei protocolli del pacchetto selezionato nella tabella;
3. Vista in versione binaria dei dati del pacchetto selezionato nella tabella.



E' anche possibile vedere un sommario sulle statistiche scegliendo il menu Statistics/Summary.

## 2.4 Regole di colorazione

E' possibile migliorare la visualizzazione dei vari pacchetti nella tabella principale colorando le righe in base al tipo di protocolli o indirizzi coinvolti. I filtri di coloramento si possono impostare nel menu View/Colouring rules.

E' possibile creare nuovi filtri di colori con il bottone New (Figura 5) che apre una finestra in cui si può impostare nome, regole e colori. Le regole si impostano con un linguaggio diverso da quello dei filtri di cattura (si noti il pulsante Expression che semplifica la scrittura delle regole combinando campi dei pacchetti, valori e operatori logici).

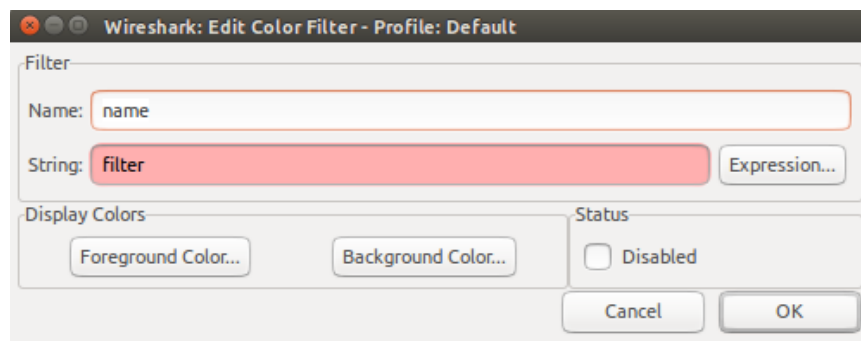
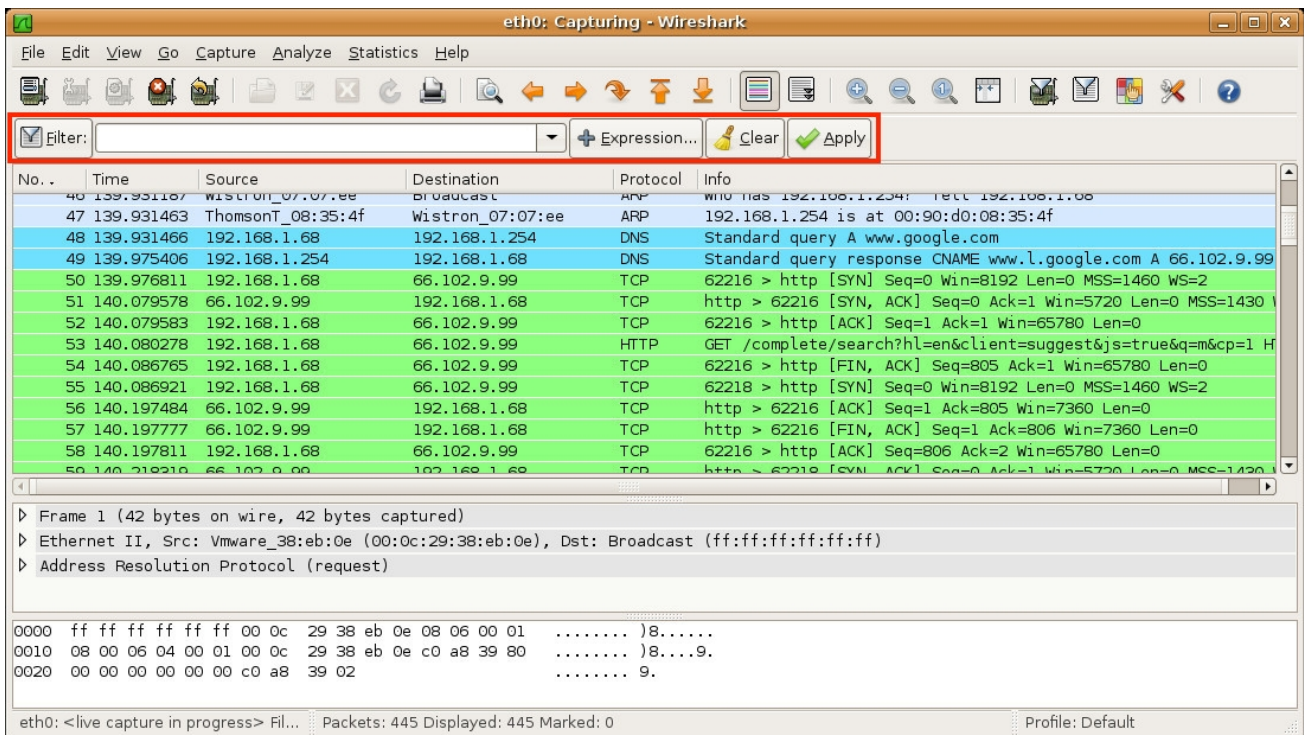


Figura 5 Creazione di una nuova regola di colorazione

## 2.5 Applicazione dei filtri di visualizzazione nella finestra principale

E' possibile creare un filtro per limitare il numero di pacchetti visualizzati in una cattura già avvenuta, per fare questo, si utilizza la barra dei filtri presente nella schermata principale.



Tramite il tasto Filter è possibile specificare un filtro esistente o crearne di nuovi e per la stesura dei filtri ci aiuta il tasto Expression che fornisce un tool automatico di stesura filtri simile a quello per la colorazione delle righe.



## 2.6 Analisi del flusso TCP

Per quanto riguarda TCP, selezionando un pacchetto TCP nella finestra principale, è possibile seguire l'intero flusso di dati di quella “conversazione” mediante la voce **Analyze/Follow TCP Stream** e studiare l'andamento di alcuni parametri del protocollo mediante la voce **Statistics/TCP stream graph**.

Wireshark 1.6.7 interface showing a packet capture on eth0. The filter is set to `tcp.stream eq 27`. The packet list shows several packets, including an HTTP GET request and a TCP segment. The 'Follow TCP Stream' dialog is open, displaying the stream content for the selected packet.

Stream Content:

```
GET /images/I/41TfxNV5QiL._SL125_.jpg HTTP/1.1
Host: ecx.images-amazon.com
Connection: keep-alive
User-Agent: Mozilla/5.0 (X11; Linux i686) AppleWebKit/537.31 (KHTML, like Gecko)
Chrome/26.0.1410.63 Safari/537.31
Accept: */*
Referer: http://astore.amazon.co.uk/androidsmart-21
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-GB,en-US;q=0.8,en;q=0.6
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3

HTTP/1.0 200 OK
Content-Type: image/jpeg
Content-Length: 3115
Connection: keep-alive
Server: nginx
Date: Sat, 02 Mar 2013 08:34:47 GMT
Cache-Control: max-age=630720000,public
Expires: Wed, 18 May 2033 03:33:20 GMT
Last-Modified: Wed, 22 Feb 2012 10:19:11 GMT
Age: 5124356
Via: 1.0 96047b1d560fcd73b9669160a7899897.cloudfront.net (CloudFront)
X-Cache: Hit from cloudfront
```

Entire conversation (19984 bytes)

Find Save As Print ☐ ASCII ☐ EBCDIC ☐ Hex Dump ☐ C Arrays ☒ Raw

Help Filter Out This Stream Close

### 3. Comando Ping

```
marco@marco-XPS-13-9350:~$ ping
Usage: ping [-aAbBdDfhLnOqrRUvV64] [-c count] [-i interval] [-I interface]
          [-m mark] [-M pmtudisc_option] [-l preload] [-p pattern] [-Q tos]
          [-s packetsize] [-S sndbuf] [-t ttl] [-T timestamp_option]
          [-w deadline] [-W timeout] [hop1 ...] destination
Usage: ping -6 [-aAbBdDfhLnOqrRUvV] [-c count] [-i interval] [-I interface]
          [-l preload] [-m mark] [-M pmtudisc_option]
          [-N nodeinfo_option] [-p pattern] [-Q tclass] [-s packetsize]
          [-S sndbuf] [-t ttl] [-T timestamp_option] [-w deadline]
          [-W timeout] destination
```

Il comando Ping è un semplice strumento molto usato nell'amministrazione di reti per computer per verificare la raggiungibilità di un computer connesso alla rete e il relativo Round Trip Time (RTT) ossia il tempo dalla partenza del pacchetto inviato verso il computer bersaglio al ritorno indietro della risposta. Viene utilizzato il protocollo ICMP.

**ICMP (Internet Control Message Protocol)** è un protocollo di servizio utilizzato dal protocollo IP per trasmettere informazioni riguardanti malfunzionamenti (ad es. TimeExceeded, Destination Unreachable), informazioni di controllo (ad es. Echo Request, Echo Replay) o messaggi tra i vari componenti di una rete di calcolatori.

```
PING www.google.com (216.58.210.164): 56 data bytes
64 bytes from 216.58.210.164: icmp_seq=0 ttl=55 time=31.004 ms
64 bytes from 216.58.210.164: icmp_seq=1 ttl=55 time=31.695 ms
64 bytes from 216.58.210.164: icmp_seq=2 ttl=55 time=30.773 ms
64 bytes from 216.58.210.164: icmp_seq=3 ttl=55 time=32.544 ms
64 bytes from 216.58.210.164: icmp_seq=4 ttl=55 time=33.416 ms
64 bytes from 216.58.210.164: icmp_seq=5 ttl=55 time=32.221 ms
64 bytes from 216.58.210.164: icmp_seq=6 ttl=55 time=31.984 ms
64 bytes from 216.58.210.164: icmp_seq=7 ttl=55 time=32.039 ms
64 bytes from 216.58.210.164: icmp_seq=8 ttl=55 time=32.371 ms
^C
--- www.google.com ping statistics ---
9 packets transmitted, 9 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 30.773/32.005/33.416/0.751 ms
```

Figura 6 Esempio di esecuzione del comando ping

Il pacchetto ICMP viene inviato con il messaggio di ECHO\_REQUEST verso l'indirizzo IP o il nome DNS di destinazione; se la destinazione è attiva, il software ICMP sull'host o sul router risponde con un messaggio di ECHO\_REPLY che trasporta gli stessi dati del messaggio di richiesta (Figura 6).

Dopo aver avviato il comando ping da terminale è possibile visualizzare lo scambio dei messaggi tra il nostro calcolatore e la destinazione. Per fare ciò è sufficiente avviare una cattura da Wireshark applicando il filtro "icmp" per la visualizzazione. Notate che nell'elenco dei pacchetti vengono mostrati esattamente il numero di pacchetti trasmessi e ricevuti indicati nel terminale (Figura 7).

No.	Time	Source	Destination	Protocol	Length	Info
70	3.656263	157.27.143.46	216.58.210.164	ICMP	98	Echo (ping) request id=0x4806, seq=0/0, ttl=64 (reply in 71)
71	3.687200	216.58.210.164	157.27.143.46	ICMP	98	Echo (ping) reply id=0x4806, seq=0/0, ttl=55 (request in 70)
72	4.659745	157.27.143.46	216.58.210.164	ICMP	98	Echo (ping) request id=0x4806, seq=1/256, ttl=64 (reply in 73)
73	4.691308	216.58.210.164	157.27.143.46	ICMP	98	Echo (ping) reply id=0x4806, seq=1/256, ttl=55 (request in 72)
87	5.664656	157.27.143.46	216.58.210.164	ICMP	98	Echo (ping) request id=0x4806, seq=2/512, ttl=64 (reply in 88)
88	5.695357	216.58.210.164	157.27.143.46	ICMP	98	Echo (ping) reply id=0x4806, seq=2/512, ttl=55 (request in 87)
90	6.669878	157.27.143.46	216.58.210.164	ICMP	98	Echo (ping) request id=0x4806, seq=3/768, ttl=64 (reply in 91)
91	6.702238	216.58.210.164	157.27.143.46	ICMP	98	Echo (ping) reply id=0x4806, seq=3/768, ttl=55 (request in 90)
92	7.675057	157.27.143.46	216.58.210.164	ICMP	98	Echo (ping) request id=0x4806, seq=4/1024, ttl=64 (reply in 93)
93	7.708296	216.58.210.164	157.27.143.46	ICMP	98	Echo (ping) reply id=0x4806, seq=4/1024, ttl=55 (request in 92)
94	8.678485	157.27.143.46	216.58.210.164	ICMP	98	Echo (ping) request id=0x4806, seq=5/1280, ttl=64 (reply in 95)
95	8.710585	216.58.210.164	157.27.143.46	ICMP	98	Echo (ping) reply id=0x4806, seq=5/1280, ttl=55 (request in 94)
98	9.683684	157.27.143.46	216.58.210.164	ICMP	98	Echo (ping) request id=0x4806, seq=6/1536, ttl=64 (reply in 99)
99	9.715487	216.58.210.164	157.27.143.46	ICMP	98	Echo (ping) reply id=0x4806, seq=6/1536, ttl=55 (request in 98)
101	10.688858	157.27.143.46	216.58.210.164	ICMP	98	Echo (ping) request id=0x4806, seq=7/1792, ttl=64 (reply in 104)
104	10.720713	216.58.210.164	157.27.143.46	ICMP	98	Echo (ping) reply id=0x4806, seq=7/1792, ttl=55 (request in 101)
105	11.692241	157.27.143.46	216.58.210.164	ICMP	98	Echo (ping) request id=0x4806, seq=8/2048, ttl=64 (reply in 106)
106	11.724436	216.58.210.164	157.27.143.46	ICMP	98	Echo (ping) reply id=0x4806, seq=8/2048, ttl=55 (request in 105)

Figura 7 Cattura Wireshark relativa al ping

La natura di Ping secondo cui il destinatario risponde automaticamente all' ECHO\_REQUEST con il messaggio di ECHO\_REPLY, ha reso Ping lo uno strumento molto utilizzato per attacchi informatici di tipo denial-of-service (DoS) e distributed denial-of-service (DdoS). Per esempio, se si dispone di una grande quantità di banda è possibile infatti effettuare un attacco chiamato Ping Flood che consiste nell'invio continuato di pacchetti che può portare alla congestione della rete del destinatario, rendendolo quindi inaccessibile dagli altri utenti (Figura 8).

```

C:\Users\labnol>ping -t 8.8.8.8

Pinging 8.8.8.8 with 32 bytes of data:
Request timed out.
Reply from 8.8.8.8: bytes=32 time=137ms TTL=56
Reply from 8.8.8.8: bytes=32 time=145ms TTL=56
Reply from 8.8.8.8: bytes=32 time=147ms TTL=56
Reply from 8.8.8.8: bytes=32 time=106ms TTL=56
Reply from 8.8.8.8: bytes=32 time=310ms TTL=56
Reply from 8.8.8.8: bytes=32 time=430ms TTL=56
Request timed out.
Request timed out.
Request timed out.
Reply from 192.168.1.1: Destination net unreachable.
Request timed out.
Request timed out.
Request timed out.
Reply from 192.168.1.1: Destination net unreachable.
Request timed out.
Request timed out.
Request timed out.
Reply from 8.8.8.8: bytes=32 time=427ms TTL=56
Reply from 8.8.8.8: bytes=32 time=507ms TTL=56
Reply from 8.8.8.8: bytes=32 time=509ms TTL=56
Reply from 8.8.8.8: bytes=32 time=479ms TTL=56
Reply from 8.8.8.8: bytes=32 time=35ms TTL=56
Reply from 8.8.8.8: bytes=32 time=33ms TTL=56
Reply from 8.8.8.8: bytes=32 time=32ms TTL=56
Reply from 8.8.8.8: bytes=32 time=32ms TTL=56
Reply from 8.8.8.8: bytes=32 time=258ms TTL=56

```

ping command  
with -t switch

connected to  
the Internet

offline,  
disconnected

connection  
restored, online

Figura 8 Esempio di Ping Flood

## 4. Comando Traceroute

Il comando traceroute (tracert in Windows) invece è un semplice strumento per scoprire il



percorso che un pacchetto segue dalla sorgente alla destinazione. Il comando mostra un elenco di tutte le interfacce di router che il pacchetto attraversa finché raggiunge la destinazione.

Per capire il funzionamento di questo strumento occorre ricordare che ogni pacchetto IP contiene un campo Time To Live (TTL) con un valore intero che viene decrementato ad ogni router del percorso tra l'IP sorgente e l'IP destinazione. Quando un router trova TTL=1 scarta il pacchetto inviando all'IP sorgente il messaggio ICMP TIME-TO-LIVE\_EXCEEDED mettendo come IP sorgente se stesso e quindi rivelando la propria identità. Il programma Traceroute quindi invia una sequenza di pacchetti "sonda" verso la destinazione con TTL con valori crescenti a partire da 1 generando via via messaggi ICMP TIME-TO-LIVE\_EXCEEDED dalle interfacce dei router attraversati e in questo modo viene a conoscere il percorso del pacchetto. In Unix il pacchetto "sonda" è un UDP ma usando l'opzione "-I" è possibile usare un ICMP ECHO\_REQUEST. In Windows il programma usa sempre ICMP ECHO\_REQUEST. Quando un pacchetto "sonda" raggiungerà la destinazione finale, questa invierà indietro un pacchetto ICMP diverso da TIME-TO-LIVE\_EXCEEDED e il processo terminerà.

In Figura 8 si può vedere che per ogni valore di TTL il programma ha inviato tre pacchetti "sonda" e per ognuno visualizza il RTT e l'indirizzo IP dell'interfaccia del router attraversato (se possibile anche il nome DNS associato).

```
bob@ubuntu-comp:/home$ traceroute www.google.com
traceroute to www.google.com (173.194.116.145), 30 hops max, 60 byte packets
 1 speedtouch.lan (192.168.1.1) 12.669 ms 11.902 ms 11.053 ms
 2 78.134.144.1-dsl.net.metronet.hr (78.134.144.1) 13.987 ms 15.648 ms 17.37
1 ms
 3 10.50.0.73 (10.50.0.73) 22.473 ms 23.213 ms 26.523 ms
 4 10.50.0.74 (10.50.0.74) 29.124 ms 30.102 ms 34.318 ms
 5 213.147.96.110 (213.147.96.110) 35.266 ms 38.136 ms 39.979 ms
 6 212.162.29.1 (212.162.29.1) 41.924 ms 36.542 ms 38.233 ms
 7 ae-2-3.bar1.Ljubljana1.Level3.net (4.69.151.233) 36.919 ms 13.695 ms 15.1
55 ms
 8 * * *
 9 * * *
10 ae-3-80.edge3.Frankfurt1.Level3.net (4.69.154.135) 81.414 ms ae-4-90.edge3.
Frankfurt1.Level3.net (4.69.154.199) 39.745 ms ae-1-60.edge3.Frankfurt1.Level3.
net (4.69.154.7) 42.889 ms
11 4.68.70.186 (4.68.70.186) 45.590 ms 48.088 ms 50.605 ms
12 209.85.240.64 (209.85.240.64) 65.602 ms 55.049 ms 57.458 ms
13 66.249.94.69 (66.249.94.69) 60.435 ms 23.370 ms 23.618 ms
14 173.194.116.145 (173.194.116.145) 24.013 ms 23.452 ms 23.585 ms
```

Figura 8 Exmple of traceroute command

Dopo aver avviato il comando traceroute da terminale è possibile visualizzare in Wireshark lo scambio dei messaggi tra il nostro calcolatore e la sorgente di destinazione. Notate che nell'elenco dei pacchetti vengono mostrati esattamente il numero di pacchetti trasmessi, notare che si visualizzano per ogni hop i tre pacchetti "sonda" (Figura 9).

No.	Time	Source	Destination	Protocol	Length	Info
102	9.082596	192.168.1.1	192.168.1.75	ICMP	82	Time-to-live exceeded (Time to live exceeded in transit)
106	9.086306	192.168.1.1	192.168.1.75	ICMP	82	Time-to-live exceeded (Time to live exceeded in transit)
108	9.087392	192.168.1.1	192.168.1.75	ICMP	82	Time-to-live exceeded (Time to live exceeded in transit)
330	24.103483	172.17.224.156	192.168.1.75	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
337	25.135966	172.17.224.158	192.168.1.75	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
374	26.168144	172.17.224.114	192.168.1.75	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
420	27.205908	172.17.224.186	192.168.1.75	ICMP	94	Time-to-live exceeded (Time to live exceeded in transit)
430	28.235384	172.17.224.190	192.168.1.75	ICMP	94	Time-to-live exceeded (Time to live exceeded in transit)
437	29.274343	172.17.224.37	192.168.1.75	ICMP	94	Time-to-live exceeded (Time to live exceeded in transit)
557	44.547025	172.19.177.26	192.168.1.75	ICMP	94	Time-to-live exceeded (Time to live exceeded in transit)
562	45.313707	195.22.205.116	192.168.1.75	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
566	45.344174	195.22.205.98	192.168.1.75	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
570	45.372869	195.22.205.98	192.168.1.75	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
572	45.388634	74.125.146.168	192.168.1.75	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
587	46.457741	72.14.209.236	192.168.1.75	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
613	47.511276	72.14.204.72	192.168.1.75	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
2532	63.597676	216.239.42.21	192.168.1.75	ICMP	94	Time-to-live exceeded (Time to live exceeded in transit)
2537	64.652943	216.239.42.23	192.168.1.75	ICMP	94	Time-to-live exceeded (Time to live exceeded in transit)
2552	65.718147	216.239.42.21	192.168.1.75	ICMP	94	Time-to-live exceeded (Time to live exceeded in transit)
2556	65.733129	216.239.42.21	192.168.1.75	ICMP	94	Time-to-live exceeded (Time to live exceeded in transit)
2559	65.750461	216.58.205.163	192.168.1.75	ICMP	70	Destination unreachable (Port unreachable)
2565	65.787707	216.58.205.163	192.168.1.75	ICMP	70	Destination unreachable (Port unreachable)

Figura 9 Cattura Wireshark di pacchetti generati da traceroute

## 5 nslookup

È uno strumento presente in tutti i sistemi operativi che utilizzano il protocollo TCP/IP (Linux, Unix, Windows, MacOS). Nslookup consente di effettuare interrogazioni ai server DNS per poter ottenere da un hostname il relativo indirizzo IP o viceversa. Si può utilizzare in due modalità: interattivo e non interattivo.

**DNS (Domain Name System)** è un sistema di server organizzato gerarchicamente, che serve per la gestione del *namespace* (Domain Name Space). Il compito principale di questo servizio è quello di rispondere alle richieste della risoluzione del nome di dominio, ovvero la conversione dei nomi di dominio in indirizzi IP.

Modo interattivo: permette di effettuare più query e visualizza i singoli risultati. Viene abilitato in maniera automatica quando il comando non è seguito da alcun argomento.

```
valentina@Valentinas-MBP:~$ nslookup
> www.google.it
Server:      192.168.1.1
Address:     192.168.1.1#53

Non-authoritative answer:
Name:   www.google.it
Address: 216.58.198.35
> www.univr.it
Server:      192.168.1.1
Address:     192.168.1.1#53

Non-authoritative answer:
www.univr.it canonical name = webserv.univr.it.
Name:   webserv.univr.it
Address: 157.27.6.235
```

Modo non interattivo: permette di effettuare una sola query e ovviamente visualizza il risultato della singola query. Abilitato ogni qualvolta si specifichi l'*host-to-find*.

```
sam@osa-Q500A: ~  
sam@osa-Q500A:~$ nslookup linuxandubuntu.com  
Server:      127.0.1.1  
Address:     127.0.1.1#53  
  
Non-authoritative answer:  
Name:   linuxandubuntu.com  
Address: 199.34.228.65  
  
sam@osa-Q500A:~$
```

## 6 ifconfig

Ifconfig (ipconfig in Windows) è un comando utilizzato per configurare e controllare un'interfaccia di rete TCP/IP da riga di comando. L'esecuzione del comando con l'opzione **-a** (oppure **/all** in ipconfig su Windows) mostra a video le informazioni di tutte le interfacce di rete.

```
eth0      Link encap:Ethernet  HWaddr 44:a8:42:ed:33:a8  
          UP BROADCAST MULTICAST  MTU:1500  Metric:1  
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0  
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0  
          collisions:0 txqueuelen:1000  
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)  
  
lo        Link encap:Local Loopback  
          inet addr:127.0.0.1  Mask:255.0.0.0  
          inet6 addr: ::1/128 Scope:Host  
          UP LOOPBACK RUNNING  MTU:65536  Metric:1  
          RX packets:40228 errors:0 dropped:0 overruns:0 frame:0  
          TX packets:40228 errors:0 dropped:0 overruns:0 carrier:0  
          collisions:0 txqueuelen:1  
          RX bytes:3153028 (3.1 MB)  TX bytes:3153028 (3.1 MB)  
  
wlan0     Link encap:Ethernet  HWaddr 5c:e0:c5:e7:9f:c8  
          inet addr:157.27.146.128  Bcast:157.27.151.255  Mask:255.255.248.0  
          inet6 addr: fe80::72c5:7e66:929f:de69/64 Scope:Link  
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1  
          RX packets:455547 errors:0 dropped:0 overruns:0 frame:0  
          TX packets:301399 errors:0 dropped:0 overruns:0 carrier:0  
          collisions:0 txqueuelen:1000  
          RX bytes:348798058 (348.7 MB)  TX bytes:182681107 (182.6 MB)
```

- **eth0** è la prima interfaccia Ethernet (ulteriori interfacce, se presenti, saranno denominate **eth1**, **eth2**, etc.)
- **lo** è l'interfaccia loopback, sempre presente. È un'interfaccia di rete speciale che il Sistema usa per comunicare con se stesso.
- **wlan0** è il nome della prima interfaccia di rete wireless del Sistema. Ulteriori interfacce wireless saranno denominate **wlan1**, **wlan2**, etc.

Questa è la tradizionale convenzione dei nomi per le interfacce di rete sotto sistema Linux, altri sistemi operativi potrebbero avere nomenclature diverse. Per determinare in modo esatto il nome delle interfacce è necessario verificare la propria configurazione consultando la relativa documentazione.

## 7 route

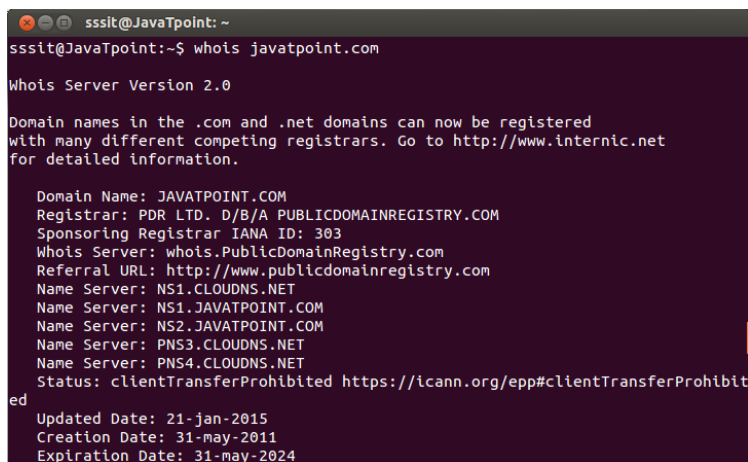
Il comando è utilizzato per vedere e modificare le tabelle di routing. L'esecuzione del comando route su Linux (route PRINT su Windows) permette di visualizzare la tabella di routing dell'host.

```
kaos@kaos:~$ route -n
Kernel IP routing table
Destination      Gateway          Genmask          Flags Metric Ref    Use Iface
192.168.101.0    192.168.102.102 255.255.255.0    UG    0      0      0 eth0
192.168.102.0    0.0.0.0          255.255.255.0    U      0      0      0 eth0
192.168.103.0    192.168.102.102 255.255.255.0    UG    0      0      0 eth0
192.168.12.0     0.0.0.0          255.255.255.0    U      0      0      0 eth0
0.0.0.0          192.168.12.1    0.0.0.0          UG    0      0      0 eth0
```

## 8 whois

Whois è un protocollo di rete che consente, mediante l'interrogazione di appositi database server da parte di un client, di stabilire a quale provider Internet appartenga un determinato indirizzo IP o uno specifico dominio DNS. Nel whois vengono solitamente mostrate anche informazioni riguardanti l'intestatario, data di registrazione e la data di scadenza.

Whois si può consultare tradizionalmente da riga di comando, anche se ora esistono numerosi strumenti web-based per consultare dai database.



```
ssstt@JavaTpoint: ~
ssstt@JavaTpoint:~$ whois javatpoint.com

Whois Server Version 2.0

Domain names in the .com and .net domains can now be registered
with many different competing registrars. Go to http://www.internic.net
for detailed information.

Domain Name: JAVATPOINT.COM
Registrar: PDR LTD. D/B/A PUBLICDOMAINREGISTRY.COM
Sponsoring Registrar IANA ID: 303
Whois Server: whois.PublicDomainRegistry.com
Referral URL: http://www.publicdomainregistry.com
Name Server: NS1.CLOUDNS.NET
Name Server: NS1.JAVATPOINT.COM
Name Server: NS2.JAVATPOINT.COM
Name Server: PNS3.CLOUDNS.NET
Name Server: PNS4.CLOUDNS.NET
Status: clientTransferProhibited https://icann.org/epp#clientTransferProhibit
ed
Updated Date: 21-jan-2015
Creation Date: 31-may-2011
Expiration Date: 31-may-2024
```

## 9. Esercitazione

Per motivi di sicurezza non è possibile analizzare il traffico reale presente nella rete del laboratorio e quindi, per questa esercitazione, useremo del traffico precedentemente catturato.

### Esercizio 1

Occorre avviare wireshark, aprire il menu File/Open e selezionare il file capture.cap.

Si prenda in considerazione il pacchetto numero 9 e si risponda alle seguenti domande/esercizi:

1. che tipo di protocollo di livello datalink è usato? Come fa Wireshark a capirlo?
2. disegnare la PDU di livello datalink indicando il valore dei vari campi

3. qual è il MAC sorgente ? Di che tipo è: unicast o broadcast ?
4. qual è il MAC destinazione ? Di che tipo è: unicast o broadcast ?
5. che tipo di protocollo di livello network è usato? Come fa Wireshark a capirlo?
6. qual è la lunghezza dell'header IP?
7. quali sono gli indirizzi IP sorgente e destinazione?
8. che tipo di protocollo di livello trasporto è contenuto in IP? Come fa Wireshark a capirlo?
9. quali sono le porte sorgente e destinazione a livello trasporto?
10. creare un filtro per visualizzare solo i pacchetti che hanno ARP come protocollo (suggerimento: basta scrivere “arp” nella barra Filter sotto la toolbar; si ricordi di premere su APPLY dopo aver scritto “arp”)
11. dopo aver applicato il filtro precedente qual è la percentuale di pacchetti che rimangono visualizzati rispetto al totale ? (suggerimento: vedere entrambi i valori nella barra di stato in basso)
12. creare un filtro per visualizzare solo i pacchetti che hanno sorgente MAC 00:22:19:c7:2b:ee (suggerimento: usare l'editor di espressioni; la categoria da selezionare è Ethernet; per l'indirizzo MAC usare la notazione esadecimale con i due punti come separatori; si ricordi di premere su APPLY dopo aver creato l'espressione)
13. dopo aver applicato il filtro precedente qual è la percentuale di pacchetti che rimangono visualizzati rispetto al totale ? (suggerimento: vedere entrambi i valori nella barra di stato in basso)
14. creare un filtro per visualizzare solo i pacchetti che hanno destinazione MAC broadcast (suggerimento: nell'editor di espressioni la categoria da usare è Ethernet; per l'indirizzo MAC usare la notazione esadecimale con i due punti come separatori; si ricordi di premere su APPLY dopo aver creato l'espressione)
15. dopo aver applicato il filtro precedente qual è la percentuale di pacchetti che rimangono visualizzati rispetto al totale ? Sono molti ? Perché ?

## Esercizio 2

Occorre aprire il menu File/Open e selezionare il file simpleHTTP.cap.

1. Colorare di rosso tutti i pacchetti che contengono UDP e di verde tutti i pacchetti che contengono TCP (suggerimento: nell'editor delle regole di colorazione è sufficiente portare in alto due regole già esistenti e modificarle per cambiarne i colori di sfondo)
2. Cosa contengono i primi due pacchetti della sessione di cattura?
  - IP sorgente, IP destinazione
  - tipo di protocollo di trasporto
  - tipo di protocollo di livello applicazione. Come fa Wireshark a capirlo?
  - messaggio contenuto nel payload di livello applicazione
3. Prendere in considerazione il pacchetto 3
  - IP sorgente, IP destinazione
  - tipo di protocollo di trasporto
  - IP sorgente e destinazione sono in qualche modo collegati con i messaggi scambiati a



livello applicazione nei primi due pacchetti ? E' possibile fare delle ipotesi su cosa serve il protocollo di livello applicazione dei primi due pacchetti ?

4. Descrivere la struttura del pacchetto 6
  - IP sorgente, IP destinazione
  - tipo di protocollo di trasporto
  - tipo di protocollo di livello applicazione
  - Perché prima della trasmissione del primo messaggio HTTP c'è lo scambio di tre pacchetti puramente TCP? Quali sono i flag settati nell'header TCP di questi tre pacchetti?
4. creare un filtro per visualizzare solo i pacchetti TCP (compresi i pacchetti HTTP) e determinarne il numero.
5. creare un filtro per visualizzare solo i pacchetti TCP (esclusi i pacchetti HTTP) e determinarne il numero.
  - Qual è la percentuale sul totale dei pacchetti TCP trovata al punto 5?
  - A cosa servono tali pacchetti?
  - Se il protocollo DNS dei pacchetti 1 e 2 avesse usato il protocollo TCP, quanti pacchetti IP sarebbero stati generati? Sarebbe stato utile?
4. Selezionare il pacchetto 3 e seguire lo stream TCP col comando da menu Analyze/Follow TCP Stream
  - cosa si può leggere?
  - qual è il messaggio contenuto nel payload della PDU di livello applicazione?

### Esercizio 3

Occorre aprire il menu File/Open e selezionare il file busyNetwork.cap.

1. Elencare i protocolli di livello applicazione entrano in azione in questa cattura classificandoli in base al livello trasporto utilizzato.
2. Provare ad analizzare diversi stream TCP con sopra diversi protocolli di livello applicazione.
3. Che differenza c'è tra il contenuto trasmesso in una connessione TCP per il protocollo FTP e quello trasmesso per il protocollo SSH?

### Esercizio 4

Occorre aprire il menu File/Open e selezionare il file pingCapture.cap.

1. Individuare le richieste ping inviate e le relative risposte. Quante sono? Quali sono i valori dei campi type e code del pacchetto ICMP? Vedere la tabella [ICMP type and code](https://en.wikipedia.org/wiki/Internet_Control_Message_Protocol) ([https://en.wikipedia.org/wiki/Internet\\_Control\\_Message\\_Protocol](https://en.wikipedia.org/wiki/Internet_Control_Message_Protocol)) per la codifica dei codici e tipi.
2. Quali sono IP sorgente e destinazione della richiesta ICMP? Chi li ha registrati?

### Esercizio 5

Occorre aprire il menu File/Open e selezionare il file tracerouteCapture.cap.

1. Individuate i pacchetti “sonda” del traceroute. Qual è l’indirizzo IP dell’ host di partenza? E quale quello di destinazione?
2. Esaminate il campo TTL dei pacchetti “sonda”. Cosa notate?
3. Esaminate i pacchetti ICMP ricevuti dall’host di partenza. Che tipo di messaggi contengono? Da quali indirizzi IP provengono (riportarli in un elenco)?

## **Esercizio 6**

1. Cercare quali interfacce sono attualmente attive. Qual è l’indirizzo IP dell’interfaccia che state utilizzando sul vostro host? E la netmask corrispondente?
2. Qual è l’indirizzo IP di [www.univr.it](http://www.univr.it) ?

## APPENDICE A. Sintassi dei filtri di cattura

La descrizione della sintassi è presa dalla man page di tcpdump.

### SYNOPSIS

```
tcpdump [ -adeflnNOpqStvx ] [ -c count ] [ -F file ]  
        [ -i interface ] [ -r file ] [ -s snaplen ]  
        [ -T type ] [ -w file ] [ expression ]
```

### DESCRIPTION

*Tcpdump* prints out the headers of packets on a network interface that match the boolean *expression*.

### OPTIONS

- c** Exit after receiving *count* packets.
- F** Use *file* as input for the filter expression. An additional expression given on the command line is ignored.
- i** Listen on *interface*. If unspecified, *tcpdump* searches the system interface list for the lowest numbered, configured up interface (excluding loop-back). Ties are broken by choosing the earliest match.
- r** Read packets from *file* (which was created with the *-w* option). Standard input is used if *file* is *``-''*.
- w** Write the raw packets to *file* rather than parsing and printing them out. They can later be printed with the *-r* option. Standard output is used if *file* is *``-''*.

#### *expression*

selects which packets will be dumped. If no *expression* is given, all packets on the net will be dumped. Otherwise, only packets for which *expression* is *`true'* will be dumped.

The *expression* consists of one or more *primitives*. Primitives usually consist of an *id* (name or number) preceded by one or more qualifiers. There are three different kinds of qualifier:

*type* qualifiers say what kind of thing the *id* name or number refers to. Possible types are **host**, **net** and **port**. E.g., *`host foo'*, *`net 128.3'*, *`port 20'*. If there is no type qualifier, **host** is assumed.

*dir* qualifiers specify a particular transfer direction to and/or from *id*. Possible directions are **src**, **dst**, **src or dst** and **src and dst**. E.g., *`src foo'*, *`dst net 128.3'*, *`src or dst port ftp-data'*. If there is no *dir* qualifier, **src or dst** is assumed. For *`null'* link layers (i.e. point to point protocols such as slip) the **inbound** and **outbound** qualifiers can be used to specify a desired direction.

*proto* qualifiers restrict the match to a particu-

lar protocol. Possible protos are: **ether**, **fddi**, **ip**, **arp**, **rarp**, **decnet**, **lat**, **sca**, **moprc**, **mopdl**, **tcp** and **udp**. E.g., ``ether src foo'`, ``arp net 128.3'`, ``tcp port 21'`. If there is no proto qualifier, all protocols consistent with the type are assumed. E.g., ``src foo'` means ``(ip or arp or rarp) src foo'` (except the latter is not legal syntax), ``net bar'` means ``(ip or arp or rarp) net bar'` and ``port 53'` means ``(tcp or udp) port 53'`.

[``fddi'` is actually an alias for ``ether'`; the parser treats them identically as meaning ``the data link level used on the specified network interface.'` FDDI headers contain Ethernet-like source and destination addresses, and often contain Ethernet-like packet types, so you can filter on these FDDI fields just as with the analogous Ethernet fields. FDDI headers also contain other fields, but you cannot name them explicitly in a filter expression.]

In addition to the above, there are some special ``primitive'` keywords that don't follow the pattern: **gateway**, **broadcast**, **less**, **greater** and arithmetic expressions. All of these are described below.

More complex filter expressions are built up by using the words **and**, **or** and **not** to combine primitives. E.g., ``host foo and not port ftp and not port ftp-data'`. To save typing, identical qualifier lists can be omitted. E.g., ``tcp dst port ftp or ftp-data or domain'` is exactly the same as ``tcp dst port ftp or tcp dst port ftp-data or tcp dst port domain'`.

Allowable primitives are:

**dst host** *host*

True if the IP destination field of the packet is *host*, which may be either an address or a name.

**src host** *host*

True if the IP source field of the packet is *host*.

**host** *host*

True if either the IP source or destination of the packet is *host*. Any of the above host expressions can be prepended with the keywords, **ip**, **arp**, or **rarp** as in:

**ip host** *host*

which is equivalent to:

**ether proto** *\ip and host host*

If *host* is a name with multiple IP addresses, each address will be checked for a match.

**ether dst** *ehost*

True if the ethernet destination address is *ehost*. *Ehost* may be either a name from `/etc/ethers` or a number (see `ethers(3N)` for numeric format).

**ether src** *ehost*  
 True if the ethernet source address is *ehost*.

**ether host** *ehost*  
 True if either the ethernet source or destination address is *ehost*.

**gateway** *host*  
 True if the packet used *host* as a gateway. I.e., the ethernet source or destination address was *host* but neither the IP source nor the IP destination was *host*. *Host* must be a name and must be found in both */etc/hosts* and */etc/ethers*. (An equivalent expression is  
**ether host** *ehost* **and not** **host** *host*  
 which can be used with either names or numbers for *host* / *ehost*.)

**dst net** *net*  
 True if the IP destination address of the packet has a network number of *net*. *Net* may be either a name from */etc/networks* or a network number (see *networks(4)* for details).

**src net** *net*  
 True if the IP source address of the packet has a network number of *net*.

**net** *net*  
 True if either the IP source or destination address of the packet has a network number of *net*.

**net net mask** *mask*  
 True if the IP address matches *net* with the specific netmask. May be qualified with **src** or **dst**.

**net net/len**  
 True if the IP address matches *net* a netmask *len* bits wide. May be qualified with **src** or **dst**.

**dst port** *port*  
 True if the packet is ip/tcp or ip/udp and has a destination port value of *port*. The *port* can be a number or a name used in */etc/services* (see *tcp(4P)* and *udp(4P)*). If a name is used, both the port number and protocol are checked. If a number or ambiguous name is used, only the port number is checked (e.g., **dst port 513** will print both tcp/login traffic and udp/who traffic, and **port domain** will print both tcp/domain and udp/domain traffic).

**src port** *port*  
 True if the packet has a source port value of *port*.

**port** *port*  
 True if either the source or destination port of the packet is *port*. Any of the



above port expressions can be prepended with the keywords, **tcp** or **udp**, as in:

**tcp src port** *port*

which matches only tcp packets whose source port is *port*.

**less** *length*

True if the packet has a length less than or equal to *length*. This is equivalent to:

**len** <= *length*.

**greater** *length*

True if the packet has a length greater than or equal to *length*. This is equivalent to:

**len** >= *length*.

**ip proto** *protocol*

True if the packet is an ip packet (see *ip(4P)*) of protocol type *protocol*. *Protocol* can be a number or one of the names *icmp*, *igrp*, *udp*, *nd*, or *tcp*. Note that the identifiers *tcp*, *udp*, and *icmp* are also keywords and must be escaped via backslash (\), which is \\ in the C-shell.

**ether broadcast**

True if the packet is an ethernet broadcast packet. The *ether* keyword is optional.

**ip broadcast**

True if the packet is an IP broadcast packet. It checks for both the all-zeroes and all-ones broadcast conventions, and looks up the local subnet mask.

**ether multicast**

True if the packet is an ethernet multicast packet. The *ether* keyword is optional. This is shorthand for **ether[0] & 1 != 0**.

**ip multicast**

True if the packet is an IP multicast packet.

**ether proto** *protocol*

True if the packet is of ether type *protocol*. *Protocol* can be a number or a name like *ip*, *arp*, or *rarp*. Note these identifiers are also keywords and must be escaped via backslash (\). [In the case of FDDI (e.g., **fddi protocol arp**), the protocol identification comes from the 802.2 Logical Link Control (LLC) header, which is usually layered on top of the FDDI header. *Tcpdump* assumes, when filtering on the protocol identifier, that all FDDI packets include an LLC header, and that the LLC header is in so-called SNAP format.]

**ip, arp, rarp, decnet**

Abbreviations for:

**ether proto** *p*

where *p* is one of the above protocols.

**lat, moprc, mopdl**

Abbreviations for:

**ether proto *p***

where *p* is one of the above protocols. Note that *tcpdump* does not currently know how to parse these protocols.

**tcp, udp, icmp**

Abbreviations for:

**ip proto *p***

where *p* is one of the above protocols.

***expr relop expr***

True if the relation holds, where *relop* is one of **>**, **<**, **>=**, **<=**, **=**, **!=**, and *expr* is an arithmetic expression composed of integer constants (expressed in standard C syntax), the normal binary operators **+**, **-**, **\***, **/**, **&**, **|**, a length operator, and special packet data accessors. To access data inside the packet, use the following syntax:

***proto* [*expr* : *size* ]**

*Proto* is one of **ether**, **fddi**, **ip**, **arp**, **rarp**, **tcp**, **udp**, or **icmp**, and indicates the protocol layer for the index operation. The byte offset, relative to the indicated protocol layer, is given by *expr*. *Size* is optional and indicates the number of bytes in the field of interest; it can be either one, two, or four, and defaults to one. The length operator, indicated by the keyword **len**, gives the length of the packet.

For example, **`ether[0] & 1 != 0`** catches all multicast traffic. The expression **`ip[0] & 0xf != 5`** catches all IP packets with options. The expression **`ip[6:2] & 0x1fff = 0`** catches only unfragmented datagrams and frag zero of fragmented datagrams. This check is implicitly applied to the **tcp** and **udp** index operations. For instance, **tcp[0]** always means the first byte of the TCP header, and never means the first byte of an intervening fragment.

Primitives may be combined using:

A parenthesized group of primitives and operators (parentheses are special to the Shell and must be escaped).

Negation (**`!`** or **`not`**).

Concatenation (**`&&`** or **`and`**).

Alternation (**`||`** or **`or`**).

Negation has highest precedence. Alternation and concatenation have equal precedence and associate left to right. Note that explicit **and** tokens, not juxtaposition, are now required for concatenation.

If an identifier is given without a keyword, the most recent keyword is assumed. For example,

**not host vs and ace**

is short for

**not host vs and host ace**

which should not be confused with

## **not ( host vs or ace )**

Expression arguments can be passed to `tcpdump` as either a single argument or as multiple arguments, whichever is more convenient. Generally, if the expression contains Shell metacharacters, it is easier to pass it as a single, quoted argument. Multiple arguments are concatenated with spaces before being parsed.

### **EXAMPLES**

To print all packets arriving at or departing from *sun-down*:

```
tcpdump host sundown
```

To print traffic between *helios* and either *hot* or *ace*:

```
tcpdump host helios and \( hot or ace \)
```

To print all IP packets between *ace* and any host except *helios*:

```
tcpdump ip host ace and not helios
```

To print all traffic between local hosts and hosts at Berkeley:

```
tcpdump net ucb-ether
```

To print all ftp traffic through internet gateway *snup*: (note that the expression is quoted to prevent the shell from (mis-)interpreting the parentheses):

```
tcpdump 'gateway snup and (port ftp or ftp-data)'
```

To print traffic neither sourced from nor destined for local hosts (if you gateway to one other net, this stuff should never make it onto your local net).

```
tcpdump ip and not net localnet
```

To print the start and end packets (the SYN and FIN packets) of each TCP conversation that involves a non-local host.

```
tcpdump 'tcp[13] & 3 != 0 and not src and dst net localnet'
```

To print IP packets longer than 576 bytes sent through gateway *snup*:

```
tcpdump 'gateway snup and ip[2:2] > 576'
```

To print IP broadcast or multicast packets that were *not* sent via ethernet broadcast or multicast:

```
tcpdump 'ether[0] & 1 = 0 and ip[16] >= 224'
```

To print all ICMP packets that are not echo requests/replies (i.e., not ping packets):

```
tcpdump 'icmp[0] != 8 and icmp[0] != 0'
```