Un po’ di pratica

Network tools

- `netstat`
- `netcat`
- `ping`
- `host`
- `nslookup`
- `wireshark`
- ..and many others

DISCLAIMER

- You are free to use your favourite operating system, but during this and the following practical lectures, we will only refer to **GNU/Linux**.
    - other operating systems may have slightly different behaviours or tool implementations we won’t discuss (although there might be some exception to this rule)
- It is **strongly** recommended to run the examples at home
- For Windows/OSX users:
    - you can run Linux on a virtual machine
    - **VirtualBox** is free and easy to use
    - You can download the image of a XUbuntu distribution from: http://virtualboxes.org/images/xubuntu/
        - it’s very lightweight, should run on older computers too
- Another possibility would be to use a XUbuntu as a **Live** distribution http://xubuntu.org/getxubuntu/ (does not require to install software)

`netstat`

a command line tool that displays network connections, routing tables, interface statistics and so on..

`netstat`

- by default, **`netstat`** shows only the established connections
- using the **`-a`** option, it shows both established and **listening** connections
  - a connection in **`LISTEN`** state typically belongs to a server waiting for clients to connect
- **`netstat`** shows both TCP/UDP/**TCPv6**/**UDPv6** connections and **UNIX sockets**
  - TCPv6, UDPv6: TCP and UDP connections on top of the IPv6 protocol (check out the lectures on IP)
  - UNIX sockets are roughly like a TCP/UDP connection used only for local inter-process communication purposes (**not** covered by this course. Check out the Operating Systems course)

netstat

```
root@bt:~# netstat -a
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address      Foreign Address            State
tcp   0      0      *:ssh               *:*                      LISTEN
tcp   0      0      *:12345             *:*                      LISTEN
tcp   0      0      localhost:7337      *:*                      LISTEN
tcp   0      0      192.168.1.130:51051 mil01s19-in-f12.1:https  ESTABLISHED
tcp   0      0      192.168.1.130:44305 fa-in-f84.1e100.n:https  ESTABLISHED
tcp   0      0      192.168.1.130:41495 zrh04s05-in-f20.1e1:www  ESTABLISHED
tcp   1      0      192.168.1.130:45425 zrh04s05-in-f31.1e1:www  CLOSE_WAIT
tcp   0      0      192.168.1.130:41640 OCSP.AMS1.VERISIGN.:www  TIME_WAIT
...
tcp6  0      0      [::]:ssh            [::]:*                   LISTEN
Active UNIX domain sockets (servers and established)
Proto RefCnt Flags   Type     State        I-Node   Path
unix  2      [ ACC ] STREAM   LISTENING    24790    /tmp/.X11-unix/X0
unix  2      [ ACC ] STREAM   LISTENING    25029    /tmp/.ICE-unix/3732
...
```

netstat

```
root@bt:~# netstat -a
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address        Foreign Address            State
tcp   0      0      *:ssh                 *:*                       LISTEN
tcp   0      0      *:12345               *:*                       LISTEN
tcp   0                  host:7337        *:*                       LISTEN
tcp   0                 68.1.130:51051 mil01s19-in-f12.1:https  ESTABLISHED
tcp   0      0      192.168.1.130:44305 fa-in-f84.1e100.n:https  ESTABLISHED
tcp   0      0      192.168.1.130:41495 zrh04s05-in-f20.1e1:www  ESTABLISHED
tcp   1      0      192.168.1.130:45425 zrh04s05-in-f31.1e1:www  CLOSE_WAIT
tcp   0      0      192.168.1.130:41640 OCSP.AMS1.VERISIGN.:www  TIME_WAIT
...
tcp6  0      0      [::]:ssh              [::]:*                    LISTEN
Active UNIX domain sockets (servers and established)
Proto RefCnt Flags    Type     State        I-Node   Path
unix  2      [ ACC ]  STREAM   LISTENING    24790    /tmp/.X11-unix/X0
unix  2      [ ACC ]  STREAM   LISTENING    25029    /tmp/.ICE-unix/3732
...
```

Protocol used

netstat

```
root@bt:~# netstat -a
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address       Foreign Address           State
tcp   0      0      *:ssh                *:*                      LISTEN
tcp   0      0      *:12345              *:*                      LISTEN
tcp   0      0      localhost:7337       *:*                      LISTEN
tcp   0      0      192.168.1.130:51051 mil01s19-in-f12.1:https  ESTABLISHED
tcp   0      0      192.168.1.130:44305 fa-in-f84.1e100.n:https  ESTABLISHED
tcp   0      0      192.168.1.130:41495 zrh04s05-in-f20.1e1:www  ESTABLISHED
tcp   1      0      192.168.1.130:45425 zrh04s05-in-f31.1e1:www  CLOSE_WAIT
tcp   0      0      192.168.1.130:41640 OCSP.AMS1.VERISIGN.:www  TIME_WAIT
...
tcp6  0      0       [::]:ssh             [::]:*                  LISTEN
Active UNIX domain sockets (servers and established)
Proto RefCnt Flags    Type     State        I-Node   Path
unix  2      [ ACC ]  STREAM   LISTENING    24790   /tmp/.X11-unix/X0
unix  2      [ ACC ]  STREAM   LISTENING    25029   /tmp/.ICE-unix/3732
...
```

netstat

```
root@bt:~# netstat -a
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address         Foreign Address              State
tcp    0      0     *:ssh                  *:*                        LISTEN
tcp    0      0     *:12345                *:*                        LISTEN
                                                                        SHED
                                                                        SHED
                                                                        SHED
                                                                        AIT
                                                                        IT
```

- connections endpoints, in the form addr:port
    - netstat gives a name to any known port (*e.g.,* 22 becomes ‘ssh’, 80 becomes ‘http’, and so on). You can use the **`-n`** option to disable this feature
    - ‘*’ means ‘any’
    - ‘*:ssh’ in the **`Local Address`** column means that a process is listening on the ‘ssh’ (22) port from any interface (*e.g.,* both ethernet and WiFi)
    - For listening connections, ‘*:*’ in the **`Foreign Address`** column means that the server accepts connections from any client

netstat

```
root@bt:~# netstat -a
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address      Foreign Address          State
tcp   0      0      *:ssh               *:*                    LISTEN
tcp   0      0      *:12345             *:*                     LISTEN
tcp   0      0      localhost:7337      *:*                    LISTEN
tcp   0      0      192.168.1.130:51051 mil01s19-in-f12.1:https ESTABLISHED
tcp   0      0      192.168.1.130:44305 fa-in-f84.1e100.n:https ESTABLISHED
tcp   0      0      192.168.1.130:41495 zrh04s05-in-f20.1e1:www ESTABLISHED
                                                                   AIT
                                                                   IT
...
```

- connections endpoints, in the form addr:port
  - for established connections, the **`Foreign Address`** column shows the address:port of the remote endpoint of the connections
  - for established connection, the **`Local Address`** column shows the address:port of the local endpoint of the connections

netstat

```
root@bt:~# netstat -a
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address      Foreign Address           State
tcp   0      0      *:ssh               *:*                     LISTEN
tcp   0      0      *:12345             *:*                     LISTEN
tcp   0      0      localhost:7337      *:*                     LISTEN
tcp   0      0      192.168.1.130:51051 mil01s19-in-f12.1:https ESTABLISHED
                                                                ESTABLISHED
                                                                ESTABLISHED
                                                                CLOSE_WAIT
                                                                TIME_WAIT

                                                                LISTEN

unix  2      [ ACC ]  STREAM   LISTENING     24790    /tmp/.X11-unix/X0
unix  2      [ ACC ]  STREAM   LISTENING     25029    /tmp/.ICE-unix/3732
...
```

- Example of connection states:
  - **LISTEN**: waiting for connections
  - **ESTABLISHED**: the connection is opened
  - **CLOSE_WAIT/TIME_WAIT**: the connection is about to be closed

`netstat`

- Other **`netstat`** options:
  - **`-p`** shows the name of the process that opened the connections
    - needs root privileges
  - **`-t`** shows TCP connections only
  - **`-l`** shows listening connections only
  - **`-4`** does not show TCPv4 or IPv4 connections
  - **`-n`** does not resolve addresses or ports
- Options can be combined together:
  - for example: **`-t4l`** shows only listening TCP connections
- **`netstat -r`** shows the local ***routing table*** (check out the lectures on IP)
  - not very interesting for typical desktop/laptops configurations
- **`netstat -i`** shows info on the available network interfaces (*e.g.,* ethernet, WiFi, local loop)

netstat

Example. Let’s check how many connections Spotify uses (next slide)

netstat

```
netstat -t -l -p -4
```

Means: show all the TCP connections (**`-t`**) based on IPv4 (**`-4`**) that are in the LISTEN state (**`-l`**). Print also the PID of the process associated to each connection (**`-p`**)

```
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address Foreign Address State    PID/Program name
tcp      0   0 localhost:4371          *:*         LISTEN   9269/spotify
tcp      0   0 *:57621                 *:*         LISTEN   9269/spotify
tcp      0   0 *:ssh                   *:*         LISTEN   1146/sshd
tcp      0   0 localhost:4381          *:*         LISTEN   9269/spotify
tcp      0   0 localhost:7337          *:*         LISTEN   1041/postgres.bin
tcp      0   0 *:29642                 *:*         LISTEN   9269/spotify
```

netstat

netstat -t -l -p -4

```
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address Foreign Address State     PID/Program name
tcp        0      0 localhost:4371          *:*             LISTEN     9269/spotify
                                                                        9269/spotify
                                                                        1146/sshd
                                                                        9269/spotify
                                                                        1041/postgres.bin
                                                                        9269/spotify
```

- This column appeared because we used the **`-p`** option
- It shows, for each entry, the PID and the name of the process the relative connection is associated to. What is a PID? Check out the Operating Systems course.

netstat

```
netstat -t -l -p -4
```

```
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address Foreign Address State    PID/Program name
tcp        0      0 localhost:4371          *:*             LISTEN     9269/spotify
tcp        0      0 *:57621                 *:*             LISTEN     9269/spotify
tcp        0      0 *:ssh                   *:*             LISTEN     1146/sshd
tcp        0      0 localhost:4381          *:*             LISTEN     9269/spotify
tcp        0      0 localhost:7337          *:*             LISTEN     1041/postgres.bin
tcp        0      0 *:29642                 *:*             LISTEN     9269/spotify
```

So, Spotify is waiting for connections to ports 57621 and 29642 coming from **ANY** network interface (*e.g.,* WiFi and ethernet)

netstat

```
netstat -t -l -p -4
```

```
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address Foreign Address State    PID/Program name
tcp      0   0 localhost:4371           *:*           LISTEN   9269/spotify
tcp      0   0 *:57621                  *:*           LISTEN   9269/spotify
tcp      0   0 *:ssh                    *:*           LISTEN   1146/sshd
tcp      0   0 localhost:4381           *:*           LISTEN   9269/spotify
tcp      0   0 localhost:7337           *:*           LISTEN   1041/postgres.bin
tcp      0   0 *:29642                  *:*           LISTEN   9269/spotify
```

.. and on ports 4371 and 4381 from the virtual
internal interface only

netstat

Means: show all the TCP connections (**-t**) based on IPv4 (**-4**) that are not in the LISTEN state (**-l** is omitted). Print also the PID of the process associated to each connection (**-p**)

netstat -t -p -4

```
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 192.168.1.128:42948     host81-148-21-127:18671 ESTABLISHED 9691/spotify
tcp        0      0 192.168.1.128:44735     host109-153-120-2:26071 ESTABLISHED 9691/spotify
tcp        0      0 192.168.1.128:39386     i19-les02-ntr-176:17048 ESTABLISHED 9691/spotify
tcp        0      0 192.168.1.128:45017     cpc8-seac19-2-0-c:47488 ESTABLISHED 9691/spotify
tcp        0      0 192.168.1.128:58314     96.29.82.79.rev.s:18428 ESTABLISHED 9691/spotify
tcp        0      0 192.168.1.128:54971     fa-in-f189.1e100.:https ESTABLISHED 8997/firefox
tcp        0      0 192.168.1.128:44571     178-26-158-174-dy:63235 ESTABLISHED 9691/spotify
tcp        0      0 192.168.1.128:42548     bl10-81-202.dsl.t:13687 ESTABLISHED 9691/spotify
tcp        0      0 192.168.1.128:34983     68.232.34.151:www       ESTABLISHED 9691/spotify
tcp        0      1 192.168.1.128:60928     bl15-104-193.dsl.:39711 SYN_SENT    9691/spotify
tcp        0    726 192.168.1.128:53426     host109-145-57-13:24432 ESTABLISHED 9691/spotify
tcp        0      0 192.168.1.128:42496     cdt33-1-88-177-70:43360 ESTABLISHED 9691/spotify
tcp        0      0 192.168.1.128:40787     169.130.79.188.dy:32885 ESTABLISHED 9691/spotify
tcp        0      0 192.168.1.128:46408     5.226-134-109.ads:40967 ESTABLISHED 9691/spotify
tcp        0      1 192.168.1.128:45038     24.133.118.209:16100    SYN_SENT    9691/spotify
tcp        0      1 192.168.1.128:55793     78-21-193-22.acce:55959 SYN_SENT    9691/spotify
tcp        0      1 192.168.1.128:37999     195-132-159-157.r:24555 SYN_SENT    9691/spotify
tcp        0      0 192.168.1.128:38959     greta.lon.spotify:https ESTABLISHED 9691/spotify
tcp        0      0 192.168.1.128:54784     host109-145-62-17:54001 ESTABLISHED 9691/spotify
tcp        0      0 192.168.1.128:33482     thebreakfa96.pnds:55664 ESTABLISHED 9691/spotify
tcp        0      0 192.168.1.128:34698     82-135-201-51.sta:35423 ESTABLISHED 9691/spotify
tcp        0      0 192.168.1.128:34333     bl18-112-171.dsl.:26916 ESTABLISHED 9691/spotify
tcp        0      0 192.168.1.128:44186     ip-178-201-42-170:44792 ESTABLISHED 9691/spotify
```

netstat

netstat -t -p -4

```
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 192.168.1.128:42948     host81-148-21-127:18671 ESTABLISHED 9691/spotify
tcp        0      0 192.168.1.128:44735     host109-153-120-2:26071 ESTABLISHED 9691/spotify
tcp        0      0 192.168.1.128:39386     i19-les02-ntr-176:17048 ESTABLISHED 9691/spotify
tcp        0      0 192.168.1.128:45017     cpc8-seac19-2-0-c:47488 ESTABLISHED 9691/spotify
tcp        0      0 192.168.1.128:58314     96.29.82.79.rev.s:18428 ESTABLISHED 9691/spotify
tcp        0      0 192.168.1.128:54971     fa-in-f189.1e100.:https ESTABLISHED 8997/firefox
tcp        0      0 192.168.1.128:44571     178-26-158-174-dy:63235 ESTABLISHED 9691/spotify
tcp                                                                          spotify
tcp                                                                          spotify
tcp                                                                          spotify
tcp                                                                          spotify
tcp                                                                          spotify
tcp                                                                          spotify
tcp        0      0 192.168.1.128:46408     5.226-134-109.ads:40967 ESTABLISHED 9691/spotify
tcp        0      1 192.168.1.128:45038     24.133.118.209:16100    SYN_SENT    9691/spotify
tcp        0      1 192.168.1.128:55793     78-21-193-22.acce:55959 SYN_SENT    9691/spotify
tcp        0      1 192.168.1.128:37999     195-132-159-157.r:24555 SYN_SENT    9691/spotify
tcp        0      0 192.168.1.128:38959     greta.lon.spotify:https ESTABLISHED 9691/spotify
tcp        0      0 192.168.1.128:54784     host109-145-62-17:54001 ESTABLISHED 9691/spotify
tcp        0      0 192.168.1.128:33482     thebreakfa96.pnds:55664 ESTABLISHED 9691/spotify
tcp        0      0 192.168.1.128:34698     82-135-201-51.sta:35423 ESTABLISHED 9691/spotify
tcp        0      0 192.168.1.128:34333     bl18-112-171.dsl.:26916 ESTABLISHED 9691/spotify
tcp        0      0 192.168.1.128:44186     ip-178-201-42-170:44792 ESTABLISHED 9691/spotify
```

A connection belonging to the Firefox web browser

netstat

netstat -t -p -4

```
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 192.168.1.128:42948     host81-148-21-127:18671 ESTABLISHED 9691/spotify
tcp        0      0 192.168.1.128:44735     host109-153-120-2:26071 ESTABLISHED 9691/spotify
tcp        0      0 192.168.1.128:39386     i19-les02-ntr-176:17048 ESTABLISHED 9691/spotify
tcp        0      0 192.168.1.128:45017     cpc8-seac19-2-0-c:47488 ESTABLISHED 9691/spotify
tcp        0      0 192.168.1.128:58314     96.29.82.79.rev.s:18428 ESTABLISHED 9691/spotify
tcp        0      0 192.168.1.128:54971     fa-in-f189.1e100.:https ESTABLISHED 8997/firefox
tcp        0      0 192.168.1.128:44571     178-26-158-174-dy:63235 ESTABLISHED 9691/spotify
tcp                                                                             /spotify
tcp                                                                             /spotify
tcp                                                                             /spotify
tcp                                                                             /spotify
tcp                                                                             /spotify
tcp                                                                             /spotify
tcp        0      0 192.168.1.128:46408     5.226-134-109.ads:40967 ESTABLISHED 9691/spotify
tcp        0      1 192.168.1.128:45038     24.133.118.209:16100    SYN_SENT    9691/spotify
tcp        0      1 192.168.1.128:55793     78-21-193-22.acce:55959 SYN_SENT    9691/spotify
tcp        0      1 192.168.1.128:37999     195-132-159-157.r:24555 SYN_SENT    9691/spotify
tcp        0      0 192.168.1.128:38959     greta.lon.spotify:https ESTABLISHED 9691/spotify
tcp        0      0 192.168.1.128:54784     host109-145-62-17:54001 ESTABLISHED 9691/spotify
tcp        0      0 192.168.1.128:33482     thebreakfa96.pnds:55664 ESTABLISHED 9691/spotify
tcp        0      0 192.168.1.128:34698     82-135-201-51.sta:35423 ESTABLISHED 9691/spotify
tcp        0      0 192.168.1.128:34333     bl18-112-171.dsl.:26916 ESTABLISHED 9691/spotify
tcp        0      0 192.168.1.128:44186     ip-178-201-42-170:44792 ESTABLISHED 9691/spotify
```

`SYN_SENT`: means Spotify is trying to open a connection (check out the lectures on TCP)

netstat

netstat -t -p -4

```
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address            Foreign Address         State     PID/Program name
tcp          0      0 192.168.1.128:42948      host81-148-21-127:18671 ESTABLISHED 9691/spotify
tcp          0      0 192.168.1.128:44735      host109-153-120-2:26071 ESTABLISHED 9691/spotify
tcp          0      0 192.168.1.128:39386      i19-les02-ntr-176:17048 ESTABLISHED 9691/spotify
tcp          0      0 192.168.1.128:45017      cpc8-seac19-2-0-c:47488 ESTABLISHED 9691/spotify
tcp          0      0 192.168.1.128:58314      96.29.82.79.rev.s:18428 ESTABLISHED 9691/spotify
tcp          0      0 192.168.1.128:54971      fa-in-f189.1e100.:https ESTABLISHED 8997/firefox
tcp          0      0 192.168.1.128:44571      178-26-158-174-dy:63235 ESTABLISHED 9691/spotify
tcp          0      0 192.168.1.128:42548      bl10-81-202.dsl.t:13687 ESTABLISHED 9691/spotify
tcp          0      0 192.168.1.128:34983      68.232.34.151:www       ESTABLISHED 9691/spotify
tcp          0      1 192.168.1.128:60928      bl15-104-193.dsl.:39711 SYN_SENT    9691/spotify
tcp          0    726 192.168.1.128:53426      host109-145-57-13:24432 ESTABLISHED 9691/spotify
tcp                                                                            HED 9691/spotify
tcp                                                                            HED 9691/spotify
tcp                                                                            HED 9691/spotify
tcp                                                                                9691/spotify
tcp                                                                                9691/spotify
tcp          0      1 192.168.1.128:37999      195-132-159-157.r:24555 SYN_SENT    9691/spotify
tcp          0      0 192.168.1.128:38959      greta.lon.spotify:https ESTABLISHED 9691/spotify
tcp          0      0 192.168.1.128:54784      host109-145-62-17:54001 ESTABLISHED 9691/spotify
tcp          0      0 192.168.1.128:33482      thebreakfa96.pnds:55664 ESTABLISHED 9691/spotify
tcp          0      0 192.168.1.128:34698      82-135-201-51.sta:35423 ESTABLISHED 9691/spotify
tcp          0      0 192.168.1.128:34333      bl18-112-171.dsl.:26916 ESTABLISHED 9691/spotify
tcp          0      0 192.168.1.128:44186      ip-178-201-42-170:44792 ESTABLISHED 9691/spotify
```

An https connection towards a Spotify Server

netstat

`netstat -t -n -4`

Just like the previous command, but with the **`-n`** option, telling netstat to not give names to addresses (through reverse DNS queries)

```
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address              Foreign Address            State
tcp           0     0 192.168.1.128:42948      81.148.21.127:18671       ESTABLISHED
tcp           0     0 192.168.1.128:44735      109.153.120.206:26071     ESTABLISHED
tcp           0     0 192.168.1.128:39386      176.186.160.160:17048     ESTABLISHED
tcp           0     0 192.168.1.128:45017      81.108.153.96:47488       ESTABLISHED
tcp           0     0 192.168.1.128:58314      79.82.29.96:18428         ESTABLISHED
tcp           0     0 192.168.1.128:54971      173.194.70.189:443        ESTABLISHED
tcp           0     0 192.168.1.128:44571      178.26.158.174:63235      ESTABLISHED
tcp           0     0 192.168.1.128:42548      85.243.81.202:13687       ESTABLISHED
tcp           0     1 192.168.1.128:34049      78.146.230.119:52451      SYN_SENT
tcp           0     0 192.168.1.128:59208      173.194.116.14:443        ESTABLISHED
tcp           0     0 192.168.1.128:46408      109.134.226.5:40967       ESTABLISHED
tcp           0     1 192.168.1.128:41011      2.240.42.97:24628         SYN_SENT
tcp           0     0 192.168.1.128:38959      78.31.8.16:443            ESTABLISHED
tcp           0     1 192.168.1.128:34426      84.30.100.93:43383        SYN_SENT
tcp           0     0 192.168.1.128:54784      109.145.62.171:54001      ESTABLISHED
tcp           0     0 192.168.1.128:34957      68.232.34.151:80          TIME_WAIT
tcp           0     0 192.168.1.128:33482      80.229.251.184:55664      ESTABLISHED
tcp           0     0 192.168.1.128:34698      82.135.201.51:35423       ESTABLISHED
tcp           0     0 192.168.1.128:34333      188.83.112.171:26916      ESTABLISHED
tcp           0     0 192.168.1.128:44186      178.201.42.170:44792      ESTABLISHED
```

netstat

netstat -t -n -4

Just like the previous command, but with the **-n** option, telling netstat to not give names to addresses (through reverse DNS queries)

```
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address              Foreign Address          State
tcp          0      0 192.168.1.128:42948      81.148.21.127:18671      ESTABLISHED
tcp          0      0 192.168.1.128:44735      109.153.120.206:26071    ESTABLISHED
tcp          0      0 192.168.1.128:39386      176.186.160.160:17048    ESTABLISHED
tcp          0      0 192.168.1.128:45017      81.108.153.96:47488      ESTABLISHED
tcp          0      0 192.168.1.128:58314      79.82.29.96:18428        ESTABLISHED
tcp          0      0 192.168.1.128:54971      173.194.70.189:443       ESTABLISHED
tcp          0      0 192.168.1.128:44571      178.26.158.174:63235     ESTABLISHED
tcp          0      0 192.168.1.128:42548      85.243.81.202:13687      ESTABLISHED
tcp          0      1 192.168.1.128:34049      78.146.230.119:52451     SYN_SENT
                                               173.194.116.14:443       ESTABLISHED
                                               109.134.226.5:40967      ESTABLISHED
                                               2.240.42.97:24628        SYN_SENT
                                               78.31.8.16:443           ESTABLISHED
                                               84.30.100.93:43383       SYN_SENT
tcp          0      0 192.168.1.128:54784      109.145.62.171:54001     ESTABLISHED
tcp          0      0 192.168.1.128:34957      68.232.34.151:80         TIME_WAIT
tcp          0      0 192.168.1.128:33482      80.229.251.184:55664     ESTABLISHED
tcp          0      0 192.168.1.128:34698      82.135.201.51:35423      ESTABLISHED
tcp          0      0 192.168.1.128:34333      188.83.112.171:26916     ESTABLISHED
tcp          0      0 192.168.1.128:44186      178.201.42.170:44792     ESTABLISHED
```

Names have been replaced by IP addresses (and ports).

netstat

`netstat -t -n -4`

Just like the previous command, but with the **-n** option, telling netstat to not give names to addresses (through reverse DNS queries)

```
Active Internet connections (w/o servers)
                                          State
671                                     ESTABLISHED
26071                                   ESTABLISHED
17048                                   ESTABLISHED
488                                     ESTABLISHED
8                                       ESTABLISHED
43                                      ESTABLISHED
3235                                    ESTABLISHED
687                                     ESTABLISHED
2451                                    SYN_SENT
43                                      ESTABLISHED
967                                     ESTABLISHED
8                                       SYN_SENT
                                        ESTABLISHED
83                                      SYN_SENT
4001                                    ESTABLISHED
                                        TIME_WAIT
5664                                    ESTABLISHED
423                                     ESTABLISHED
6916                                    ESTABLISHED
4792                                    ESTABLISHED
```

Overall:

- 1 connection to Belgium
- 3 connections to Germany
- 2 connections to France
- 6 connections to U.K.
- 1 connection to Lithuania
- 1 connection to Luxembourg
- 1 connection to Netherlands
- 2 connections to Portugal
- 3 connections to the U.S.

Spotify **truly is** a world-wide P2P network!

`nslookup`

`nslookup` is a command-line tool to query Internet **D**omain **N**ame **S**ervers (DNS) interactively

`nslookup`

(simplified) syntax:

```
nslookup [-type=TYPE] name [server]
```

by default, it tells the name server to perform a **recursive query**

`nslookup`

(simplified) syntax:

```
nslookup [-type=TYPE] name [server]
```

default: **`-type=A`**

default
(system configured)
`/etc/resolv.conf`
on Linux

`nslookup`

possible types are:

Type	**Meaning**
A	IPv4 address of a host
AAAA	IPv6 address of a host
MX	Domain willing to accept mail
NS	Name of a server for this domain
PTR	Alias for an IP address
CNAME	Alias of one name to another
…	...

nslookup

Example N.1: **`nslookup uniroma1.it`**

Means: give me the address of the domain **`uniroma1.it`**

```
Server:      8.8.8.8
Address:     8.8.8.8#53
```

My default name server

```
Non-authoritative answer:
Name:    uniroma1.it
Address: 151.100.101.67
```

the answer is **not authorative** because 8.8.8.8 is not the manager of the root of the tree. These values are coming from 8.8.8.8’s **cache**

IP address associated to the **`uniroma1.it`** domain

so.. **who** is responsible for the domain **`uniroma1.it`** ?

nslookup

Means: give me the name server responsible for the domain **uniroma1.it**

Example N.2: **nslookup -type=NS uniroma1.it**

```
Server:     8.8.8.8
Address:    8.8.8.8#53

Non-authoritative answer:
uniroma1.it    nameserver = risc-ns.cics.uniroma1.it.
uniroma1.it    nameserver = desiree.cics.uniroma1.it.
uniroma1.it    nameserver = ns1.garr.net.

Authoritative answers can be found from:
```

3 nameservers for **uniroma1.it** (may be for fault tolerance reasons)

let’s ask one of them for an authorative answer for **uniroma1.it**

nslookup

IP address of **`desiree.cics.uniroma1.it`**

Example N.3: **`nslookup uniroma1.it 151.100.4.13`**

```
Server:    151.100.4.13
Address:   151.100.4.13#53

Name:    uniroma1.it
Address: 151.100.101.67
```

Finally! Next question is: who is responsible for the **root** of the tree?

nslookup

Means: give me the name server responsible for the **root** domain ‘.’

Example N.4: **`nslookup -type=NS .`**

```
Server:     8.8.8.8
Address:    8.8.8.8#53

Non-authoritative answer:
.    nameserver = b.root-servers.net.
.    nameserver = e.root-servers.net.
.    nameserver = f.root-servers.net.
.    nameserver = j.root-servers.net.
…
.    nameserver = h.root-servers.net.

Authoritative answers can be found from:
```

the answer is **not authorative** because 8.8.8.8 is not responsible for the root of the tree. These values are coming from 8.8.8.8’s **cache**

root name servers (there are 13)

nslookup

Means: give me the alternative names of '**phd.di.uniroma1.it**'

Example N.5: **`nslookup -type=CNAME phd.di.uniroma1.it`**

```
Server:		8.8.8.8
Address:	8.8.8.8#53

Non-authoritative answer:
phd.di.uniroma1.it	canonical name = ccalcolo.di.uniroma1.it.

Authoritative answers can be found from:
```

`ccalcolo` is the actual name of the **`phd`** host

nslookup

Example N.6, the **-norecurse** option:

```
nslookup -norecurse venere.di.uniroma1.it
```

```
Server:     8.8.8.8
Address:    8.8.8.8#53

Non-authoritative answer:
*** Can't find venere.di.uniroma1.it: No answer
```

OK, venere.di.uniroma1.it does not exist ...?
To be sure, let’s ask to **desiree.cics.uniroma1.it** (next slide)

nslookup

```
nslookup -norecurse venere.di.uniroma1.it 151.100.4.13
```

```
Server:         151.100.4.13
Address:        151.100.4.13#53

Name:      venere.di.uniroma1.it
Address: 151.100.17.16
```

venere’s address is **151.100.17.16**
Wait.. what? According to **8.8.8.8**, venere.di.uniroma1.it does not exist!
Well, let’s ask again to **8.8.8.8**, but without the **-norecurse** option (next slide)

nslookup

nslookup venere.di.uniroma1.it

```
Server:     8.8.8.8
Address:    8.8.8.8#53

Non-authoritative answer:
Name:    venere.di.uniroma1.it
Address: 151.100.17.16
```

8.8.8.8 was able to find venere.di.uniroma1.it now
So, it was the **-norecurse** option’s fault!
Are we sure? Let’s double check, using the **-norecurse** option again (next slide)

nslookup

```
nslookup -norecurse venere.di.uniroma1.it
```

```
Server:     8.8.8.8
Address:    8.8.8.8#53

Non-authoritative answer:
Name:    venere.di.uniroma1.it
Address: 151.100.17.16
```

OK, now I’m confused. Why could **8.8.8.8** find the address of **venere.di.uniroma1.it**?
(check out next slide)

nslookup

Explaination: **`8.8.8.8`** is **`not`** responsible for the **`venere.di.uniroma1.it`** domain (whereas **`desiree.cics.uniroma1.it`** is). By using the **`-norecurse`** option, we are not allowing **`8.8.8.8`** to navigate the domain tree to retrieve the IP of **`venere.di.uniroma1.it`**. That’s why it could not find it. However, when the **`8.8.8.8`** is allowed to perform a recursive search, not only it successfully finds **`venere.di.uniroma1.it`**, but it also **caches** the answer, so as to speed up the search next time someone (*i.e.,* you, or some other user) asks for the same information. Once the reply it’s cached, even if the **`-norecurse`** option is used, **`8.8.8.8`** can retrieve the answer from its cache.

`nslookup`

Exercise: use **`nslookup`** to find out what are the name servers responsible for the domains:

- `.` (root)
- **`it.`**
- **`uniroma1.it.`**
- **`di.uniroma1.it.`**
- **`redi.uniroma1.it.`**

Discuss the results based on what you know about the structure of **DNS**

netcat

- It’s the “TCP/IP swiss army knife”:
  - reads and writes data across network connections, using TCP or UDP protocol.
  - it is a feature-rich network debugging and exploration tool, since it can create almost any kind of connection you would need and has several interesting built-in capabilities.

`netcat`

Simple example: 2 users chat:

- open a new terminal window and type:
  - **`nc -l -p 12345`**
  - means: act as a server and listen for a new connection (**`-l`**) on port (**`-p`**) 12345. Listens for connections from **any** interface
- open another terminal window and type:
  - **`nc localhost 12345`**
  - means: act as a client and connect to localhost on port 12345
- whatever is written on a terminal (followed by a new line) will appear on the other terminal (and viceversa)

`netcat`

Simple example: 2 users chat:

- the same example works between two ***remote*** machines
  - the machine acting as a server has to be reachable by the client
  - ‘localhost’ must be replaced by the address (or name) of the server

`netcat`

More useful example: **copy** ‘picture.png’ between two remote machines:

- on the receiver side (address a.b.c.d), open a new terminal window and type:
  - **`nc -l -p 12345 > picture.png`**
  - **`‘>’`** is a shell command that redirects the output of nc to the file **`picture.png`**
- on the sender side:
  - **`nc a.b.c.d 12345 < picture.png`**
  - **`‘<’`** is a shell command that writes the contents of the file **`picture.png`** to the input of **`nc`**

`netcat`

The same example works by switching the roles:

- on the sender side (address a.b.c.d), open a new terminal window and type:
  - **`nc -l -p 12345 < picture.png`**
  - sends the content of the **`picture.png`** file to *any* client
- on the receiver side:
  - **`nc a.b.c.d 12345 > picture.png`**
  - writes the output of the server to the **`picture.png`** file

`netcat`

`netcat` can talk to ***any*** server/client, not just other netcat instances!

Example, retrieve a page from a web server:

- type:

```
nc google.it 80
GET / HTTP/1.1
```

- followed by two new lines

netcat

Server response:

```
HTTP/1.1 302 Found
Location: http://www.google.it/?gws_rd=cr&ei=WGVeUpWIAsjGtQaLsoDIDA
Cache-Control: private
Content-Type: text/html; charset=UTF-8
Set-Cookie: PREF=ID=6df6a36cfeac9258:FF=0:TM=1381918040:LM=1381918040:S=TxsHtJMBvvGYb-XB; expires=Fri, 16-Oct-2015 10:07:20 GMT; path=/; domain=.google.com
...
<HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=utf-8">
<TITLE>302 Moved</TITLE></HEAD><BODY>
<H1>302 Moved</H1>
The document has moved
<A HREF="http://www.google.it/?gws_rd=cr&amp;ei=WGVeUpWIAsjGtQaLsoDIDA">here</A>.
</BODY></HTML>
```

`netcat`

Notice:

- we got a “**`302 moved`**” message from the server (a redirection to http://www.google.it/...WGVeUpWIAsjGtQaLsoDIDA)
- **`netcat`** does not talk HTTP, so
  - it won’t follow the redirect
  - it won’t download the other page contents and so on
- But, *in principle*, with **A LOT** of patience, you could use netcat to browse (part of) the web manually (just pretend to be a browser)
  - (don’t try this at home!)

netcat

Similar example: act as a Web Server!

- type
  - nc -l -p 80
- use your favourite web browser to go to:
  - http://localhost:80
- go back to the terminal, you’ll see something like:

```
GET / HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:27.0) Gecko/20100101 Firefox/27.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

netcat

The browser is asking for the / page in our (fake?) Web Server

If we don’t reply, the connection is eventually going to be closed (timeout). So, type in terminal:

```
<html>
Hello, <b>world</b>!
</html>
```

then close the connection with CTRL+C. Now, go back to the browser window. The page should have been loaded by now.

`netcat`

Interesting fact:

- our `netcat`-based web server probably **violated** the HTTP protocol (our response did not include the header!)
- This is **BAD!!!** Still, the web browser did not complain, and figured out how to display the webpage nicely
- This is because web browsers have become **very good** at talking to careless web servers who do not comply with standards

`netcat`

With a similar approach, you can use netcat to talk to:

- Mail servers
- DNS servers
- FTP servers
- …

It may help getting a better idea about how some protocols work. **BUT, always** refer to the relative **RFC** to know what it is allowed or not by the protocol! Complying to protocols is the only right way to keep the Internet working (though being tolerant to protocol violations of *other* people helps a lot)

`netcat`

`netcat` VS `telnet`:

- `telnet` is a command line tool that speaks the Telnet protocol
  - for instance, it requires a carriage return character to be followed by a null (‘`\0`’) character
- since the Telnet protocol is very simple (just a bidirectional text oriented protocol), `telnet` may be used to open raw TCP connections to any server
- `netcat`, on the other hand, has been built with the specific purpose of opening raw connections. **It does not have** any protocol to comply to
  - everything is ***always*** transmitted as-it-is from source to destination (and viceversa)
- `netcat` supports a much richer set of features with respect to `telnet,` for example
  - can be used to send arbitrary binary data
  - supports both TCP and UDP
  - allows to perform TCP port scanning
  - ...

`wireshark`

Enough with the application level. Let’s **dive** in the TCP/IP stack with **`wireshark`**!

`wireshark`

- Wireshark is a software (packet analyzer) that allows to monitor the incoming/outgoing network frames
  - it captures **a copy** of the frames
  - does not inject traffic
- it can expose the **whole** content of each frame (*i.e.,* the whole protocol stack)
- very useful for
  - learning how TCP/IP works
  - network administrators
- it is **not** a security tool
- Wireshark is a rather complex and powerful tool, whose complete set of functionalities cannot be discussed with a single lecture
  - we will cover its basics only
- other packet analyzers:
  - **`tcpdump`, `tshark`**

wireshark

To install Wireshark on Windows or OSX, go to http://www.wireshark.org

On a Debian-based GNU/Linux distribution (*e.g.*, Ubuntu, Linux Mint.. and Debian), just open a terminal window and type:

- **`apt-get install wireshark`**

When the installation is complete, just type

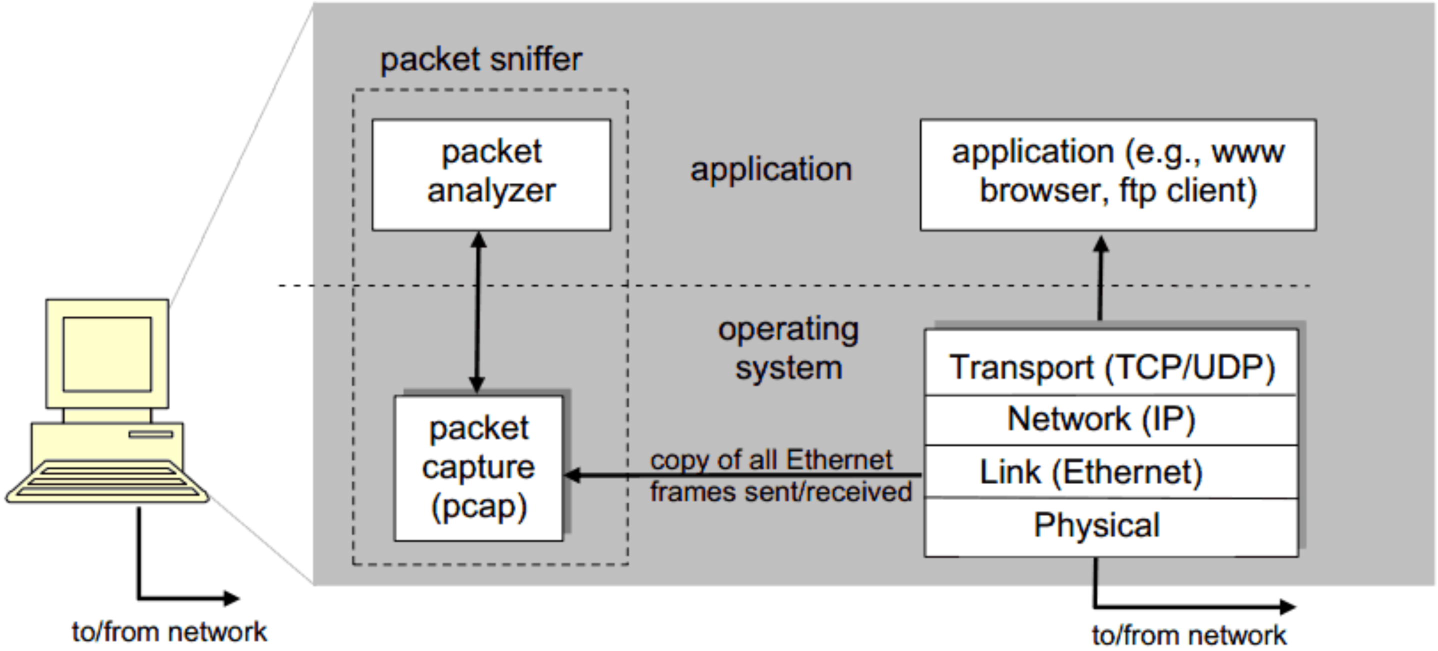
- **`wireshark`**

on a terminal (or run it from the applications menu)

Useful links:

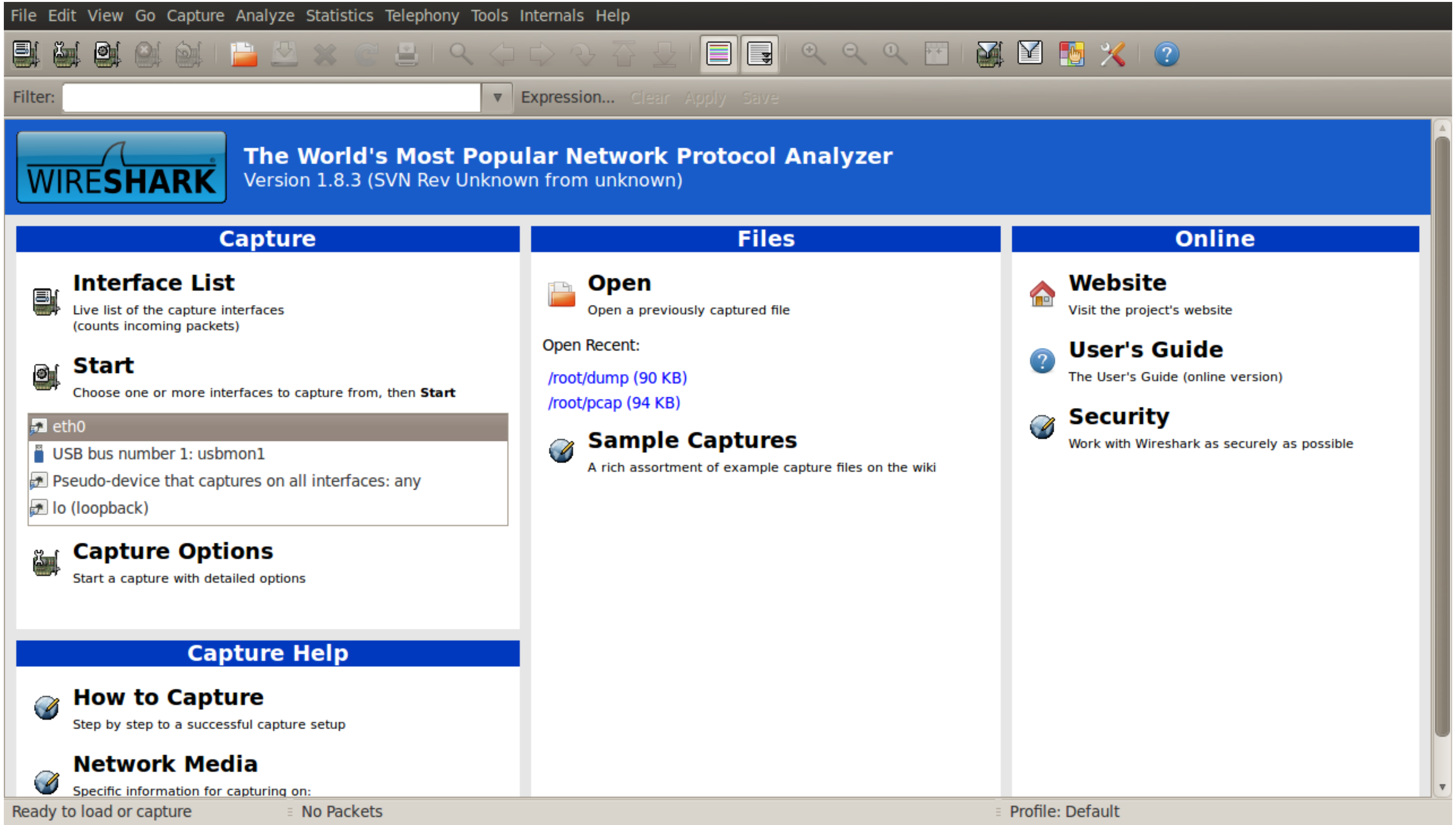
- http://wiki.wireshark.org/CaptureSetup
- https://www.wireshark.org/docs/wsug_html_chunked/
- http://wiki.wireshark.org/SampleCaptures

wireshark



pcap: Packet capture library

wireshark



File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression...
Clear
Apply
Save
WIRESHARK

The World's Most Popular Network Protocol Analyzer
Version 1.8.3 (SVN Rev Unknown from unknown)

Capture

Interface List
Live list of the capture interfaces
(counts incoming packets)

Start
Choose one or more interfaces to capture from, then **Start**

eth0
USB bus number 1: usbmon1
Pseudo-device that captures on all interfaces: any
lo (loopback)

Capture Options
Start a capture with detailed options

Capture Help

How to Capture
Step by step to a successful capture setup

Network Media
Specific information for capturing on:

Files

Open
Open a previously captured file

Open Recent:
/root/dump (90 KB)
/root/pcap (94 KB)

Sample Captures
A rich assortment of example capture files on the wiki

Online

Website
Visit the project's website

User's Guide
The User's Guide (online version)

Security
Work with Wireshark as securely as possible

Ready to load or capture | No Packets | Profile: Default

wireshark

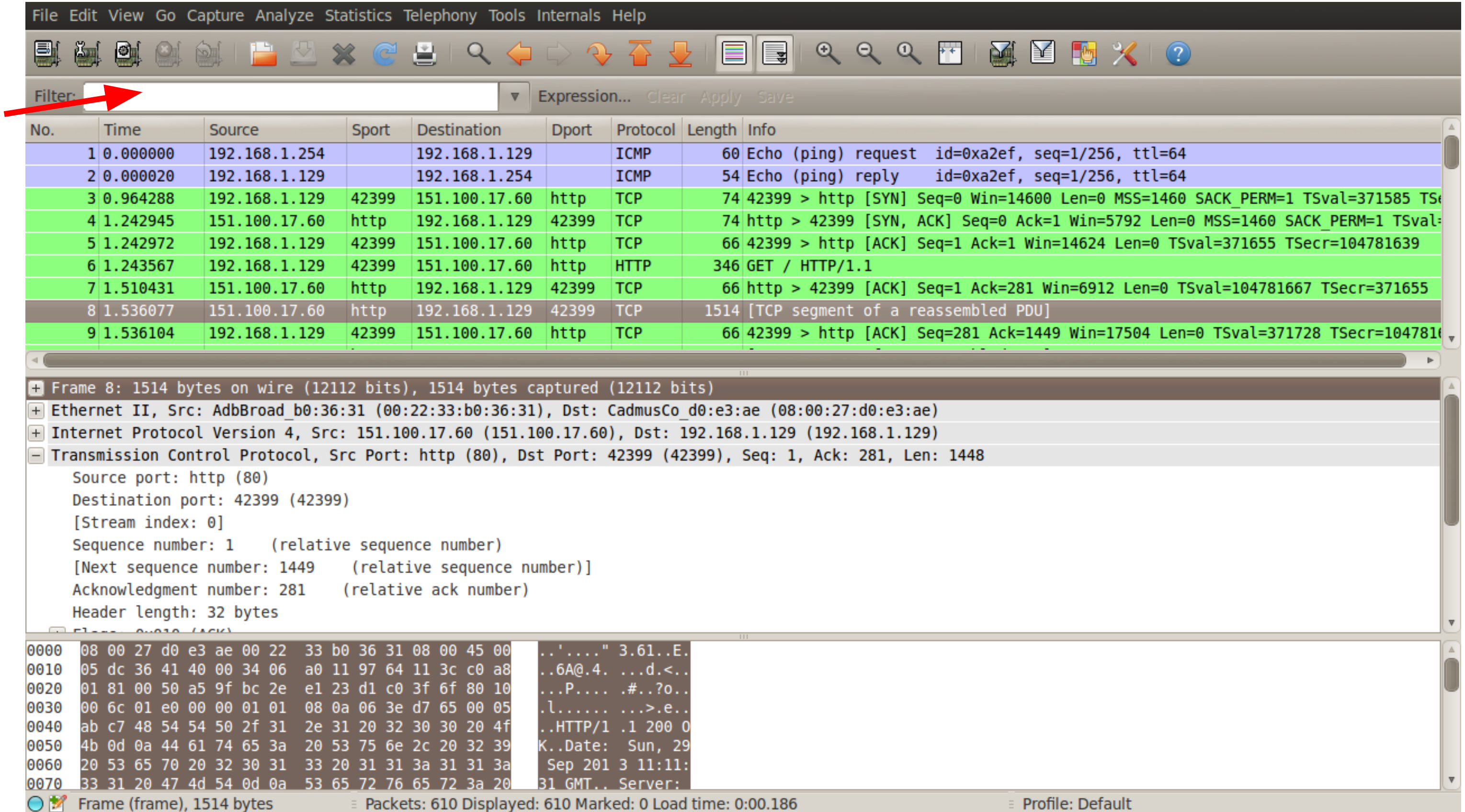
commands menu

filter specification

list of packets captured

details of selected header

packet content in hexadecimal and ASCII


File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression... Clear Apply Save

No.	Time	Source	Sport	Destination	Dport	Protocol	Length	Info
1	0.000000	192.168.1.254		192.168.1.129		ICMP	60	Echo (ping) request id=0xa2ef, seq=1/256, ttl=64
2	0.000020	192.168.1.129		192.168.1.254		ICMP	54	Echo (ping) reply id=0xa2ef, seq=1/256, ttl=64
3	0.964288	192.168.1.129	42399	151.100.17.60	http	TCP	74	42399 > http [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 TSval=371585 TSe
4	1.242945	151.100.17.60	http	192.168.1.129	42399	TCP	74	http > 42399 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 SACK_PERM=1 TSval=
5	1.242972	192.168.1.129	42399	151.100.17.60	http	TCP	66	42399 > http [ACK] Seq=1 Ack=1 Win=14624 Len=0 TSval=371655 TSecr=104781639
6	1.243567	192.168.1.129	42399	151.100.17.60	http	HTTP	346	GET / HTTP/1.1
7	1.510431	151.100.17.60	http	192.168.1.129	42399	TCP	66	http > 42399 [ACK] Seq=1 Ack=281 Win=6912 Len=0 TSval=104781667 TSecr=371655
8	1.536077	151.100.17.60	http	192.168.1.129	42399	TCP	1514	[TCP segment of a reassembled PDU]
9	1.536104	192.168.1.129	42399	151.100.17.60	http	TCP	66	42399 > http [ACK] Seq=281 Ack=1449 Win=17504 Len=0 TSval=371728 TSecr=104781(

```
⊞ Frame 8: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits)
⊞ Ethernet II, Src: AdbBroad_b0:36:31 (00:22:33:b0:36:31), Dst: CadmusCo_d0:e3:ae (08:00:27:d0:e3:ae)
⊞ Internet Protocol Version 4, Src: 151.100.17.60 (151.100.17.60), Dst: 192.168.1.129 (192.168.1.129)
⊟ Transmission Control Protocol, Src Port: http (80), Dst Port: 42399 (42399), Seq: 1, Ack: 281, Len: 1448
    Source port: http (80)
    Destination port: 42399 (42399)
    [Stream index: 0]
    Sequence number: 1    (relative sequence number)
    [Next sequence number: 1449    (relative sequence number)]
    Acknowledgment number: 281    (relative ack number)
    Header length: 32 bytes
  ⊞ Flags: 0x010 (ACK)
```

```
0000  08 00 27 d0 e3 ae 00 22  33 b0 36 31 08 00 45 00   ..'....." 3.61..E.
0010  05 dc 36 41 40 00 34 06  a0 11 97 64 11 3c c0 a8   ..6A@.4. ...d.<..
0020  01 81 00 50 a5 9f bc 2e  e1 23 d1 c0 3f 6f 80 10   ...P.... .#..?o..
0030  00 6c 01 e0 00 00 01 01  08 0a 06 3e d7 65 00 05   .l...... ...>.e..
0040  ab c7 48 54 54 50 2f 31  2e 31 20 32 30 30 20 4f   ..HTTP/1 .1 200 O
0050  4b 0d 0a 44 61 74 65 3a  20 53 75 6e 2c 20 32 39   K..Date:  Sun, 29
0060  20 53 65 70 20 32 30 31  33 20 31 31 3a 31 31 3a    Sep 201 3 11:11:
0070  33 31 20 47 4d 54 0d 0a  53 65 72 76 65 72 3a 20   31 GMT.. Server:
```

Frame (frame), 1514 bytes   Packets: 610 Displayed: 610 Marked: 0 Load time: 0:00.186   Profile: Default

wireshark

Exercise N.1 (simple):

1. Download and open the following **capture file** using Wireshark http://wiki.wireshark.org/SampleCaptures?action=AttachFile&do=get&target=http.cap
2. Apply the filter:
   - `ip.addr == 65.208.228.223`
   - remember to hit ‘enter’ to see the effects of the filter
3. Observe the list of exchanged packets
   - what are the HTTP connection endpoints?
4. Select an HTTP packet, then:
   - Analyze → Follow TCP Stream
   - How many TCP connections have been opened?
5. Notice: no DNS packets! (they have been probably removed by the author of the capture file

wireshark

Exercise N.2 (more tricky):

1. Use Wireshark to start a capture session on the pseudo device that captures on all the interfaces
2. Open your favourite browser, clean its cache (it may not be necessary), and go to:
   - http://gaia.cs.umass.edu/wireshark-labs/
3. Wait for the page to finish loading, go back to Wireshark and stop the capture session
4. Apply the filter
   - **`dns`**
5. Search for the DNS query relative to **`gaia.cs.umass.edu`** and look for the resolved IP address on the packet's payload (hint, it's probably: 128.119.245.12)
6. Apply the filter
   - **`ip.addr == 128.119.245.12`**
7. Now analyze the HTTP flow like we did in Exercise N.1
   - is HTTP's 'keep-alive' used?

`wireshark`

Further exercises. Use Wireshark to analyze the traffic generated when:

1. A web page with text and pictures is downloaded
2. A DNS request is performed with **`nslookup`**
3. A file gets downloaded through FTP