Chapter 4
WLAN labs

Marco Mellia

Lab requirements

- Your laptop with WiFi card, your mobile phone, an open and/or a WPA-personal AP
- Linux OS
  - Kali Linux or any other distro, with installed the tools below
- MacOS
  - It may work also with the standard Apple Airport interface (WiFi)
  - Use homebrew to install tools - https://brew.sh
- Windows
  - You can try but don't ask

- Tools:
  - iperf and/or iperf3 - https://sourceforge.net/projects/iperf2/ https://iperf.fr
  - Wireshark to capture packets and filter packets - https://www.wireshark.org
    - tcpdump to force wifi in monitor mode - https://www.tcpdump.org – if WiFi driver is picky
  - aircrack-ng – to capture and inject packets in the network - https://aircrack-ng.org
    - Check http://aircrack-ng.org/doku.php?id=aireplay-ng for details
  - hashcat – to crack passwords - https://hashcat.net/hashcat/
  - zizzania - https://github.com/cyrus-and/zizzania [may not work]
  - Plenty of tools/tutorial

Part 1 – WiFi performance test

Part 1 – WiFi performance test

- Setup a simple testbed where you have two hosts:
  - Both hosts connected via Ethernet
  - One host connected via Ethernet, one connected via WiFi
  - Both hosts connected via WiFi

- Check the configuration of the hosts
  - Ethernet speed
    - Linux: `ethtool eth0`
    - MacOS: `ifconfig -v -n en1` `# -v => verbose mode; -m => list supported media`
    - Windows: `Get-NetAdapter` [from an admin powershell]
  - WiFi speed
    - Linux: `iw dev wlan0 link` or `iwconfig wlan0`
    - MacOS: `sudo wdutil info` or option + click on WiFi icon
    - Windows: `netsh wlan show interfaces` [from an admin powershell]

Expected Goodput

- Goodput:

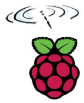
  G = Useful data at the Application Layer / Time to complete the transfer

- Maximum efficiency
  - Using TCP, on Ethernet μ_{TCP} = (1460)/(1460+20+20)+38 = 0.949
  - Using UDP, on Ethernet μ_{udp} = (1472)/(1472+8+20)+38 = 0.957
  - Using TCP, on Half Duplex Ethernet link $\mu_{TCP\text{-}HD}$ = (1460)/[(1460+20+20+38)+(20+20+6+38] = 0.900
    - Question: what about TCP options like Timestamp, SACK, MSS, Window Scaling?
    - Question: what if you use IPv6?
  - On WiFi, depends on the configuration. Typically
    - $\mu_{TCP} < 0.5$
    - $\mu_{udp} < 0.55$

- Given the capacity C of the *bottleneck* link, then

$$G < \mu C$$

How to measure the goodput with iperf

- Run a server 

- Run a client 

`iperf -s -i 1` or `iperf3 -s`

`iperf -c <Server_IP> -i 1` or
`iperf3 -c <Server_IP>`

`-u`: perform a UDP test

`-R`: receive instead of sending

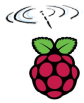
`-l`: length of the application buffer (1472 for UDP)

`-b`: set target bitrate to n bits/sec (default 1 Mbit/sec for UDP)

Prepare a script to repeat the test 10 times and extract min, max, avg, std from the results

TCP Example

- Run a server



- Run a client

`iperf -s -i 1` or `iperf3 -s`

`iperf -c <Server_IP> -i 1` or
`iperf3 -c <Server_IP>`

```
pi@raspberrypi: ~                                                    ⌥⌘1
pi@raspberrypi:~ $ iperf -s -i 1
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  4] local 192.168.1.155 port 5001 connected with 192.168.1.112 port 57220
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0- 1.0 sec  2.78 MBytes  23.3 Mbits/sec
[  4]  1.0- 2.0 sec  3.60 MBytes  30.2 Mbits/sec
[  4]  2.0- 3.0 sec  4.07 MBytes  34.2 Mbits/sec
[  4]  3.0- 4.0 sec  4.52 MBytes  37.9 Mbits/sec
[  4]  4.0- 5.0 sec  2.85 MBytes  23.9 Mbits/sec
[  4]  5.0- 6.0 sec  3.77 MBytes  31.6 Mbits/sec
[  4]  6.0- 7.0 sec  2.92 MBytes  24.5 Mbits/sec
[  4]  7.0- 8.0 sec  2.75 MBytes  23.0 Mbits/sec
[  4]  8.0- 9.0 sec  3.75 MBytes  31.5 Mbits/sec
[  4]  9.0-10.0 sec  3.09 MBytes  26.0 Mbits/sec
[  4]  0.0-10.3 sec  35.0 MBytes  28.5 Mbits/sec
```

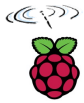
-u: pe
-R: re
-l: len
UDP)
-b: se
Mbit/

ts

```
-zsh                                                                 ⌥⌘2
------------------------------------------------------------
Client connecting to raspberrypi.local, TCP port 5001
TCP window size:  128 KByte (default)
------------------------------------------------------------
[  1] local 192.168.1.112 port 57220 connected with 192.168.1.155 port 5001
 (icwnd/mss/irtt=11/1448/13000)
[ ID] Interval       Transfer     Bandwidth
[  1] 0.00-1.00 sec  4.63 MBytes  38.8 Mbits/sec
[  1] 1.00-2.00 sec  3.62 MBytes  30.4 Mbits/sec
[  1] 2.00-3.00 sec  4.12 MBytes  34.6 Mbits/sec
[  1] 3.00-4.00 sec  4.50 MBytes  37.7 Mbits/sec
[  1] 4.00-5.00 sec  1.62 MBytes  13.6 Mbits/sec
[  1] 5.00-6.00 sec  5.00 MBytes  41.9 Mbits/sec
[  1] 6.00-7.00 sec  2.88 MBytes  24.1 Mbits/sec
[  1] 7.00-8.00 sec  1.25 MBytes  10.5 Mbits/sec
[  1] 8.00-9.00 sec  5.25 MBytes  44.0 Mbits/sec
[  1] 9.00-10.00 sec  2.00 MBytes  16.8 Mbits/sec
[  1] 10.00-10.34 sec   128 KBytes  3.07 Mbits/sec
[  1] 0.00-10.34 sec  35.0 MBytes  28.4 Mbits/sec
(base) marcomellia@MacBook1diMarco ~ %
```

TCP Example

- Run a server



- Run a client

`iperf -s -i 1` or `iperf3 -s`

`iperf -c <Server_IP> -i 1` or
`iperf3 -c <Server_IP>`

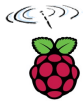
```
pi@raspberrypi: ~                                                    ⌥⌘1
pi@raspberrypi:~ $ iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
Accepted connection from 192.168.1.112, port 57155
[  5] local 192.168.1.155 port 5201 connected to 192.168.1.112 port 57156
[ ID] Interval           Transfer     Bandwidth
[  5]   0.00-1.00   sec  2.57 MBytes  21.6 Mbits/sec
[  5]   1.00-2.00   sec  2.46 MBytes  20.6 Mbits/sec
[  5]   2.00-3.00   sec  3.19 MBytes  26.8 Mbits/sec
[  5]   3.00-4.00   sec  2.22 MBytes  18.6 Mbits/sec
[  5]   4.00-5.00   sec  2.10 MBytes  17.6 Mbits/sec
[  5]   5.00-6.00   sec  2.70 MBytes  22.6 Mbits/sec
[  5]   6.00-7.00   sec  2.88 MBytes  24.2 Mbits/sec
[  5]   7.00-8.00   sec  2.53 MBytes  21.2 Mbits/sec
[  5]   8.00-9.00   sec  2.56 MBytes  21.5 Mbits/sec
[  5]   9.00-10.00  sec  3.74 MBytes  31.4 Mbits/sec
[  5]  10.00-10.42  sec  1.52 MBytes  30.1 Mbits/sec
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth
[  5]   0.00-10.42  sec  0.00 Bytes  0.00 bits/sec                  sender
[  5]   0.00-10.42  sec  28.5 MBytes  22.9 Mbits/sec                receiver
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

```
-zsh                                                                 ⌥⌘2
Last login: Sun Apr 14 08:50:09 on ttys000
(base) marcomellia@MacBook1diMarco ~ % iperf3 -c raspberrypi.local
Connecting to host raspberrypi.local, port 5201
[  8] local 192.168.1.112 port 57156 connected to 192.168.1.155 port 5201
[ ID] Interval           Transfer     Bitrate
[  8]   0.00-1.01   sec  4.62 MBytes  38.6 Mbits/sec
[  8]   1.01-2.01   sec  1.00 MBytes  8.33 Mbits/sec
[  8]   2.01-3.01   sec  4.25 MBytes  35.9 Mbits/sec
[  8]   3.01-4.00   sec  2.50 MBytes  21.1 Mbits/sec
[  8]   4.00-5.01   sec  2.12 MBytes  17.7 Mbits/sec
[  8]   5.01-6.01   sec  2.75 MBytes  23.1 Mbits/sec
[  8]   6.01-7.00   sec  2.62 MBytes  22.1 Mbits/sec
[  8]   7.00-8.02   sec  2.75 MBytes  22.7 Mbits/sec
[  8]   8.02-9.01   sec  1.62 MBytes  13.8 Mbits/sec
[  8]   9.01-10.01  sec  4.62 MBytes  38.5 Mbits/sec
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate
[  8]   0.00-10.01  sec  28.9 MBytes  24.2 Mbits/sec                  sender
[  8]   0.00-10.01  sec  28.5 MBytes  23.8 Mbits/sec                  receiver

iperf Done.
(base) marcomellia@MacBook1diMarco ~ %
```

Questions

- Why are the results different?
- Why did the goodput change during the 10s of the experiment?
- Why are the receiver and the sender measurements different? Which is the correct goodput estimation?
- What changes if we invert the sender and the receiver role?

UDP Example

- Run a server 

- Run a client 

`iperf -s -i 1` or `iperf3 -s`

`iperf -c <Server_IP> -i 1` or `iperf3 -c <Server_IP>`

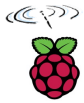
pi@raspberrypi: ~ ⌥⌘1

```
pi@raspberrypi:~ $ iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
Accepted connection from 192.168.1.112, port 57247
[  5] local 192.168.1.155 port 5201 connected to 192.168.1.112 port 61831
[ ID] Interval           Transfer     Bandwidth       Jitter    Lost/Total Datagrams
[  5]   0.00-1.00   sec   127 KBytes  1.04 Mbits/sec  342831107.035 ms  0/90 (0%)
[  5]   1.00-2.00   sec   127 KBytes  1.04 Mbits/sec  1029190.311 ms  0/90 (0%)
[  5]   2.00-3.00   sec   126 KBytes  1.03 Mbits/sec  3300.163 ms  0/89 (0%)
[  5]   3.00-4.00   sec   130 KBytes  1.07 Mbits/sec  10.825 ms  0/92 (0%)
[  5]   4.00-5.00   sec   129 KBytes  1.05 Mbits/sec  3.739 ms  0/91 (0%)
[  5]   5.00-6.00   sec   127 KBytes  1.04 Mbits/sec  2.476 ms  0/90 (0%)
[  5]   6.00-7.00   sec   129 KBytes  1.05 Mbits/sec  2.342 ms  0/91 (0%)
[  5]   7.00-8.00   sec   127 KBytes  1.04 Mbits/sec  4.157 ms  0/90 (0%)
[  5]   8.00-9.00   sec   129 KBytes  1.05 Mbits/sec  1.563 ms  0/91 (0%)
[  5]   9.00-10.00  sec   127 KBytes  1.04 Mbits/sec  2.131 ms  0/90 (0%)
[  5]  10.00-10.01  sec  2.83 KBytes  1.64 Mbits/sec  1.933 ms  0/2 (0%)
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Jitter    Lost/Total Datagrams
[  5]   0.00-10.01  sec  0.00 Bytes  0.00 bits/sec  1.933 ms  0/906 (0%)
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

-zsh ⌥⌘2

```
(base) marcomellia@MacBook1diMarco ~ % iperf3 -c raspberrypi.local -i 1 -u
Connecting to host raspberrypi.local, port 5201
[  8] local 192.168.1.112 port 61831 connected to 192.168.1.155 port 5201
[ ID] Interval           Transfer     Bitrate         Total Datagrams
[  8]   0.00-1.00   sec   129 KBytes  1.05 Mbits/sec  91
[  8]   1.00-2.00   sec   127 KBytes  1.04 Mbits/sec  90
[  8]   2.00-3.00   sec   129 KBytes  1.05 Mbits/sec  91
[  8]   3.00-4.00   sec   129 KBytes  1.05 Mbits/sec  91
[  8]   4.00-5.00   sec   127 KBytes  1.04 Mbits/sec  90
[  8]   5.00-6.00   sec   129 KBytes  1.05 Mbits/sec  91
[  8]   6.00-7.00   sec   127 KBytes  1.04 Mbits/sec  90
[  8]   7.00-8.00   sec   129 KBytes  1.05 Mbits/sec  91
[  8]   8.00-9.00   sec   127 KBytes  1.04 Mbits/sec  90
[  8]   9.00-10.00  sec   129 KBytes  1.05 Mbits/sec  91
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Jitter    Lost/Total Datagrams
[  8]   0.00-10.00  sec  1.25 MBytes  1.05 Mbits/sec  0.000 ms  0/906 (0%)  sender
[  8]   0.00-10.00  sec  1.25 MBytes  1.05 Mbits/sec  1.933 ms  0/906 (0%)  receiver

iperf Done.
(base) marcomellia@MacBook1diMarco ~ %
```

UDP Example

- Run a server 

- Run a client 

`iperf -s -i 1` or `iperf3 -s`

`iperf -c <Server_IP> -i 1` or `iperf3 -c <Server_IP>`

pi@raspberrypi: ~ ⌥⌘1

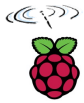
```
------------------------------------------------------------
Accepted connection from 192.168.1.112, port 57249
[  5] local 192.168.1.155 port 5201 connected to 192.168.1.112 port 49192
[ ID] Interval           Transfer     Bandwidth       Jitter    Lost/Total Datagrams
[  5]   0.00-1.00   sec  4.28 MBytes  35.9 Mbits/sec  0.421 ms  0/3096 (0%)
[  5]   1.00-2.00   sec  4.36 MBytes  36.6 Mbits/sec  0.477 ms  0/3157 (0%)
[  5]   2.00-3.00   sec  3.66 MBytes  30.7 Mbits/sec  0.409 ms  0/2654 (0%)
[  5]   3.00-4.00   sec  3.74 MBytes  31.4 Mbits/sec  0.500 ms  0/2707 (0%)
[  5]   4.00-5.00   sec  4.79 MBytes  40.2 Mbits/sec  0.383 ms  0/3469 (0%)
[  5]   5.00-6.00   sec  4.17 MBytes  35.0 Mbits/sec  0.445 ms  0/3022 (0%)
[  5]   6.00-7.00   sec  4.17 MBytes  35.0 Mbits/sec  0.442 ms  0/3018 (0%)
[  5]   7.00-8.00   sec  4.46 MBytes  37.4 Mbits/sec  0.472 ms  0/3232 (0%)
[  5]   8.00-9.00   sec  4.73 MBytes  39.7 Mbits/sec  0.459 ms  0/3423 (0%)
[  5]   9.00-10.00  sec  4.19 MBytes  35.1 Mbits/sec  0.345 ms  0/3034 (0%)
[  5]  10.00-11.00  sec  4.32 MBytes  36.3 Mbits/sec  0.816 ms  0/3131 (0%)
[  5]  11.00-12.00  sec  4.01 MBytes  33.6 Mbits/sec  0.406 ms  0/2902 (0%)
[  5]  12.00-13.00  sec  3.95 MBytes  33.1 Mbits/sec  0.441 ms  0/2858 (0%)
[  5]  13.00-13.80  sec  3.38 MBytes  35.5 Mbits/sec  0.411 ms  1005/3453 (29%)
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Jitter    Lost/Total Datagrams
[  5]   0.00-13.80  sec  0.00 Bytes  0.00 bits/sec  0.411 ms  1005/43156 (2.3%)
------------------------------------------------------------
Server listening on 5201
------------------------------------------------------------
```

-zsh ⌥⌘2

```
[  8]   0.00-10.00  sec  1.25 MBytes  1.05 Mbits/sec  0.000 ms  0/906 (0%)  sender
[  8]   0.00-10.00  sec  1.25 MBytes  1.05 Mbits/sec  1.933 ms  0/906 (0%)  receiver

iperf Done.
(base) marcomellia@MacBook1diMarco ~ % iperf3 -c raspberrypi.local -i 1 -u -b 50M
Connecting to host raspberrypi.local, port 5201
[  8] local 192.168.1.112 port 49192 connected to 192.168.1.155 port 5201
[ ID] Interval           Transfer     Bitrate         Total Datagrams
[  8]   0.00-1.00   sec  5.96 MBytes  50.0 Mbits/sec  4313
[  8]   1.00-2.00   sec  5.96 MBytes  50.0 Mbits/sec  4317
[  8]   2.00-3.00   sec  5.96 MBytes  50.0 Mbits/sec  4316
[  8]   3.00-4.00   sec  5.96 MBytes  50.0 Mbits/sec  4316
[  8]   4.00-5.00   sec  5.96 MBytes  50.0 Mbits/sec  4317
[  8]   5.00-6.00   sec  5.96 MBytes  50.0 Mbits/sec  4316
[  8]   6.00-7.00   sec  5.96 MBytes  50.0 Mbits/sec  4316
[  8]   7.00-8.00   sec  5.96 MBytes  50.0 Mbits/sec  4317
[  8]   8.00-9.00   sec  5.96 MBytes  50.0 Mbits/sec  4316
[  8]   9.00-10.00  sec  5.96 MBytes  50.0 Mbits/sec  4316
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Jitter    Lost/Total Datagrams
[  8]   0.00-10.00  sec  59.6 MBytes  50.0 Mbits/sec  0.000 ms  0/43160 (0%)  sender
[  8]   0.00-10.00  sec  58.2 MBytes  48.8 Mbits/sec  0.411 ms  0/43156 (0%)  receiver

iperf Done.
(base) marcomellia@MacBook1diMarco ~ %
```

UDP Example

- Run a server



- Run a client

`iperf -s -i 1` or `iperf3 -s`

`iperf -c <Se`

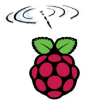
`iperf3 -c <Se`

```
pi@raspberrypi: ~
⌥⌘1
------------------------------------------------------------
Accepted connection from 192.168.1.112, port 57249
[  5] local 192.168.1.155 port 5201 connected to 192.168.1.112 port 49192
[ ID] Interval           Transfer     Bandwidth       Jitter    Lost/Total Datagrams
[  5]   0.00-1.00   sec  4.28 MBytes  35.9 Mbits/sec  0.421 ms  0/3096 (0%)
[  5]   1.00-2.00   sec  4.36 MBytes  36.6 Mbits/sec  0.477 ms  0/3157 (0%)
[  5]   2.00-3.00   sec  3.66 MBytes  30.7 Mbits/sec  0.409 ms  0/2654 (0%)
[  5]   3.00-4.00   sec  3.74 MBytes  31.4 Mbits/sec  0.500 ms  0/2707 (0%)
[  5]   4.00-5.00   sec  4.79 MBytes  40.2 Mbits/sec  0.383 ms  0/3469 (0%)
[  5]   5.00-6.00   sec  4.17 MBytes  35.0 Mbits/sec  0.445 ms  0/3022 (0%)
[  5]   6.00-7.00   sec  4.17 MBytes  35.0 Mbits/sec  0.442 ms  0/3018 (0%)
[  5]   7.00-8.00   sec  4.46 MBytes  37.4 Mbits/sec  0.472 ms  0/3232 (0%)
[  5]   8.00-9.00   sec  4.73 MBytes  39.7 Mbits/sec  0.459 ms  0/3423 (0%)
[  5]   9.00-10.00  sec  4.19 MBytes  35.1 Mbits/sec  0.345 ms  0/3034 (0%)
[  5]  10.00-11.00  sec  4.32 MBytes  36.3 Mbits/sec  0.816 ms  0/3131 (0%)
[  5]  11.00-12.00  sec  4.01 MBytes  33.6 Mbits/sec  0.406 ms  0/2902 (0%)
[  5]  12.00-13.00  sec  3.95 MBytes  33.1 Mbits/sec  0.441 ms  0/2858 (0%)
[  5]  13.00-13.80  sec  3.38 MBytes  35.5 Mbits/sec  0.411 ms  1005/3453 (29%)
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Jitter    Lost/Total Datagrams
[  5]   0.00-13.80  sec  0.00 Bytes  0.00 bits/sec  0.411 ms  1005/43156 (2.3%)
------------------------------------------------------------
Server listening on 5201
------------------------------------------------------------
```

```
[  8]   0.00-10.00  sec
[  8]   0.00-10.00  sec

iperf Done.
(base) marcomellia@MacB
Connecting to host rasp
[  8] local 192.168.1.1
[ ID] Interval
[  8]   0.00-1.00   sec
[  8]   1.00-2.00   sec
[  8]   2.00-3.00   sec
[  8]   3.00-4.00   sec
[  8]   4.00-5.00   sec
[  8]   5.00-6.00   sec
[  8]   6.00-7.00   sec
[  8]   7.00-8.00   sec
[  8]   8.00-9.00   sec
[  8]   9.00-10.00  sec
- - - - - - - - - - - -
[ ID] Interval
[  8]   0.00-10.00  sec
[  8]   0.00-10.00  sec

iperf Done.
(base) marcomellia@MacBook1diMarco ~ %
```

```
-zsh
⌥⌘3
(base) marcomellia@MacBook1diMarco ~ % ping -i 0.5 raspberrypi
PING raspberrypi.lan (192.168.1.155): 56 data bytes
64 bytes from 192.168.1.155: icmp_seq=0 ttl=64 time=8.839 ms
64 bytes from 192.168.1.155: icmp_seq=1 ttl=64 time=23.628 ms
64 bytes from 192.168.1.155: icmp_seq=2 ttl=64 time=20.094 ms
64 bytes from 192.168.1.155: icmp_seq=3 ttl=64 time=37.050 ms
64 bytes from 192.168.1.155: icmp_seq=4 ttl=64 time=14.000 ms
64 bytes from 192.168.1.155: icmp_seq=5 ttl=64 time=19.183 ms
64 bytes from 192.168.1.155: icmp_seq=6 ttl=64 time=14.199 ms
64 bytes from 192.168.1.155: icmp_seq=7 ttl=64 time=10.905 ms
64 bytes from 192.168.1.155: icmp_seq=8 ttl=64 time=24.885 ms
64 bytes from 192.168.1.155: icmp_seq=9 ttl=64 time=83.806 ms
64 bytes from 192.168.1.155: icmp_seq=10 ttl=64 time=209.323 ms
Request timeout for icmp_seq 11
64 bytes from 192.168.1.155: icmp_seq=11 ttl=64 time=525.101 ms
64 bytes from 192.168.1.155: icmp_seq=12 ttl=64 time=570.726 ms
64 bytes from 192.168.1.155: icmp_seq=13 ttl=64 time=736.485 ms
64 bytes from 192.168.1.155: icmp_seq=14 ttl=64 time=961.899 ms
Request timeout for icmp_seq 16
64 bytes from 192.168.1.155: icmp_seq=15 ttl=64 time=1207.734 ms
64 bytes from 192.168.1.155: icmp_seq=16 ttl=64 time=1488.189 ms
Request timeout for icmp_seq 19
64 bytes from 192.168.1.155: icmp_seq=17 ttl=64 time=1744.154 ms
Request timeout for icmp_seq 21
64 bytes from 192.168.1.155: icmp_seq=18 ttl=64 time=2104.530 ms
64 bytes from 192.168.1.155: icmp_seq=19 ttl=64 time=2427.429 ms
Request timeout for icmp_seq 24
64 bytes from 192.168.1.155: icmp_seq=20 ttl=64 time=2694.272 ms
64 bytes from 192.168.1.155: icmp_seq=21 ttl=64 time=2998.922 ms
Request timeout for icmp_seq 27
64 bytes from 192.168.1.155: icmp_seq=22 ttl=64 time=3295.730 ms
64 bytes from 192.168.1.155: icmp_seq=23 ttl=64 time=3440.713 ms
Request timeout for icmp_seq 30
64 bytes from 192.168.1.155: icmp_seq=24 ttl=64 time=3529.455 ms
64 bytes from 192.168.1.155: icmp_seq=25 ttl=64 time=3969.113 ms
Request timeout for icmp_seq 33
Request timeout for icmp_seq 34
Request timeout for icmp_seq 35
64 bytes from 192.168.1.155: icmp_seq=29 ttl=64 time=3580.658 ms
64 bytes from 192.168.1.155: icmp_seq=30 ttl=64 time=3075.526 ms
64 bytes from 192.168.1.155: icmp_seq=31 ttl=64 time=2570.451 ms
64 bytes from 192.168.1.155: icmp_seq=32 ttl=64 time=2070.287 ms
64 bytes from 192.168.1.155: icmp_seq=33 ttl=64 time=1569.305 ms
64 bytes from 192.168.1.155: icmp_seq=34 ttl=64 time=1064.225 ms
64 bytes from 192.168.1.155: icmp_seq=35 ttl=64 time=562.946 ms
64 bytes from 192.168.1.155: icmp_seq=36 ttl=64 time=58.636 ms
64 bytes from 192.168.1.155: icmp_seq=37 ttl=64 time=21.459 ms
64 bytes from 192.168.1.155: icmp_seq=38 ttl=64 time=15.991 ms
64 bytes from 192.168.1.155: icmp_seq=39 ttl=64 time=14.666 ms
64 bytes from 192.168.1.155: icmp_seq=40 ttl=64 time=9.836 ms
^C
--- raspberrypi.lan ping statistics ---
41 packets transmitted, 38 packets received, 7.3% packet loss
round-trip min/avg/max/stddev = 8.839/1230.904/3969.113/1316.293 ms
(base) marcomellia@MacBook1diMarco ~ %
```

UDP Example

- Run a server 

- Run a client 

`iperf -s -i 1` or `iperf3 -s`

`iperf -c <Server_IP> -i 1` or `iperf3 -c <Server_IP>`

pi@raspberrypi: ~ ⌥⌘1

```
[  5] local 192.168.1.155 port 5201 connected to 192.168.1.112 port 60007
[ ID] Interval           Transfer     Bandwidth       Jitter    Lost/Total Datagrams
[  5]   0.00-1.00   sec  4.58 MBytes  38.4 Mbits/sec  1.139 ms  0/586 (0%)
[  5]   1.00-2.00   sec  4.78 MBytes  40.1 Mbits/sec  0.881 ms  0/612 (0%)
[  5]   2.00-3.00   sec  4.50 MBytes  37.8 Mbits/sec  0.999 ms  0/576 (0%)
[  5]   3.00-4.00   sec  4.31 MBytes  36.2 Mbits/sec  1.269 ms  0/552 (0%)
[  5]   4.00-5.00   sec  4.80 MBytes  40.2 Mbits/sec  1.406 ms  0/614 (0%)
[  5]   5.00-6.00   sec  4.43 MBytes  37.2 Mbits/sec  1.783 ms  710/1277 (56%)
[  5]   6.00-7.00   sec  3.97 MBytes  33.3 Mbits/sec  2.642 ms  1026/1534 (67%)
[  5]   7.00-8.00   sec  4.41 MBytes  37.0 Mbits/sec  1.587 ms  1023/1587 (64%)
[  5]   8.00-9.00   sec  4.05 MBytes  34.0 Mbits/sec  1.335 ms  771/1290 (60%)
[  5]   9.00-10.00  sec  3.98 MBytes  33.4 Mbits/sec  2.207 ms  1050/1559 (67%)
[  5]  10.00-11.00  sec  4.48 MBytes  37.6 Mbits/sec  2.033 ms  1052/1625 (65%)
[  5]  11.00-12.00  sec  4.38 MBytes  36.8 Mbits/sec  1.921 ms  1095/1656 (66%)
iperf3: OUT OF ORDER - incoming packet = 14146 and received packet = 0 AND SP = 14175
[  5]  12.00-13.00  sec  3.81 MBytes  32.0 Mbits/sec  1.710 ms  910/1397 (65%)
[  5]  13.00-13.14  sec  0.00 Bytes  0.00 bits/sec  1.710 ms  0/0 (0%)
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Jitter    Lost/Total Datagrams
[  5]   0.00-13.14  sec  0.00 Bytes  0.00 bits/sec  1.710 ms  7637/14865 (51%)
[SUM]  0.0-13.1 sec  1 datagrams received out-of-order
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

-zsh ⌥⌘2

```
iperf Done.
(base) marcomellia@MacBook1diMarco ~ % iperf3 -c raspberrypi.local -l 8192 -u -b 100M

warning: UDP block size 8192 exceeds TCP MSS 1448, may result in fragmentation / drops
Connecting to host raspberrypi.local, port 5201
[  8] local 192.168.1.112 port 60007 connected to 192.168.1.155 port 5201
[ ID] Interval           Transfer     Bitrate         Total Datagrams
[  8]   0.00-1.00   sec  11.9 MBytes  99.9 Mbits/sec  1525
[  8]   1.00-2.00   sec  11.9 MBytes   100 Mbits/sec  1526
[  8]   2.00-3.00   sec  11.9 MBytes   100 Mbits/sec  1526
[  8]   3.00-4.00   sec  11.9 MBytes   100 Mbits/sec  1526
[  8]   4.00-5.00   sec  11.9 MBytes   100 Mbits/sec  1526
[  8]   5.00-6.00   sec  11.9 MBytes   100 Mbits/sec  1526
[  8]   6.00-7.00   sec  11.9 MBytes  99.9 Mbits/sec  1525
[  8]   7.00-8.00   sec  11.9 MBytes   100 Mbits/sec  1526
[  8]   8.00-9.00   sec  11.9 MBytes   100 Mbits/sec  1526
[  8]   9.00-10.00  sec  11.9 MBytes   100 Mbits/sec  1526
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Jitter    Lost/Total Datagrams
[  8]   0.00-10.00  sec   119 MBytes   100 Mbits/sec  0.000 ms  0/15258 (0%)  sender
[  8]   0.00-10.00  sec  56.5 MBytes  47.4 Mbits/sec  1.710 ms  0/14865 (0%)  receiver

iperf Done.
(base) marcomellia@MacBook1diMarco ~ %
```

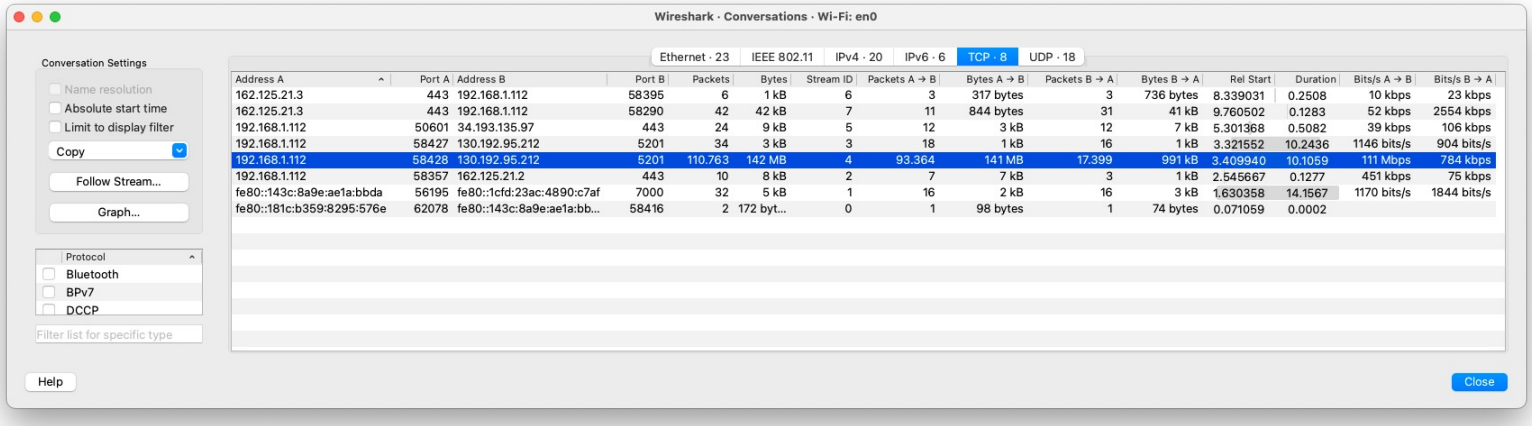
Questions

- Why are the results different?
- Why did the goodput change during the 10s of the experiment?
- Why are the receiver and the sender measurements different? Which is the correct goodput estimation?
- What changes if we invert the sender and the receiver role?
- How to properly choose the data generation rate with the `-b` parameter?

Wireshark check conversations

From the `Statistics menu -> Conversations`

- Shows four interesting columns:
  - the start time of the conversation (“Rel Start”) or (“Abs Start”)
  - the duration of the conversation in seconds
  - the average bits (not bytes) per second in each direction. This is a measure of the throughput since it includes all bytes sent/received, including all headers, signalling packets, and eventually retransmitted packets



Wireshark to check the results

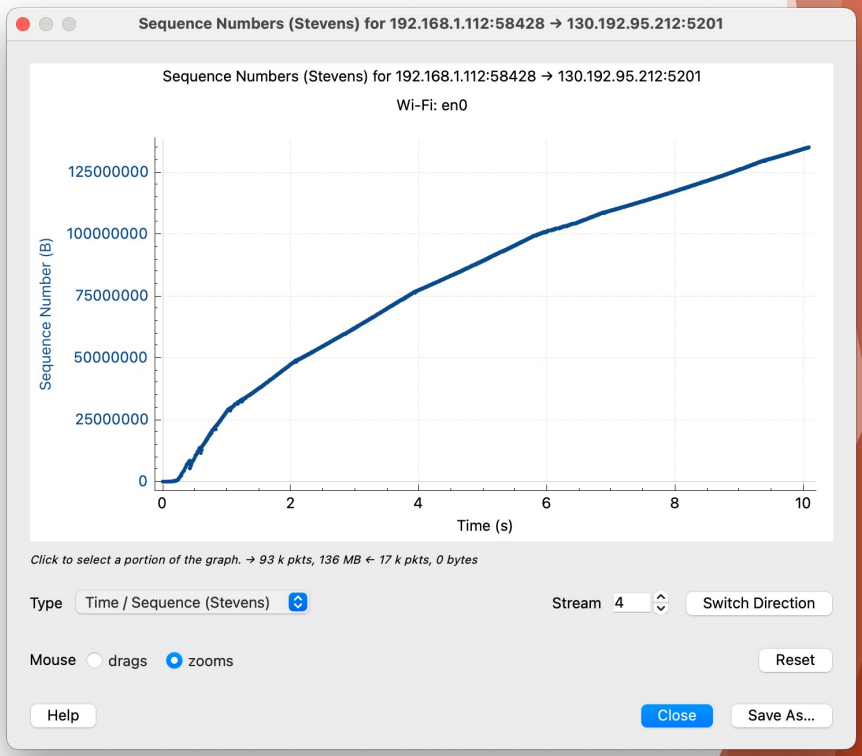
From the `Statistics -> I/O plot`

- Shows the evolution over time of the desired quantity
- Use `Filters` to select what to plot
- Add as many lines as you like
- Use `Y-Axis` to choose the unit (bytes, bits, packets)
- User `Y Axis Factor` to ease the visualization

TCP Stream graph to get

From the `Statistics -> TCP Stream Graphs`

- `Stevens TCP Graph plot`: plots the evolution of the sequence numbers over time
  - The slope is the throughput
- `Throughput plot`: plots the smoothed average throughput observed in a Moving Average Window
  - Shows also the Segment length
- `Round Trip Time`: plots the time elapsed since the observation of the TCP segment with a given sequence number and the reception of the corresponding Acknowledgement
  - It is an indication of possible congestion



Test to run

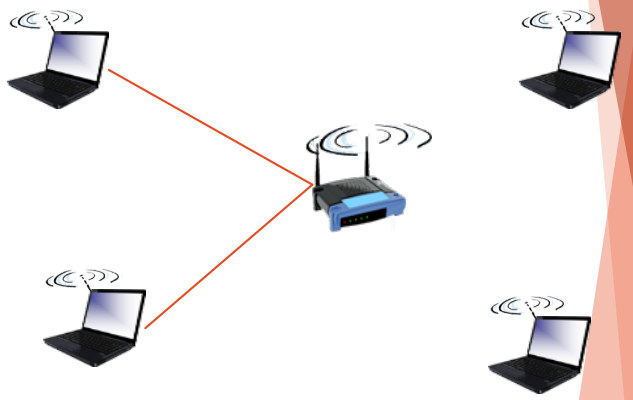
1. **Predict the goodput you expect** when TCP or UDP is used at the transport layer

2. **Perform the test**, repeating it at least 10 times. Summarize the results reporting the average, minimum, maximum and standard deviation of the observed goodput at the **receiver side**

3. Check for eventual **unexpected cases**. For instance, do you observe packet loss? Are those negligible?

Scenarios to consider

- 1 - **Both Ethernet**: client and server are connected via Ethernet
  - A. Single flow: the client sends data to the server
  - B. Single flow: the client receives data from the server
- 2 - **Both WiFi**: client is WiFi and server is WiFi
  - C. Single flow: the client sends data to the server
  - D. Single flow: the client receives data from the server
- 3 - **Mixed mode**: Server is Ethernet, client is WiFi
  - E. Single flow: the client sends data to the server
  - F. Single flow: the client receives data from the server



WARNING: Do NOT connect your device with BOTH Ethernet and WiFi at the same time! Which card will be used to transmit and receive in fact?

Results for TCP

	Test	TCP: Goodput per flow					Comment
		Prediction	Average	Min	Max	Std	
Both Ethernet	A						
	B						
Both WiFi	D						
	E						
Mixed mode	G						
	H						

Results for UDP

	Test	UDP: Goodput per flow					Comment
		Prediction	Average	Min	Max	Std	
Both Ethernet	A						
	B						
Both WiFi	D						
	E						
Mixed mode	G						
	H						

Report preparation

- Use LaTex and Overleaf with ACM CCS template
  - See course intro for some details
  - Use the “review” option: `\documentclass[sigconf, review]{acmart}`
  - Change authors, title, etc.
  - Remove the copyright part to save space:
    - `\setcopyright{none} %remove the (c) section`
    - `\acmConference{Wireless Security Report}{Torino}{2024} % set a possibly significant conference name`
    - `\acmPrice{} % leave the rest empty. These will still appear.`
    - `\acmISBN{}`
    - `\acmDOI{}`

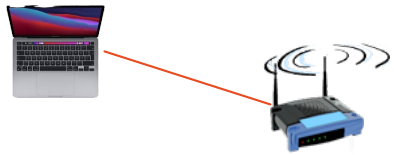
Report preparation

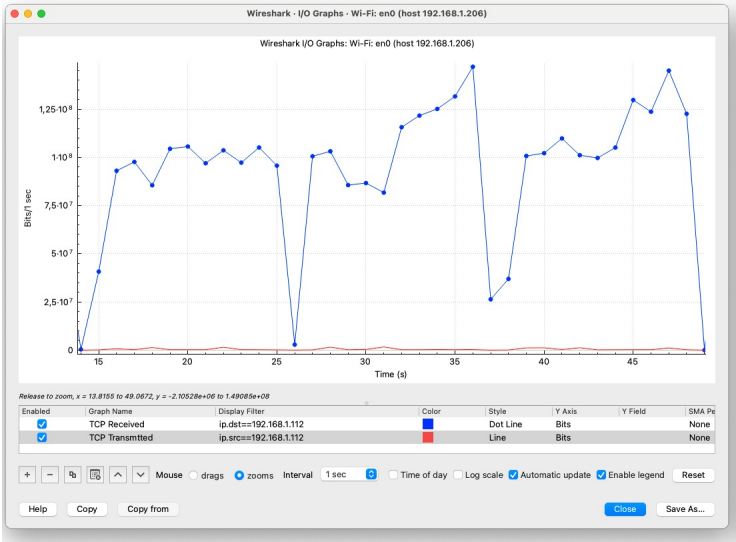
- Prepare a group report of 4 pages + appendix. Here is a possible organization
  - **Introduction**: briefly introduce the motivations and goals of the lab.
  - **Tools and theory**: summarise the expected goodput estimation and the iperf/iperf3 tool you use to run the experiments.
  - **Lab setup and scenarios**: describe the scenarios, the nominal link capacities, the IP setup (addresses, network, default gateway), and the operating system(s). You can add a figure to clarify the setup. Next, describe the scenarios that you will consider for your experiments (A,…,F). In case you cannot consider all scenarios (e.g., due to the lack of hardware), I suggest repeating the same experiments with multiple O.S. and devices or considering multiple setups (e.g., changing the AP, the devices, the tools, …).
  - **Results**: for each scenario, report the results by comparing your predictions with the actual results. You can use a table to summarize all cases. In case results are not in line with the prediction, discuss eventual sources of possible impairments that can be present (especially with WiFi, e.g., changes in the physical link rate due to changing channel conditions, interference of other devices over the same WiFi Channel, etc.). Add some plots to illustrate some specific case, e.g., comparing the performance over ethernet vs over WiFi, illustrating the “noisy” WiFi case, or those cases where the prediction is not consistent with the actual result. Consider using the Wireshark features to produce I/O graphs and TCP sequence number evolution over time.
  - **Conclusions**: briefly summarize the results
  - **Appedix**: [mandatory] insert here the scripts you created to make the experiment automatic and extract the results.

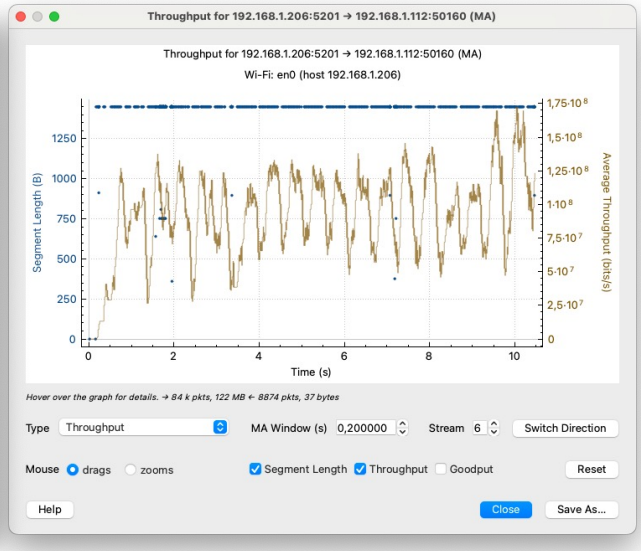
Quick and dirty test

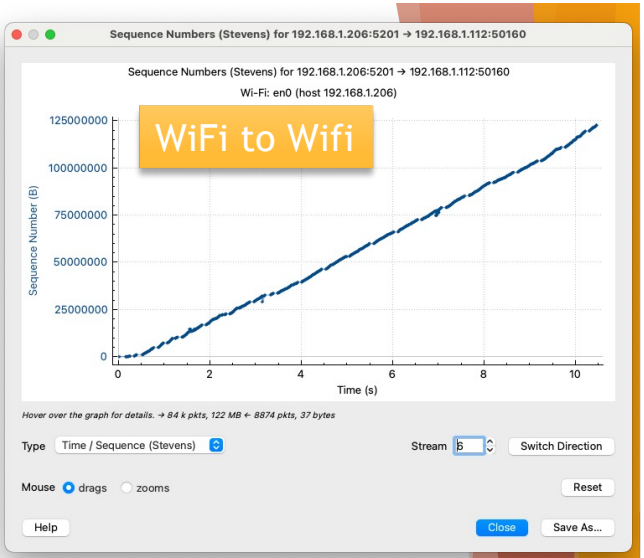
- On home WiFi 802.11ax - 5GHz, Channel 36
  - Client: MacBook @192.168.1.112 as a **WiFi** client, @192.168.1.153 as a 100Mb/s **Ethernet**
  - iPad pro @192.168.1.206 as a **Wifi** server running Hurricane Network Tools (add -s on the iperf3 panel)
  - iperf3 tool

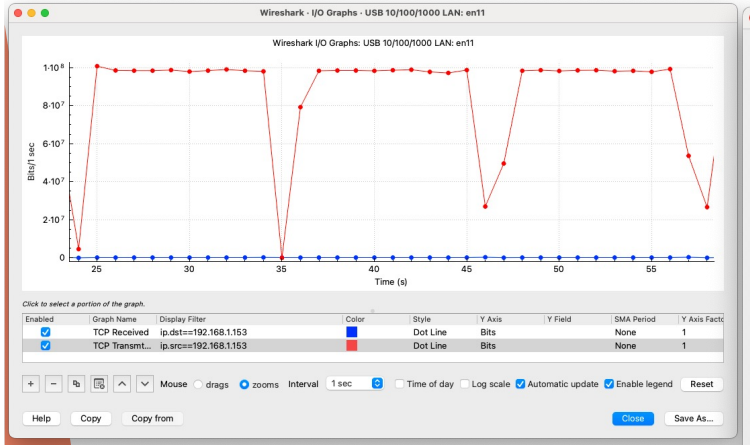
```
rm /tmp/gput.dat
for i in `seq 10`; do
     iperf3 -c 192.168.1.206 -B 192.168.1.112 -R |
          grep receiver |
          tr -s ' '|
          cut -d ' ' -f 7 >>/tmp/gput.dat;
          sleep 1;
     done
cat /tmp/gput.dat | awk '
     BEGIN{max=0;min=99999999}
          {x+=$0;y+=$0^2;if ($0<min){min=$0}; if ($0>max) {max=$0}}
     END{print "avg=", x/NR, "\nmin=", min, "\nmax=", max,"\nstd=", sqrt(y/NR-(x/NR)^2)}'
avg= 82,375
min= 97.5
max= 105.0
std= 24,8693
```

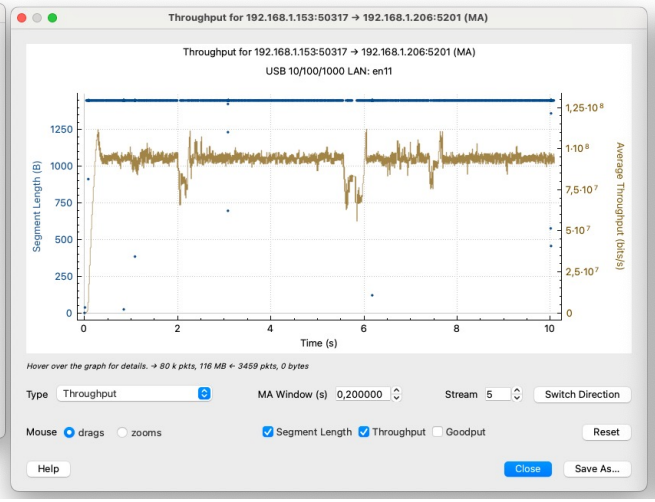


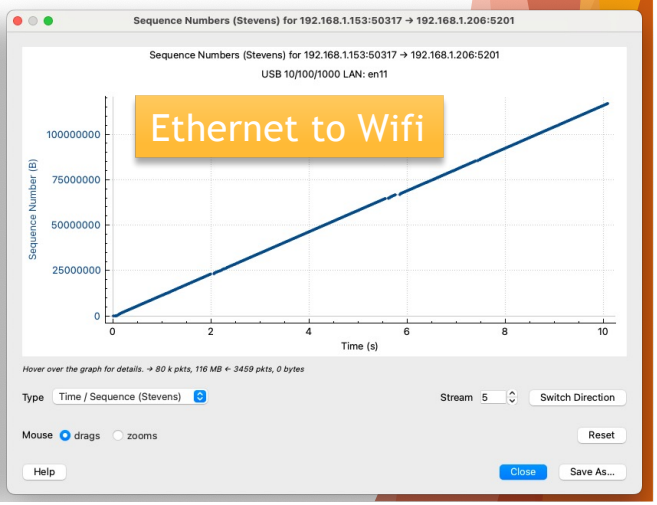

Wireshark · I/O Graphs · Wi-Fi: en0 (host 192.168.1.206)
Wireshark I/O Graphs: Wi-Fi: en0 (host 192.168.1.206)
Bits/1 sec
Time (s)
Release to zoom, x = 13.8155 to 49.0672, y = -2.10528e+06 to 1.49085e+08
Enabled
Graph Name
Display Filter
Color
Style
Y Axis
Y Field
SMA Pe
TCP Received
ip.dst==192.168.1.112
Dot Line
Bits
None
TCP Transmtted
ip.src==192.168.1.112
Line
Bits
None
Mouse
drags
zooms
Interval
1 sec
Time of day
Log scale
Automatic update
Enable legend
Reset
Help
Copy
Copy from
Close
Save As...


Throughput for 192.168.1.206:5201 → 192.168.1.112:50160 (MA)
Throughput for 192.168.1.206:5201 → 192.168.1.112:50160 (MA)
Wi-Fi: en0 (host 192.168.1.206)
Segment Length (B)
Average Throughput (bits/s)
Time (s)
Hover over the graph for details. → 84 k pkts, 122 MB ← 8874 pkts, 37 bytes
Type
Throughput
MA Window (s)
0,200000
Stream
6
Switch Direction
Mouse
drags
zooms
Segment Length
Throughput
Goodput
Reset
Help
Close
Save As...


Sequence Numbers (Stevens) for 192.168.1.206:5201 → 192.168.1.112:50160
Sequence Numbers (Stevens) for 192.168.1.206:5201 → 192.168.1.112:50160
Wi-Fi: en0 (host 192.168.1.206)
WiFi to Wifi
Sequence Number (B)
Time (s)
Hover over the graph for details. → 84 k pkts, 122 MB ← 8874 pkts, 37 bytes
Type
Time / Sequence (Stevens)
Stream
6
Switch Direction
Mouse
drags
zooms
Reset
Help
Close
Save As...

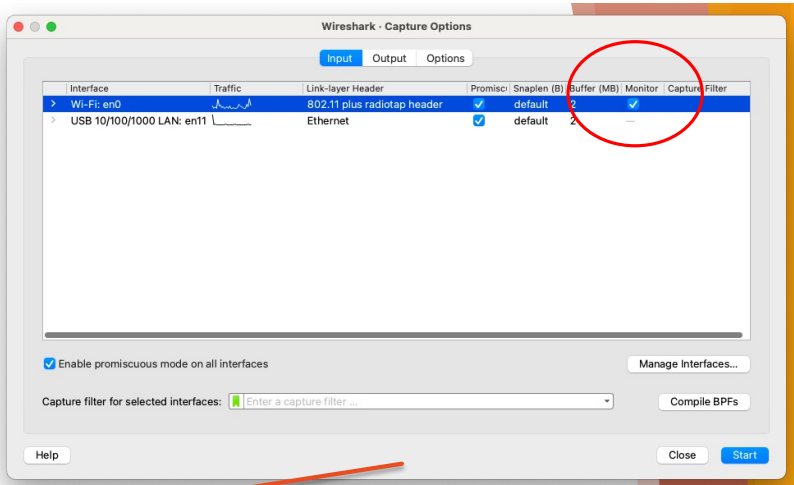

Wireshark · I/O Graphs · USB 10/100/1000 LAN: en11
Wireshark I/O Graphs: USB 10/100/1000 LAN: en11
Bits/1 sec
Time (s)
Click to select a portion of the graph.
Enabled
Graph Name
Display Filter
Color
Style
Y Axis
Y Field
SMA Period
Y Axis Facto
TCP Received
ip.dst==192.168.1.153
Dot Line
Bits
None
1
TCP Transmt...
ip.src==192.168.1.153
Dot Line
Bits
None
1
Mouse
drags
zooms
Interval
1 sec
Time of day
Log scale
Automatic update
Enable legend
Reset
Help
Copy
Copy from
Close
Save As...


Throughput for 192.168.1.153:50317 → 192.168.1.206:5201 (MA)
Throughput for 192.168.1.153:50317 → 192.168.1.206:5201 (MA)
USB 10/100/1000 LAN: en11
Segment Length (B)
Average Throughput (bits/s)
Time (s)
Hover over the graph for details. → 80 k pkts, 116 MB ← 3459 pkts, 0 bytes
Type
Throughput
MA Window (s)
0,200000
Stream
5
Switch Direction
Mouse
drags
zooms
Segment Length
Throughput
Goodput
Reset
Help
Close
Save As...


Sequence Numbers (Stevens) for 192.168.1.153:50317 → 192.168.1.206:5201
Sequence Numbers (Stevens) for 192.168.1.153:50317 → 192.168.1.206:5201
USB 10/100/1000 LAN: en11
Ethernet to Wifi
Sequence Number (B)
Time (s)
Hover over the graph for details. → 80 k pkts, 116 MB ← 3459 pkts, 0 bytes
Type
Time / Sequence (Stevens)
Stream
5
Switch Direction
Mouse
drags
zooms
Reset
Help
Close
Save As...

Part 2 – monitor wifi packets

Part 2 – monitor wifi packets



- Scan for available networks
  - `sudo iwlist wlan0 scan`
- Choose the network channel to monitor
  - `sudo iwconfig wlan0 mode monitor channel 1`
- On MacOS, use the airport utility

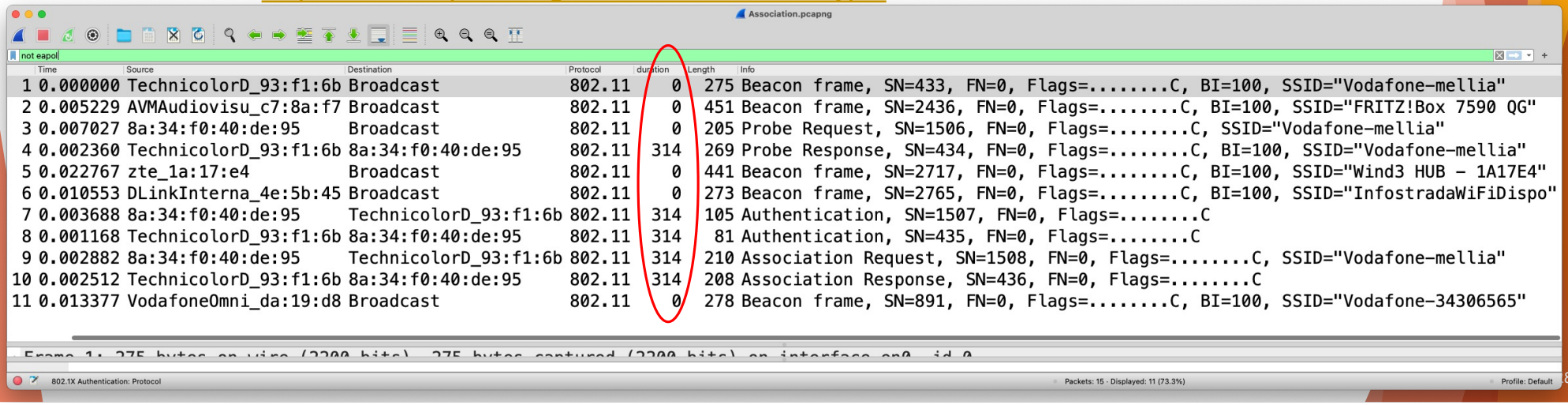
```
ln -s /System/Library/PrivateFrameworks/Apple80211.framework/Versions/Current/Resources/airport
/opt/homebrew/bin // link the airport cmd in a $PATH folder for convenience. Do this once!

sudo airport en0 -z -s //disconnect and scan for available networks
sudo airport en0 sniff 2g1/20 // sniff on channel 1 in the 2.4GHz band - 20MHz wide channel
```

- Using wireshark, monitor a wifi network
  - Put the interface in monitor mode and start capturing
    - If this does not not work use tcpdump or iwconfig to force the card in monitor mode

```
sudo tcpdump -i wlan0 -I
sudo ifconfig wlan0 down;sudo iwconfig wlan0 mode monitor; sudo ifconfig wlan0 up
```

Authentication & Association

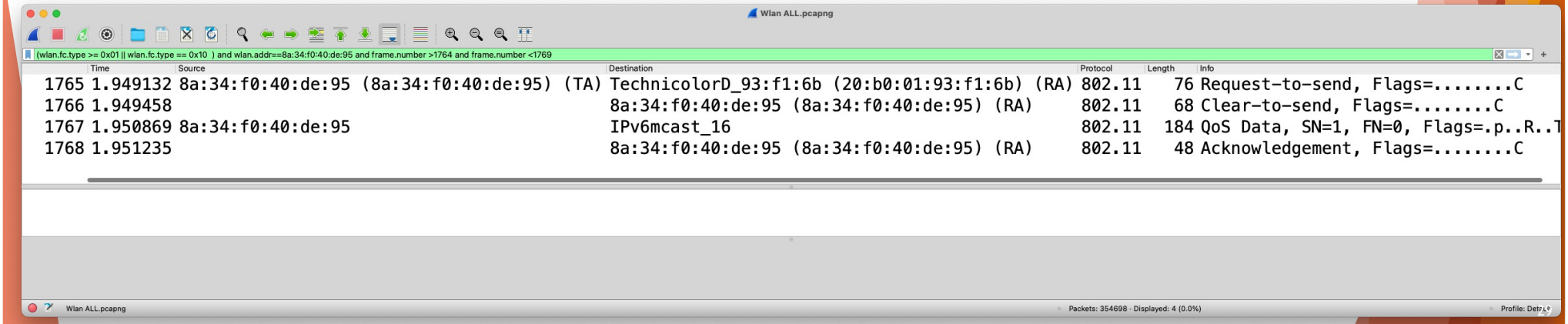
- Capture and analyse packets identifying
  - Beacon frames sent by the AP(s)
  - A complete Probe + Authentication + Association request from your mobile phone
    - hint, you can use your mobile phone to generate request by turning off/on the wifi and joining the WLAN
  - Check the duration frame
    - Add a «column» to the quick view (right-click on the column header) to show the duration
- Define filters to select only packets that are interesting for you
  - Check https://en.wikipedia.org/wiki/802.11_Frame_Types for control frame values



CSMA-CA in practice

- Check the CSMA-CD frames
  - Can you see the RTS/CTS?
  - Can you see the DATA/ACK packets?

- Define filters to select only packets that are interesting for you
  - Check https://en.wikipedia.org/wiki/802.11_Frame_Types for control frame values
- Why some frames may be missing? Why there might be a lot more RTS than CTS?



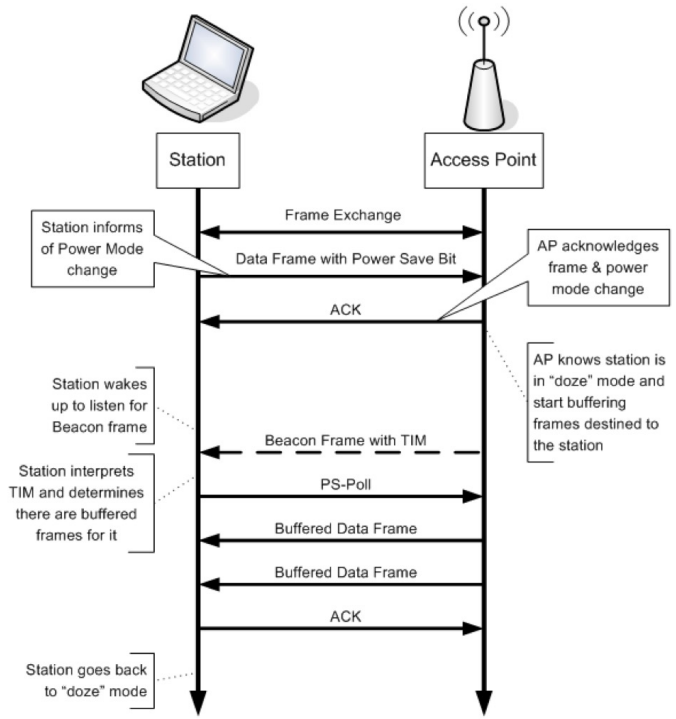
Possible statistics

- How many APs are present in the trace?
  - Hint: prepare a filter to show only beacon frames and then use the Statistics->Endpoint menu
- How many ESSID are present in the trace? Which channels do they use?
  - Hint: Check the “Wireless” menu entry
- How many STA are present in the trace?
  - Hint: prepare a filter to show the probe request and then use the Statistics->Endpoint menu

- Decrypt the packets
  - https://wiki.wireshark.org/HowToDecrypt802.11
- How many IP addresses are present?
- Pick one IP address and try to identify its OS
  - Hint: filter some DNS queries and check names
  - Hint: resolve MAC/IP address names (View->name resolution)

Power Saving in 802.11

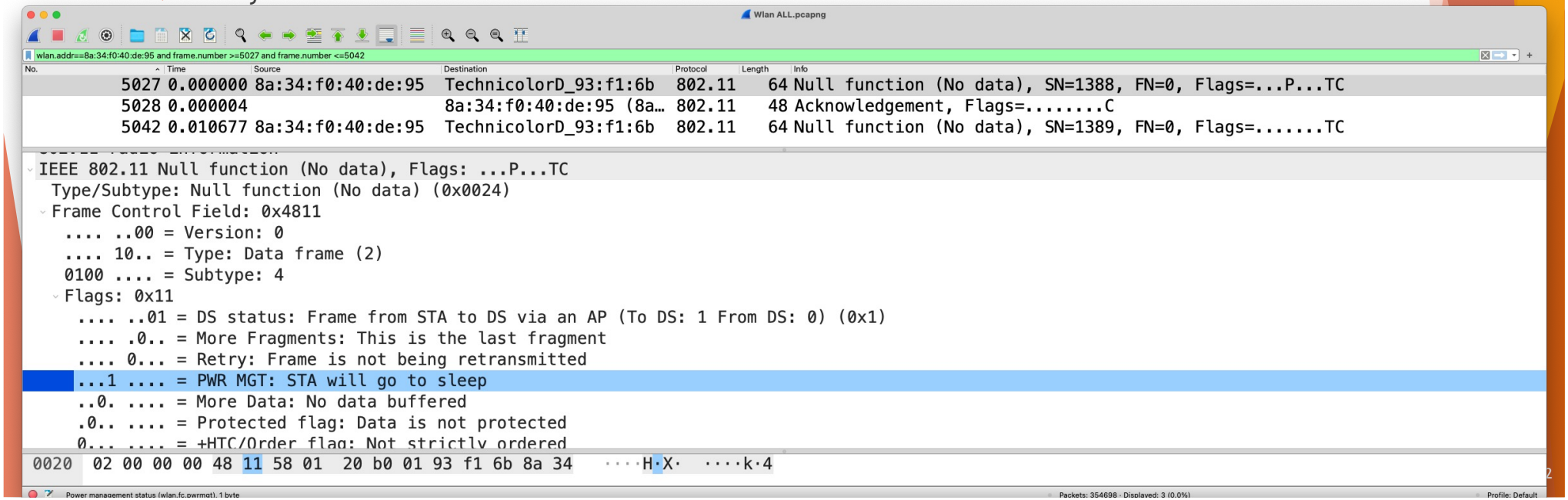
- Nodes “sleep” to conserve energy
- AP will buffer clients’ packets until requested with a poll message
- TIM (traffic indication map) is a periodic packet sent by AP to notify client of buffered data
- Relies on sync of packets so client is awake when the TIM is sent



Picture from Meiners, L. F. (2009). *But... my station is awake! Power Save Denial of Service in 802.11 Networks. Core Secur. Technol., Eden Prairie, MN, USA, Tech. Rep.*

Power management

- Filter packets that announce the STA is going to sleep
- Check when the STA is going to wake up
  - How does the STA tell the AP what is going to happen?
- From another device, send data to the STA and check if and when it goes to sleep
  - Can you see the TIM header in the beacon frames?



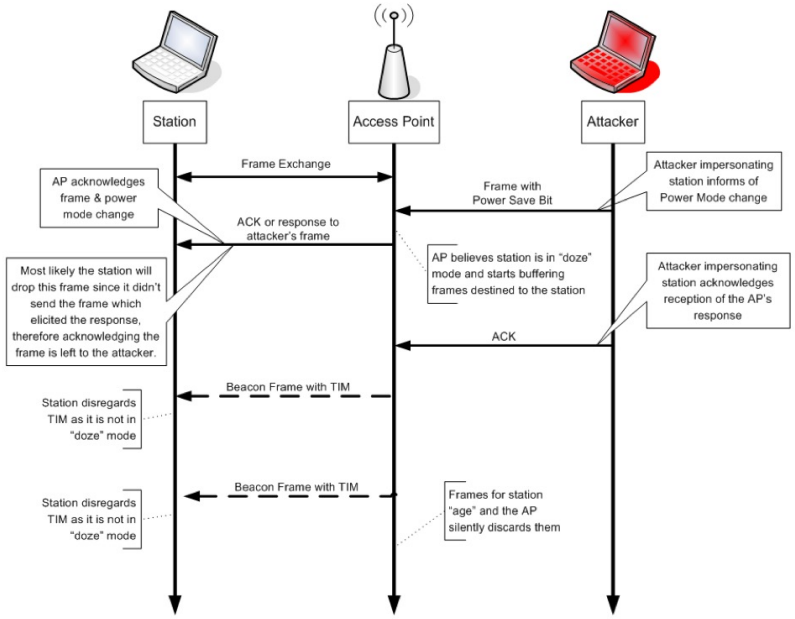
Part 2: testing some attacks

- In this second part of the lab, we will try to run some attacks:
- Power management attacks
  - Inject fake PM control frame to isolate a victim
- Deauthentication/Deassociation attack
  - Force a station offline so to observe an authentication phase
- On WP2-personal, mount a brute force attack

Part-2: power management attack

- The attacker impersonates the victim station
  - Just need to know the AP and victim MAC address
- The attacker keeps **injecting false Power Management frames** with the bit set to 1
- The AP believes the victim is in sleep mode
  - Starts buffering packets
- Try using aircrack to mount this attack

http://aircrack-ng.org/doku.php?id=interactive_packet_replay



Picture from Meiners, L. F. (2009). *But... my station is awake! Power Save Denial of Service in 802.11 Networks. Core Secur. Technol., Eden Prairie, MN, USA, Tech. Rep.*

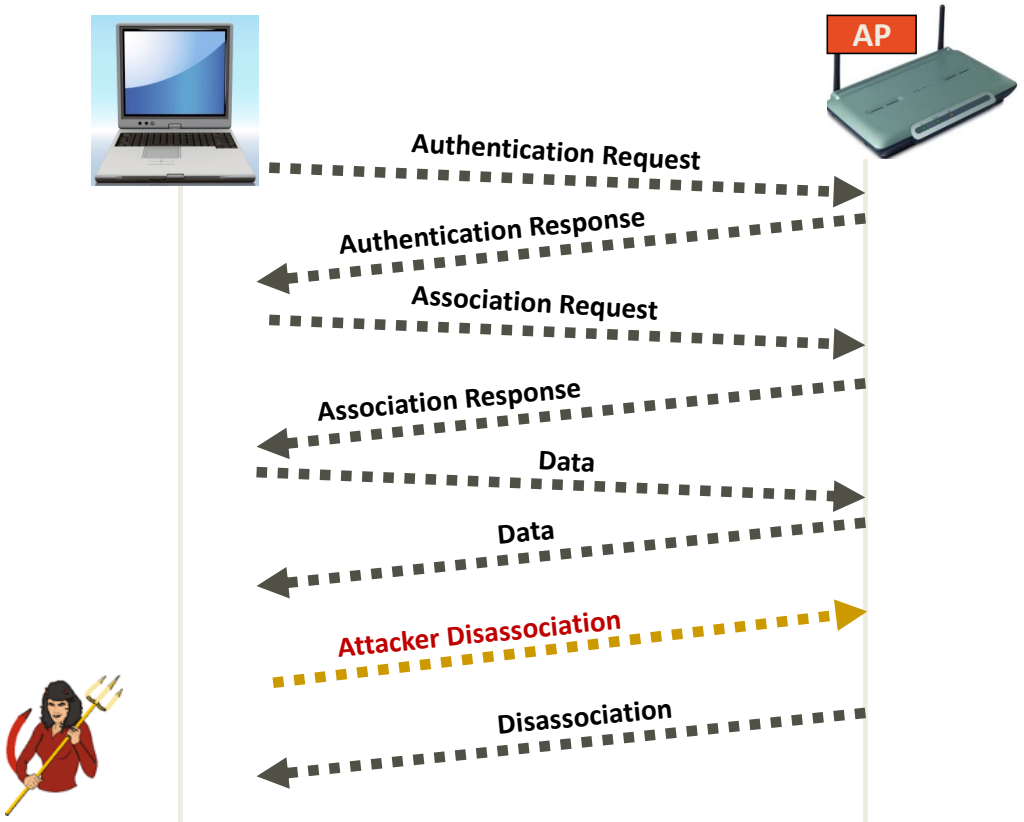
Part-2: Disassociation or deauthentication attack

- A client can authenticate with multiple APs but associate with one to allow the correct AP to forward packets
  - E.g., polito WiFi has several AP. Your STA can authenticate with many, but associate with only one

- Association frames are **unauthenticated**
- 802.11 provides both a **disassociation** and deauthentication messages

- Vulnerability is **spoofed message causing the AP to disassociate/deauthenticate the client**

Part-2: Disassociation attack



Part-2: Deauthentication attack

- This attack sends deauthentication frame to one or more clients which are currently associated with a particular access point
- This allows:
  - Recovering a hidden ESSID. This is an ESSID which is not being broadcast. Another term for this is “cloaked”.
  - Capturing WPA/WPA2 handshakes by forcing clients to reauthenticate
  - Generate ARP requests (Windows clients sometimes flush their ARP cache when disconnected)
- You need two laptops and a third device (the victim)
  - Capture packets on Wireshark on one laptop
  - Identify the MAC address of your victim
  - Use the second laptop to run a deauthentication attack with airmon-ng (note: your card must allow packet injection)

Part-2: Deauthentication attack

```
aireplay-ng -0 1 -a 00:11:22:33:44:55 -c aa:bb:cc:dd:ee:ff wlan0
```

- Where:
  - -0 means deauthentication
  - 1 is the number of deauths to send (you can send multiple if you wish)
  - -a 00:11:22:33:44:55 is the MAC address of the access point
  - -c aa:bb:cc:dd:ee:ff is the MAC address of the client you are deauthing
  - wlan0 is the interface name
- Observe the trace on wireshark and the network status on the victim
- Why does deauthentication not work?
  - You are physically too far away from the client(s). You need enough transmit power for the packets to reach and be heard by the clients. If you do a full packet capture, each packet sent to the client should result in an “ack” packet back. This means the client heard the packet. If there is no “ack” then likely it did not receive the packet.
  - Wireless cards work in particular modes such b, g, n and so on. If your card is in a different mode then the client card there is good chance that the client will not be able to correctly receive your transmission.
  - Some clients ignore broadcast deauthentications. If this is the case, you will need to send a deauthentication directed at the particular client.
  - Clients may reconnect too fast for you to see that they had been disconnected. If you do a full packet capture, you will be able to look for the reassociation packets in the capture to confirm deauthentication worked.

Part-2: Brute force attack against WPA

- This is the most common attack against WPA/WPA2 networks
- Captures the 4-way handshake that allows the authentication key to be cracked offline
- During the 4-way handshake, several pieces of information are required
  - SSID
  - The key (not directly present in the exchanges)
  - The MAC address of the two parties
  - The nonce
- You will need to capture the challenge and the result
- Brute-force: Once the handshake has been captured, the attacker only has to try all possible combinations offline to guess the key using a dictionary

- Enable monitor mode and capture packets as usual
- Identify and capture a 4-way handshake

```
airodump-ng -w wpafile -c X --bssid
AA:BB:CC:DD:EE:FF wlan0
```

- Deauthenticate a device to force a reauthentication phase
  - or manually associate another device

```
aireplay-ng -0 0 -a 11:22:33:44:55:66 wlan0
```

- Password Cracking with Aircrack-ng

```
airodump-ng -a2 -w dictionary wpafile.cap
```

- Can use the GPU to speed up cracking
  - https://hashcat.net/hashcat/

Part-2: Brute force attack against WPA

Alternatives:

- Use zizzania to mount the attack
  - https://github.com/cyrus-and/zizzania
- Download some password dictionary
  - https://github.com/Legoclones/password-cracking
  - https://github.com/zacheller/rockyou
  - https://github.com/berzerk0/Probable-Wordlists/tree/master/Real-Passwords/WPA-Length
  - …
- Use hashcat to speed up the crack
  https://hashcat.net/wiki/doku.php?id=cracking_wpawpa2
  - Convert pcap into hash file

```
hcxpcapngtool -o hash.hc22000 -E essids dump.pcapng
```

  - Brute force the hash with a dictionary

```
hashcat -m 22000 hash.hc22000 wordlist.txt
```

- **Mount the attack on your home WiFi and check that you can recover the pwd**

Challange instructions

- In the room there is a NEW WLAN network
  - On the ISM band – 2.4GHz range
  - Unknown channel and modulation
  - Unknown SSID
  - Protected by a WPA2 personal access
  - Password is present in the 01GYSUV_2024.csv file
- There is one device connected to it
  - Unknown MAC address
  - Unknown modulation
- Goal: Crack the
  - Find the WLAN name
  - Find the MAC address of the connected device
  - Mount a deauthentication attack
  - Capture at least one complete Authentication exchange
  - Find the WPA2 password via a brute-force attack

Proof of success

- Prepare and send an email address to marco.mellia@polito.it
  - Sender must be your **official email address**

- Containing
  - You Name, Lastname, Group name, matricola ID
  - The successful password found message
  - The **FILTERED** packet capture trace (pcap file) produced by the `tcpdump` command
    - The last 4 packets MUST be the EAPOL messages (or more than 4 packets in case of retransmissions)
  - The MD5 of the pcap file
  - The output of the `date` command
- Send **BOTH** a screenshot and a cut&paste of the txt

- The first 30 received emails will get the extra point