Linux For Embedded Systems

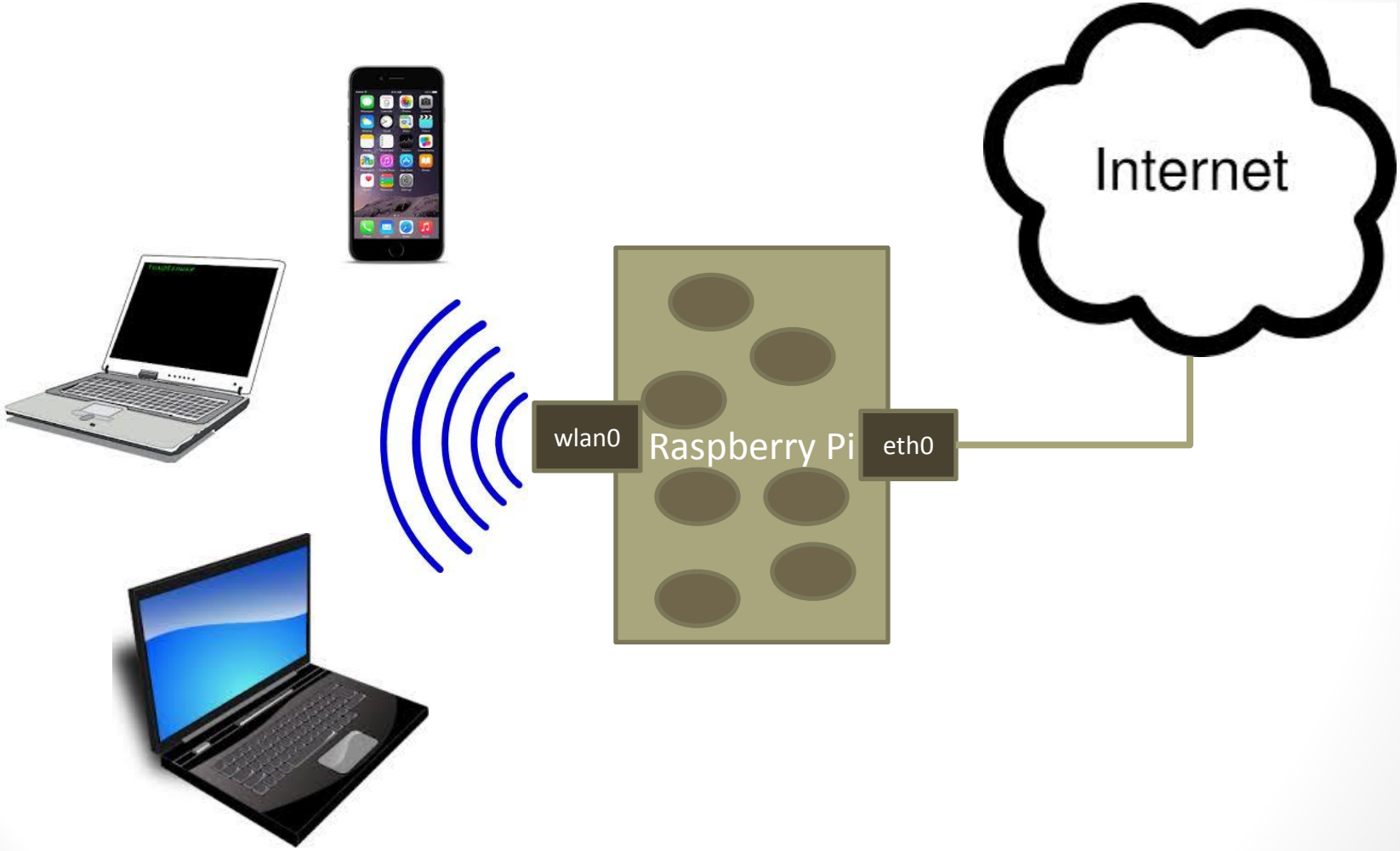
For Arabs

Cairo University
Computer Eng. Dept.
CMP445-Embedded Systems

Ahmed ElArabawy

Lecture 8:
Lab 1:
Building a Pi Based WiFi Access Point

Target



AP Building Blocks

Open Source Apps

Apps & Utilities

Linux Kernel

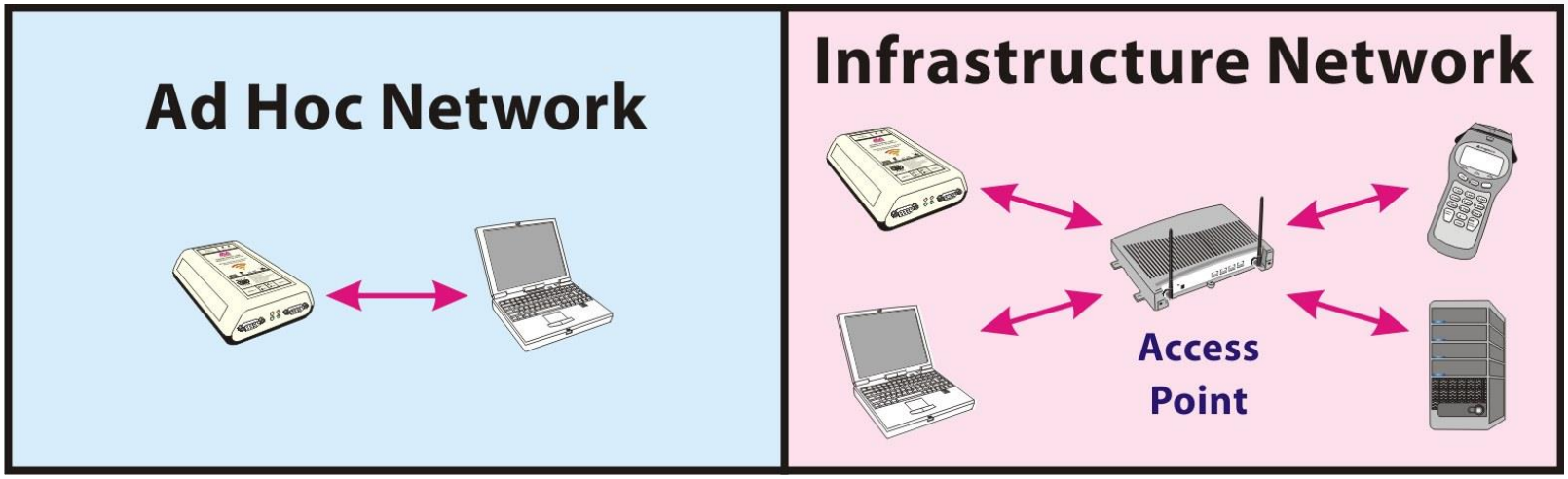
WiFi Driver

Building a Pi Based WiFi Access Point

- We can use the Raspberry Pi board to build a WiFi Access Point
  - Some WiFi chipset can be configured to provide AP functionality
  - Also, the Linux kernel has strong support for different WiFi modes of operation
  - All we need is to install and configure user plane apps that will enable us to switch the Pi into a fully functional WiFi AP
- Why do we need to build that,
  - Extend the WiFi coverage
    - But keep in mind that the Pi will be less cost effective, and less powerful than commercial products
  - Learning purposes
    - To learn more about WiFi, its modes of operation, configuration parameters, and its software components
  - Build a special featured access point such as more security, special firewall, adding anonymity (onion server)
    - A lot of libraries and applications provide a lot of functionality, so you can customize your own WiFi AP
  - Hack the AP software to test new functionality and features
    - All user plane applications are open source with a lot of community support, so you can develop your own features
    - The WiFi drivers running in the Linux Kernel are also open source, so you can also hack it
    - Linux provides a well documented API to enable developers to connect to the different components of the WiFi System
  - Build a product where the WiFi AP support is just one part of it

Introduction to WiFi Networks

WiFi Network Types



- Two types of WiFi networks (WLANs)
  - **Ad-hoc** Network
    - No access point
    - Just a group of WiFi Devices talking directly to each other
    - This is called peer-to-peer communication
    - Not very popular except for very limited environments
  - **Infrastructure** Network (Managed Network)
    - Access point needed
    - WiFi devices are clients and they only talk to the Access Point (AP)
    - Most common setup

Generations of WiFi Networks

- The WiFi network use the IEEE 802.11 protocol
- There are multiple generations for this protocol, most popular are,

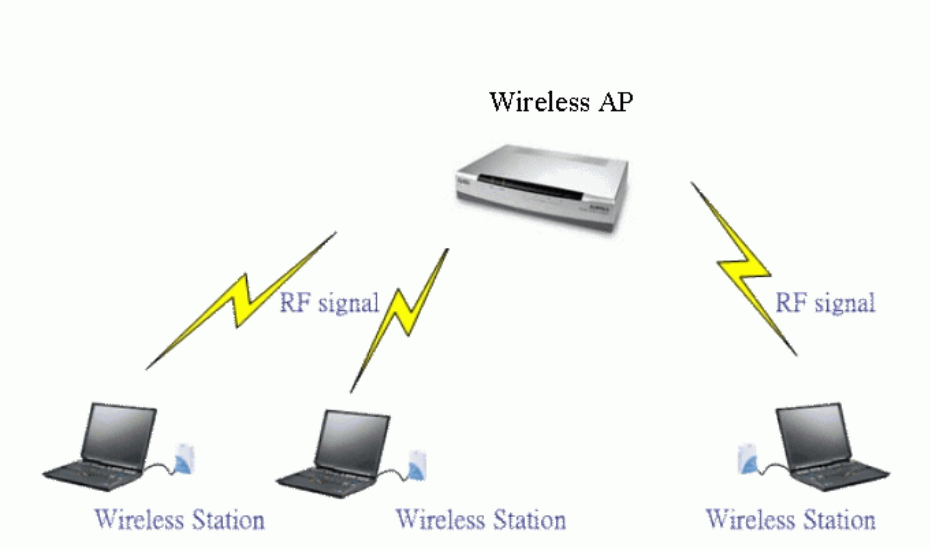
Protocol	Freq. Band (in GHz)	Ch. BW (in MHz)	Max Bit Rate (in Mbps)
802.11a	5	20	54
802.11b	2.4	20	11
802.11g	2.4	20	54
802.11n	2.4/5	20/40	75/150
802.11ac	5	20/40/80/160	100/200/433/866

Wireless Modes of a WiFi Device

Wireless Modes

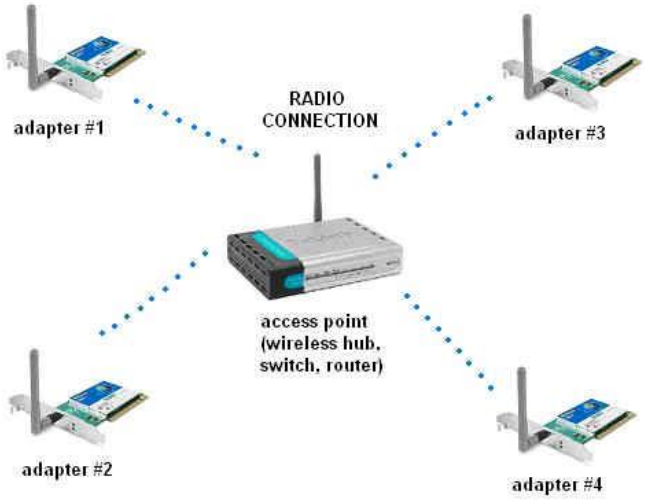
- The WiFi device (named a ***WNIC***) can work in one of the following modes
  - Station (STA) infrastructure mode
  - AccessPoint (AP) infrastructure mode
  - Ad-Hoc (IBSS) mode
  - Monitor (MON) mode
  - Wireless Distribution System (WDS) mode
  - Mesh mode
- In some ***WNICs***, it is possible to run in multiple modes at the same time

Station (STA) Infrastructure Mode



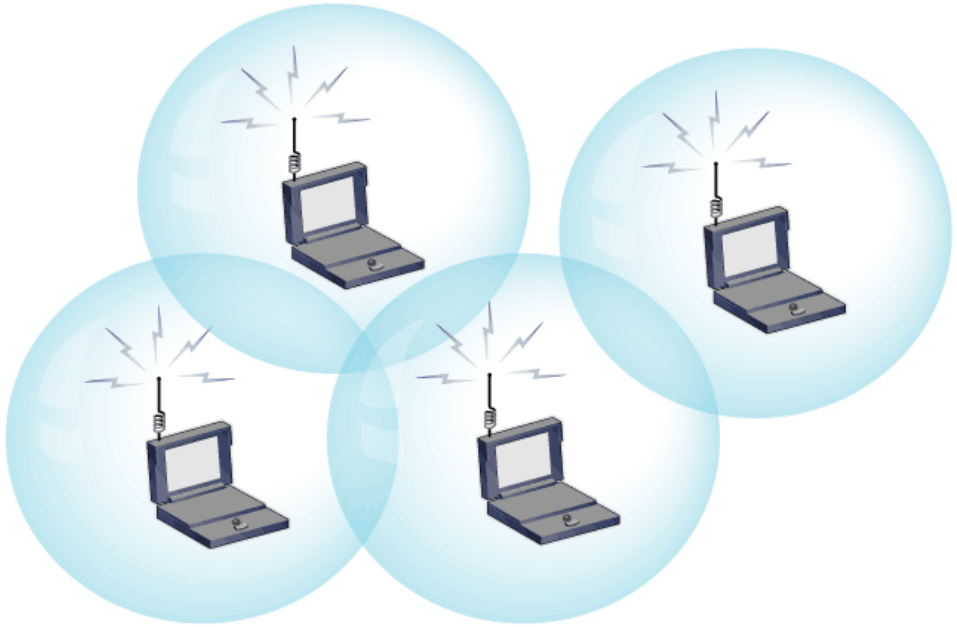
- This is the default mode for any WNIC
- In this mode the WLAN is called "***managed***" since it is managed by the Access Point (AP)
- Two WNICs in ***STA*** mode, cannot connect to one another directly
- They require a third WNIC in AP mode to manage the network
- A station in STA mode needs to join a WNIC in AP mode by,
  - Authenticate with the AP
  - Associate itself with the AP

Access Point Infrastructure Mode



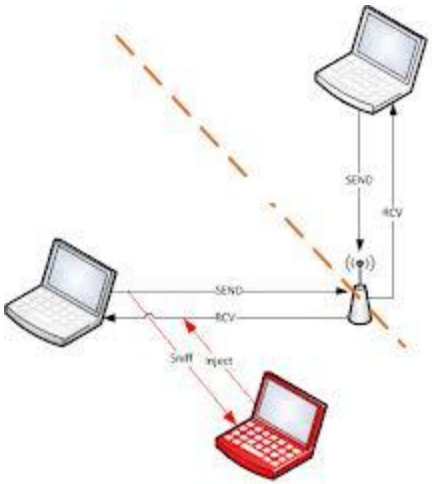
- This is the master WNIC of this WLAN
- It manages traffic between the STA nodes as well as security policies
- All STA nodes will need to authenticate/associate with the AP to join this WLAN
- The AP broadcasts its MAC Address which becomes the ***BSSID*** of the this network
- The ***SSID*** is a human readable name set by the AP and it is also broadcasted by it

Ad-Hoc (IBSS) Mode



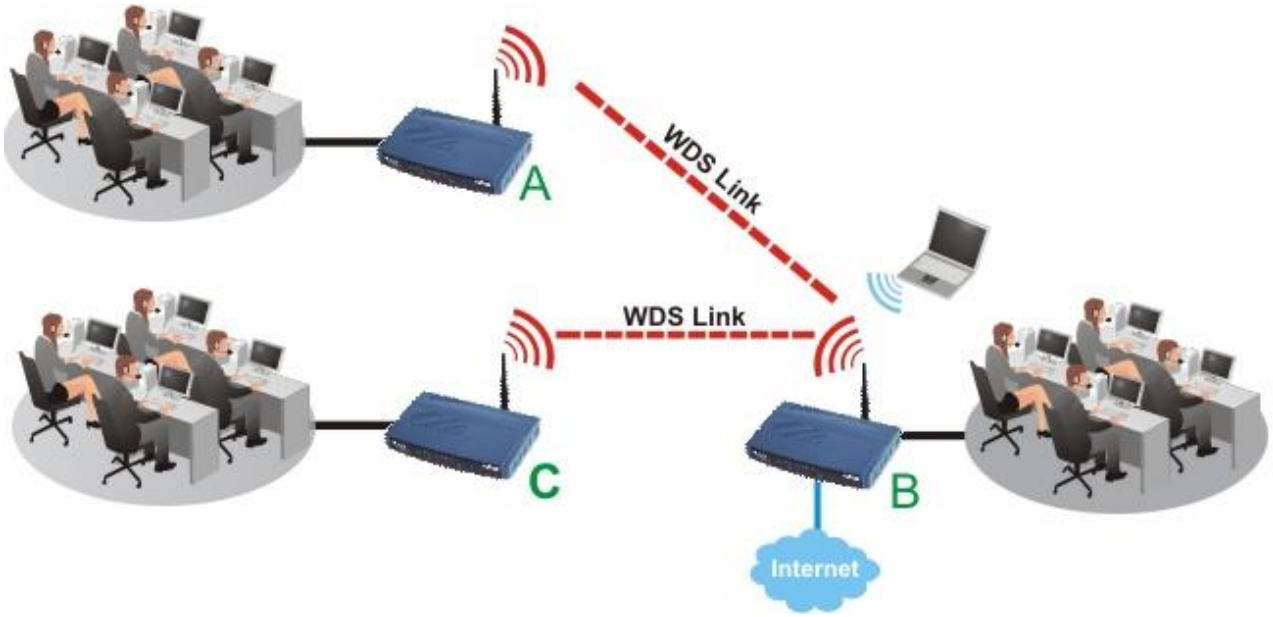
- An ***IBSS*** (Independent Basic Service Set) is a WLAN that runs without an AP
- All WNICs in the IBSS are set in this mode
- WNICs in IBSS mode manage the network in a distributed manner

Monitor (MON) Mode



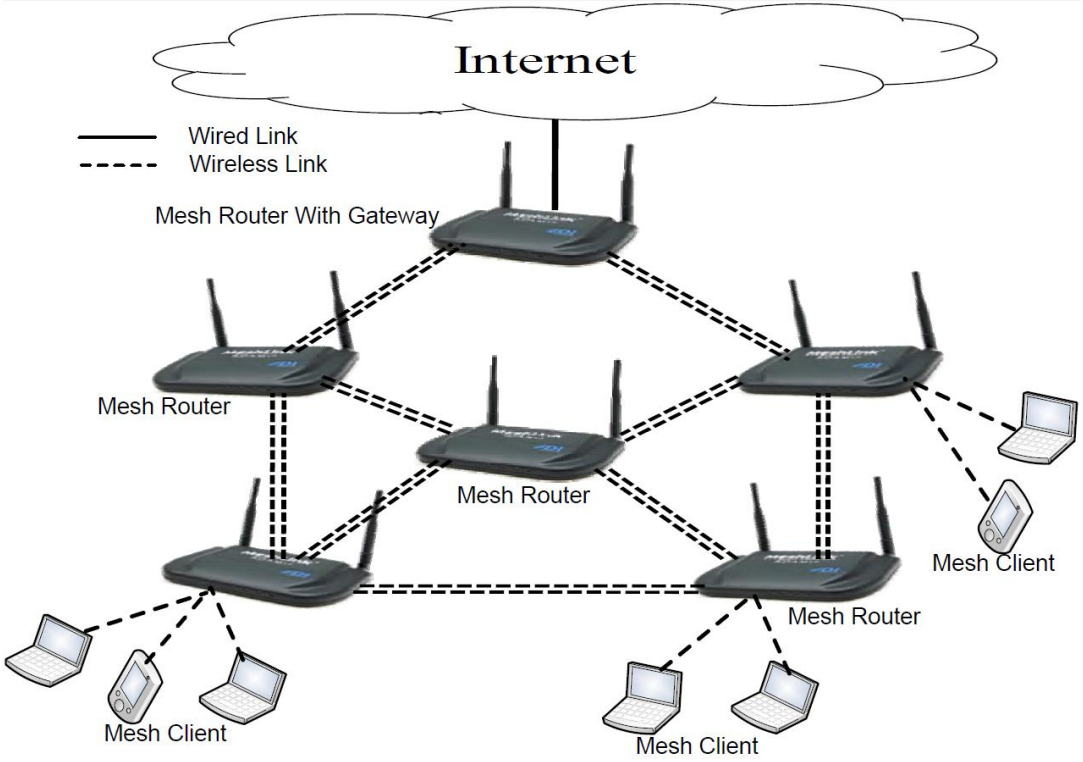
- All incoming packets are passed to the host computer
- Similar to promiscuous mode in wired NIC cards
- It is possible to have a WNIC in monitor mode in addition to a a regular device mode (if supported by hardware)
- This mode also allows transmission of packets (called ***injection***)

Wireless Distribution System (WDS) Mode



- Normally APs are connected to each other through the wired network to form a Distributed System
- However, this can also happen using the wireless link
- A WNIC which is running in ***WDS*** mode is an AP that communicates with other APs wirelessly

Mesh Mode



- Mesh interfaces are used to allow multiple devices to communication with each other by establishing intelligent routes between each other dynamically.
- This is achieved through ***802.11s***

WiFi Security

WiFi Security

- Security in WiFi has two aspects,
  - ***Authentication***
    - Verifying the identity of the station when it is joining the network
  - ***Data Integrity (Encryption)***
    - Ability to hide data content in traffic in the network
- Both authentication and encryption use ***ciphering*** to hide keys and traffic
- When the initial WiFi protocol (802.11) was introduced in 1999 it came with the security protocol named ***WEP*** (Wired Equivalent Protocol)
- WEP uses ***RC4*** ciphering for both authentication and encryption
- WEP ciphering keys,
  - 40 bit (10 digits) for ***WEP-64***
  - 104 bit (26 digits) for ***WEP-128***
- Soon, it was discovered that WEP is very insecure, and can be easily breakable, so effort was done to come up with better security protocols
- This triggered the effort done by to introduce the 802.11i Protocol

Introduction of WPA

- It was discovered in 802.11i, that a new security protocol needs to be completely different than WEP
- That would need long time to draft and will not be a simple software upgrade (we have to change the ciphering algorithm which is using hardware)
- Accordingly, current chips would not be able to support it
- It was agreed to introduce an interim protocol until the final protocol is drafted
- That interim protocol is named ***WPA*** (WiFi Protected Access)
- WPA uses a new encryption algorithm that is stronger than WEP. The new algorithm is named ***TKIP*** (Temporary Key Integrity Protocol)
- TKIP improves key handling of WEP but it uses the same ciphering algorithm (***RC4***) to be able to run on existing hardware with software upgrade (ciphering is normally run using hardware accelerators)
- ***WPA-TKIP*** is much more secure than WEP but not the final outcome of 802.11i

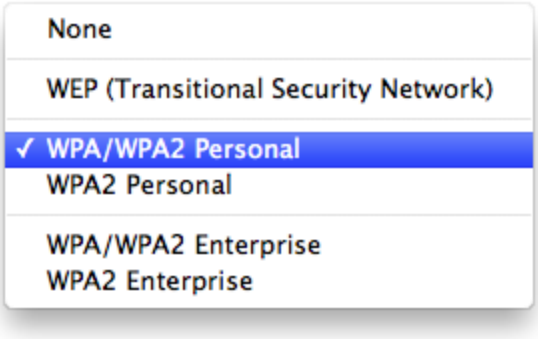
Introduction of WPA2 (RSN)

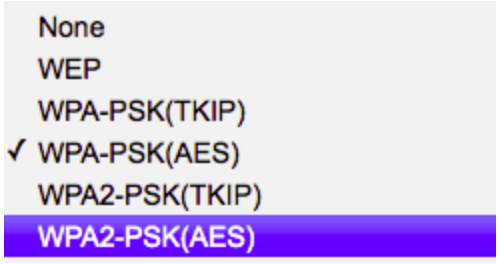
- In the final draft of 802.11i (2004), a new security mechanism was introduced and called ***RSN*** (Robust Security Network).
- The protocol is also named ***WPA2***
- WPA2 uses a more robust ciphering algorithm ***AES-CCMP*** which can not run on older chips
- In 2006, WPA2 support was a mandatory feature for any WiFi certified product
- For compatibility purposes with old devices, access points today come with both **WPA** and **WPA2** protocols, and each protocol supports both **TKIP** and **AES-CCMP** encryption
  - ***WPA-TKIP*** (the traditional WPA)
  - ***WPA-AES*** (rarely used, very close to WPA2-AES)
  - ***WPA2-TKIP*** (used to run WPA2 on the old devices)
  - ***WPA2-AES*** (the standard WPA2)

PSK and EAP

- As far as authentication 802.11i supports two modes of authentication:
  - **WPA-Personal (PSK)**:
    - Useful for residential and personal use
    - Relies on a shared passphrase between the two entities
    - Does not require a separate authentication server
    - Also named ***PSK*** (Pre-Shared Key)
  - **WPA-Enterprise (EAP)**:
    - Useful for enterprises that require stronger authentication procedures
    - No shared passphrases
    - Uses a central ***RADIUS*** server for authentication
    - Follow ***802.1X*** protocol
    - Authentication protocol is ***EAP*** (Extensible Authentication Protocol)
    - EAP is just a wrapper protocol of other protocols, hence we have ***EAP-TLS***, ***EAP-TTLS***, ***EAP-PEAP***, …

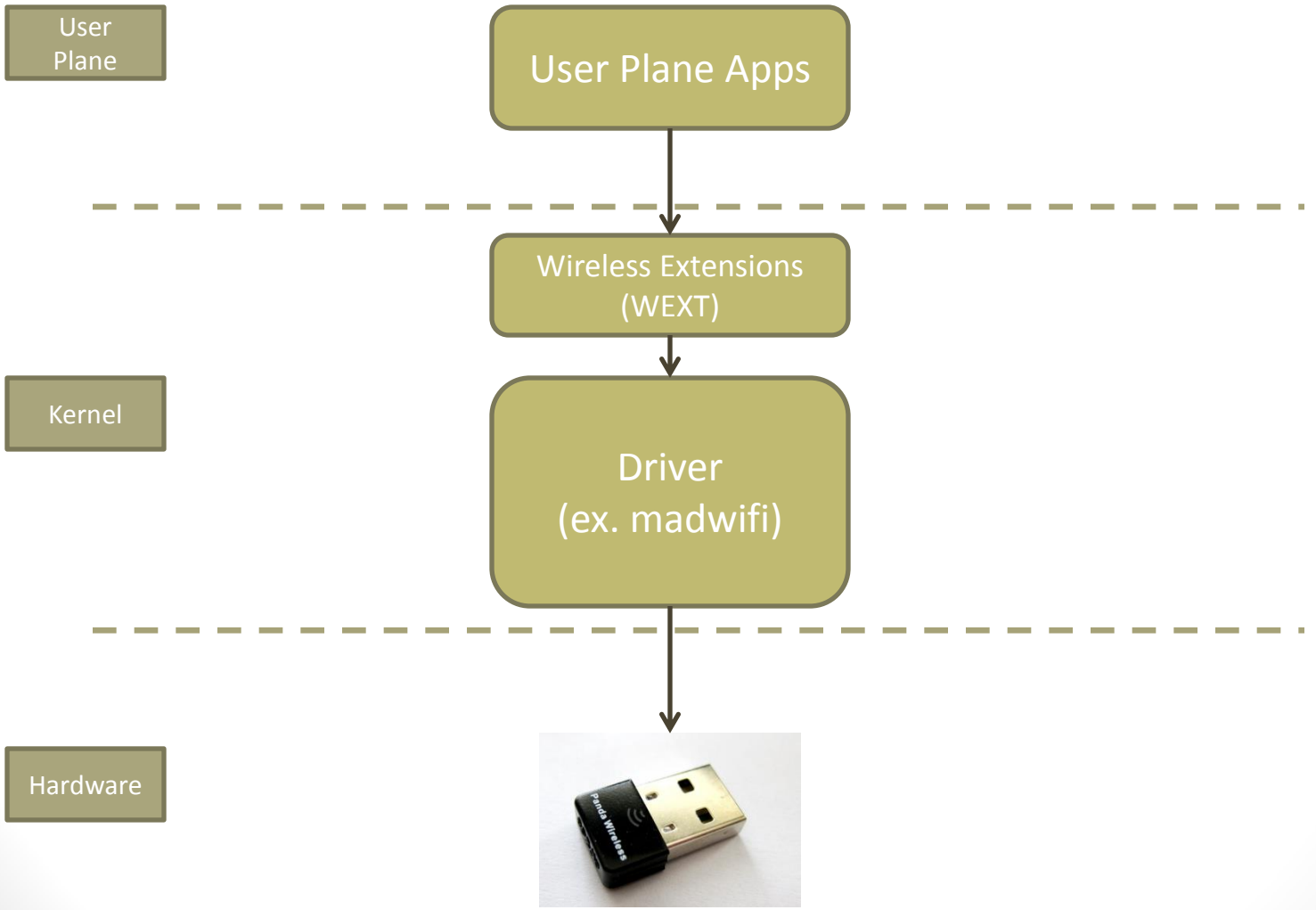
Security Selections



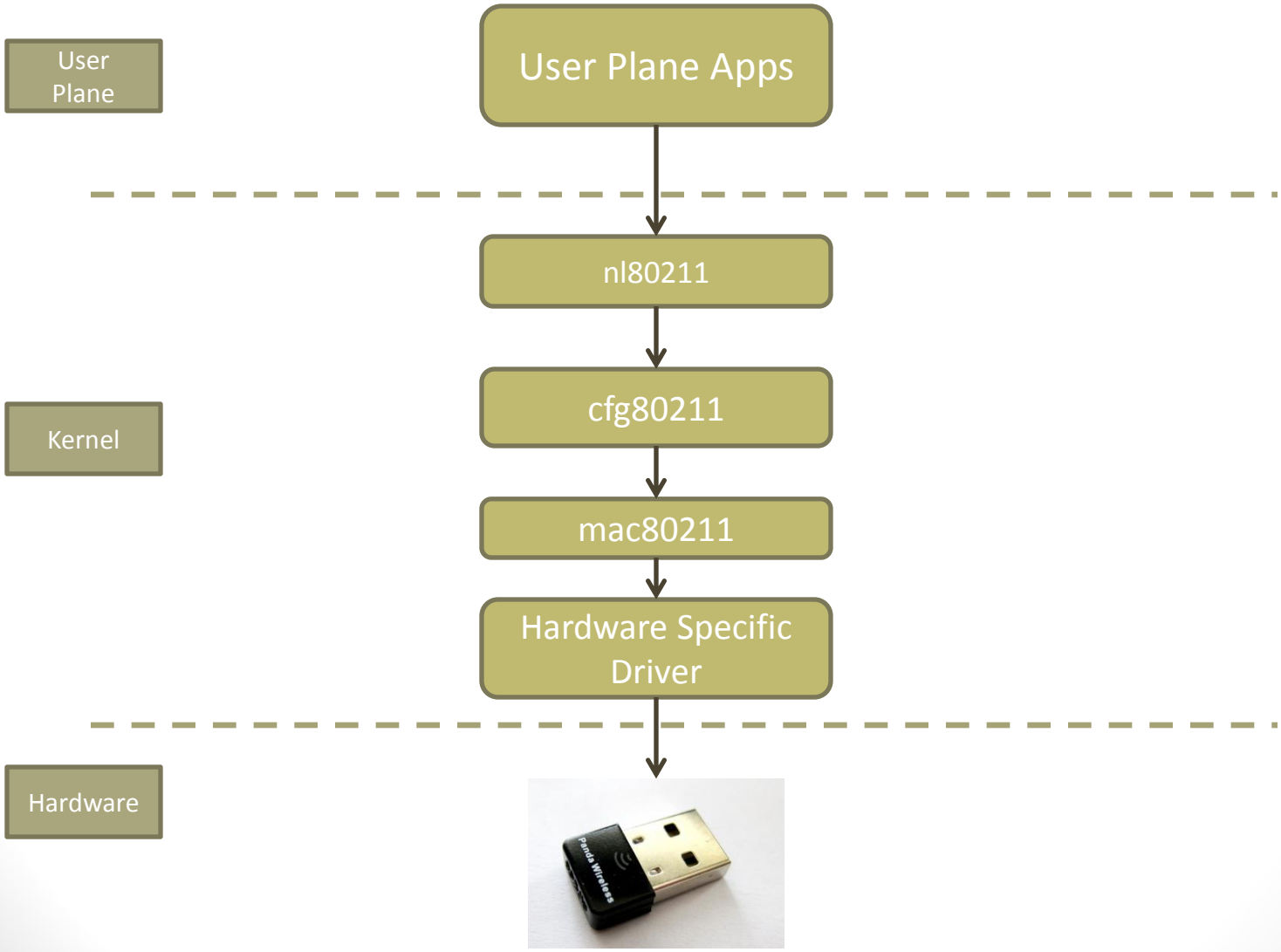


Linux WiFi Support

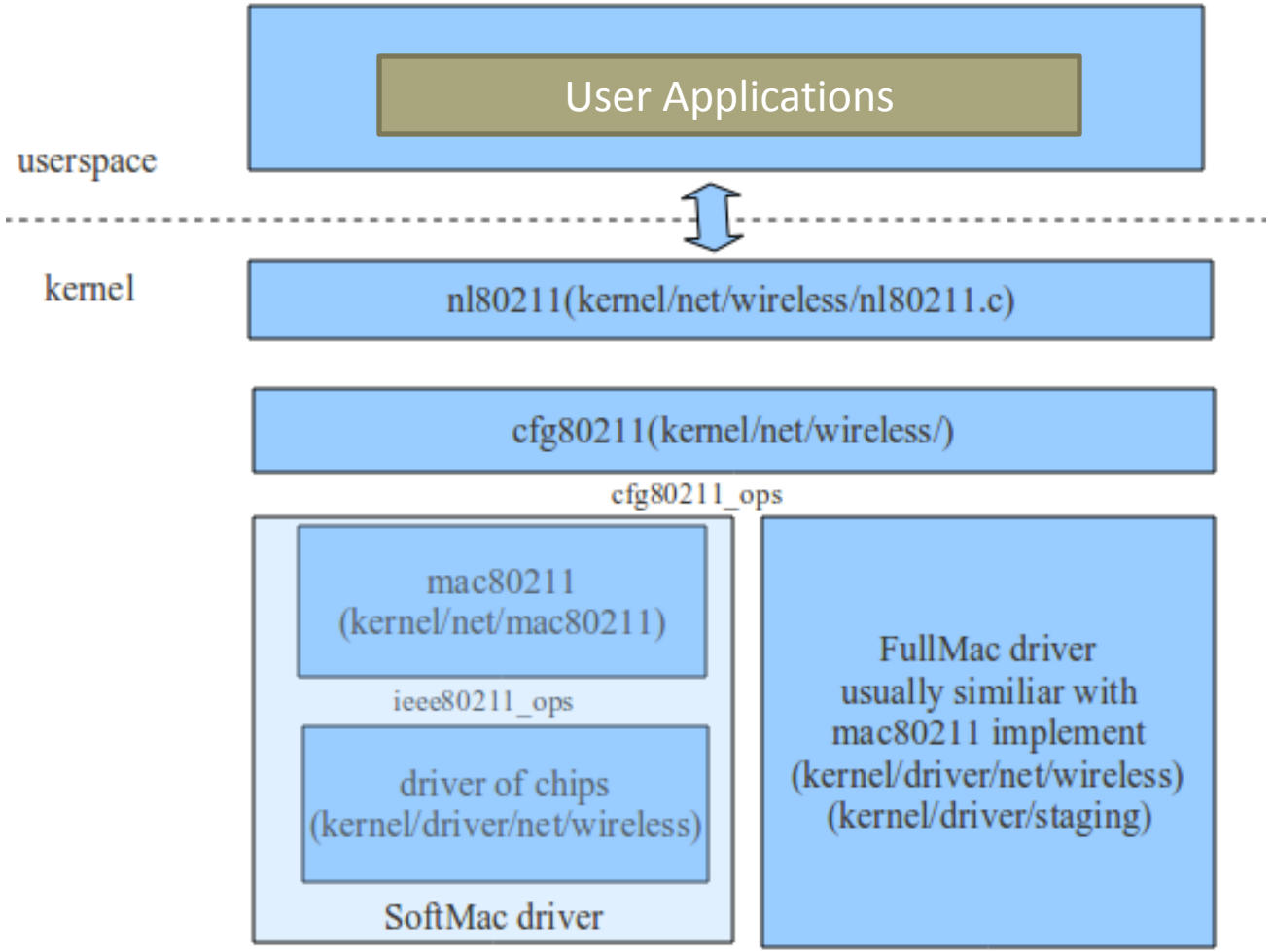
Legacy WiFi Support in Linux



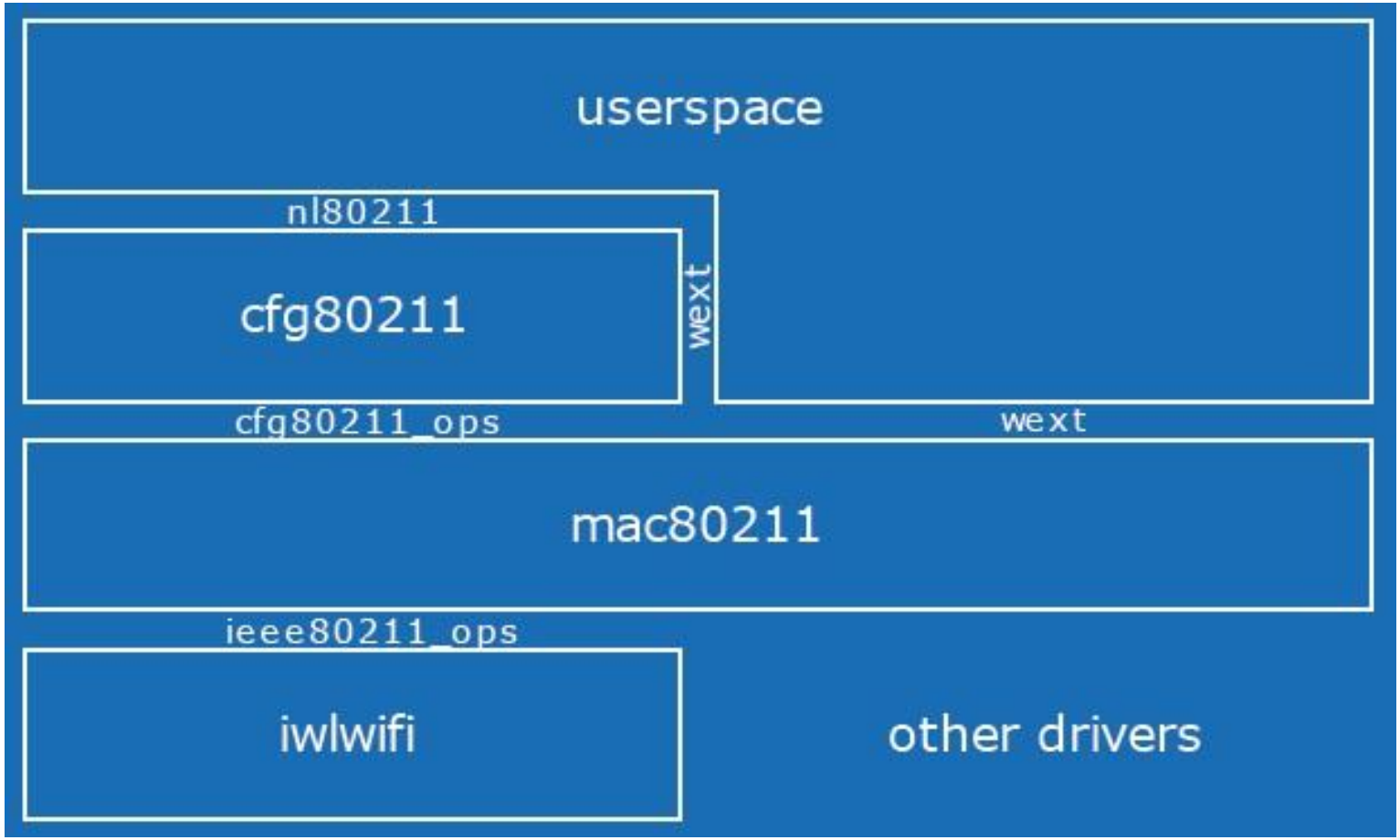
New WiFi Support in Linux



New WiFi Support in Linux



Backward Compatibility



WiFi Chipsets

- There are a lot of WiFi products in the market
- What matters is the used **chipset** in the product
- This defines the capability of the device,
  - What wireless modes does it support (Station, AP, Monitor, ..)
  - Can it support multiple modes simultaneously ?
  - What type of device architecture does it work with (***wext/nl80211***)

User Applications

- Most popular applications,
  - ***wpa-supplicant***
    - User application that implements supplicant functionality
    - Used in WNICs running in station mode (STA)
    - Performs authentication, association, security key management, …
  - ***hostapd***
    - User application that implements the access point functionality
    - Used in WNICs running in AP mode
  - ***iw***
    - Utility application that can be used to communicate with the kernel components through ***nl80211***
    - It is used to configure the driver, and read its statistics
- Keep in mind that some user plane applications only work with one driver architecture
  - For example the utility ***iw*** works with the **nl80211** driver, while ***iwconfig*** is a similar utility that works with drivers using **wext**
  - Also the official ***hostapd*** application only works with the **nl80211** architecture, there is a modified version that supports the ***wext*** architecture

Using the iw Configuration Utility

What is **iw**

- The ***iw*** is a CLI utility for configuration of Wireless devices and interfaces
- It is based on the Linux new architecture of WiFi support (using the ***nl80211*** interface)
- It replaces the old utility ***iwconfig*** which was based on the deprecated ***Wireless Extensions*** interface
- For WiFi modules that don't use the new architecture, when you try to use the ***iw*** utility, you will get an error message,

  $ iw list

  nl80211 not found
- Accordingly, for those modules, you will have to use old utility ***iwconfig*** instead of ***iw***

- For more info about ***iw***, http://wireless.kernel.org/en/users/Documentation/iw

Listing device Capability (iw list Command)

- Using the iw utility, you can list the capability of the WNIC device such as,
  - Supported modes of operation
  - Supported channels
  - Supported bit rates
  - Supported security protocols
  - Transmit power ranges
  - Supported features (Power save mode, Advanced modulation types, ..)
  - And a lot of other features

$ iw list

pi@raspberrypi: ~

```
Do NOT screenscrape this tool, we don't consider its output stable.

pi@raspberrypi ~ $ iw list
Wiphy phy0
        Band 1:
                Capabilities: 0x172
                        HT20/HT40
                        Static SM Power Save
                        RX Greenfield
                        RX HT20 SGI
                        RX HT40 SGI
                        RX STBC 1-stream
                        Max AMSDU length: 3839 bytes
                        No DSSS/CCK HT40
                Maximum RX AMPDU length 65535 bytes (exponent: 0x003)
                Minimum RX AMPDU time spacing: 2 usec (0x04)
                HT RX MCS rate indexes supported: 0-7, 32
                TX unequal modulation not supported
                HT TX Max spatial streams: 1
                HT TX MCS rate indexes supported may differ
                Frequencies:
                        * 2412 MHz [1] (20.0 dBm)
                        * 2417 MHz [2] (20.0 dBm)
                        * 2422 MHz [3] (20.0 dBm)
                        * 2427 MHz [4] (20.0 dBm)
                        * 2432 MHz [5] (20.0 dBm)
                        * 2437 MHz [6] (20.0 dBm)
                        * 2442 MHz [7] (20.0 dBm)
                        * 2447 MHz [8] (20.0 dBm)
                        * 2452 MHz [9] (20.0 dBm)
                        * 2457 MHz [10] (20.0 dBm)
                        * 2462 MHz [11] (20.0 dBm)
                        * 2467 MHz [12] (20.0 dBm) (passive scanning, no IBSS)
                        * 2472 MHz [13] (20.0 dBm) (passive scanning, no IBSS)
                        * 2484 MHz [14] (20.0 dBm) (passive scanning, no IBSS)
                Bitrates (non-HT):
```

$ iw list

pi@raspberrypi: ~

```
            Bitrates (non-HT):
                    * 1.0 Mbps
                    * 2.0 Mbps (short preamble supported)
                    * 5.5 Mbps (short preamble supported)
                    * 11.0 Mbps (short preamble supported)
                    * 6.0 Mbps
                    * 9.0 Mbps
                    * 12.0 Mbps
                    * 18.0 Mbps
                    * 24.0 Mbps
                    * 36.0 Mbps
                    * 48.0 Mbps
                    * 54.0 Mbps
    max # scan SSIDs: 4
    max scan IEs length: 2257 bytes
    Coverage class: 0 (up to 0m)
    Supported Ciphers:
            * WEP40 (00-0f-ac:1)
            * WEP104 (00-0f-ac:5)
            * TKIP (00-0f-ac:2)
            * CCMP (00-0f-ac:4)
    Available Antennas: TX 0 RX 0
    Supported interface modes:
             * IBSS
             * managed
             * AP
             * AP/VLAN
             * WDS
             * monitor
             * mesh point
    software interface modes (can always be added):
             * AP/VLAN
             * monitor
    valid interface combinations:
             * #{ AP, mesh point } <= 8,
               total <= 8, #channels <= 1
    Supported commands:
```

$ iw list


```
pi@raspberrypi: ~
        Supported commands:
                 * new_interface
                 * set_interface
                 * new_key
                 * new_beacon
                 * new_station
                 * new_mpath
                 * set_mesh_params
                 * set_bss
                 * authenticate
                 * associate
                 * deauthenticate
                 * disassociate
                 * join_ibss
                 * join_mesh
                 * set_tx_bitrate_mask
                 * action
                 * frame_wait_cancel
                 * set_wiphy_netns
                 * set_channel
                 * set_wds_peer
                 * Unknown command (84)
                 * Unknown command (87)
                 * Unknown command (85)
                 * Unknown command (89)
                 * Unknown command (92)
                 * connect
                 * disconnect
        Supported TX frame types:
                 * IBSS: 0x00 0x10 0x20 0x30 0x40 0x50 0x60 0x70 0x80 0x90 0xa0 0xb0 0xc0 0xd0 0xe0 0xf0
                 * managed: 0x00 0x10 0x20 0x30 0x40 0x50 0x60 0x70 0x80 0x90 0xa0 0xb0 0xc0 0xd0 0xe0 0xf0
                 * AP: 0x00 0x10 0x20 0x30 0x40 0x50 0x60 0x70 0x80 0x90 0xa0 0xb0 0xc0 0xd0 0xe0 0xf0
                 * AP/VLAN: 0x00 0x10 0x20 0x30 0x40 0x50 0x60 0x70 0x80 0x90 0xa0 0xb0 0xc0 0xd0 0xe0 0xf0
                 * mesh point: 0x00 0x10 0x20 0x30 0x40 0x50 0x60 0x70 0x80 0x90 0xa0 0xb0 0xc0 0xd0 0xe0 0xf0
                 * P2P-client: 0x00 0x10 0x20 0x30 0x40 0x50 0x60 0x70 0x80 0x90 0xa0 0xb0 0xc0 0xd0 0xe0 0xf0
                 * P2P-GO: 0x00 0x10 0x20 0x30 0x40 0x50 0x60 0x70 0x80 0x90 0xa0 0xb0 0xc0 0xd0 0xe0 0xf0
                 * Unknown mode (10): 0x00 0x10 0x20 0x30 0x40 0x50 0x60 0x70 0x80 0x90 0xa0 0xb0 0xc0 0xd0 0xe0 0xf0
```

$ iw list


```
pi@raspberrypi: ~
                * set_wds_peer
                * Unknown command (84)
                * Unknown command (87)
                * Unknown command (85)
                * Unknown command (89)
                * Unknown command (92)
                * connect
                * disconnect
        Supported TX frame types:
                * IBSS: 0x00 0x10 0x20 0x30 0x40 0x50 0x60 0x70 0x80 0x90 0xa0 0xb0 0xc0 0xd0 0xe0 0xf0
                * managed: 0x00 0x10 0x20 0x30 0x40 0x50 0x60 0x70 0x80 0x90 0xa0 0xb0 0xc0 0xd0 0xe0 0xf0
                * AP: 0x00 0x10 0x20 0x30 0x40 0x50 0x60 0x70 0x80 0x90 0xa0 0xb0 0xc0 0xd0 0xe0 0xf0
                * AP/VLAN: 0x00 0x10 0x20 0x30 0x40 0x50 0x60 0x70 0x80 0x90 0xa0 0xb0 0xc0 0xd0 0xe0 0xf0
                * mesh point: 0x00 0x10 0x20 0x30 0x40 0x50 0x60 0x70 0x80 0x90 0xa0 0xb0 0xc0 0xd0 0xe0 0xf0
                * P2P-client: 0x00 0x10 0x20 0x30 0x40 0x50 0x60 0x70 0x80 0x90 0xa0 0xb0 0xc0 0xd0 0xe0 0xf0
                * P2P-GO: 0x00 0x10 0x20 0x30 0x40 0x50 0x60 0x70 0x80 0x90 0xa0 0xb0 0xc0 0xd0 0xe0 0xf0
                * Unknown mode (10): 0x00 0x10 0x20 0x30 0x40 0x50 0x60 0x70 0x80 0x90 0xa0 0xb0 0xc0 0xd0 0xe0 0xf0
        Supported RX frame types:
                * IBSS: 0x40 0xb0 0xc0 0xd0
                * managed: 0x40 0xd0
                * AP: 0x00 0x20 0x40 0xa0 0xb0 0xc0 0xd0
                * AP/VLAN: 0x00 0x20 0x40 0xa0 0xb0 0xc0 0xd0
                * mesh point: 0xb0 0xc0 0xd0
                * P2P-client: 0x40 0xd0
                * P2P-GO: 0x00 0x20 0x40 0xa0 0xb0 0xc0 0xd0
                * Unknown mode (10): 0x40 0xd0
        Device supports RSN-IBSS.
        HT Capability overrides:
                * MCS: ff ff ff ff ff ff ff ff ff ff
                * maximum A-MSDU length
                * supported channel width
                * short GI for 40 MHz
                * max A-MPDU length exponent
                * min MPDU start spacing
        Device supports TX status socket option.
        Device supports HT-IBSS.
pi@raspberrypi ~ $
```

Scanning For Wireless Networks (iw scan Command)

$ iw dev <dev name> scan

- This command scans for the available networks seen by the device
- This includes both AP based and ad-hoc networks

  $ sudo iw dev wlan0 scan

  Note that we need root access for this command

$ sudo iw dev wlan0 scan


```
pi@raspberrypi: ~
pi@raspberrypi ~ $ sudo iw dev wlan0 scan
BSS 2e:a4:3c:81:7a:42 (on wlan0)
        TSF: 440066900 usec (0d, 00:07:20)
        freq: 2437
        beacon interval: 100
        capability: ESS Privacy ShortPreamble ShortSlotTime (0x0431)
        signal: -63.00 dBm
        last seen: 2690 ms ago
        Information elements from Probe Response frame:
        SSID: Peachjar
        Supported rates: 1.0* 2.0* 5.5* 11.0* 6.0 9.0 12.0 18.0
        DS Parameter set: channel 6
        ERP: <no flags>
        RSN:     * Version: 1
                 * Group cipher: TKIP
                 * Pairwise ciphers: CCMP TKIP
                 * Authentication suites: PSK
                 * Capabilities: (0x0000)
        Extended supported rates: 24.0 36.0 48.0 54.0
        HT capabilities:
                Capabilities: 0x11cc
                        HT20
                        SM Power Save disabled
                        RX HT40 SGI
                        TX STBC
                        RX STBC 1-stream
                        Max AMSDU length: 3839 bytes
                        DSSS/CCK HT40
                Maximum RX AMPDU length 65535 bytes (exponent: 0x003)
                Minimum RX AMPDU time spacing: 8 usec (0x06)
                HT RX MCS rate indexes supported: 0-15
                HT TX MCS rate indexes are undefined
        HT operation:
                 * primary channel: 6
                 * secondary channel offset: no secondary
                 * STA channel width: 20 MHz
                 * RIFS: 1
```

$ sudo iw dev wlan0 scan

pi@raspberrypi: ~

```
        HT operation:
                 * primary channel: 6
                 * secondary channel offset: no secondary
                 * STA channel width: 20 MHz
                 * RIFS: 1
                 * HT protection: no
                 * non-GF present: 1
                 * OBSS non-GF present: 0
                 * dual beacon: 0
                 * dual CTS protection: 0
                 * STBC beacon: 0
                 * L-SIG TXOP Prot: 0
                 * PCO active: 0
                 * PCO phase: 0
        WMM:     * Parameter version 1
                 * BE: CW 15-63, AIFSN 3
                 * BK: CW 15-1023, AIFSN 7
                 * VI: CW 7-15, AIFSN 1, TXOP 3008 usec
                 * VO: CW 3-7, AIFSN 1, TXOP 1504 usec
BSS 2a:a4:3c:81:7a:42 (on wlan0)
        TSF: 440061122 usec (0d, 00:07:20)
        freq: 2437
        beacon interval: 100
        capability: ESS Privacy ShortPreamble ShortSlotTime (0x0431)
        signal: -63.00 dBm
        last seen: 2690 ms ago
        Information elements from Probe Response frame:
        SSID: Peachjar Guest
        Supported rates: 1.0* 2.0* 5.5* 11.0* 6.0 9.0 12.0 18.0
        DS Parameter set: channel 6
        ERP: <no flags>
        RSN:     * Version: 1
                 * Group cipher: TKIP
                 * Pairwise ciphers: CCMP TKIP
                 * Authentication suites: PSK
                 * Capabilities: (0x0000)
        Extended supported rates: 24.0 36.0 48.0 54.0
```

Listening to Events (iw event Command)

$ iw event

- This command is used to display the different network events
- This is useful in debugging

  $ iw event

- To show the different management frames such as authentication and association frames

  $ iw event -f

- To show timing information

  $ iw event -t

$ iw event -f -t

pi@raspberrypi: ~

pi@raspberrypi: ~ | aelarabawy@aelarabawy-demo-backup64: ~/work/rpi/rpi-projects/wifi-projects/e...

```
pi@raspberrypi ~ $ iw event -f -t
========================================
I WILL DISCONNECT FROM THE WLAN NOW
========================================
1415829416.046939: wlan0: del station d6:8c:b5:68:2f:87
1415829416.093329: wlan0 (phy #0): deauth 00:0f:60:01:7d:d3 -> d6:8c:b5:68:2f:87 reason 3: Deauthenticated because sending station is
f 87 00 0f 60 01 7d d3 d6 8c b5 68 2f 87 00 00 03 00]
1415829416.094930: wlan0 (phy #0): disconnected (local request)
1415829416.246893: phy #0: regulatory domain change: set to world roaming by the wireless core upon initialization request


========================================
I WILL START TO SCAN NOW
========================================
1415829450.052700: wlan0 (phy #0): scan started
1415829451.435991: wlan0 (phy #0): scan finished: 2412 2417 2422 2427 2432 2437 2442 2447 2452 2457 2462 2467 2472 2484, ""



========================================
I WILL CONNECT BACK NOW
========================================
1415829494.424095: wlan0 (phy #0): scan started
1415829495.736243: wlan0 (phy #0): scan finished: 2412 2417 2422 2427 2432 2437 2442 2447 2452 2457 2462 2467 2472 2484, ""
1415829495.813669: wlan0: new station d6:8c:b5:68:2f:87
1415829495.815666: wlan0 (phy #0): auth d6:8c:b5:68:2f:87 -> 00:0f:60:01:7d:d3 status: 0: Successful [frame: b0 00 40 01 00 0f 60 01
1415829495.841696: wlan0 (phy #0): assoc d6:8c:b5:68:2f:87 -> 00:0f:60:01:7d:d3 status: 0: Successful [frame: 10 00 40 01 00 0f 60 01
 82 84 8b 0c 12 96 18 24 32 04 30 48 60 6c dd 18 00 50 f2 02 01 01 8e 00 03 a4 00 00 27 a4 00 00 42 43 5e 00 62 32 2f 00 dd 1e 00 90
00 00 00 00 2d 1a 4c 10 1b ff ff 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 dd 1a 00 90 4c 34 04 08 08 00 00 00 0
0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 4a 0e 14 00 0a 00 2c 01 c8 00 14 00 05 00 19 00 7f 01 01 dd 09 00 03 7f 01 01 00 00 ff
1415829495.848282: wlan0 (phy #0): connected to d6:8c:b5:68:2f:87
1415829539.396112: wlan0 (phy #0): connection quality monitor event: RSSI went below threshold
```

More iw Commands

- With ***iw***, you can also,
  - Collect device statistics
  - Collect link information
  - Read/set transmit power
  - Enable/Disable power saving mode
  - Setting frequency channel and BW
  - Select bit rate
  - Adding / Deleting interfaces

Building the Access Point

Pre-requisites

- Prepare the SD Card with a Raspbian OS
- Setup the Ethernet Connection (eth0)
- Setup the WiFi Connection (wlan0) using a WiFi USB Module

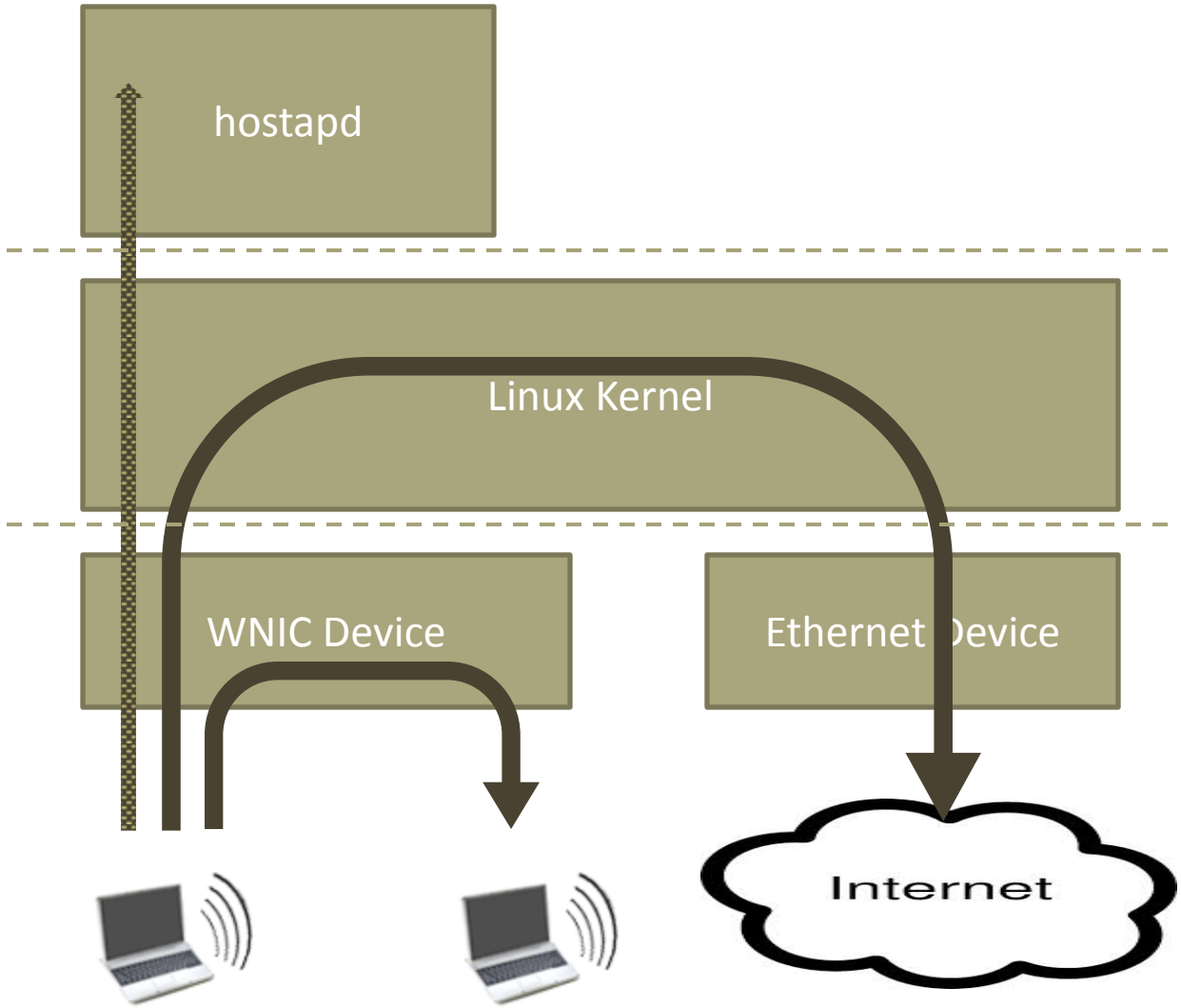
First Step: hostapd

- ***hostapd*** is a user plane application that handles WNIC AP functionality
- To run the WNIC device in AP mode, ***hostapd*** needs to be running (instead of ***wpa-supplicant*** used for STA mode)
- The ***hostapd*** program works with WNICs running with ***nl80211*** drivers only, a modified version may be used for other drivers
- For example, for WiFi modules using the chipset ***Realtek RTL8188CUS***

  http://www.jenssegers.be/blog/43/Realtek-RTL8188-based-access-point-on-Raspberry-Pi

- ***hostapd*** is responsible for handling management operations from stations such as ,
  - Authentication procedure of Stations
  - Association procedure of stations
- Note that ***hostapd*** has nothing to do with data packets handling, this is completely done in the kernel (or in the hardware device)

Data vs. Management Traffic



Setting Up hostapd

- Install ***hostapd***

  $ sudo apt-get install hostapd
- Create a configuration file for ***hostapd***:

  $ sudo vi /etc/hostapd/hostapd.conf

  Add these Lines:

```
interface=wlan0
driver=nl80211
ssid=Pi_AP
hw_mode=g
channel=6
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=raspberry
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
```

- Set the configuration file name for ***hostapd***

  $ sudo vi /etc/default/hostapd

```
DAEMON_CONF="/etc/hostapd/hostapd.conf"
```

Run hostapd

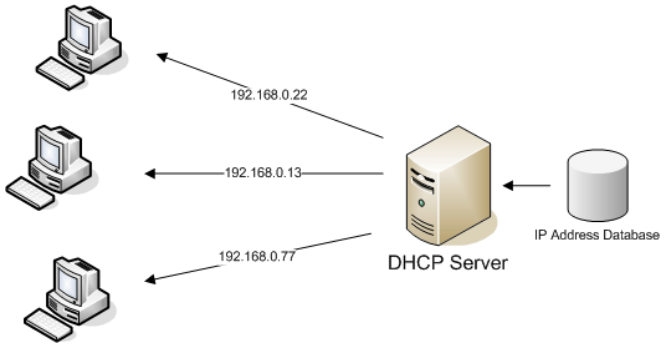
- Now we can run ***hostapd***

  $ sudo /usr/sbin/hostapd /etc/hostapd/hostapd.conf

- Note: use **-dd** option for detailed log

- The AP is now active, and it can be seen by other computers
- However, when trying to connect to it, connection fail !!!
- This is because, the access point is not able to provide IP Addresses to the connecting stations
- We need a DHCP server, to provide the IP addressing

Second Step: DHCP Server



- WiFi Stations need to acquire an IP Address when connecting to the AP
- This means, an AP needs to have a dhcp server to allocate IP Addresses to the connecting devices
- The dhcp server needs to be configured with the range of addresses that it uses for allocation
- The addresses should belong to the same subnet, and the AP should be the default gateway for this subnet

Setting up DHCP Server

- Install a DHCP Server on the Pi

  $ sudo apt-get install isc-dhcp-server
- Configure isc-dhcp-server with which interface to use

  $ sudo vi /etc/default/isc-dhcp-server

  INTERFACES="wlan0"
- Configure the dhcp server

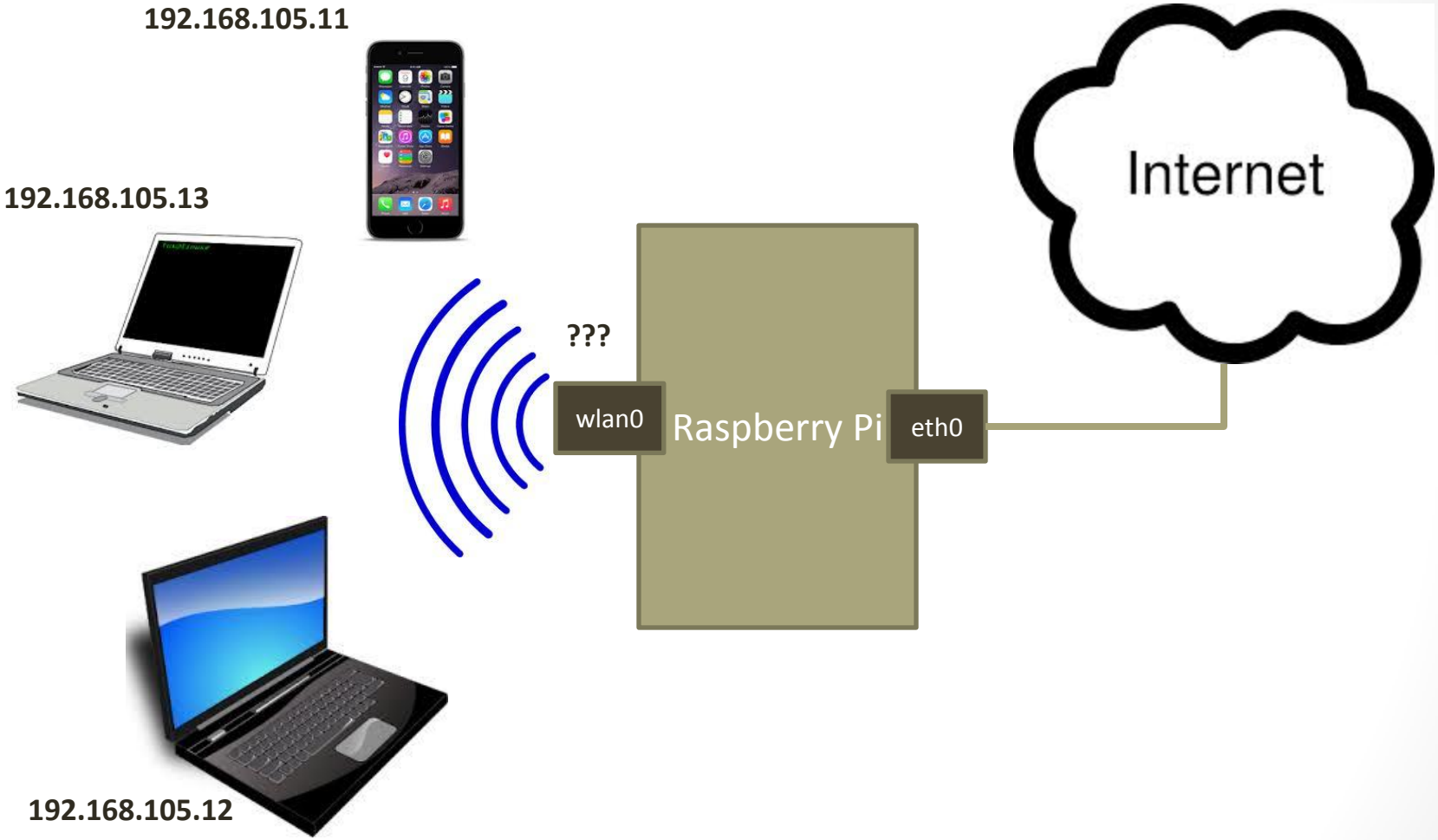
  $ sudo vi /etc/dhcp/dhcpd.conf
  - Comment out the lines:

    ```
    option domain-name "example.org";
    option domain-name-servers ns1.example.org ns2.example.org;
    ```
  - Uncomment the line

    ```
    Authoritative
    ```
  - Add the configurations

    ```
    subnet 192.168.105.0 netmask 255.255.255.0 {
    range 192.168.105.10 192.168.105.50;
    option broadcast-address 192.168.105.255;
    option routers 192.168.105.1
    default-lease-time 600;
    max-lease-time 7200
    option domain-name "local"
    option domain-name-servers 8.8.8.8, 8.8.4.4;
    }
    ```
- Now we need to start the dhcp service

  $ sudo service isc-dhcp-server start


192.168.105.11
192.168.105.13
192.168.105.12
???
wlan0
Raspberry Pi
eth0
Internet

Set Up wlan0 IP Address

- We need to make the IP address of wlan0 within the same subnet as the station connected to the AP
- This require that we set the IP address for this interface as a static IP address

  $ sudo vi /etc/network/interfaces

  - Comment all lines configuring wlan0 under the line

    allow-hotplug wlan0

  - At the end of the file, add the lines

    Iface wlan0 inet static
    address 192.168.105.1
    netmask 255.255.255.0

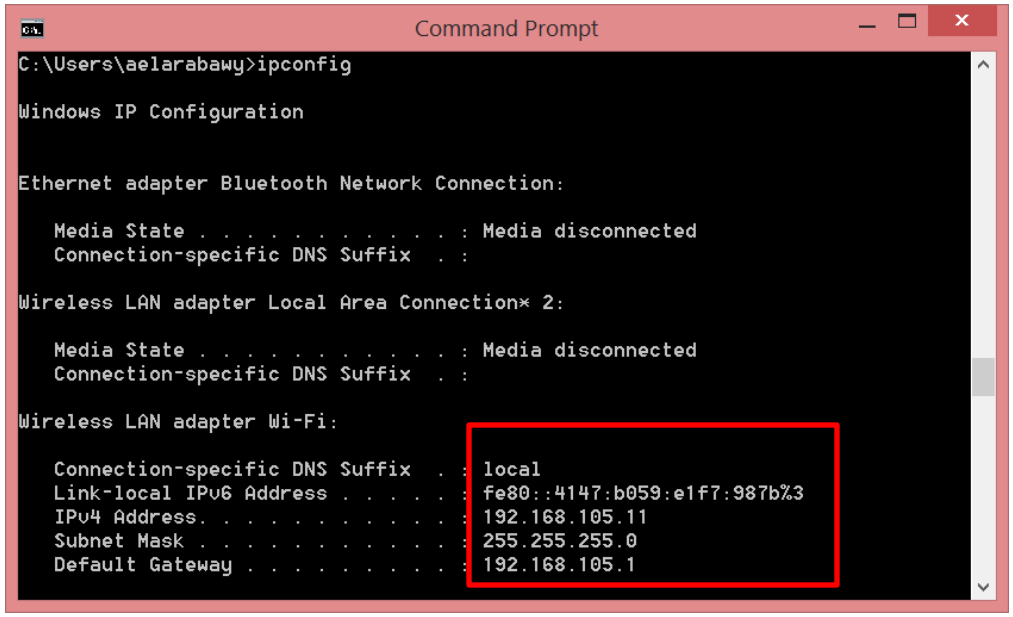
- We need now to restart the wlan0 interface so it will pick up the new IP Address:

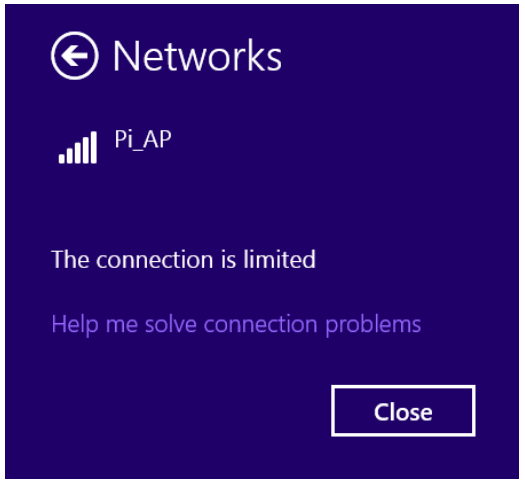
  $ sudo ifdown wlan0

  $ sudo ifup wlan0

Now Let Us Try Again

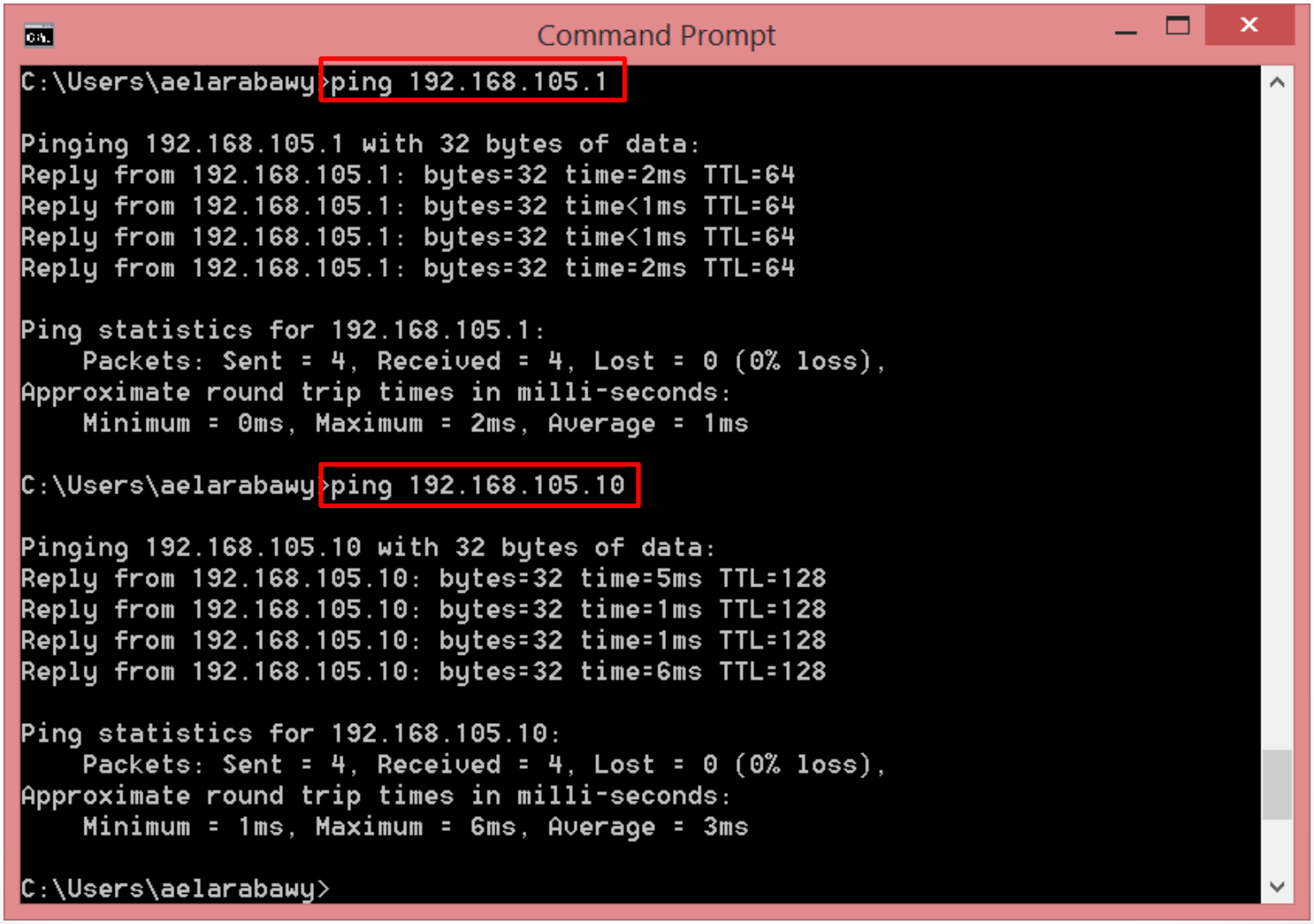
$ sudo /usr/sbin/hostapd /etc/hostapd/hostapd.conf



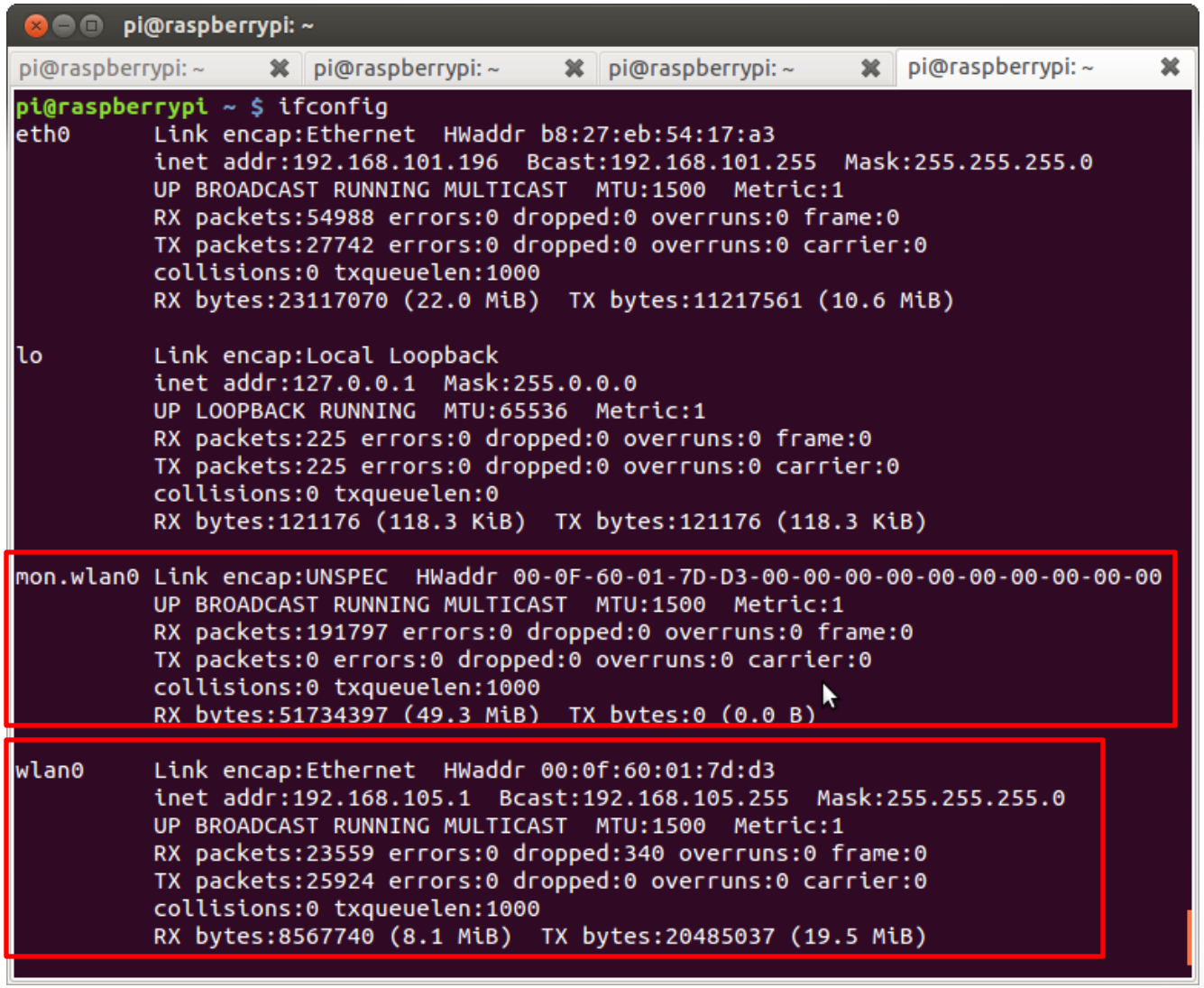


- Now we moved forward a little, and connection is established successfully
- Also the IP Address is allocated within the provided range
- However, we have **limited connection** !!
- The reason for that, is that packets from the station arrives to the Pi, but the Pi fails to pass it outside the WLAN

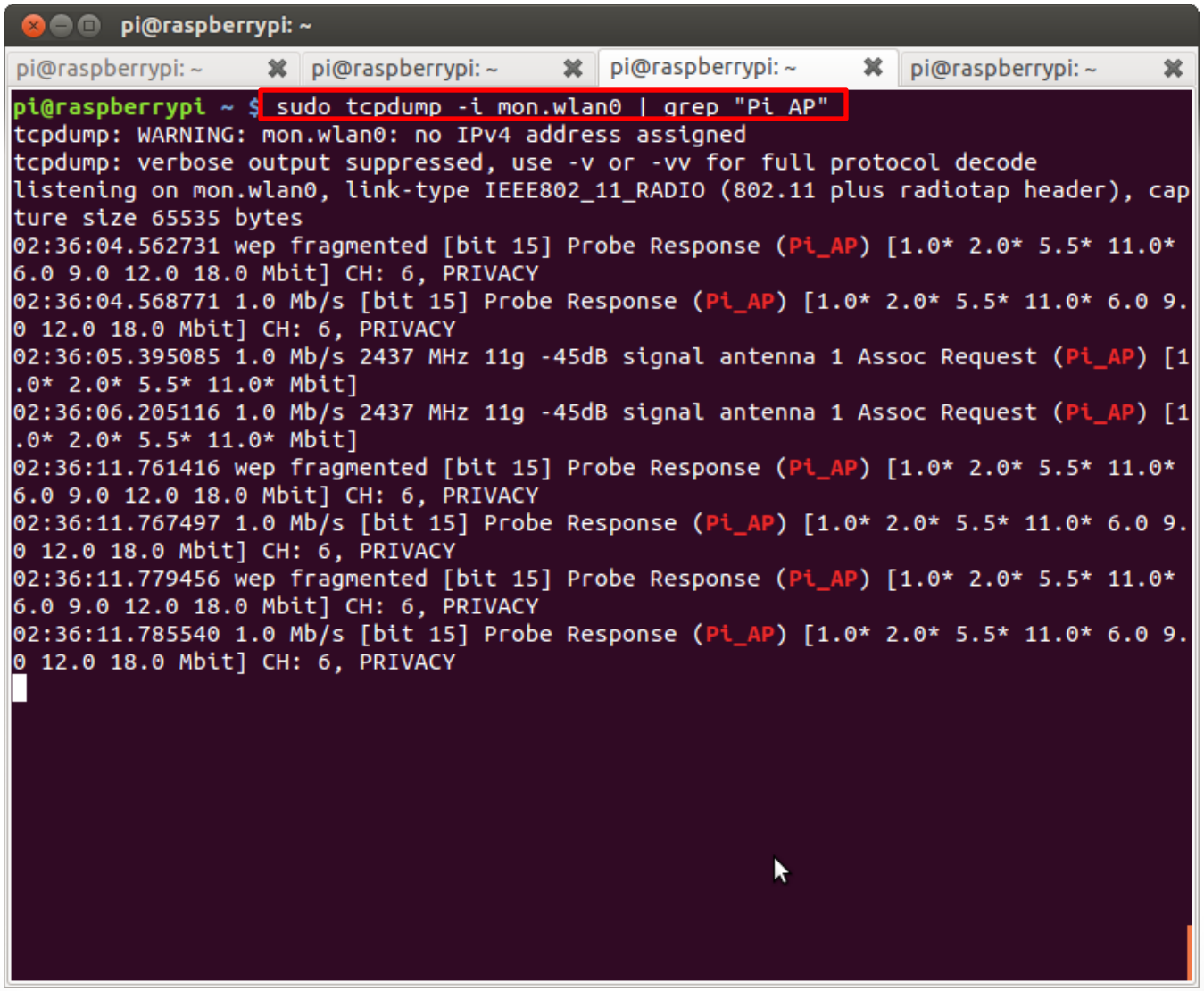
Pinging within the WLAN



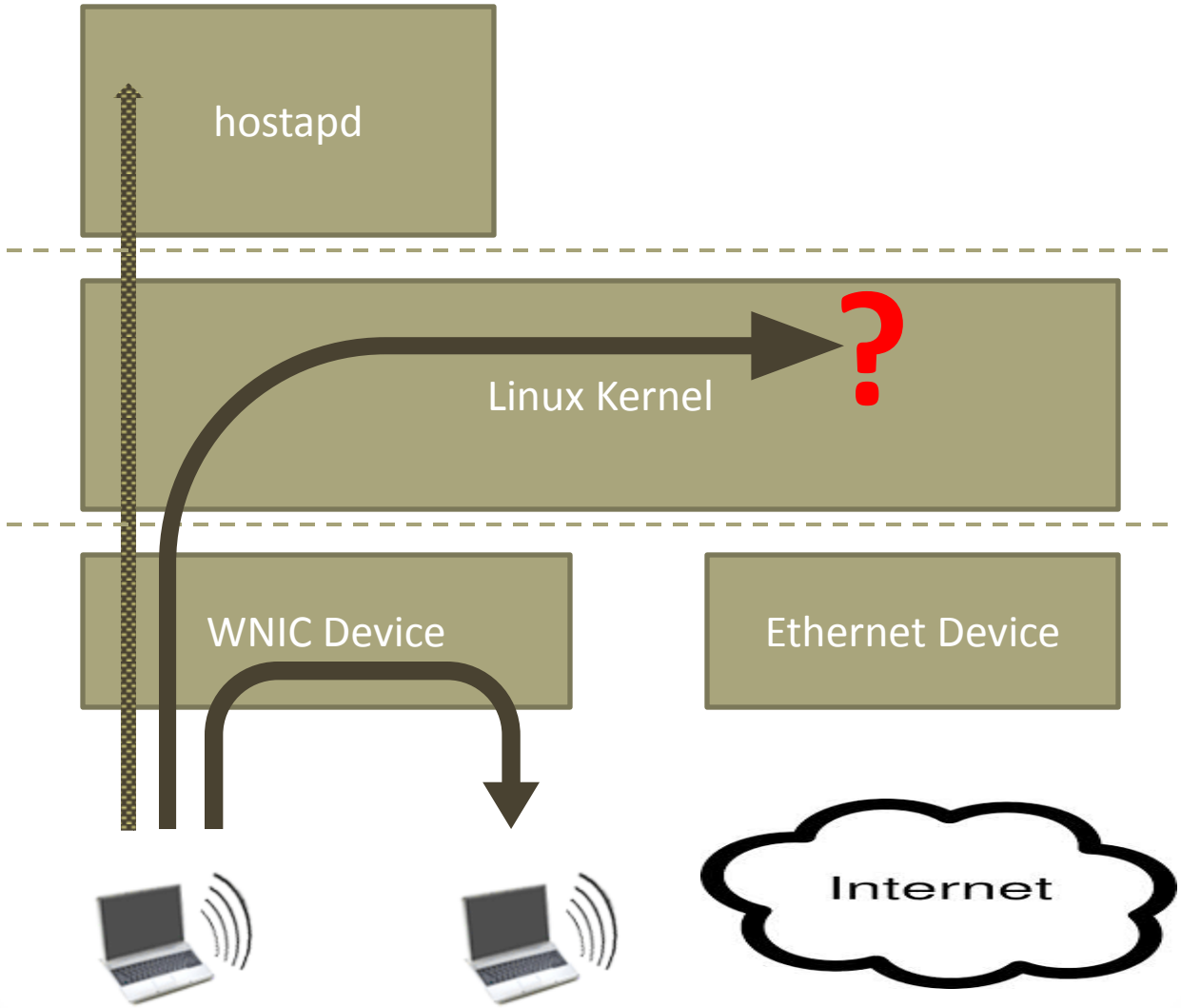
ifconfig


pi@raspberrypi: ~
pi@raspberrypi: ~
pi@raspberrypi: ~
pi@raspberrypi: ~
pi@raspberrypi: ~
pi@raspberrypi ~ $ ifconfig
eth0 Link encap:Ethernet HWaddr b8:27:eb:54:17:a3
inet addr:192.168.101.196 Bcast:192.168.101.255 Mask:255.255.255.0
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:54988 errors:0 dropped:0 overruns:0 frame:0
TX packets:27742 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:23117070 (22.0 MiB) TX bytes:11217561 (10.6 MiB)
lo Link encap:Local Loopback
inet addr:127.0.0.1 Mask:255.0.0.0
UP LOOPBACK RUNNING MTU:65536 Metric:1
RX packets:225 errors:0 dropped:0 overruns:0 frame:0
TX packets:225 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:121176 (118.3 KiB) TX bytes:121176 (118.3 KiB)
mon.wlan0 Link encap:UNSPEC HWaddr 00-0F-60-01-7D-D3-00-00-00-00-00-00-00-00-00-00
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:191797 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:51734397 (49.3 MiB) TX bytes:0 (0.0 B)
wlan0 Link encap:Ethernet HWaddr 00:0f:60:01:7d:d3
inet addr:192.168.105.1 Bcast:192.168.105.255 Mask:255.255.255.0
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:23559 errors:0 dropped:340 overruns:0 frame:0
TX packets:25924 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:8567740 (8.1 MiB) TX bytes:20485037 (19.5 MiB)

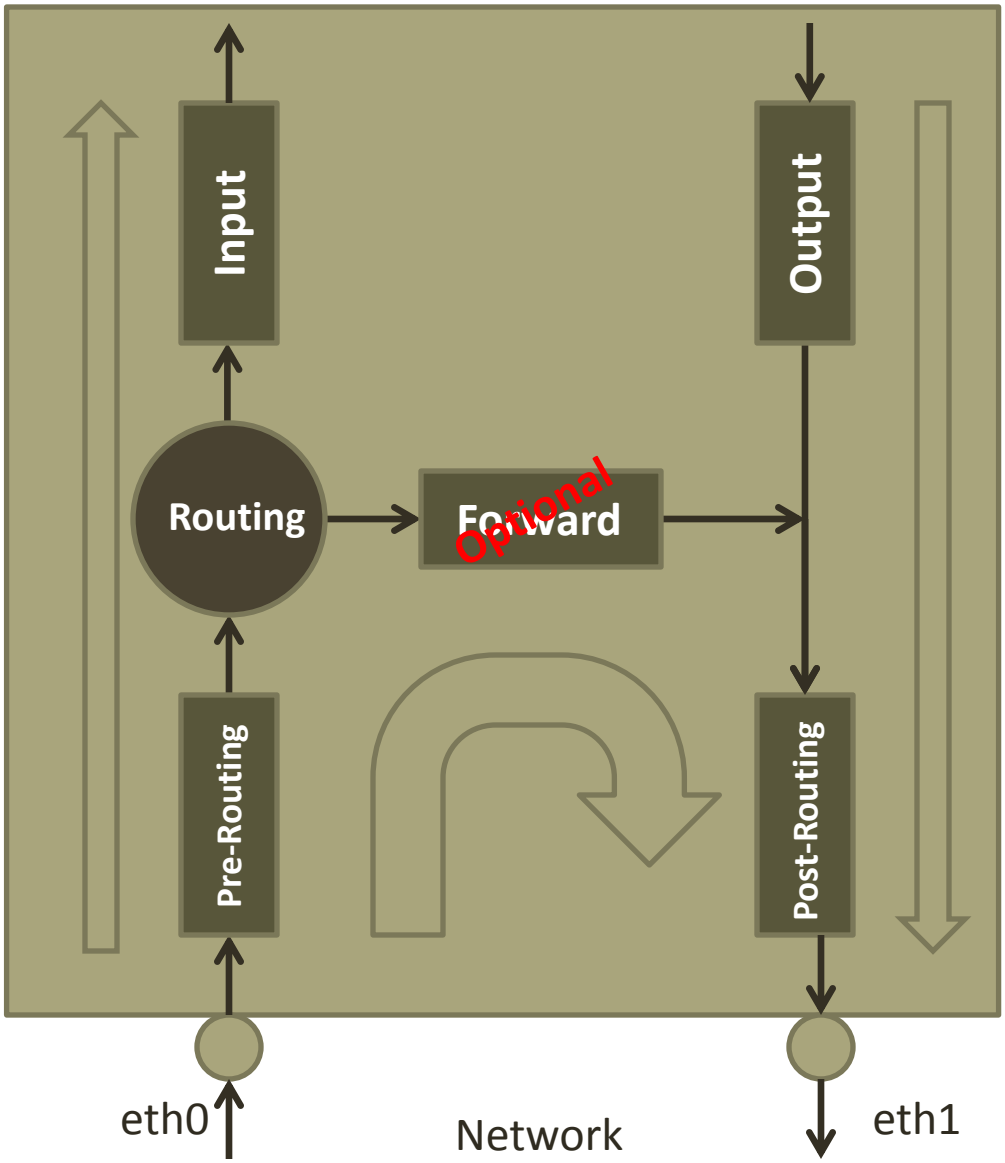
mon.wlan0 Interface



Current Status



Routing In Linux



Packet Forwarding

- Linux supports packet forwarding as part of its routing capabilities
- However, this support needs to be enabled (it is disabled by default)
- To check if packet forwarding is enabled,

  $ cat /proc/sys/net/ipv4/ip_forward

  1 means enabled, 0 means disabled
- To enable packet forwarding,

  $ sudo sh -c "echo 1 > /proc/sys/net/ipv4/ip_forward"
- To enable packet forwarding automatically at startup,

  $ sudo vi /etc/sysctl.conf

  Add the following line at the end of the file

  net.ipv4.ip_forward=1

Adding Forwarding Rules

- Now packet forwarding is enabled, we need now to add forwarding rules
- The tool used for that purpose is called ***iptables***
- With ***iptables*** you can add a rule on any leg in the Linux Packet path (called chains)
  - PREROUTING
  - POSTROUTING
  - FORWARD
  - INPUT
  - OUTPUT
- In our case we will use these commands,

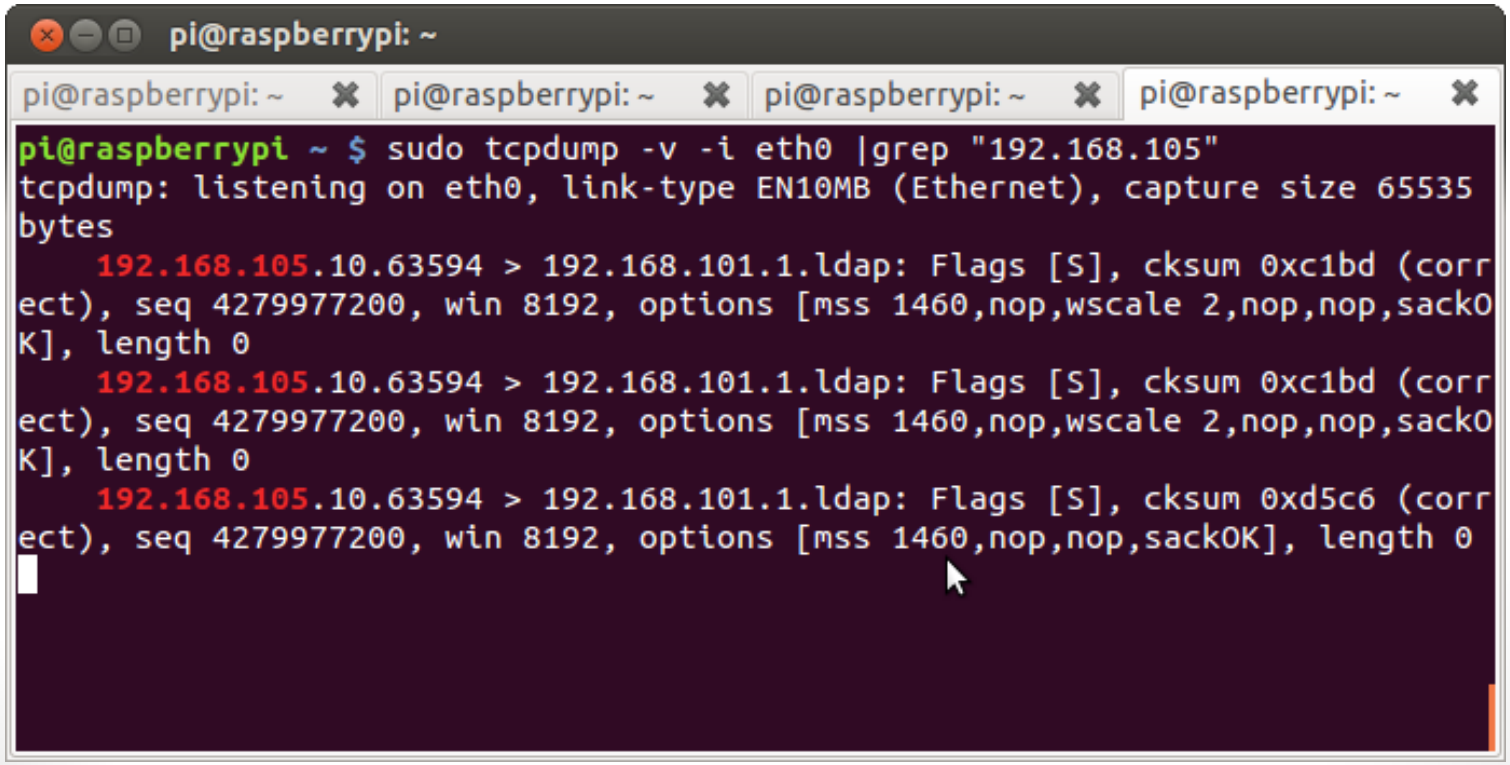
  $ sudo iptables -A FORWARD -i wlan0 -o eth0 -j ACCEPT

  $ sudo iptables -A FORWARD -i eth0 -o wlan0 -m state --state RELATED,ESTABLISHED -j ACCEPT
- To check your changes

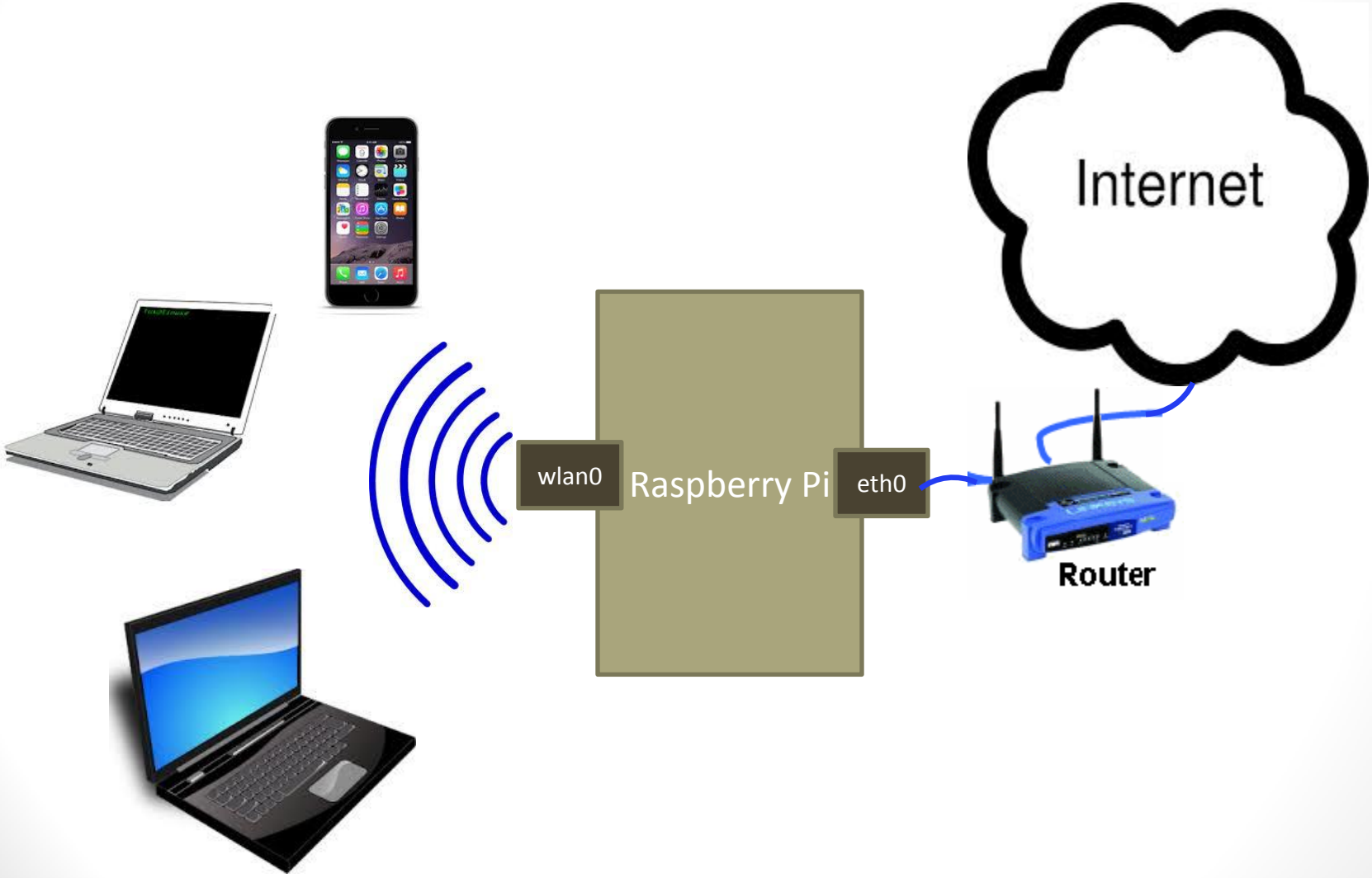
  $ sudo iptables -S

Should It Work Now ??

- No, ….
- We use ***tcpdump tool*** to debug the situation,
- The log shows that packets from wlan0 are forwarded successfully to eth0 and leave the Pi
- **But there are no packets that come back in response**



What is the Problem ??



NAT

- The WLAN subnet is not known to the router
- Accordingly, any traffic from the network towards the WLAN will be discarded by the router
- Solution is to configure the Pi to perform NATing to the WLAN IP Addresses
- So, we need to add a rule, that any traffic going out of the eth0 is NATed to the eth0 IP Address Subnet
- This is done via this command,

  $ sudo iptables -t nat -A POSTROUTING –o eth0 -j MASQUERADE
- To check your changes

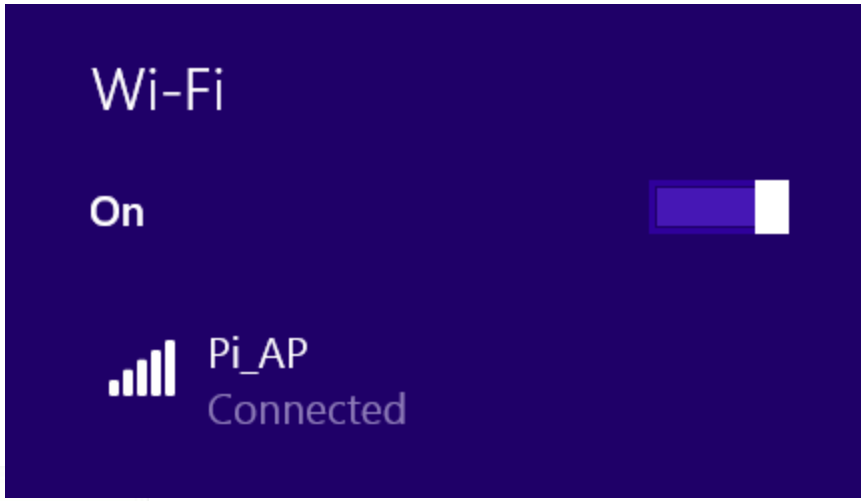
  $ sudo iptables -t nat -S
- This way, traffic from the network destined to the WLAN will be sent to the Pi, which will convert it back to its correct address and forward it to wlan0

Saving iptables Rules

- The rules we set for iptables will take effect immediately, but will be lost when the Pi reboots
- To save these settings,

  $ sudo sh -c "iptables-save > /etc/iptables.ipv4.nat"

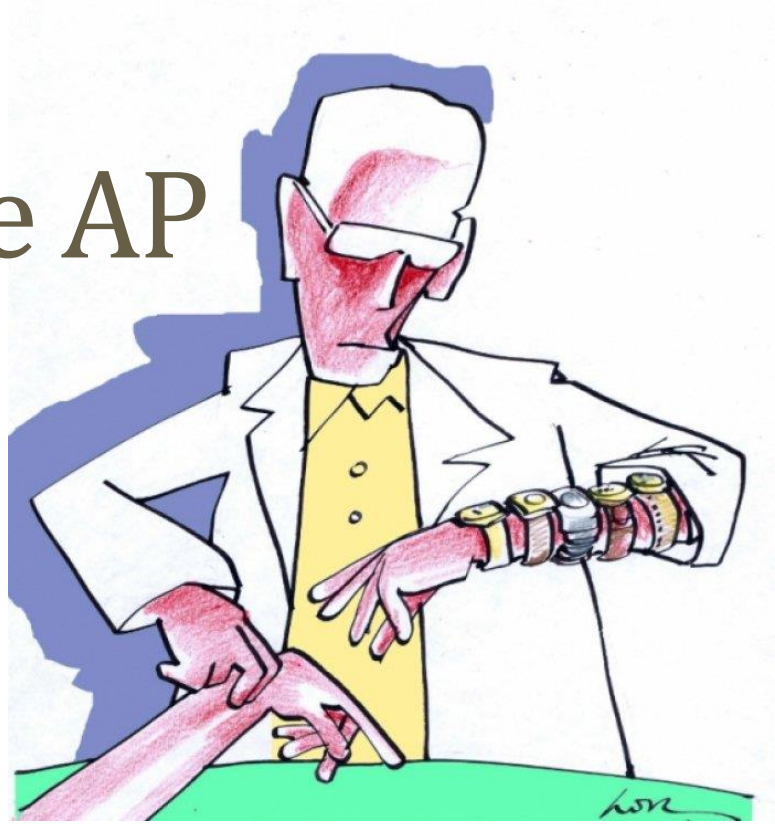
Finally



Notes

- On the bright side,
  - You have now a functioning WiFi AP that you can use for real network connectivity
  - Using the configuration files that we have touched on, you can modify the behavior of your AP
  - You can add new features for your AP by adding more libraries and applications
  - More interestingly, you can also add your own ideas and functionality (remember that all the tools and libraries we used are open source and can accept added functionality)
  - You can build your own embedded device (sensors, motors, …) along with WiFi AP functionality as well
- However,
  - Although the Pi functions as an AP, we need to remember that commercial WiFi APs are based on much more powerful boards (most products in the market use multicore chips with stronger cores than that is used in the Pi)
  - This means, the Pi will not be able to handle very high throughput that can be handled by commercial 802.11n and 802.11ac products

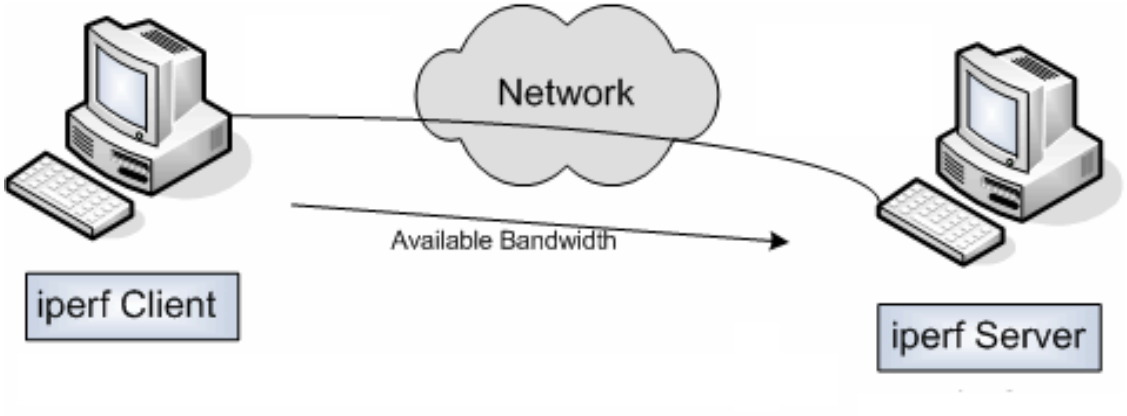
Evaluating the AP Throughput



Evaluating the AP Throughput

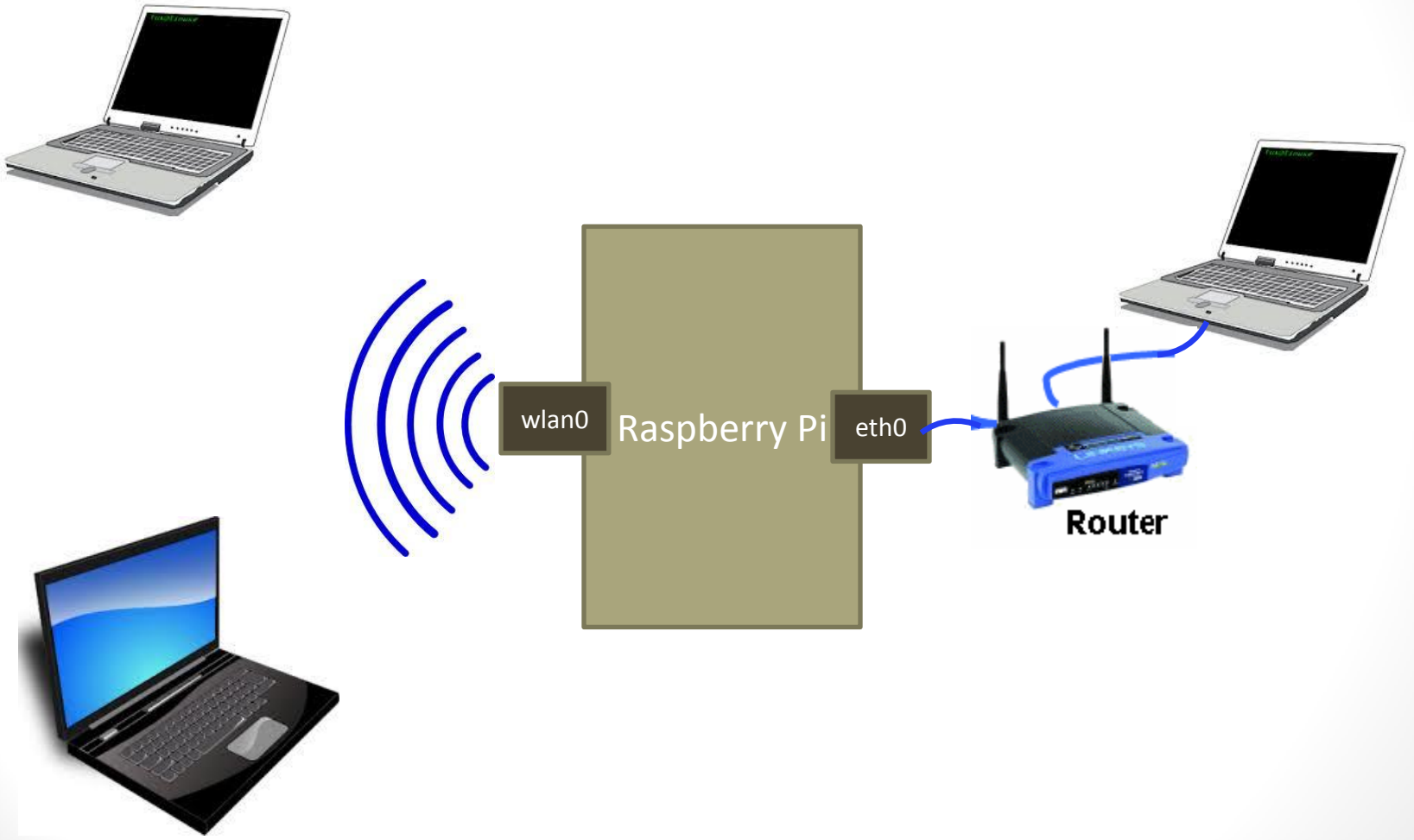
- Now we have the AP operational, we need to measure its performance
- Things to measure:
  - Max bit rate it can handle between stations connected to it
  - Max bit rate it can handle between a station and the internet
  - Delays and latency
- We can measure that using both UDP and TCP traffic
- To perform these measurements, we use the ***iperf tool***

Using Iperf



- ***Iperf*** is a tool to measure the bandwidth and the quality of a network link
- There is a java graphical frontend for it, called, ***Jperf***
- Iperf should be run on two machines, one machine should be running the server side and the other one should be running the client side
- Then the client side will be sending traffic to the server side, which performs different measurements such as,
  - Latency (response time or RTT)
  - Jitter (latency variation)
  - Datagram loss

Using Iperf



Installing Iperf

- There are 2 working versions of Iperf
  - **Iperf v2.x**:
    - The traditional version
    - You can install it using the apt tool,

      $ sudo apt-get install iperf
  - **Iperf v3.x**:
    - A complete rewrite of the tool with improved operation and reduced size
    - Also named Iperf3
    - This version is not backward compatible with Iperf 2.x
    - You can install it as follows,
      - Download the iperf3 code from the website, http://downloads.es.net/pub/iperf/
      - Extract the compressed file you downloaded

        $ tar xzf iperf-3.x.y.tar.gz
      - Install this pre-requisite

        $ sudo apt-get install uuid-dev
      - Build the code

        $ cd iperf-3.x.y

        $ ./configure; make; sudo make install

Running Iperf (iperf Command)

$ iperf (-s|-c) <Options>

- To run the iperf tool, specify if you are running a client or a server, and define the options you need
- To see all the options

  $ iperf -h

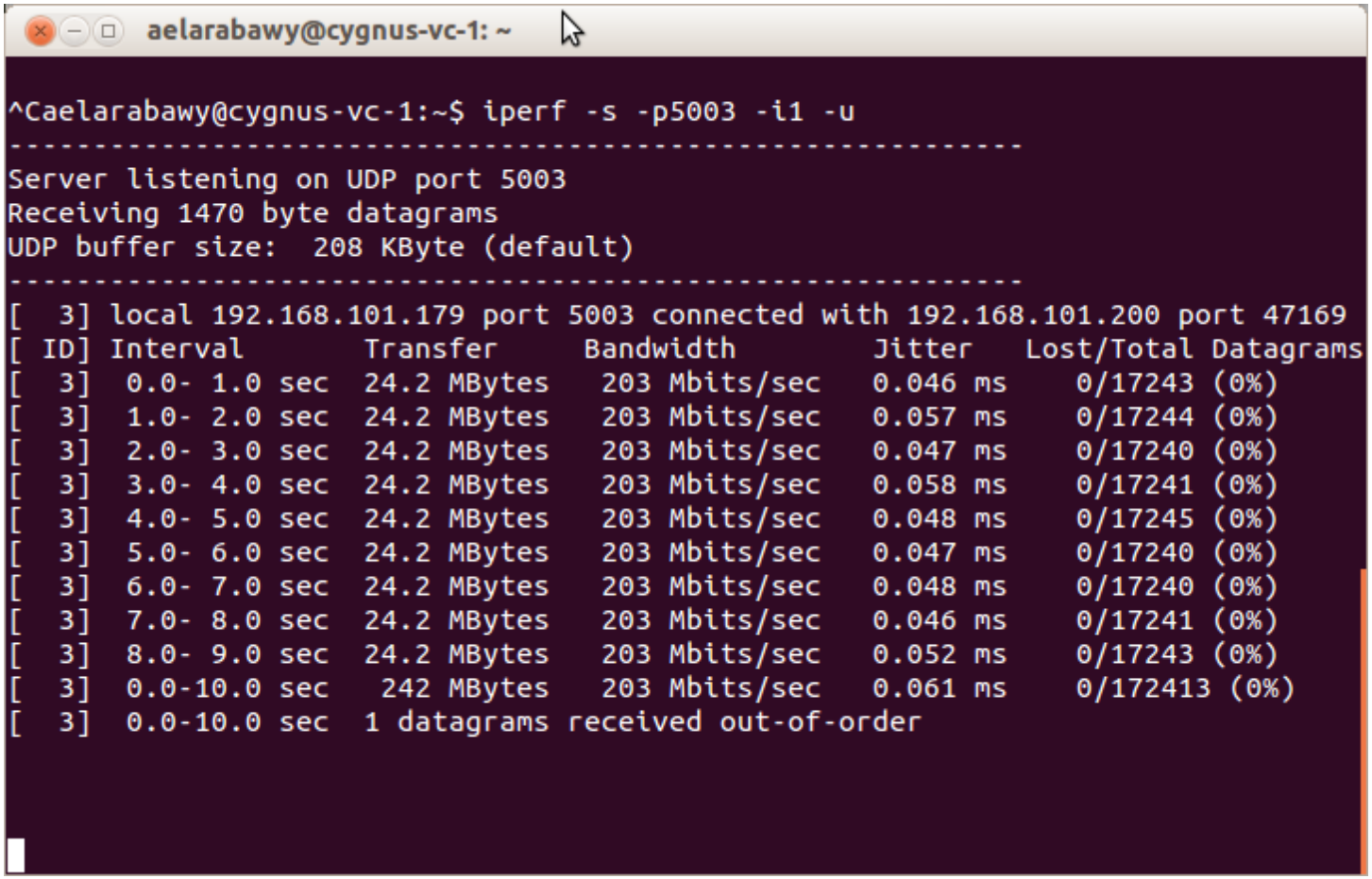
- To start Iperf as a server:

  $ iperf -s -p 5003 -i1 -u

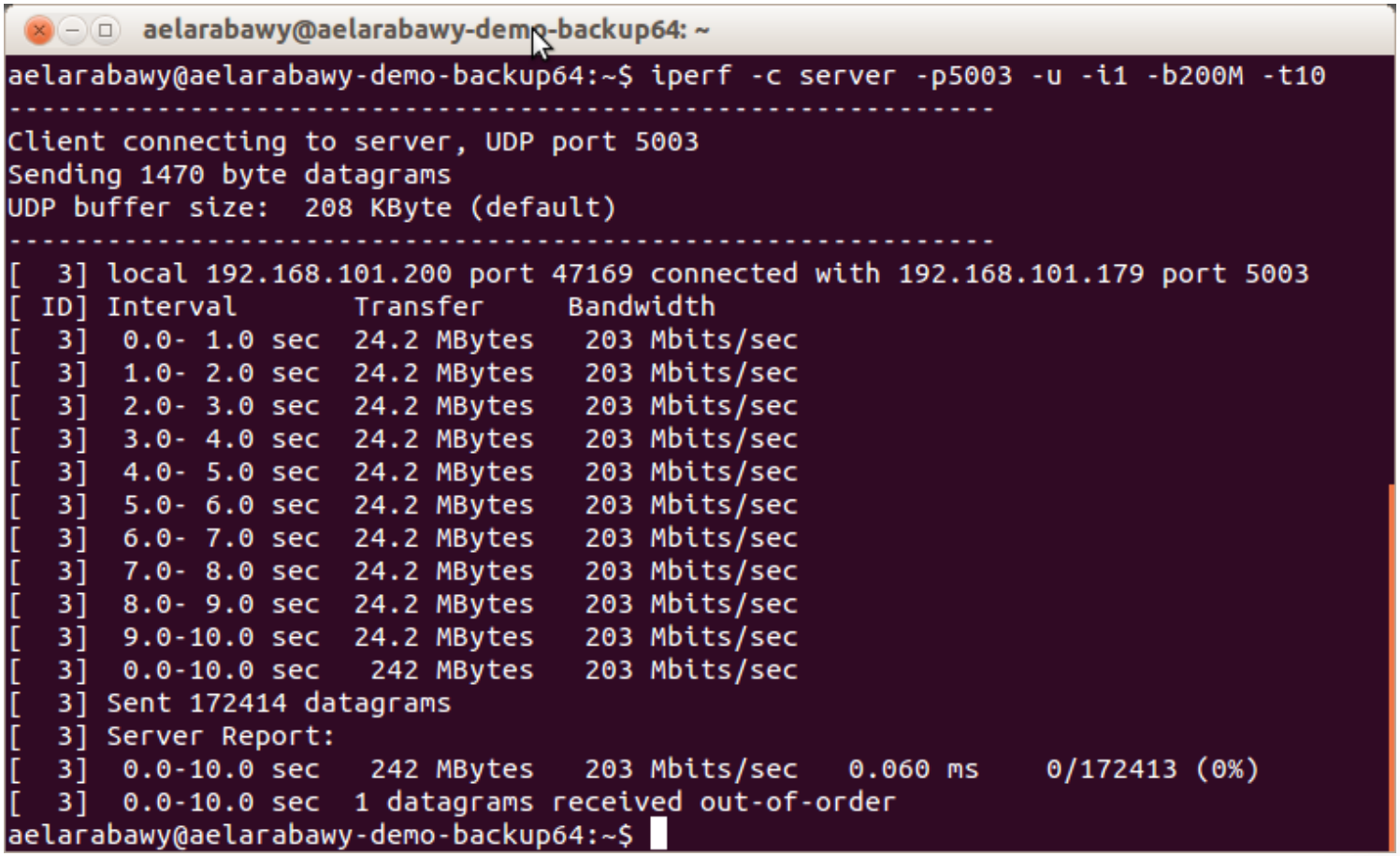
- To start Iperf as a client:

  $ iperf -c 192.168.0.111 -p 5003 -u -i 1 -b 200M -t 100

Starting the Iperf Server



Starting the Iperf Client



Linux 4 Embedded Systems

http://Linux4EmbeddedSystems.com