# Wi-Fi sniffer – functional requirements

## Introduction

The system is designed to enable Wi-Fi sniffing in-order to debug Wi-Fi enabled systems with specific designation for IoT systems.

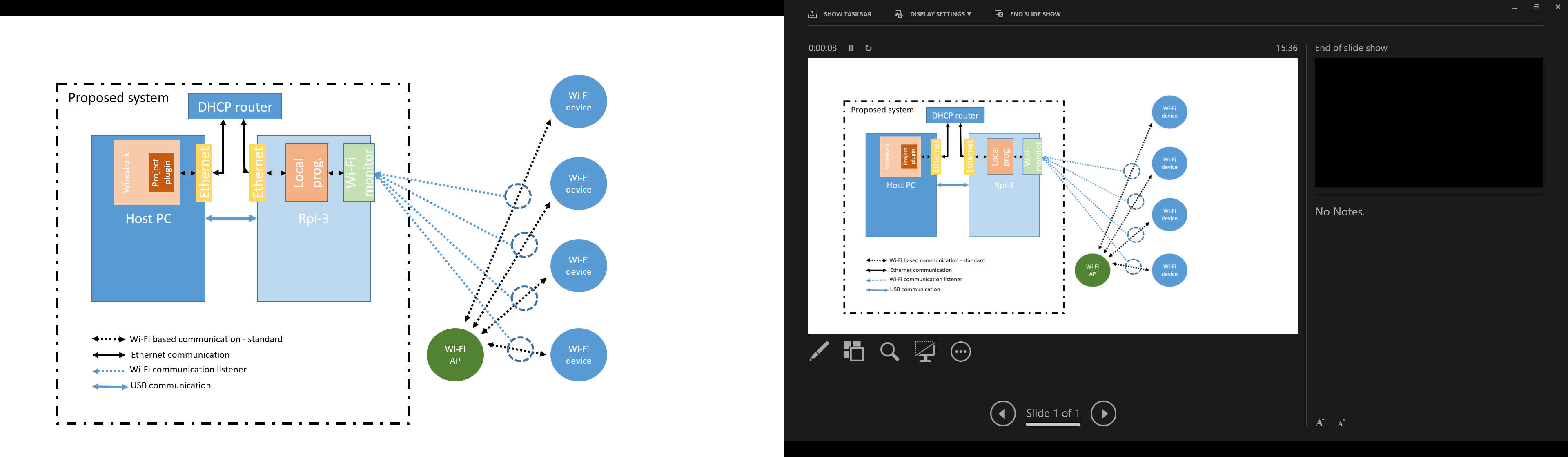
IoT systems usually hold a Wi-Fi enabled devices based on a closed Firmware with a UART (serial port) connection which usually do not provide much control or data to the user regarding the system state.

The main motivation is to allow system-developers to track Wi-Fi communication in order to ease development of Wi-Fi based systems.  
Currently there are available solutions but they are usually very expansive, so the main development achievement is to match performance of existing solutions but at a lower price tag.

Test-case for example:

Assume that one has an IoT based network of Wi-Fi enabled sensors that all connect to the same AP and transmit data to a central location.  
The system should be able to allow the system developer to track all the stations to see if they have finished authentication and registration to the AP. It should be also able to track a single sensor and verify that it is transmitting in the correct interval.  
We would also hope to track the data from the station and verify that the data is formatted according to spec (this requires data decryption).

## System schematic



## Terms and abbreviations

|  |  |
| --- | --- |
| TCP | Transmission Control Protocol – reliable, ordered and error-checked data protocol – meant to transfer critical data |
| UDP | [User Datagram Protocol](https://en.wikipedia.org/wiki/User_Datagram_Protocol)  - meant to transfer data where it is not crucial that the all data will arrive |
| Ethernet | Physical connection between devices – RJ45 connector |
| AP | Access Point – a device which opens a Wi-Fi network and allows devices to connect to it and allow for data transfer to\from the device |
| Wi-Fi device | Endpoint – any device which is Wi-Fi capable |
| Host-PC | The PC that the system will be connected to – a PC with display, keyboard etc. |
| Wireshark | Program – de-facto standard of analysis of network data – open source |
| Raspberry-Pi \ RPi | A System-On-module – basically a low-power low-performance fully-capable PC |
| Socket | Programmatic connection between two SW elements (running on the same or different systems) that allows data transfer |
| TBD | To Be Decided – where shown item still has not been decided |

## Technical requirements

### General requirements

The system shall be able to sniff traffic in the air

1. All packet types shall be captured. I.e. management, control and data
2. The entire message shall be displayed and analyzed including all handlers (specifically 802.11 header)
3. Packets will be recorded even if the device is not a part of the transaction, i.e. messages between an AP and station will also be recorded in both cases:
   1. Sniffer is connected to the network
   2. Sniffer is not connected to any AP

### SW elements - Host PC

The current idea for implementation is for the Host-PC to simply run Wireshark as the main-SW element, due note that there should be a need to implement a plugin for Wireshark  
this solution will enable to maximize predeveloped SW features and shorten development times

System configuration – since the system configuration should be done by the host-PC there is a need to enable some form of configuration scheme, its endpoint will be a configuration file that is passed to the Raspberry Pi

Wireshark enables among other features

1. Coloring of messages – due to type
2. Filtering of messages according to process and MAC address
3. Wireshark should also provide data decryption – up to review

### SW elements - Raspberry PI - Local program

1. read configuration setup – from file transferred to device from host-PC
2. Wi-Fi interface configuration – based on the configuration setup from the Host PC control the interface – set the interface mode, band, channel etc.  
   State of the interface will be restored upon execution end
3. Listen to data over the air, encapsulate and transfer via socket to host PC

### Configuration

1. The Host-PC side will be able to configure mode, band and channel to listen on – transfer to RPi
   1. Hi Prio - single channel listening
   2. Low Prio – scan on more than one channel on a band
2. The host-PC will be able to configure filters on the data (by MAC, by process TBD)
3. The Host-PC will be able to configure test time as a parameter to limit the listening time
4. The RPi – will be able to inform Host-PC of multiple available interfaces to work with  
   If only one exist it will be selected automatically without user-intervention

Configuration designated to the RPi will be packed to a file format and passed to the RPi via file-transfer protocol

RPi will read configuration file upon activation request

### Control\execution

The Host-PC will pass control\execution commands via dedicated TCP socket

### Data

1. The RPi will transfer data to the Host-PC via dedicated UDP socket
2. The Host-PC will display the data filtered and colored according to the configuration

### Interfaces - SW

According to the requirements above 3 main interfaces between Host-PC and RPi can be inferred

1. File-transfer – for configuration file and setup – FTP \ SCP TBD
2. TCP Socket – command transfer – start\stop\pause\channel switch etc.
3. UDP Socket – data from Wi-Fi through RPi 🡪 Host-PC

### Interfaces - HW

The preferred connection between Host-PC <--> RPi should be USB based

There is an option to transmit the data over Ethernet – direct connection between host and device is not preferred. I.e. device should connect to the network and make itself discoverable to the Host (research Avahi)

## Implementation directives

1. Compilation – the implementation will have a single source of compilation. i.e. the compile command will be a single line and will produce a single runnable binary (auxiliary files are allowed)
2. Documentation
   1. the source code will be well documented - functions
      1. All functions shall have a short description of its designation
      2. After the designation there shall be a list of all input parameters with a short description of the parameter including type, expected values and use-description
      3. Each function will have an output description – which includes type, expected values and description
      4. Error values – if the function can return error values there shall be a list of possible error values with a short description of the reason the error value is returned
   2. the source code will be well documented – variables
      1. all variables names and function names shall follow a set convention – said convention is up to the implementer, but I must be documented
   3. code readability – the use of enums and defines is encouraged to ease code review process – there shall be minimal to no hardcoded number use in the code
   4. Code readability – critical implementation points in the code shall include a short description it the code block itself (all comments in the code shall be in the form of multi-line comment – the use of single line comment is discouraged (i.e. use ‘/\*..\*/’ and not ‘//’ – in C)
   5. the implementation shall include a WORD document that includes
      1. function description – similar to that in the code
      2. the code convention used in the implementation
      3. critical flow description and schematics where possible
      4. any other restriction or information that a community user\developer will need in-order to use the code