# **Introduction:**

In today’s wireless age, understanding how Wi-Fi networks operate at the packet level is vital. This project focuses on building a **Wi-Fi packet sniffer** using the **ESP8266 microcontroller**, which captures network packets in promiscuous mode and transmits the data to a **Python-based analysis system** via UDP. The project provides insight into how wireless devices can monitor signal strength and track nearby device activity without connecting to the Wi-Fi network.

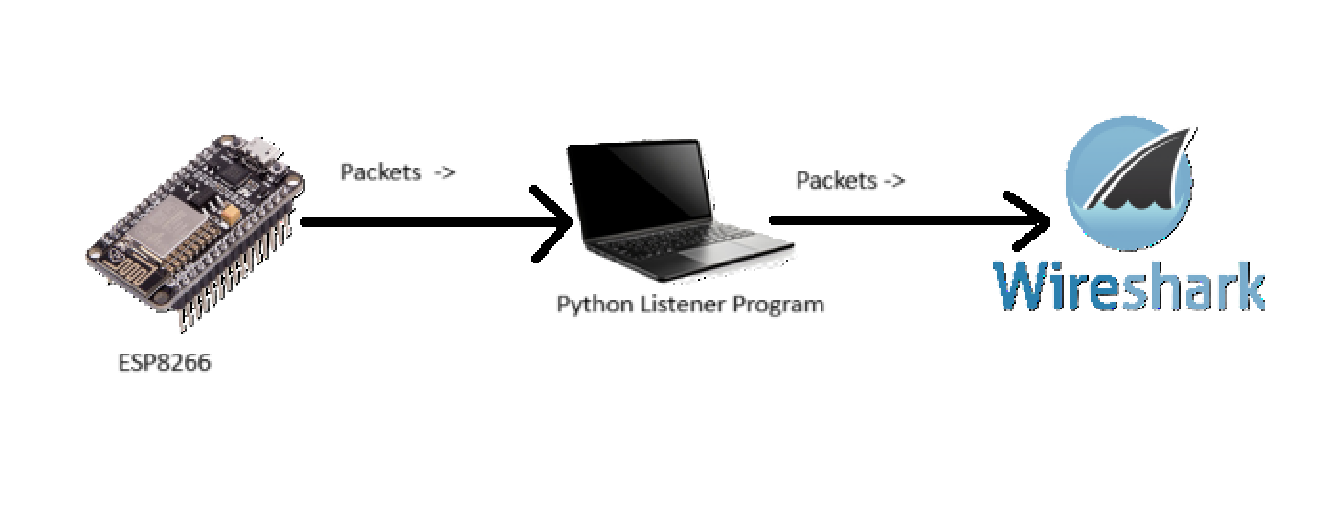
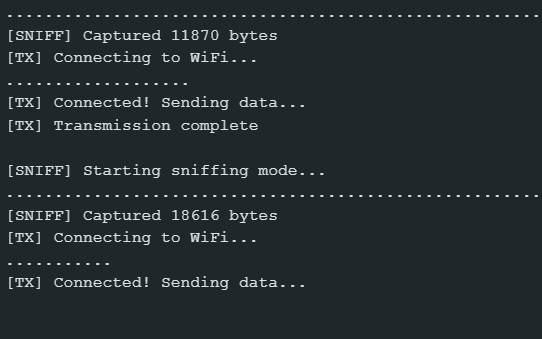


Figure : Block diagram

# **Workflow:**

The project operates in a **dual-mode loop**, with each cycle consisting of:

* **Sniffing Mode (10 seconds):**  
  The ESP8266 enters monitor mode and captures raw Wi-Fi packets on a specified channel. For each packet, it extracts information such as packet length, RSSI (signal strength), and timestamp.
* **Transmission Mode (5 seconds):**  
  After sniffing, the ESP8266 connects to a pre-configured Wi-Fi hotspot and transmits the captured data via UDP to a Python program running on a laptop.

**On the laptop side:**

* The **Python program** listens for incoming UDP packets.
* Data is **stored in a CSV file**, parsed in real-time, and **plotted as an RSSI graph**.

Figure : Capture and Transmission Cycle of Esp

* Additionally, the system tracks when a device was last seen and performs simple **analysis** (e.g., average RSSI, inactive devices).

# **Summary of Work Done:**

We divided our project into two main components—**the ESP8266 firmware** and **a Python-based desktop application**—which worked together to implement Wi-Fi packet sniffing, data transmission, analysis, and visualization. Here's a breakdown of what we accomplished:

* **ESP8266 Firmware Development:**
  + Configured the ESP8266 to operate in **promiscuous mode**, enabling it to capture all Wi-Fi packets on a specific channel, not just those addressed to the device.
  + Extracted relevant metadata from each packet such as **timestamp, packet length, Wi-Fi channel, and RSSI (signal strength)**.
  + Buffered the captured data for a fixed duration (sniffing mode) before switching to transmission mode.
  + Implemented logic to **connect to a hotspot** and send the buffered data to the laptop via **UDP packets**.
* **Python Program on Laptop:**
  + Developed a **UDP server** that listens for incoming data packets from the ESP8266 and decodes the received payload.
  + Parsed each packet and saved the data into a **CSV file** in real-time, ensuring persistence and easy access for future analysis.
  + Maintained a **list of active devices** by tracking their last seen timestamps based on incoming data.
  + Added a feature to **analyze missing devices** (those not seen in the last 60 seconds), simulating a basic tracking mechanism.
  + Enabled **RSSI-based trend visualization** using Matplotlib to get insights on signal fluctuations over time.
* **Output Logging & Visualization:**
  + Logged metadata such as **time of capture, RSSI values, and packet sizes**, storing them for later analysis.
  + Generated real-time **RSSI vs. time graphs**, helping us visualize the Wi-Fi signal environment.
  + Displayed **device activity status** and **calculated the average signal strength**, aiding in understanding network stability and proximity of devices.

This comprehensive setup allowed us to continuously cycle between sniffing Wi-Fi packets and transmitting them for further analysis, effectively turning the ESP8266 into a lightweight, real-time Wi-Fi environment monitor.

# **Analysis:**

Through this project, we were able to:

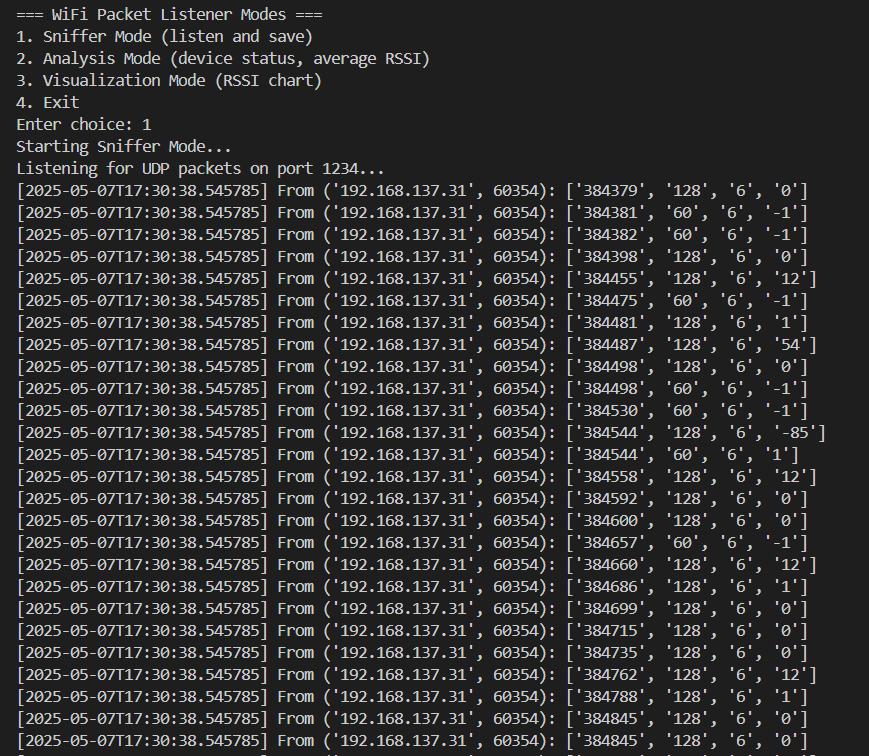
* **Observe real-world network traffic** without connecting to access points.
* Monitor the **signal strength (RSSI)** over time, which reflects the distance and environment between the sniffer and other devices.

Figure : Captured packets via UDP

* Identify devices that **have gone inactive** using last-seen timestamps and a timeout threshold.
* Maintain efficient **low-data transmission** using UDP, reducing latency and complexity.

This project not only demonstrates the fundamentals of packet sniffing and wireless communication but also integrates embedded systems with network programming and data visualization.

# **Input and Output:**

## **Input**

The "input" in this project comes from the **wireless environment**. More specifically:

* The ESP8266 **passively receives Wi-Fi packets** from any nearby devices (phones, routers, laptops, etc.).
* It does **not connect to any of these devices**, but instead reads metadata such as:
  + **Timestamp** (when the packet was seen)
  + **Packet length**
  + **Wi-Fi channel** (hardcoded to 6 in this project)
  + **RSSI (Received Signal Strength Indicator)** – an indicator of how strong the signal is from that device

This data is captured in real-time from all devices broadcasting on the monitored channel.

## 

## **Output**

The outputs are **delivered through the Python interface on the laptop**, in several forms:

* **Raw Output**:  
  Captured data is saved into a .csv file with the following fields:
  + Timestamp
  + Millisecond
  + Packet Length
  + Channel
  + RSSI

Figure : CSV file

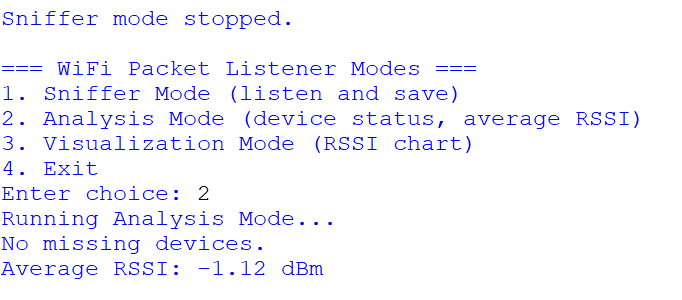
* **Analytical Output**:
  + Shows which devices were active recently.
  + Detects devices that have gone silent (not seen in the last 60 seconds).
  + Displays the **average RSSI**, which gives insight into signal strength quality.
* **Visual Output**:
  + A **line plot of RSSI values over time**, showing the strength of received signals.
  + This graph helps visualize signal fluctuations, interference, or range issues.

Figure : Various Modes of Python Listener

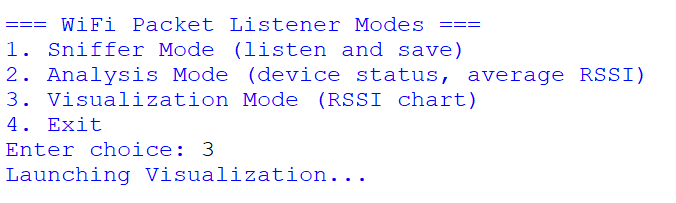


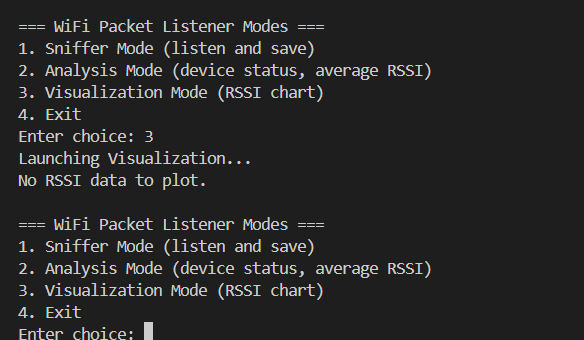
Figure : Various Modes of Python Listener

# **Anlysis:**

This project effectively **demonstrates the use of low-cost IoT hardware for wireless packet analysis**. The combination of **ESP8266 and Python** makes it both scalable and customizable.

* The **sniffer is lightweight** and doesn’t require a full Linux system like traditional sniffers.
* It is useful in scenarios like:
  + Checking Wi-Fi signal coverage in a room
  + Understanding network traffic density
  + Detecting presence or absence of known devices (e.g., for IoT security)

However, there are also limitations:

* It captures only **metadata**, not full packet contents.
* It is **restricted to one Wi-Fi channel** at a time.
* It may miss some packets during transmission mode.

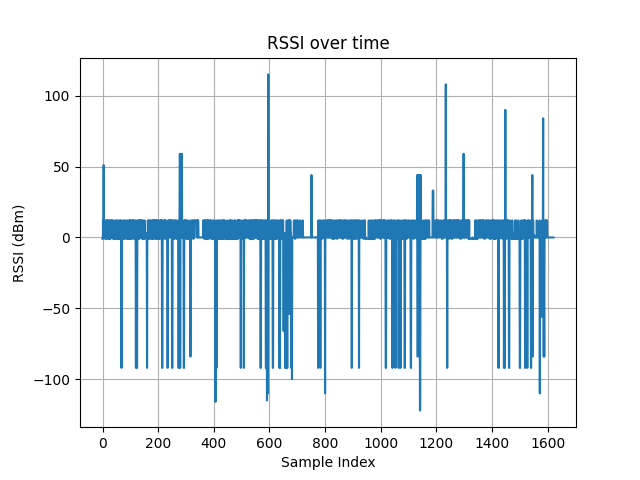
Despite these, it offers a solid foundation for further development in wireless monitoring and IoT-based network tools.

Figure : Visualization Mode output

# **Additional Enhancements:**

To make this project more comprehensive, we can implement the following:

* Support for **changing Wi-Fi channels** to detect more devices.
* Calculating **device density** or number of unique devices seen per cycle.
* Extending to support **TCP transmission** or **data encryption**.
* Implementing **auto-sync with cloud storage** for collected data.

# **Conclusion:**

This project successfully integrates embedded firmware design with Python-based data processing to create a lightweight and extensible Wi-Fi packet sniffer. It can be adapted for more advanced use-cases like security monitoring, network health analysis, or even signal mapping.