

# IOT Based Wireless Sensor Network with Linux Based Server

Sahil Goti<sup>1</sup>, Jeel Ramani<sup>2</sup>, Drashti Domadiya<sup>3</sup>, Shruti Savani<sup>4</sup>, Pritesh Saxena<sup>5</sup>, Chintan Panchal<sup>6</sup>

*Department of Electronics and Communication Engineering, Sarvajani College of Engineering and Technology  
Surat, India*

**Abstract - This paper presents the design along with the implementation of an IoT-based Wireless Sensor Network (WSN) that uses ESP32 as well as ESP8266 microcontrollers as sensor nodes with a Raspberry Pi as a centralized Linux-based server. The sensor nodes come equipped with environmental sensors that include temperature, humidity, gas, and also light detectors. Because of protocols that are like UDP or TCP, these nodes transmit data in real time to the server wirelessly. The Raspberry Pi processes data, stores data, also visualizes data because it acts as a local fog computing layer that closes the divide between edge sensing and user access. Flask creates the web interface that the server hosts while it handles many clients. This allows for real-time data to be visualized and monitored. The architecture ensures some scalability and energy efficiency as well as low-latency communication and that makes the architecture quite suitable for some applications like smart agriculture or environmental monitoring as well as health monitoring plus home automation. The system is modular and extensible in nature, which makes it easy to integrate additional nodes, connect to the cloud, and improve future features like encrypted communication and edge-based intelligence. Because of how the results acquire more data and can service multiple clients quite reliably, they can validate the idea that a Linux-based WSN model works in practical IoT deployments.**

**Keywords—ESP32, ESP8266, Raspberry Pi, Flask, Wireless Sensor Network, IoT, Fog Computing, Sensor Nodes, Real-Time Monitoring, UDP, TCP.**

## I. Introduction

Today, the Internet of Things (IoT) is virtually limitless, from smartwatches to great and all-encompassing smart cities. At its simplest, IoT is about devices communicating with each other and transmitting data without the need for human

intervention. One of its coolest and most valuable components is called a Wireless Sensor Network (WSN). A WSN is made up of tiny devices known as sensor nodes. These sensor nodes collect data from the environment and transmit that data to a central location for processing. IoT can provide significant utility through WSN in similar ways that automated farms, pollution monitoring, smart traffic systems, and hospital monitoring utilize WSN systems.

However, developing a WSN that actually works in practice is no easy task. Considerations must be made about how little power the sensor nodes can utilize (because they are often powered by batteries), delays in time for the data to be transmitted wirelessly, accommodating multiple users, and still permissively handle different types of data. Balancing all of this is no easy task.

This study presents an IoT-oriented WSN framework, consisting of inexpensive, low-power sensor nodes (ESP32 and ESP8266) based on microcontrollers, and a Linux-centric centralized server based on Raspberry Pi. The sensor nodes are configured to collect parameters such as temperature, humidity, gas levels, and ambient light through different sensor modules, to send the data wirelessly to Raspberry Pi, which acts like a fog computing unit for data processing, data logging, and user interface to monitor data collection and parameters.

The Linux-based server, which offers stability, flexibility, and capability to support a multitude of open-source applications and protocols and capable of handling multiple clients while locally storing data even in a variety of formats (CSV or JSON), in addition to a graphical interface to report real time state of the system utilizing a web interface powered by Flask.

The addition of a local processing layer minimizes dependence on cloud services and allows for

increased reliability and awareness of privacy that would be critical in sensitive or critical applications.

This paper outlines the hardware and software architecture, communication protocols, system integration, and real time performance results of the outlined architecture. The envisioned goal is to leverage the components described to generate a robust WSN framework as a scalable, low-cost, low-power, and flexible WSN solution that can easily adapt to a wide array of IoT applications.

## Ii. Technical Background

Wireless Sensor Networks (WSNs) arose from the idea of distributed sensing technology wherein geographically separated and autonomous devices (sensor nodes) monitor and record physical or environmental conditions, and work collaboratively to communicate that data to a singular location or processing unit. WSNs provide the foundation for the Internet of Things (IoT) ecosystem by supporting real time monitoring and shortly enabling automation in applications such as smart cities, agriculture, health monitoring, and industrial automation [1].

### 2.1 Microcontroller-Based Sensor Nodes

The ESP8266 and ESP32 microcontrollers from Espressif Systems are ultra-low cost, ultra-low power microcontrollers with built-in Wi-Fi that are easy to deploy as sensor nodes. The ESP32 has greater computing bandwidth because it has a dual-core processing engine while also having built-in Bluetooth, making it more capable to complete more advanced and complex edge processing tasks [2]. The ESP8266 and ESP32 microcontrollers also have the flexibility to use many different sensor interfaces to connect any number of different sensors; their integrated interfaces (I2C, SPI, ADC) were used in this project to connect sensors that include DHT11/22 (temperature and humidity measurement), MQ-series gas (detection), and LDR (Light Dependent Resistor) modules.

### 2.2 Communication Protocols

Data traveling from the sensor nodes to the central server occurs through a User Datagram Protocol (UDP) or Transmission Control Protocol (TCP). UDP is utilized where low-latency, real-time data transfer with little overhead is present and TCP is used whenever reliability is necessary [3]. Both UDP and TCP utilize Wi-Fi connections, facilitating

a flexible, and infrastructure-less communication topology over a local area network (LAN).

### 2.3 Linux-Based Server with Fog Computing Function

The Raspberry Pi, a small form-factor, low-power, ARM-based single-board computer using a Linux-based operating system (the most common operating system is Raspbian or Raspberry Pi OS), will act as a fog computing node in the setup. Fog computing is a bridge between cloud and data processing at the edge, which enables cloud storage, data processing, and analytics at the source of event data collection. This reduces the latency and reliance on cloud computing capabilities [4]. The fog computing architecture allows for edge intelligence, local data persistence, and privacy-aware applications, which improve the reliability of the system.

### 2.4 Web Interface

The Raspberry Pi runs Flask, a Python-based micro web framework to create a lightweight but powerful web interface. Flask supports multiple HTTP clients at the same time, as well as real-time data visualization. Plotting sensor data can be done through the web browser, and the data can be saved as CSV or JSON files. The data can also be shared through RESTful endpoints or a web dashboard.

### 2.5 Modularity, scalability, and extensibility

The modular architecture makes it easy to add additional sensor nodes without having to reconfigure the main system. In addition, since sensors, services, and data formats are standardized, they can be exchanged easily, making it interoperable. The architecture is scalable by supporting a hierarchical node structure, and potentially integrate with the cloud. Extensibility of the architecture is supported as new sensors or encrypted communication modules can be interfaced as plug-and-play.

## III. Wireless Sensor Network Architecture and Methodology

In an IoT-based wireless sensor network (WSN) comprising multiple sensor node (also known as edge nodes) which are deployed over some area to monitor either environmental or application-specific parameters. These nodes often have limited resources and communicate via radio or wireless communication to transmit the sensed data to a

central node (the main node) which sends the data to a Linux-based server running the processing, storage, or visualization.

### 3.1 Data Transmission Methodology

In this proposed architecture, data from the edge nodes is sent back to the main node following the shortest path first. In cases where an edge node is not in range of the main node, it first must determine its nearest neighbouring node (any node to its cluster might feasibly be the neighbouring node), and send the data there. The receiving node will send the data on to the next node in accordance with the protocol until the data finally reaches the main node. The main node will then forward the received data to the server. In addition to the edge nodes sending data back to the main node, the server also has the capability to send data requests to any node on the network, which can then also trigger data requests to be routed back to the server based on the same shortest-path routing method.

For the complete assurance that seamless data transfers are maintained also with a dynamic situation concerning nodes being disconnected etc, the routing path is not determined statically. The network uses a dynamic routing algorithm, where updated routing tables are created and propagated when the network topology changes, keeping routing adaptive and efficient, as well as the robustness and resiliency of the system.

### 3.2 Key Techniques Implemented

#### 3.2.1 Nearest Neighbour Discovery

Description: Edge nodes dynamically perceive and communicate with the nearest neighbouring node that is in communication range. This is important for multi-hop communication when a route to the main node does not exist.

Implementation: Each node broadcasts a "hello" message periodically, and maintains a current neighbour table in order to be aware of their local topology and use that information in deciding which node is the closest and which link has the highest quality to forward packets to. This approach aligns with the neighbour discovery methods described in [5], which emphasize periodic beaconing and maintenance of neighbour tables to support low-power, low-latency WSN communication.

#### 3.2.2 Dynamic Routing Table Updates

Description: The routing table in each node is dynamically updated in real-time to reflect changes to network topology, such as when a node becomes unavailable or a new node join.

Implementation: The network uses a protocol such as Ad hoc On-Demand Distance Vector (AODV) or one of the other reactive protocols that invoke route discovery only when necessary. When a topology change is detected by a node, the node informs the cluster and, if the cluster must update its routing table, it sends its updated table to all nodes. The use of reactive routing protocols such as AODV is supported by recent studies [6], which show that on-demand route discovery and real-time table updates improve performance in dynamic WSN environments.

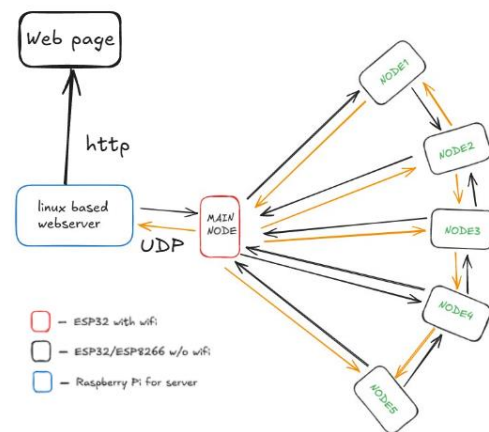


Figure 1. Architecture of WSN

Figure 1 portrays the multi-level structure of the Wireless Sensor Network (WSN) proposed in this paper that combines different layers of sensing, communication, processing, and visualization in order to allow for effective real-time monitoring in IoT applications. The first level is composed of the edge nodes, implemented on either ESP32 or ESP8266 microcontrollers. These low power Wireless Fidelity (Wi-Fi) devices are compatible with various environmental, and biomedical sensors such as, temperature/humidity sensors (DHT11), body temperature sensors (LM35), pulse rate measurement & SpO<sub>2</sub> sensors (MAX30100), and accelerometers (MPU6050) for fall detection. Each edge node can independently monitor each sensor and collect data at a specified time interval, as well as pre-process data, such as filtering out noise, or averaging, and then wirelessly communicating this data to the main nodes. When either direct connection to the central server is not possible due to distance, or there is material blocking signal transmission, the architecture includes main nodes,

usually other ESP32, that act as intermediate data aggregators. Main nodes collect data from edge nodes in the vicinity, and communicate the data wirelessly using multi-hop communication to avoid any areas of communication failure. This also allows the main nodes to collect a multi-hop dataset that other nodes would not normally be able to collect.

#### 4. Raspberry Pi as Fog Node for Local Processing and Web Display

The architecture proposed in this paper has the Raspberry Pi as a key part of the fog computing layer, which acts as a local processing and decision-making layer from the distributed Wireless Sensor Network (WSN) and any cloud infrastructure. In this framework, given the processing power of the Raspberry Pi, we can leverage a low-cost, low-power, single-board computer to provide real-time data aggregation, filtering, analysis and visualisation without always sending all data to the cloud. By using fog computing in this edge device as a part of the network, we are able to reduce communication latency, improve its reliability, and bandwidth efficiency, which is important for systems with a requirement of continuous data monitoring and timely responses, as needed by health monitoring and environmental monitoring applications.

The Raspberry Pi functions within a Debian-based Linux environment and the Pi has a lightweight but powerful web server based developed with the Flask micro web framework. Incoming data packets are transmitted via Wi-Fi as UDP or TCP from many ESP32/ESP8266-based sensor nodes. The data packets contain measurements from the sensors: body temperature, ambient temperature and humidity, heart rate, saturation of oxygen ( $\text{SpO}_2$ ), and acceleration data (for fall detection). The server does the basic data pre-processing: smoothing through averaging filters, as well as finding outliers and removing noise. Our pre-processing process isolates clean data from noisy data; only reliable data are visualized and stored. This pre-processing improves the accuracy and readability of the data that the user sees.

In addition, the fog server has the possibility for local decision-making. For example, it checks the sensor data against thresholds ( $\text{SpO}_2 < 90\%$  or a reasonable jump in heart rate) to indicate anomalies and notify the appropriate health worker.

The alerts were written to a log with an accurate timestamp and viewable on a interactive dashboard in real-time when accessed from any web browser. The dashboard sections included: (i) patient information providing identification and demographic data, (ii) visual displays for plotting the real-time time series physiological data, (iii) raw numerical data fields showing sensor input readings, and (iv) an alert log console for tracking abnormal events.

The real-time web interface is capable of simultaneous access across multiple clients and can be effectively used in the hospital setting or with multi-patient monitoring. With its asynchronous nature, Flask handles connections to human agents or machine agents, utilizing a RESTful API. In addition, the architecture allows for conditional integration with the cloud; when internet connection is available, the Raspberry Pi can send validated and compressed data packets (when needed) to a remote server or cloud database where the information can be stored for extended periods of time, be used for remote diagnostics, or be uploaded for AI-based analysis. The layered strategy of providing a functional cloud-computing capabilities allows the system as a whole to be usable and operable when offline or with sporadic or unreliable connectivity and increases the overall fault tolerance of the overall system.

The use of Raspberry Pi as a fog node again reinforces the modularity and scalability of the system. New nodes can be connected to the system with limited configuration, and new processing scripts or visualization modules can be added to the fog server.

#### 5. Implementation and System Setup

This section outlines the hardware and software implementation of the proposed IoT-based Wireless Sensor Network (WSN) for patient health monitoring using ESP32 microcontrollers and a Raspberry Pi-based fog server. The system architecture supports real-time monitoring, localized display, multi-hop data communication, and centralized visualization with alert management.

### 5.1 Edge Node Hardware Configuration

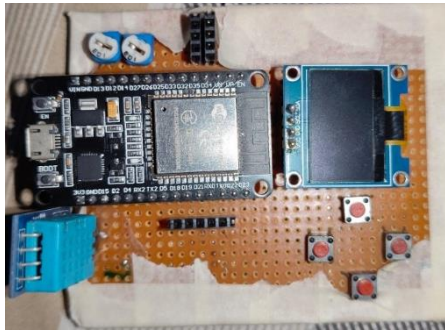


Figure 2. Edge node architecture with ESP32 and connected sensors including MAX30100, LM35, DHT11, and OLED display.

In the proposed system, each edge node is based on the world-renowned ESP32 microcontroller, which was chosen for its low power use, Wi-Fi capabilities, and ability to transmit data over a real-time medium in an environment with resource management in mind. To provide an overall look at patient health, we have added a set of biomedical sensors to each ESP32 node. First, we added the DHT11, which gives the ambient temperature (i.e., measuring environment temperature and humidity). Secondly, we added the LM35, which gives an analog reading of the body temperature. Thirdly, the MAX30100, which utilizes infrared to capture blood oxygen saturation and heart rate at the same time.

We have also used dedicated pulse sensor for the pulse of the patient to measure beats per minutes (BPM), and combine the edge node accelerometer, such as the MPU6050, to analyse changes in acceleration vector, which may suggest falling if the boundary vectors are breached. Additionally, we could integrate an optional ECG sensor to collect the underlying electrical activity detail of the heart and improve diagnosis. Each edge node will also visualize sensor data using a small TFT, or OLED, for visualization of the instant readings at the patient's location. This way, a patient can be up-to-date with their health status, even if no data is being exchanged in an inactive way directed towards server usage or communicating via the network.

### 5.2 Main Node and Communication Topology

The communication architecture proposed for the hospital-wide monitoring system is a hierarchical topology, where multiple edge nodes send monitoring data to a main node. The main node uses the ESP32 microcontroller as the focal point for

sensor data aggregation within each local cluster of sensor nodes, collecting, buffering, and forwarding sensor data packets from the nearest edge devices to the fog computing server via a secure Wi-Fi link. Reliability and low latency data communication transmissions methods will be important, especially when viewing large-scale implementations like hospital wards or care facilities.

Communication follows a multi-hop model based on proximity and communication stability for a connection between nodes. Each edge node is programmed to detect and connect with the nearest node (i.e., the main node) which helps reduce transmission power and ultimately battery depletion. The main node then immediately relays the structured data (composed of device ID, timestamp, and priority flags for critical or abnormal readings) to the Raspberry Pi-based server for aggregation. The data collected and in transit is designed to be robust and adaptable. Dynamic routing is employed with real-time updates from periodic broadcast messages, ensuring routing tables could be updated continuously and even when nodes disconnect temporarily.

### 5.3 Server Architecture and Web-Based Visualization

With the implementation of a hospital-wide monitoring system, the communication structure is defined as a hierarchy, where multiple edge nodes send data to a main node. The main node is an ESP32, which is designed to work as an aggregator of data for a local cluster of sensor nodes. The main node collects, buffers, and forwards packets of sensor data from nearby edge nodes to the fog computing server using a secure Wi-Fi channel. Either by using multiple edge deployed edge nodes, or by using a single edge (node)/ main node for clustering data packets, creates a reliable option for performing data transmissions while ensuring low-latency, especially in large-scale installations in hospitals in units like wards or care facilities.

The fog server layer is implemented as a Raspberry Pi 4 single-board computer that is running a Debian Linux distribution. It runs a Python application using the Flask library as the main application/interface for data processing, alert generation, and real-time patient health data visualization. The fog server continuously listens for incoming data from multiple clusters of main nodes across the network and is responsible for three broad operations which include

data aggregation, observation of anomalies, and visual representation of patient health data in real-time.

The server structure is a key element of the detailed IoT-based patient monitoring system. The fog computing server acts as the fog computing layer tying edge data acquisition to end-user visualization. The fog server is implemented via a Raspberry Pi 4 that is running a Debian-based Linux operating system, and utilizes a web application stack built with the Flask micro web framework. The Raspberry Pi acts as a server with the capability to receive multiple incoming data transmissions from the main nodes installed in the wireless sensor network. The server, upon receipt of a data transmission packet will perform the three fundamental tasks of data aggregation, filtering and alerting, and real-time visualization.

**Data Aggregation:** The server receives structured data streams from all patient nodes connected to it and appends them to files stored locally using a format of your choice (e.g., JSON, CSV). Our server permanently records all incoming data for you to analyse later or to process more meaningfully.

**Filtering and Alerts:** The server compares incoming data against accepted thresholds to recognize abnormal patterns. For example, if someone's blood oxygen saturation (SpO<sub>2</sub>) drops below 90%, their SpO<sub>2</sub> event will generate an email alert, or if they throw an abnormal acceleration, we might suspect they've fallen.

**Real-Time Visualization:** Using Flask, a web dashboard is automatically generated and delivered through any standard web browser from any device connected to the same Wi-Fi network. The web app provides the clinician or caretaker a way to view patient data in real time from anywhere using an intuitive interface.

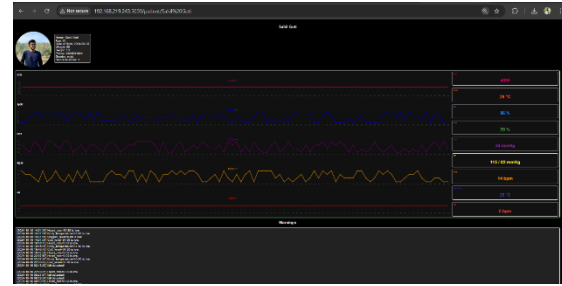


Figure 3. Real-time monitoring dashboard showing patient details, plotted graphs for SpO<sub>2</sub>, ECG, respiration, and CO<sub>2</sub>, alongside numerical data.

As shown in Figure 3, the dashboard is subdivided into four high-level categories to improve readability and functionality:

**A. Patient Information Block.** This displays metadata for each monitored patient, including the patient's name, age, gender, birth date, and their device ID

**B. Graphical Data Plot.** This shows dynamic plot(s) for monitored physiological signals including SpO<sub>2</sub>, ECG, respiration, and CO<sub>2</sub> levels.

**C. Numeric Panel.** This panel shows the most recently read sensor values in coloured numeric boxes.

**D. Alert Log Console.** This records all triggered alerts as they occur with a timestamp to help caregivers identify and address abnormal states quickly.

#### 5.4 Warning and Fall Detection System

The fog server has a severe warning and fall detection subsystem built in, so it can identify serious health events immediately. This subsystem is continuously running in real-time analysing incoming sensor data and determining if it indicates a clinically significant event. The subsystem levels of severity are based on the following indicators:

**SpO<sub>2</sub> Level:** Less than 90% is considered hypoxemia

**Heart Rate:** Less than 60 bpm (bradycardia) and greater than 120 bpm (tachycardia) are both abnormal heart rates.

**Body Temperature:** A marked variance from baseline body temperature is used for a thermal alert.

**Fall Detection:** Anomaly detection of the acceleration vector is performed using the



MPU6050 accelerometer based on predefined threshold vectors.

When one of these events occurs the system logs the event with a time stamp and visually generates a warning under the alerts component on the web dashboard. This means that medical staff and caregivers are able to intervene quickly and prevent possible health complications and take timely ambulatory action.

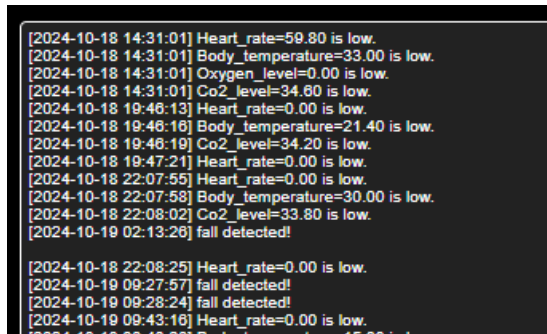


Figure 4. Warning log section displaying fall detection and vital parameter alerts with timestamps.

## 6. Conclusion

This paper has described the design and implementation of a scalable, low power, real-time IoT-based Wireless Sensor Network (WSN) for monitoring patient health in a hospital. Using ESP32 microcontrollers at the edge, the system integrates multiple biomedical sensors such as DHT11, LM35, MAX30100, pulse sensors, and accelerometers, to monitor critical health parameters, such as body temperature, SpO<sub>2</sub>, heart rate, and fall detection. Each edge node transmits data wirelessly to a centralized server via a hierarchical main node, and also displays the vital readings locally at the edge node with an integrated screen for on-site monitoring.

The fog server was implemented on a Raspberry Pi and is capable of processing incoming data on the fog server and visualizing it using a Flask-based web

dashboard that provides a real-time graphical plot, numeric indicators, and timestamped alerts for observation of outlier readings or emergencies such as falls. The architecture demonstrated is modular, dynamic, and allows for multiple clients to connect, making it a robust foundation for continuous healthcare supervision applications, particularly intensive care, elderly care, and remote health monitoring applications.

The effective deployment of edge computing, real-time alerts, and local visualization indicates the potential use ability of this WSN model in actual medical spaces. Future enhancements could include cloud integration and long-term analytics, communication encryption, and automated anomaly detection with AI processing to enhance and potentially ensure the system's accuracy, security, and predictive aspects

## 7. References

- [1] D. Rani and P. Kumar, "Wireless sensor networks in the Internet of Things: Review, techniques, challenges, and future directions," *ResearchGate*, 2024.
- [2] Espressif Systems, *ESP32 Technical Reference Manual*, 2022. [Online]. Available: <https://www.espressif.com>
- [3] B. Patel, "Performance evaluation of TCP, UDP and DCCP traffic over 4G network," *ResearchGate*, 2015.
- [4] P. Thakare and S. Patil, "A review of fog computing: Concept, architecture, application parameters, and challenges," *ResearchGate*, 2024.
- [5] X. Liu, J. Wang, and Y. Zhang, "A Practical Neighbor Discovery Framework for Wireless Sensor Networks," *Sensors*, vol. 19, no. 9, pp. 2136, 2019.
- [6] L. Zhang and Q. Li, "An Improved AODV Routing Protocol of Wireless Sensor Network," *International Conference on Information Science and Technology*, 2014.