Remote Energy Monitoring System

Rudrank Dwarkanath Datye
Electronics and Telecommunication engineering,
Goa college of engineering, Ponda, Goa, India

Devendra Sutar
Assistant Professor
Electronics and Telecommunication engineering,
Goa college of engineering, Ponda, Goa, India

Pravar Tulsidas Naik
Electronics and Telecommunication engineering,
Goa college of engineering, Ponda, Goa, India

Sharvani Tulsidas Verenkar
Electronics and Telecommunication engineering,
Goa college of engineering, Ponda, Goa, India

Vaishnavi Vithoji Rane
Electronics and Telecommunication engineering,
Goa college of engineering, Ponda, Goa, India

Abstract - The purpose of the work is to optimize the ships operated by the local carriers of the area by an energy monitoring system which is based on Internet of Things (IoT) solution. With the help of sensors, real time information on key energy consumption parameters can be measured and system is able to monitor and analyze continuously. Data gets sent to the AWS Cloud via a gateway where it then processes the data to establish KPI's like specific fuel consumption. The device also detects deviations from ideal service conditions, such as excessive fuel consumption, and provides alerts for potential servicing requirements. It also generates historic energy demand profiles that can be used to analyses trends and make proactive decisions in order to optimize operations.

I. INTRODUCTION

The real-time monitoring of energy is a key requirement to manage and optimize the consumption of energy. In this work a real-time energy monitoring system for energy optimization, based on the ESP32 microcontroller and three sensors: voltage sensor, current sensor and an ultrasonic sensor, is developed. The sensor data is read by the ESP32 and is transmitted using the lightweight MQTT protocol, that is well suited for the constrained IoT devices and the cloud. When the data hits AWS IoT Core, it's handled and processed with AWS Lambda functions. This serverless computing service enables the system to process incoming data automatically. Once ascertained, energy-related data is persisted securely to AWS DynamoDB, a high-performance NoSQL database platform. To serve this data to its users, a web application is created with Node.js. This app reads your state by the data you stored in DynamoDB and visualizes your data in a real-time way. The end-to-end system architecture that combines IoT devices, cloud services and web-based technologies of the proposed solution ensures a cost-effective, scalable and secure way of performing real time energy profiling. The rest of the paper is organized as follows Section II describes the architecture for real-time data collection and storage (IoT). Section IV presents the design and development of web-based IoT monitoring dashboards. Concluding remarks are provided in Section V.

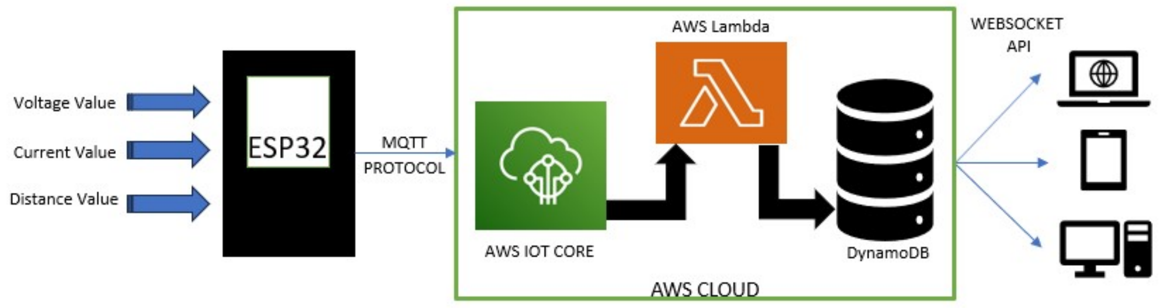


Fig 1: Dataflow architecture

II. REAL-TIME IOT DATA ACCUMULATION AND STORAGE

The ESP32 system has an architecture that supports multiple sensor integration to a single ESP32. Which enables timely data sampling and robust data acquisition from various types of sensors integrated. The Data is further sent to AWS cloud for storage purpose in DynamoDB via AWS IoT core. The data is sent using MQTT protocol

1. *Integration of Voltage Sensor with ESP32:*

The voltage sensor used is the ZMPT101B sensor, it is designed for exact AC voltage measurement in circuits. The component outputs a minor, scaled-down AC voltage which the ESP32 can safely read and process and use for further purposes. Setting it up properly with ESP32 is necessary to ensure accurate readings, involving changes to the sensitivity of the sensor module and charting the ADC values to matching real-world voltage values through program design.

2. *Integration of Current Sensor with ESP32:*

Current sensor used is ACS712 sensor, which gives output voltage proportional to the current flowing through the sensor. the output is connected to ADC pin of the ESP32 for taking the voltage value. The ESP32 reads the analog voltage and using calibration factors which are set in the Arduino program starts calculating the real current in amperes. The ACS712 sensor is sensitive to noise as it gives random values when testing, filtering techniques like averaging multiple values, are applied in the code to ensure accurate current values which are generated by the sensor

3. *Integration of Ultrasonic Sensor with ESP32:*

The HC-SR04 ultrasonic sensor is for measuring the distance of the fuel tank in the ships. It has two main pins Trigger and Echo, of which Trigger pin is connected to digital output pin on the ESP32, while the Echo pin is connected to a digital input pin of ESP32. By sending short pulse from the ESP32 and measuring the time it takes for the echo to return, we can calculate the distance using simple time-to-distance conversion formulas in the code. Longer the time longer the distance principle.

4. *Integration of ESP32 with AWS IoT Core:*

The ESP32 connects to AWS IoT Core with the help of built in Wi-Fi service, additionally using the WiFi.h library for real-time data transmission of sensor data to AWS. Secure MQTT communication is managed by the PubSubClient.h library, with data formatted in JSON using ArduinoJson.h. The ESP32 publishes data to an MQTT topic managed by AWS IoT Core. On IoT core to get Data the Thing must be generated as a virtual representation of the device that needs to be connected according to that the Certificates are auto generated by AWS IOT core which are needed for the integration purpose of AWS IoT core and ESP32. Endpoint id also required to connect the AWS and ESP32.

5. *Saving Sensor Data to DynamoDB via Lambda:*

Sensor data received by AWS IoT Core is processed using an AWS Lambda function. Lambda Function created specifically serves as a serverless processor that automatically extracts and setups the incoming MQTT data in IoT core. The data includes parameters like voltage, current, and distance are stored in AWS DynamoDB, a customizable NoSQL database. DynamoDB ensures fast recovery and robustness of real-time sensor data. Also, Lambda prepares the stored information for access by the web application. The web app that is built using Node.js and JavaScript, queries DynamoDB to fetch the latest readings from the storage, allowing users to monitor energy usage instantly.

III. REAL-TIME IOT DATA VISUALIZATION AND ALERTING

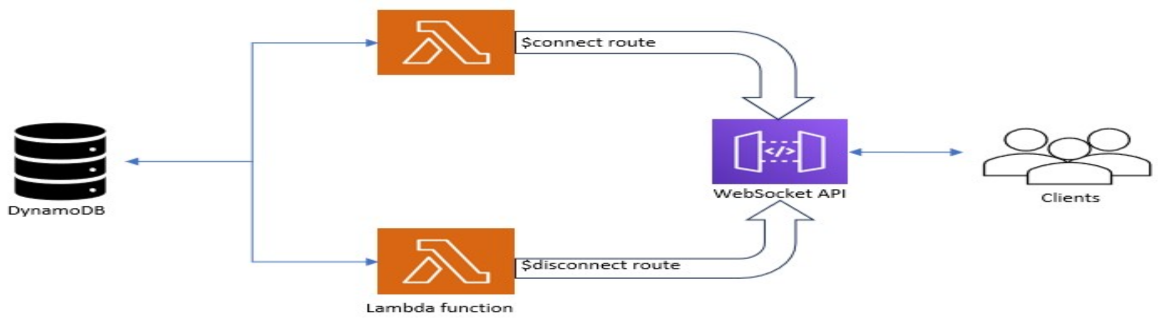


Fig 2: WebSocket APIs on AWS for Real-Time Communication

For real-time IoT data visualization, the data flow begins with IoT devices sending their data streams to AWS IoT Core there the data is passively picked up by an IoT Core Rule, which then triggers a Lambda function to process it and persist this unstructured data into a scalable DynamoDB table. The dynamo DB streams were turned on, which means every event or change is captured, and another Lambda function that consumes the stream and called the Amazon API Gateway WebSocket API when any event happens, which ensures a persistent connection in a two-way channel with any on-line website clients. Web Sockets are important because they allow the server to push updates to the client, such as a sensor value change, without having to repeatedly make API calls for updates. The server maintains client connections using the $connect and $disconnect routes, so it knows which clients are active. The Lambda function uses the API Gateway Management API's Post To Connection action to send data payloads to a specific client through the connection that was created when a client connected, in this case through their unique connection Id. The website has standard JavaScript using web requisites to define the updates being sent down so that it can dynamically display the data in near-real time.

Simultaneously, it allows for real-time alert notifications. While the original Lambda function is processing the IoT data stream, it also checks that incoming data against critical thresholds or conditions that represent alert criteria. If an alert criterion met, this function will publish a message with a description of the alert to AWS SNS Topic. Another "SNS to WebSocket" Lambda function also subscribes to that topic and will receive all the notifications for the alert. Then it would use the already constructed API Gateway WebSocket API to send the alert messages to all clients that are online. The client-side JavaScript listens for these kinds of alert messages and invokes some type of visual or audible notification strategy to let the user know about critical event.

Amazon S3 serves as the manager for static website hosting through its serverless solution. The cloud model of serverless computing enables developers to create code for deployment through infrastructure management free interfaces. Organizations can concentrate on building applications and speeding up product development since AWS manages their server provisioning and scaling alongside patch application and system availability. The web interface contains all static assets including HTML, CSS and JavaScript files which are stored in S3 buckets that allow uninterrupted public accessibility. The method provides businesses scalability and durability with cost-effective advantages. The website starts from Amazon API Gateway that functions as a managed service for creating together with publishing and protecting the API. The API Gateway service enables client HTTP(S) requests to reach AWS Lambda functions that serve as backend processing components. Dynamic content generation through Lambda functions enables access to Amazon DynamoDB data which serves as the fully managed NoSQL database for application data storage and retrieval. Without server management responsibilities the solution reduces operational expenses while maximizing cost effectiveness.

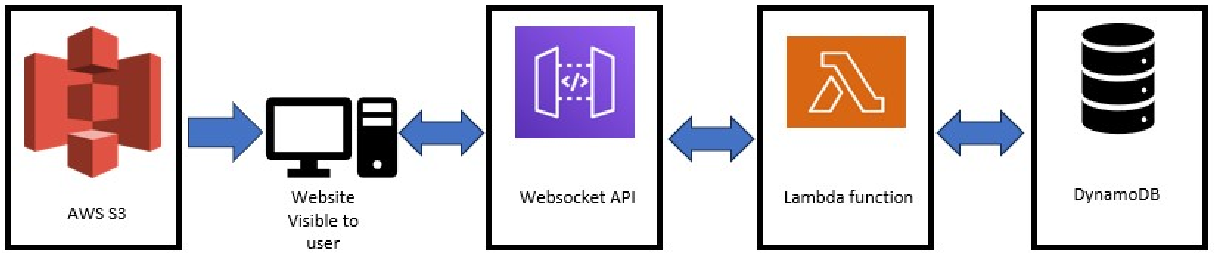


Fig 3: AWS Serverless Computing

Step	Component/Service	Purpose	Library/Protocol Used
1	ESP32	Collect sensor data	WiFi.h, WiFiClientSecure.h
2	Wi-Fi Connection	Connect ESP32 to Internet	WiFi.h
3	MQTT Communication	Publish data to AWS IoT Core	PubSubClient.h
4	AWS IoT Core	Manage incoming MQTT messages	MQTT Protocol
5	AWS Lambda	Process received data	Serverless compute
6	AWS DynamoDB	Store processed data	NoSQL database
7	WebSocket AWS API	Send real-time data from DynamoDB to clients	WebSocket Protocol
8	AWS S3	Host the static website	Static Web Hosting
9	AWS SNS	Send real-time alerts	Pub/Sub Messaging
10	Node.js Web App	Display data to users	AWS SDK, Node.js

Table 1: Real-Time Sensor Data Flow and Cloud Integration

1. *Frontend and Backend Technologies*

The frontend user interface operates with HTML together with CSS and JavaScript as enabling components. Through the implementation of Chart.js the program fetches backend API data while updating the front-end interface to display live information and alert notifications and interactive charts.

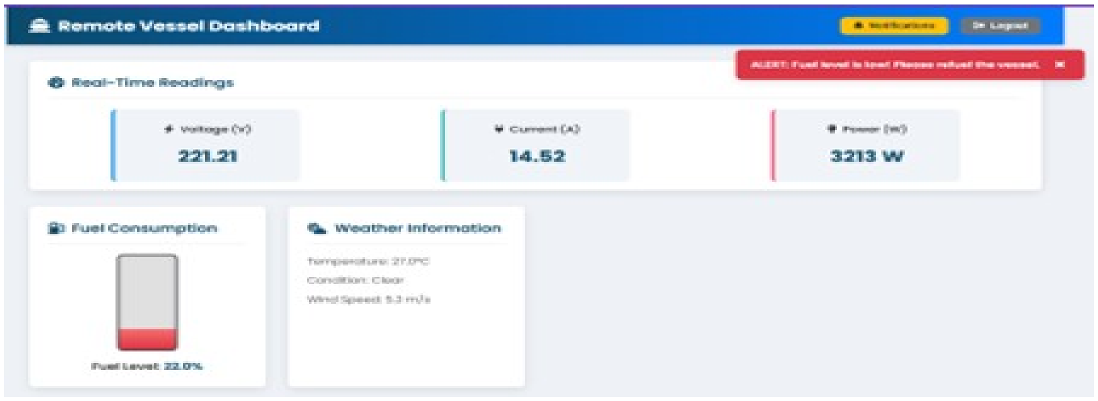
The Node.js environment includes Express.js as its backend framework to serve the server.js file which communicates with AWS DynamoDB through aws-sdk. The application uses CORS (cors) middleware to establish Cross-Origin Resource Sharing for frontend and backend interoperability. The application safely handles environment variables which contain AWS credentials through the dotenv library.

2. System Overview and Data Flow

The Remote Vessel Monitoring System is intended to provide real time monitoring and visualization of important vessel parameters.

• System Overview: The system allows users to log-in to the system and have access to a dashboard with live indicators of voltage, current, power, and fuel level. The fuel level is shown both numerically and visually as a dynamic indicator. Simulated weather data is incorporated as well. A core functionality of the system is the visualization of system performance trends over time using charts that allow interaction. The system also includes a notification functionality to notify users of any critical conditions.

• Data Flow: The frontend JavaScript (script.js) is where data flow begins, as the code periodically fetches the most recent readings from the backend API endpoint. When the backend API is called, it fetches the latest data records from the DynamoDB database. The backend also calculates the fuel level as the distance value is brought over with the latest reading. The backend then sends the processed data - voltage, current, power, and the calculated fuel level - to the frontend for display, in the form of a JSON response. The frontend will then update the appropriate UI elements dynamically and reflect these real-time measurements in the charts for users, allowing users to see the status of the vessel in near real-time.



ALERT: Fuel level is low! Please refuel the vessel.

Voltage (V) 221.21

Current (A) 14.52

Power (W) 3213 W

Fuel Level: 22.0%

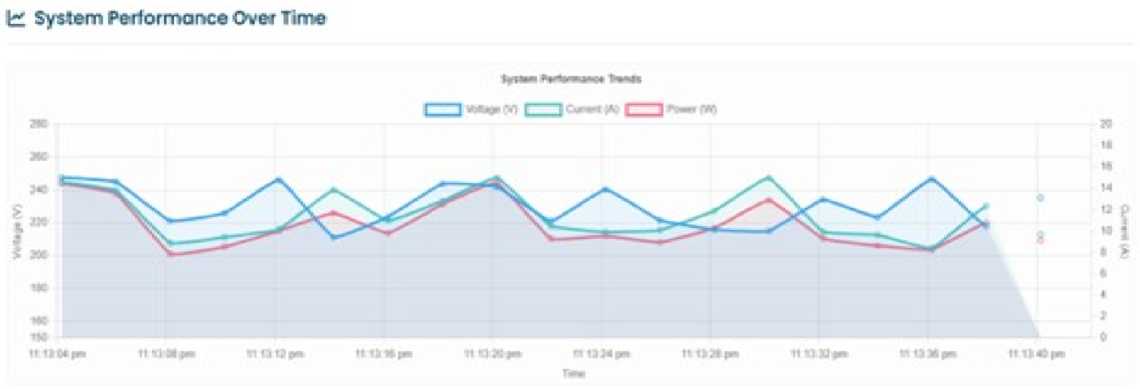
Temperature: 27.0°C

Condition: Clear

Wind Speed: 5.3 m/s

Fig 4: Real-time Vessel Monitoring Dashboard Interface with Fuel Level Alert

Fig 5: Line Chart Representation of Voltage, Current, and Power Trends



V. CONCLUSION

In this paper, we presented the process of developing an IoT-based Remote Vessel Monitoring System which presents a better way to utilize energy for local carriers. The system's key feature is its real-time monitoring and visualization of important parameters and information through the integration of multiple sensor modules and cloud-based data processing. The hardware we selected is the ESP32 microcontroller, chosen for its dual-core Xtensa LX6 processors, robust wireless capabilities (Wi-Fi and Bluetooth), and efficient power management. This hardware base permits us to acquire data from many sensors, including voltage sensors, current sensors, and the ultrasonic sensor, which gives a complete overview of the vessel's working status and the parameters in the vessel being monitored. On the cloud side, we selected Amazon Web Services (AWS) based on its scalability, reliability, and third-party management of the operational overhead. AWS IoT Core provides secure ingestion of the data, while AWS Lambda functions provide serverless programming for the data ingestion and creating the transformations. DynamoDB is used to store time series data as a NoSQL data store. Our complete system will provide real-time data and is capable of storing the data and displaying the values through our own developed web-based monitoring dashboards. The dashboards contain visualizations of the voltage, current, power, and fuel level. In conclusion, the Remote Vessel Monitoring System developed can provide a solid and efficient process to improve energy efficiency of vessel. The system has provided a real-time, display of information, monitoring capability to support informed and data-driven actions that optimize the performance and energy consumption.

VII. ACKNOWLEDGEMENT

The project was carried with generous support and assistance of Electro Controls Manufacturer of electric panels and automation system. Their collaboration was essential to the success of this project.

REFERENCES

[1] Chakraborty, S., & Aithal, P. S., (2023). Let Us Create Multiple IoT Device Controller Using AWS, ESP32 And C#. International Journal of Applied Engineering and Management Letters (IJAEML), 7(2), 27-34. DOI: https://doi.org/10.5281/zenodo.7857660
[2] https://iaeme.com/MasterAdmin/Journal_uploads/IJRCAIT/VOLUME_7_ISSUE_2/IJRCAIT_07_02_121.pdf
[3] https://aws.amazon.com/blogs/industries/real-time-operational-monitoring-of-renewable-energy-assets-with-aws-iot/
[4] Espress if Systems, "ESP32 Datasheet Version 3.9," *Espressif Documentation*, 2023. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf
[5] AWS, "AWS IoT Core Documentation," *Amazon Web Services*, 2024. [Online]. Available: https://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html
[6] AWS, "AWS Lambda Developer Guide," *Amazon Web Services*, 2024. [Online]. Available: https://docs.aws.amazon.com/lambda/latest/dg/welcome.html

[7] AWS, "Amazon DynamoDB Developer Guide," *Amazon Web Services*, 2024. [Online]. Available: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html
[8] AWS, "Amazon API Gateway WebSocket APIs," *Amazon Web Services*, 2024. [Online]. Available: https://docs.aws.amazon.com/apigateway/latest/developerguide/apigateway-websocket-api-overview.html
[9] https://docs.aws.amazon.com/pdfs/whitepapers/latest/build-static-websites-aws/build-static-websites-aws.pdf
[10] https://jaytillu.medium.com/understanding-serverless-computing-with-aws-lambda-8db9f9af1056.