UNIVERSITÀ DEGLI STUDI DE L'AQUILA

Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica

Università degli Studi dell'Aquila

Forest Fire Detection System

Department of Information Engineering, Computer Science and Mathematics

University Of L'Aquila, Italy

Professor Davide Di Ruscio

Software Engineering for Internet of Things

Members
Alex Montoya Franco
Sergo Kashuashvili
Sherkhan Azimov

UNIVERSITÀ DEGLI STUDI DE L'AQUILA

Contents

1. Introduction

Forest fires are a cause for great concern to both the ecosystem, wildlife and human settlements. Their destructiveness depends on speedy detection and rapid response. In this report, we describe a forest fire detection system based on the Internet of Things (IoT) in which environmental conditions are monitored proactively within woodlands. The Internet-based system incorporates information from multiple sensors, data analysis techniques and rapid alert capabilities. Its purpose is to enhance early warning capabilities so that any dealing with forest fires can be done quickly, in the hope of preventing or reducing damage. The application of this system aims at taking a step towards an active role regarding forest fire prevention and protection.

2. Objectives

2.1. Reliable IoT system for real-time monitoring of forested areas.

- Develop a robust IoT system capable of tracking the temperature, humidity, and air quality of multiple forested areas in real-time.

2.2. Detection of changes in temperature, humidity, and air quality indicative of potential fires.

- Implement algorithms to detect abnormal changes in temperature, humidity, and air quality, indicating potential fire risks.

2.3. Smart alert system to notify relevant authorities in case of fire detection.

- Integrate an alert mechanism that instantly notifies relevant authorities upon detecting potential fire-related anomalies.

2.4. User-friendly web-based dashboard accessible to authorities.

- Create a dashboard that provides real-time updates on environmental conditions and alerts about potential fires.

3. Functional Requirements:

Functional Requirements of the IoT-Based Fire Detection System for Forests		
Identifier	Name	Description
FDSF-FR001	Data Generation	The system must be able to generate the data for the simulation of IoT-based sensors in real-time.
FDSF-FR002	Data Collection	The system should collect the generated data from different areas of a forest.
FDSF-FR003	Fire Detection	The system should be able to detect potential fires in a forest with a specified algorithm.
FDSF-FR004	Monitoring Interface	The system should provide an interface for the user to monitor the system's current environmental conditions for each sensor in different places in a forest.
FDSF-FR005	Recent Activities	The system should provide a web interface for recent alerts.
FDSF-FR006	Use Notification	The system should notify the user through mobile notifications about identified possible fires in the area.

4. Non-Functional Requirements:

Non-Functional Requirements of the IoT-Based Fire Detection System for Forests		
Identifier	**Name**	**Description**
FDSF-NFR001	Response	The system should provide alerts within 10 seconds of detecting a potential fire.
FDSF-NFR002	Reliability	The system should ensure continuous operation with minimal downtime for maintenance.
FDSF-NFR003	Scalability	The system should be able to handle the inclusion of additional sensors in other areas.
FDSF-NFR004	Usability	The system should provide a user interface with clear visualizations, so the user can easily navigate through the real-time data of the sensors.

5. Technologies Used

The following technologies were used for the development of this project:

Python - Paho MQTT

Python is used to generate synthetic data simulating the different IoT sensors. While paho-mqtt is used as an MQTT client to publish messages to the broker. Messages are published to the topics following the structure *sensor/forest_{forest}/area_{area}* so that, down the line, we can subscribe to topics covering multiple forests with multiple areas.

The simulated sensors are temperature, humidity, light intensity, and air quality.

MQTT - Mosquitto

The Mosquitto broker uses the MQTT protocol. This messaging protocol is what we used to retrieve sensor data and then expose it to the subscribed components so that it could be retrieved and processed.

Telegraf

Telegraf connects with MQTT using its input plugin. It gathers data from MQTT topics, processes it, and seamlessly sends it to InfluxDB. This integration streamlines the transfer of real-time data from MQTT-enabled IoT devices to Influx DB for storage and analysis.

The MQTT Topics are defined as follows so that the system is not limited to a fixed number of forests and areas:

- temperature/+/+
- humidity/+/+
- light_intensity/+/+
- air_quality/+/+

Initial testing was run with one forest a.k.a forest_0 and three areas {area_0, area_1, area_2}.

For instance the topics for the temperature sensor on the forest_0 would look this way:

- temperature/forest_0/area_0
- temperature/forest_0/area_1
- temperature/forest_0/area_2

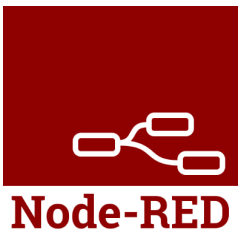
InfluxDB

The constant stream of data flowing through the system is stored in InfluxDB. The time series database is optimized for high-availability retrieval of data, faster and storage of time series data, making it ideal for IoT sensor data storage, and real-time analytics.

Grafana

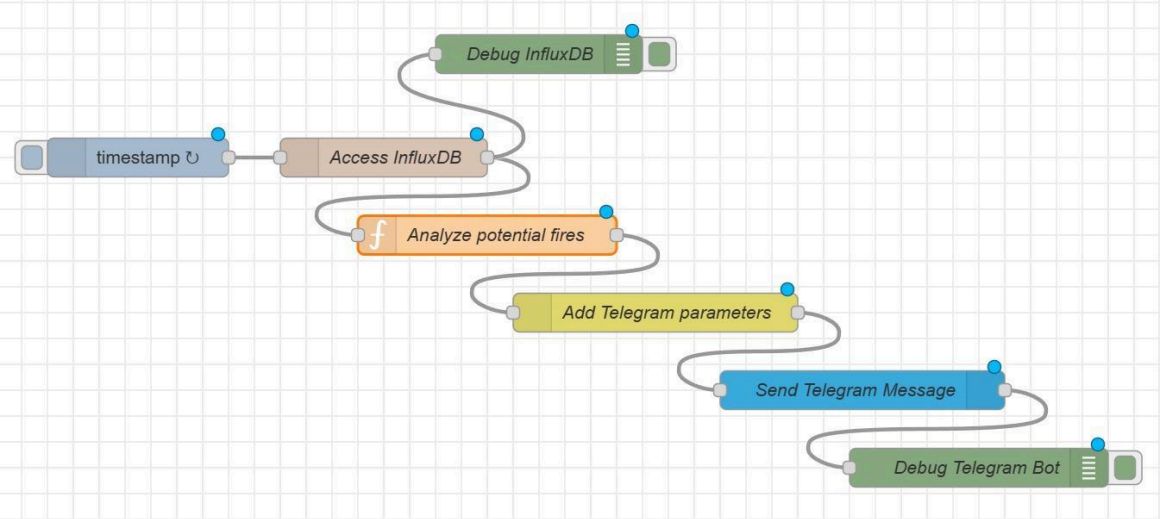
Grafana was utilized to create a dashboard for data visualization and comprehension. The key advantage we discovered was its ability to generate multiple dashboards simultaneously, which facilitated better overview of the data from different forested areas.

Node-RED + Telegram



We employed Node-RED, an IoT visual programming tool, to analyze data retrieved from InfluxDB. This analysis was used to trigger alerts via a Telegram bot whenever the system detected indications of potential fires.

From the Telegram side, we used “BotFather”, which is a bot created by Telegram that allows you to create and manage your own bots. This functionality is currently limited since it requires Telegram credentials to configure the bot and generate the alerts as expected. This functionality will be shown in the respective demo.



NodeRed flow that analyzes the information from InfluxDB to alert through the telegram bot of potential fires

Telegram Configuration Instructions

Instructions adapted from:

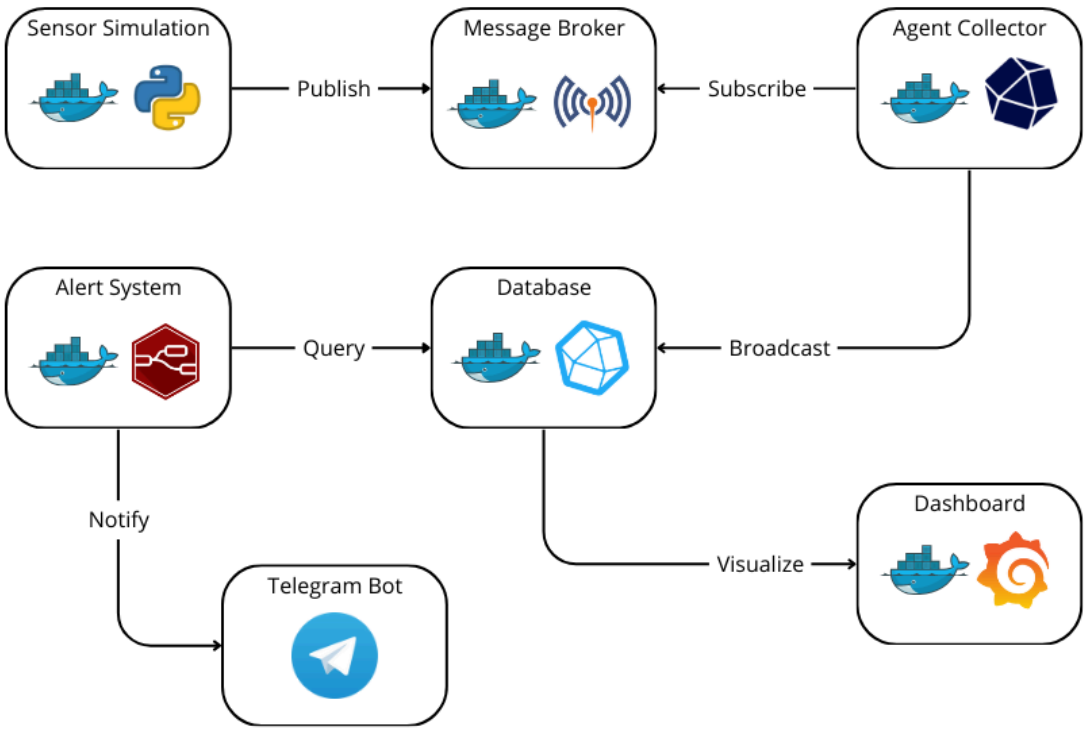
- How to Create a Telegram Bot using Python (freecodecamp.org)
- node-red-contrib-telegrambot (node) - Node-RED (nodered.org)
- Telegram Bot - how to get a group chat id? - Stack Overflow
- How to get your Telegram chat ID » DIY Usthad

1. Create the Telegram Bot
    a. Create/open a telegram account
    b. Search for @botfather in the Telegram search bar
    c. Click on the Start button to interact with the BotFather
    d. Type /newbot, and follow the prompts to set up a new bot.
    e. Save the token the BotFather provides. You will use it to authenticate to the Telegram API and interact with your bot.
    f. In your newly created bot, send the message /start to activate the bot.

2. Configure Node-RED
    a. Make sure node-red-contrib-telegrambot is installed, either from the Node-RED palette or in the Dockerfile of the container as follows: RUN npm install node-red-contrib-telegrambot
    b. Locate the data/ folder of the nodered service and update the settings.js file with the following line: process.env.BOT_TOKEN = "insert_telegram_token_here"; Add this at the beginning of the file, outside of the module.exports block.
    c. Run the containers using docker-compose up
    d. Access Node-RED on http://localhost:1880/

e. Open the "Add Telegram parameters" node and set the values for username and chatId.
   i. *Username:* If you just created your telegram account, open the mobile app, go to the settings section and click on Username to create a username for your account.
   ii. *ChatId:* One way to get the chat id is to visit the following url https://api.telegram.org/bot<YourBOTToken>/getUpdates and look for the chat object on the returned json. Please be aware that in some instances you might get the following json instead {"ok":true,"result":[]} in which case try sending a message to the bot and visiting the url again. Alternatively, you can search for "@myidbot" on Telegram, start a conversation, and type /getid, the bot will respond with your chatId. You can also explore other ways to get the chatId.
f. Open the "Send Telegram Message" node and edit the current Bot properties making sure to update the Bot-Name to align with your recently created Bot and update Users copying the same Telegram username as before.
g. Save your changes and deploy the flow.
h. Alerts should be arriving through your Telegram bot when the sensor values have exceeded the given thresholds.

6. System Architecture



Software Architecture with Technologies

7. System Functionality

Data Generation:

- Temperature, humidity, light intensity and air quality sensor data.

Data Communication:

- Transmit data from sensors to a centralized server.

Data Storage:

- Storage of time series data.

Data Analytics:

- Pattern recognition for fire detection based on the behavior of the different sensors.

Software Deployment:

- Dashboard for visualization of data.
- Notification system as an alert mechanism that can notify relevant authorities, emergency services, and nearby communities via mobile app notifications.

8. Data Visualization using Grafana

Grafana transforms sensor data into visual dashboards, presenting real-time environmental metrics like temperature, humidity, and air quality across forested regions. These customizable, intuitive displays enable quick identification of anomalies, aiding rapid response to potential fire risks.

9. Conclusion

In conclusion, our IoT-based forest fire detection system stands as a compelling demonstration of an architecture for early fire detection in forested areas. By integrating sensors, data analytics, and real-time alerts via Telegram, the system showcases a path towards quick identification of potential fires. Grafana's data visualization enhances decision-making by offering clear insights into environmental conditions.

Continuous refinement and expansion remain vital for enhancing the system's capabilities and extending its reach to safeguard larger forested areas in more efficient ways.

This system represents a step toward proactive forest protection, aiming to mitigate the destructive impact of fires on ecosystems and communities. Through ongoing development, it holds promise in fostering a safer environment for forests and human settlements alike.