Industrial Internet Infrastructure (H04I0a)

Alexander Croes (R0853406)
Ignasi Juez (R0974979)
Jan Neys (R0786900)
Robbe Serry (R0848457)
Dries Vanspauwen (R0857931)

Academic year 2023-2024

Contents

1 Introduction

In this report we will go over every part of the system, from server to Freebot, and explain our choices of implementation and security.

2 Sensor/Actuator

The sensors and actuators are part of a platform, the Freebot. This platform has some sensors (voltage meter, motorspeed meters and motor angle meters) and actuators (motors). The goal is to drive this car around and collect data. The control script can be found in the file freebot/src/main.c. The Freebot is a NRF52840, flashed so two threads are run: one for the collection and sending of the sensor data and the other for handling the received BLE data and operate the actuators. The Freebot and BLE Gateway connect over BLE using custom UUIDs, and they can transfer data in both directions successfully. Movement instructions received from the BLE connection is stored in a FIFO-queue via a callback function and the thread responsible for operating the actuators will handle this stored data. Due to the limited time only the forward moving operation is implemented. The Freebot sends its sensor data every 1000ms to the BLE Gateway.

3 BLE Gateway

The BLE Gateway is a NRF52840. This board can communicate with the Freebot over BLE. The script can be found in the file BLE-gateway/src/main.c. It is flashed with two threads: one for the communication over BLE and the other to communicate over UART. Data received from either source is stored in their own FIFO-queue protected by their own mutex. The BLE Gateway receives the sensor data from the Freebot and sends it to the internet gateway over UART. From the internet gateway, it receives movement instructions to drive the car that it passes to the freebot over BLE.

In the end result, the sending and receiving of data over UART is not implemented. However, the code for doing communication in both directions over UART is written and tested, it is just not implemented in the UART thread due to time constraints. To simulate sending data to the internet gateway, a script *fakereadings.py* can be run on the internet gateway. To simulate incoming movement operations, a button on the BLE gateway can be pressed to send a forward moving operation to the peripheral.

4 Internet Gateway

The internet gateway in this project is a Raspberry Pi Zero. This device communicates with the server through an external mqtt server (mqtt.eclipseprojects.io). The script that runs on the internet gateway can be found in the file rpicontroller.py. To simultaneously listen for control commands and send sensor data, two different threads are used.

To receive controls to drive the car, the gateway listens to MQTT server for messages under the topic *'III2024/12/control'*. These controls get limited to one byte and are passed through to the BLE gateway over UART. The internet gateway also receives data from the BLE gateway. This contains information about the battery and motors. It sends this sensor data to the MQTT server under the topic *III2024/12/sense* so the server can display these. As mentioned previously, the sending of data over UART is not implemented by the BLE gateway, but it is tested successfully by manually sending these commands from the NRF board.

5 Server

5.1 User-friendly tool

The tool to send movement commands to the Freebot is through a CLI with a simple python script. The script first informs the user about the input of the controls, which is through simple key presses. The user can use the arrow keys to move the FreeBot up, down, left or right, the spacebar to turn it clockwise and shift for a counter-clockwise movement. Pressing 'x' will shutdown the script. Any other input will be rejected and the user will be re-informed about the expected inputs.
After a correct input the tool will publish this information to the *III2024/12/control* topic which the Raspberry Pi is subscribed to. This script can be found in the controlpub.py file.

5.2 Tick Stack

This TICK Stack deployment utilizes Telegraf for data collection, InfluxDB for high-performance time-series storage, Chronograf for data visualization, and Kapacitor for real-time data analysis, enabling alerts and automated actions. The configuration files can be found in the server directory on Github.

5.2.1 InfluxDB

InfluxDB serves as the central hub for time-series data within our TICK Stack configuration. It's a high-performance database specifically designed to efficiently store and manage data points that change over time. Telegraf, the data collection agent, feeds InfluxDB with metrics and measurements.

Our InfluxDB configuration is made through the UI and leverages an organization named iii-group12 for logical data grouping. Within this organization, a dedicated bucket named iii-group12 stores the collected time-series data. Secure access is ensured through API tokens: an operator token for basic data interaction and an admin token for broader administrative tasks. InfluxDB's Line Protocol serves as the chosen method for efficiently writing data points into the designated bucket.

5.2.2 Telegraf

Telegraf acts as the data collection agent within our TICK Stack configuration. It's a lightweight, plugin-driven server that efficiently gathers metrics and measurements. Our Telegraf configuration is made in the telegraf.conf file, more concretely on the input/output plugins. In the input mqtt consumer we modified it to listen to our server where data is published, the topics that we are subscribed to and the data format that we process. In the output influxdbv2 we modified it to send the data to our url where we have our influx deployed, with its respective organization and bucket, and the token for the permission.

5.2.3 Chronograf

Chronograf acts as the visualization component within our TICK Stack configuration. It's a web-based application that allows us to explore and visualize the time-series data stored in InfluxDB. Our Chronograf configuration is made through the UI where we establish an authorized connection with influxdb.

5.2.4 Kapacitor

Kapacitor acts as the real-time processing and analysis engine. It continuously monitors the data flowing into InfluxDB and allows to define automated actions based on specific conditions. Unfortunately, during our initial setup, we encountered some technical difficulties with Kapacitor alarms. Despite ensuring proper connection with InfluxDB, the alarms weren't triggering as expected. To maintain the desired functionality, we opted to leverage InfluxDB's built-in alert capabilities as a temporary

solution. To achieve our desired notification when the super-capacitor charge drops by an additional 10%, we used InfluxDB tasks as custom alerts. These tasks utilize Flux query scripts to monitor the charge level and trigger notifications when the specified threshold is breached.

5.3 Security

5.3.1 Authorization

InfluxDB no longer employs a native authentication and authorization system in Version 2. Instead, we utilize API tokens to control access to the database. These tokens grant specific permissions to users or applications, ensuring only authorized entities can interact with the data.

Telegraf, the data collection agent, transmits messages it gathers from publishers to InfluxDB. During this process, Telegraf leverages an API token for authorization, guaranteeing only authorized data streams with valid tokens can be written to the database. This configuration is explicitly enabled within the telegraf.conf file.

When establishing a connection between Chronograf and InfluxDB during initial setup, the InfluxDB v2 authentication mechanism is used. This involves "logging in" using a combination of the organization name and a designated API token. This ensures only authorized Chronograf instances can access and visualize the data stored within the specified organization.

5.3.2 TLS

To improve our overall security, we've implemented HTTPS communication for accessing the InfluxDB user interface. A self-signed certificate was created and integrated into the InfluxDB configuration file, establishing a secure encrypted connection between clients and the server. The effectiveness of this setup is verified using curl commands, confirming successful connection establishment and TLS handshake.

It's crucial to understand that self-signed certificates are not issued by a trusted Certificate Authority (CA), leading to warning messages displayed by web browsers when accessing the UI. While the warning persists, the underlying communication remains encrypted, protecting data privacy and integrity.

To adapt Telegraf for utilizing the new HTTPS URL, the telegraf.conf file was modified in order to publish to the new URL. Additionally, the insecure-skip-verify option was included to allow Telegraf to accept the self-signed certificate and establish the HTTPS connection.

To adapt Chronograf we created a certificate and key in one file with OpenSSL, and set the environment variable TLS-CERTIFICATE. Once the tls is enabled in chronograf we updated the configuration of the connection with influxdb to now connect with the new HTTPS URL.

5.4 End-to-end encryption

We started with an implementation of end-to-end encryption using AES-128 but due to time constraints we didn't get to implement it at the embedded end of our system.