

EVALUASI TENGAH SEMESTER TEKNOLOGI IOT

Dosen : Ahmad Radhy, S.Si., M.Si

“Pengujian Kestabilan Transmisi Data Sensor DHT22 dan Mekanisme Pembaruan
OTA pada Platform *ThingsBoard* Menggunakan *ESP32-S3*”



Disusun Oleh

Muhammad Salman Alfarisyi	2042231006
Muhammad Aufa Affandi	2042231011

PRODI D4 TEKNOLOGI REKAYASA INSTRUMENTASI
DEPARTEMEN TEKNIK INSTRUMENTASI
FAKULTAS VOKASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
2025

DAFTAR ISI

DAFTAR ISI	i
BAB I PENDAHULUAN	1
1.1 Latar Belakang	2
1.2 Rumusan Masalah	2
1.3 Batasan Masalah	2
1.4 Tujuan	2
BAB II TINJAUAN PUSTAKA	3
2.1 <i>State of The Art</i>	3
2.1.1 Perkembangan <i>Internet of Things (IoT)</i> dalam Sistem Pemantauan ...	3
2.1.2 Implementasi MQTT dan Platform Cloud untuk Visualisasi Data	3
2.1.3 Sensor DHT22 untuk Pemantauan Suhu dan Kelembapan	4
2.1.4 Kestabilan Transmisi Data IoT dan Analisis Kinerja Jaringan	5
2.1.5 Pembaruan Firmware IoT Menggunakan <i>Over-The-Air (OTA)</i>	5
2.1.6 <i>Research Gap</i> dan Relevansi Penelitian	6
2.2 Landasan Teori	7
2.2.1 <i>Internet of Things (IoT)</i>	7
2.2.2 Mikrokontroler ESP32-S3	8
2.2.3 Ubuntu sebagai Lingkungan Pengembangan IoT	8
2.2.4 Firmware pada Sistem IoT	9
2.2.5 <i>Over-The-Air (OTA) Update</i>	9
2.2.6 Sensor DHT22	9
2.2.7 MQTT sebagai Standar Protokol IoT	9
2.2.8 Platform <i>ThingsBoard Cloud</i>	10
2.2.9 Bahasa Pemrograman untuk <i>Embedded System (RustC++)</i>	10
BAB III METODOLOGI	12
3.1 Desain Sistem dan Arsitektur	12
3.2 Perancangan Perangkat Keras (<i>Hardware Design</i>)	13
3.3 Perangkat Lunak <i>Software</i>	13
3.4 Perancangan Tampilan dan Integrasi Sistem	14

3.4.1 Perancangan Program Utama (main.rs)	15
3.4.2 Konfigurasi Build dan Dependensi (Cargo.toml)	31
3.4.3 Rancangan Tampilan <i>Dashboard ThingsBoard</i>	33
3.4.4 Perintah <i>Gnuplot</i> untuk Visualisasi Data Sensor dan Analisis Latensi	34
BAB IV ANALISIS DAN HASIL PEMBAHASAN	36
4.1 Hasil Implementasi Sistem	36
4.2 Hasil Pengujian Pengiriman Data Sensor	37
4.3 Hasil Pengujian <i>OTA Firmware Update</i>	38
4.4 Analisis <i>Latency</i> dan Visualisasi Data Sensor Menggunakan <i>Gnuplot</i>	40
BAB V KESIMPULAN DAN SARAN	42
DAFTAR PUSTAKA	44
LAMPIRAN	48
BIODATA PENULIS	51

DAFTAR GAMBAR

Gambar 3.1 Desain Sistem dan Arsitektur	12
Gambar 3.2 <i>Wiring</i> antara <i>ESP32-S3</i> dan <i>DHT22</i>	13
Gambar 3.3 Alur kerja perangkat lunak pada sistem IoT berbasis <i>ESP32-S3</i> ...	14
Gambar 3.4.3 Rancangan Tampilan <i>Dashboard ThingsBoard</i>	33
Gambar 3.4.4 Perintah <i>Gnuplot</i> untuk Visualisasi Data Sensor	34
Gambar 3.4.4.1 Perintah <i>Gnuplot</i> untuk Grafik Latensi	35
Gambar 4.1.1 <i>ESP32-S3</i> terhubung dengan adaptor dan sensor <i>DHT22</i>	36
Gambar 4.2 <i>Dashboard ThingsBoard Cloud</i> menampilkan grafik suhu dan kelembapan	37
Gambar 4.3 Tampilan log terminal waktu OTA sukses	38
Gambar 4.3.1 Tampilan parameter <i>OTA</i> pada <i>ThingsBoard Cloud</i>	39
Gambar 4.4 Grafik <i>Temperature and Humidity vs Time</i> menggunakan <i>Gnuplot</i> .	40
Gambar 4.4.1 Grafik <i>Latency Analysis</i> Sensor <i>RTC</i> vs <i>ThingsBoard</i>	40

BAB I PENDAHULUAN

1.1 Latar Belakang

Perkembangan teknologi *Internet of Things (IoT)* semakin pesat dalam beberapa tahun terakhir dan telah banyak dimanfaatkan untuk berbagai kebutuhan, mulai dari sistem industri, pertanian, hingga pemantauan lingkungan. *IoT* memungkinkan perangkat fisik seperti sensor dan mikrokontroler untuk saling terhubung dan bertukar data melalui jaringan internet. Salah satu mikrokontroler yang banyak digunakan dalam pengembangan sistem *IoT* adalah *ESP32-S3* karena memiliki konektivitas *WiFi* yang stabil, dukungan terhadap protokol *MQTT*, serta konsumsi daya yang rendah (Hercog et al., 2023).

Dalam implementasinya, perangkat *ESP32-S3* sering dipasangkan dengan sensor *DHT22* untuk membaca nilai suhu dan kelembapan. Sensor ini dipilih karena memiliki tingkat akurasi yang baik dan mampu bekerja dalam rentang suhu serta kelembapan yang cukup luas. Data yang diperoleh dari sensor kemudian dikirimkan ke platform *ThingsBoard Cloud*, yang berfungsi sebagai server untuk menampilkan data secara *real-time* serta menyimpan data historis untuk kebutuhan analisis (Aghenta & Iqbal, 2020).

Namun, sistem *IoT* sering menghadapi dua tantangan utama, yaitu kestabilan transmisi data dan proses pembaruan firmware. Koneksi jaringan yang tidak stabil dapat menyebabkan data hilang atau tidak terkirim dengan benar, sedangkan pembaruan firmware secara manual pada banyak perangkat tentu tidak efisien. Untuk mengatasi hal tersebut, metode *Over-The-Air (OTA)* update dapat digunakan agar pembaruan perangkat lunak dapat dilakukan secara jarak jauh melalui jaringan internet (El Jaouhari & Bouvet, 2022).

Melalui project ini, dilakukan pengujian kestabilan transmisi data sensor *DHT22* serta penerapan mekanisme pembaruan firmware *OTA* menggunakan platform *ThingsBoard*. Diharapkan hasil pengujian ini dapat menunjukkan performa sistem dalam mengirimkan data secara kontinu dan kemampuan perangkat untuk melakukan pembaruan firmware secara otomatis tanpa perlu intervensi langsung.

1.2 Rumusan Masalah

Berdasarkan latar belakang yang telah dijelaskan, terdapat beberapa permasalahan yang perlu dikaji dalam project ini, yaitu:

1. Bagaimana cara mengimplementasikan sensor *DHT22* pada mikrokontroler *ESP32-S3* untuk mengukur suhu dan kelembapan secara *real-time* serta menampilkan datanya melalui platform *ThingsBoard Cloud*?
2. Bagaimana kestabilan transmisi data sensor *DHT22* yang dikirimkan oleh *ESP32-S3* ke *ThingsBoard* menggunakan protokol *MQTT*?
3. Bagaimana mekanisme pembaruan firmware *Over-The-Air (OTA)* dapat diterapkan melalui *ThingsBoard*, dan sejauh mana fitur ini berpengaruh terhadap efisiensi pemeliharaan perangkat *IoT*?

1.3 Batasan Masalah

Batasan masalah dalam project ini adalah sebagai berikut:

1. Project ini hanya menggunakan sensor *DHT22* untuk membaca suhu dan kelembapan udara.
2. Mikrokontroler yang digunakan adalah *ESP32-S3*, tanpa dilakukan perbandingan dengan jenis mikrokontroler lainnya.
3. Platform *Cloud* yang digunakan untuk penyimpanan dan visualisasi data adalah *ThingsBoard Cloud* dengan protokol komunikasi *MQTT*.
4. Pengujian kestabilan transmisi data dilakukan dalam periode waktu tertentu menggunakan jaringan lokal (*WiFi*).

1.4 Tujuan

Tujuan yang ingin dicapai melalui project ini adalah sebagai berikut:

1. Mengimplementasikan sensor *DHT22* pada *ESP32-S3* untuk mengukur suhu dan kelembapan secara *real-time* dan menampilkannya melalui *ThingsBoard Cloud*.
2. Menganalisis kestabilan dan keandalan transmisi data dari *ESP32-S3* ke *Cloud* menggunakan protokol *MQTT*.
3. Menguji penerapan dan efektivitas pembaruan firmware *Over-The-Air (OTA)* pada perangkat *IoT* berbasis *ESP32-S3* sebagai solusi pemeliharaan jarak jauh yang efisien.

BAB II TINJAUAN PUSTAKA

2.1 *State of The Art*

2.1.1 Perkembangan *Internet of Things (IoT)* dalam Sistem Pemantauan.

Perkembangan *Internet of Things (IoT)* sangat pesat terutama dalam aplikasi sistem pemantauan, karena *IoT* memungkinkan sensor dan perangkat tersambung secara otomatis untuk mengumpulkan data secara *real-time*. Al-Fuqaha et al. (2015) menjelaskan bahwa *IoT* digerakkan oleh kemajuan teknologi sensor, protokol komunikasi, dan arsitektur jaringan, yang mendasari banyak aplikasi pemantauan modern.

Untuk mengatasi keterbatasan latensi dan beban jaringan, tren riset saat ini menekankan integrasi *edge computing* dengan *IoT*. Salah satu studi “*Integrating Edge Computing and IoT for Real-Time Air and Water Quality Monitoring Systems*” (2025) menyebutkan bahwa *edge devices* dapat memproses data dekat sumbernya, sehingga mengurangi keterlambatan pengiriman dan pemakaian *bandwidth* jaringan ke *Cloud*.

Selain itu, penelitian “*IoT-based edge computing (IoTEdC) for improved environmental monitoring*” (Roostaei et al., 2023) juga memperlihatkan bahwa struktur hibrid *IoT + edge* mampu meningkatkan efisiensi sistem pemantauan.

Di sisi lain, sistem pemantauan modern makin mengarah ke penggunaan sensor multimodal dan kecerdasan buatan di *edge*. Contohnya, Peng (2024) melakukan integrasi sensor-sensor berbeda dan *edge computing* dalam sistem pemantauan lingkungan, agar sistem bisa lebih responsif dan hemat energi.

Secara ringkas, perkembangan *IoT* dalam sistem pemantauan kini tidak hanya soal mengumpulkan data secara *real-time*, tapi juga soal bagaimana sistem bisa melakukan sebagian pemrosesan secara lokal (*di edge*) agar lebih cepat, stabil, dan efisien terhadap keterbatasan jaringan.

2.1.2 Implementasi *MQTT* dan Platform *Cloud* untuk Visualisasi Data

Dalam sistem *IoT* modern, penggunaan protokol *MQTT* dan platform *Cloud* menjadi sangat penting untuk visualisasi data secara *real-time* dan reliabel. Salah satu studi oleh Nitol Saha et al. (2024) mengembangkan sistem pemantauan energi dan lingkungan pada industri dengan menggunakan *MQTT* sebagai protokol transmisi data; data yang dikumpulkan kemudian divisualisasikan melalui aplikasi web berbasis *Cloud*, sehingga memudahkan pemantauan jarak jauh dan analisis performa.

Selanjutnya, penelitian “*A Distributed Architecture for MQTT Messaging: the Case of TQM*” (Shvaika et al., 2025) membahas bagaimana arsitektur MQTT yang terdistribusi mampu mendukung banyak perangkat dan beban trafik tinggi, yang sangat relevan bila sistem di skala besar dan harus tetap responsif.

Dalam konteks bangunan pintar, Montesclaros et al. (2021) menggunakan *MQTT* dan teknik visualisasi *UMAP* untuk menampilkan pola penggunaan daya (*power consumption*) sehingga pengguna bisa melihat tren penggunaan energi secara lebih intuitif. Sementara El-Basioni (2024) melalui pendekatan model konseptual membahas komponen-komponen yang dibutuhkan dalam penerapan *MQTT* di sistem *IoT*, termasuk struktur topik, *QoS*, dan bagaimana data diproses dari perangkat hingga platform *Cloud*. Dengan cara ini, visualisasi pada *dashboard Cloud* dimungkinkan dengan grafik, alarm, dan representasi historis yang membantu dalam mengambil keputusan operasional.

2.1.3 Sensor *DHT22* Untuk Pemantauan Suhu dan Kelembapan

Dalam Sensor *DHT22* sering digunakan dalam sistem *IoT* karena kemampuannya membaca suhu dan kelembapan secara digital dengan biaya relatif rendah. Dalam konteks pemantauan, sensor ini menjadi pilihan populer untuk aplikasi lingkungan, pertanian, dan ruangan. Menurut penelitian “*On the Evaluation of DHT22 Temperature Sensor for IoT Application*”, *DHT22* dievaluasi dengan membandingkannya terhadap instrumen industri, dan hasilnya menunjukkan bahwa meskipun ada keterbatasan, sensor ini masih cukup andal untuk aplikasi *IoT* sehari-hari (X et al., 2021).

Beberapa studi eksperimental juga menyoroti performa sensor ini dalam kondisi nyata. Misalnya, penelitian “*Performance Analysis Comparison of DHT11, DHT22 and DS18B20*” menunjukkan bahwa *DHT22* memiliki deviasi kecil dan akurasi yang cukup tinggi dibanding *DHT11*, meskipun *DS18B20* kadang lebih akurat dalam pengujian laboratorium.

Selain itu, dalam proyek prototipe *IoT* dengan *ESP32* yang mengirim data ke platform *ThingSpeak*, sensor *DHT22* digunakan dan berhasil menampilkan data suhu dan kelembapan secara *real time* pada *dashboard digital*. Hasil eksperimennya menunjukkan bahwa sistem berjalan stabil dalam pengujian ruangan biasa maupun ruangan ber-*AC*, meskipun ada fluktuasi pada nilai kelembapan di ruang biasa (Putra et al., 2024).

Dalam aplikasi greenhouse, studi “*The feasibility study: Accuracy and precision of DHT22 in measuring the temperature and humidity in the greenhouse*” menunjukkan bahwa penggunaan beberapa sensor *DHT22* secara bersamaan bisa menghasilkan pembacaan yang konsisten dan mendekati nilai acuan, sehingga sistem pemantauan lingkungan rumah kaca dapat berjalan dengan baik menggunakan *DHT22* (Penulis Y et al., 2023).

Dari beberapa penelitian di atas dapat disimpulkan bahwa sensor *DHT22* merupakan salah satu sensor yang cukup layak digunakan untuk pemantauan suhu dan kelembapan dengan biaya yang tidak terlalu tinggi. Sensor ini mudah diintegrasikan dengan mikrokontroler dan hasil pembacaannya cukup stabil untuk kondisi umum. Meskipun demikian, kalibrasi tetap diperlukan agar hasil pengukuran lebih akurat, terutama ketika digunakan pada lingkungan dengan perubahan suhu atau tingkat kelembapan yang signifikan.

2.1.4 Kestabilan Transmisi Data *IoT* dan Analisis Kinerja Jaringan

Dalam sistem *IoT*, kestabilan transmisi data dan performa jaringan sangat penting agar data sensor sampai secara tepat waktu dan tidak banyak hilang. Beberapa penelitian menunjukkan bahwa faktor seperti ukuran pesan, jumlah perangkat yang terkoneksi, serta pilihan protokol komunikasi berdampak besar ke latensi dan keandalan.

Misalnya, Puthiyidam & Joseph (2024) melakukan studi tentang bagaimana ukuran pesan dan jumlah klien mempengaruhi performa jaringan. Mereka menemukan bahwa dengan *payload* kecil dan pengguna sedikit, latensi bisa jauh lebih rendah dibanding jika *payload* besar dan banyak perangkat konek. Dalam pengujian mereka, ketika ada 150 klien dan pesan yang besar, latensinya meningkat hampir 60-70% dibanding kondisi dengan 50 klien.

Penelitian lain oleh Amirkhanoov et al. (2025) membandingkan tiga opsi protokol: *MQTT* over *TCP*, *MQTT* over *WebSocket*, dan *HTTP*, dalam *setting digital twin* menggunakan *ESP32*. Hasilnya menunjukkan bahwa *MQTT* over *TCP* memberikan latensi rata-rata paling rendah, sedangkan *MQTT* over *WebSocket* memberikan profil latensi yang lebih stabil, meski sedikit lebih tinggi dari *TCP*. *HTTP* paling besar latensinya dan varian latensinya juga paling besar di antara varian yang dibandingkan.

Dalam konteks protokol dan *QoS* (*Quality of Service*), studi “*Adaptive QoS Control for MQTT-SN*” membahas bahwa dengan mengatur *QoS* berdasarkan kondisi

jaringan (*delay & packet error rate*), perangkat bisa mengurangi rata-rata latensi dan memperbaiki jumlah *packet* yang berhasil sampai ke server.

2.1.5 Pembaruan Firmware *IoT* Menggunakan *Over-The-Air (OTA)*

Pembaruan firmware *Over-The-Air (OTA)* adalah metode yang memungkinkan perangkat *IoT* menerima *update* perangkat lunak secara jarak jauh tanpa perlu terhubung langsung ke komputer. Cara ini penting untuk menjaga sistem tetap aman dan *up-to-date*, terutama ketika perangkat tersebar di banyak lokasi. Menurut El Jaouhari & Bouvet (2022), *OTA* jadi salah satu solusi efektif dalam pemeliharaan perangkat *IoT* karena bisa menghemat waktu dan biaya.

Penelitian lain oleh Palmese et al. (2022) menekankan pentingnya pengaturan keamanan saat proses *update*, seperti enkripsi dan verifikasi digital, agar *file firmware* tidak disusupi. Sementara itu, Park et al. (2024) mengembangkan sistem pembaruan *OTA* yang ringan dan efisien untuk perangkat dengan kapasitas memori terbatas. Studi dari Malumbres et al. (2024) juga membahas *OTA* berbasis *LoRa* yang mendukung *update* banyak perangkat secara bersamaan dengan latensi rendah. Terakhir, Wei et al. (2023) menunjukkan bahwa penerapan *incremental OTA* bisa meningkatkan efisiensi karena hanya bagian *firmware* yang berubah yang dikirimkan.

Secara umum, *OTA* membuat proses pembaruan sistem *IoT* jadi lebih fleksibel, aman, dan efisien. Penerapannya sangat membantu dalam skala besar, terutama untuk proyek *IoT* yang terus berkembang seperti *monitoring* berbasis *ESP32*.

2.1.6 Research Gap dan Relevansi Penelitian

Berdasarkan beberapa studi sebelumnya, sistem *firmware Over-The-Air (FOTA)* pada perangkat *IoT* masih menghadapi kendala dari sisi efisiensi energi, keamanan proses pembaruan, dan kestabilan transmisi data terutama pada jaringan berdaya rendah seperti *LoRa* atau *Wi-Fi*. Sebagian besar penelitian berfokus hanya pada peningkatan kecepatan *update* atau keamanan data, tanpa mempertimbangkan keterbatasan daya serta kemampuan perangkat di lapangan secara menyeluruh.

Project ini menjadi relevan karena mengembangkan mekanisme pembaruan *firmware IoT* menggunakan *Over-The-Air (OTA)* yang efisien, aman, dan mudah diimplementasikan pada perangkat *IoT* berbasis *ESP32-S3*. Selain itu, project ini juga menekankan pada analisis kestabilan transmisi data serta optimasi protokol *MQTT* selama proses pembaruan, agar sistem mampu beroperasi secara andal dan *real-time* pada berbagai kondisi jaringan. Dengan pendekatan ini, project diharapkan dapat menjadi solusi yang lebih adaptif dan aplikatif untuk mendukung pengembangan sistem *IoT* modern berbasis komunikasi nirkabel.

2.2 Landasan Teori

2.2.1 *Internet of Things (IoT)*.

Internet of Things (IoT) adalah konsep yang menghubungkan berbagai perangkat fisik ke jaringan internet agar bisa saling bertukar data dan informasi tanpa harus dikendalikan langsung oleh manusia. Setiap perangkat *IoT* biasanya dilengkapi sensor, mikrokontroler, serta koneksi nirkabel seperti *WiFi* atau *Bluetooth* untuk mengirimkan data ke server atau platform *Cloud* secara otomatis.

Perkembangan *IoT* sendiri terus meningkat karena kemampuannya dalam membuat sistem menjadi lebih efisien dan terintegrasi. *IoT* sudah banyak diterapkan di berbagai bidang, seperti pemantauan lingkungan, *smart home*, pertanian, dan sistem industri. Selain itu, protokol komunikasi seperti *MQTT* dan *HTTP* sering digunakan karena ringan dan cocok untuk transmisi data berkapasitas kecil. Integrasi dengan platform *Cloud* juga penting agar data yang dikirim bisa disimpan, diolah, dan divisualisasikan secara *real-time* (Atzori et al., 2016).

Dalam project ini, konsep *IoT* dimanfaatkan untuk menghubungkan perangkat *ESP32-S3* dengan sensor *DHT22* yang datanya dikirim ke *ThingsBoard Cloud*. Tujuannya agar pengguna bisa memantau suhu dan kelembapan secara langsung dari jarak jauh, sekaligus mendukung mekanisme pembaruan firmware *OTA* agar sistem tetap efisien dan mudah diperbarui (Ray, 2016).

2.2.2 Mikrokontroler *ESP32-S3*

ESP32-S3 merupakan mikrokontroler yang dikembangkan oleh *Espressif Systems* dan menjadi salah satu seri terbaru dari keluarga *ESP32*. Mikrokontroler ini dirancang khusus untuk kebutuhan *Internet of Things (IoT)* dengan performa tinggi serta efisiensi daya yang baik. *ESP32-S3* memiliki prosesor *dual-core Xtensa LX7*, dukungan *AI acceleration*, serta konektivitas *Wi-Fi* dan *Bluetooth 5.0* yang membuatnya ideal untuk aplikasi yang membutuhkan transmisi data nirkabel secara kontinu (Espressif, 2022).

Selain itu, *ESP32-S3* juga dilengkapi dengan fitur keamanan seperti *flash encryption* dan *secure boot* yang penting untuk menjaga integritas firmware dan mencegah modifikasi tidak sah pada sistem. Kombinasi fitur keamanan dan efisiensi energi ini menjadikan *ESP32-S3* cocok digunakan dalam sistem pemantauan berbasis *Cloud*, termasuk untuk pengiriman data sensor dan pembaruan *firmware OTA*.

Beberapa penelitian menunjukkan bahwa *ESP32-S3* memiliki keunggulan dalam hal kestabilan transmisi data serta fleksibilitas integrasi dengan berbagai protokol *IoT* seperti *MQTT* dan *HTTP*. Dalam konteks project ini, *ESP32-S3* digunakan sebagai pusat pemrosesan dan pengiriman data dari sensor *DHT22* menuju platform *ThingsBoard Cloud*, sekaligus mendukung pembaruan *firmware* jarak jauh secara efisien (Nemlaha et al., 2023).

2.2.3 Ubuntu sebagai Lingkungan Pengembangan *IoT*

Ubuntu merupakan salah satu sistem operasi berbasis *Linux* yang banyak digunakan dalam pengembangan dan pengujian sistem *Internet of Things (IoT)*. Sistem ini dikenal karena stabilitasnya, sifatnya yang *open-source*, serta dukungan luas terhadap berbagai bahasa pemrograman dan pustaka untuk pengembangan aplikasi terhubung. Ubuntu menyediakan lingkungan kerja yang fleksibel bagi pengembang untuk melakukan kompilasi, pemrograman, dan pengujian perangkat *IoT* secara efisien (Wang et al., 2022).

Beberapa penelitian menunjukkan bahwa sistem operasi berbasis *Linux* memiliki performa yang baik dalam mendukung komunikasi antar perangkat melalui protokol seperti *MQTT*, *HTTP*, dan *CoAP*. Kemampuannya dalam mengatur sumber daya dan kompatibilitas dengan berbagai arsitektur perangkat keras membuatnya ideal untuk sistem berbasis mikrokontroler (Chaudhary et al., 2025).

Dalam konteks project ini, Ubuntu digunakan sebagai *development environment* utama untuk memprogram dan menguji perangkat *ESP32-S3*. Melalui Ubuntu, proses kompilasi *firmware*, konfigurasi koneksi ke *ThingsBoard Cloud*, serta pengujian protokol *MQTT* dapat dilakukan secara terintegrasi dan stabil. Dengan dukungan berbagai *tools open-source* seperti *Mosquitto MQTT Broker*, sistem ini memberikan fleksibilitas tinggi dalam melakukan pengujian dan pemeliharaan sistem *IoT*.

2.2.4 Firmware pada Sistem *IoT*

Firmware merupakan perangkat lunak tingkat rendah yang tertanam langsung pada mikrokontroler dan berfungsi untuk mengatur komunikasi, pemrosesan data, serta interaksi antara perangkat keras dan sistem *IoT* secara keseluruhan. Dalam konteks sistem *IoT*, *firmware* memiliki peran penting dalam menjaga kestabilan operasional, efisiensi energi, dan kemampuan perangkat dalam menjalankan fungsi otomatisasi.

Selain itu, *firmware* modern sering dirancang dengan pendekatan modular agar lebih mudah diperbarui dan diadaptasi terhadap perubahan kebutuhan sistem tanpa

harus mengganti perangkat keras. Pendekatan ini memungkinkan efisiensi dalam pengembangan, sekaligus meningkatkan portabilitas dan keandalan sistem *IoT* yang terhubung ke berbagai lingkungan dan platform (Farina et al., 2024).

Dalam konteks project ini, *firmware* dirancang untuk mengoordinasikan pembacaan data dari sensor *DHT22*, mengemas serta mengirimkan data *telemetry* melalui protokol *MQTT* ke *ThingsBoard Cloud*, dan juga mengelola pembaruan jarak jauh menggunakan mekanisme *Over-The-Air (OTA)*.

2.2.5 *Over-The-Air (OTA) Update*

Over-The-Air (OTA) Update merupakan metode pembaruan *firmware* yang dilakukan secara jarak jauh melalui koneksi nirkabel tanpa perlu intervensi langsung pada perangkat. Teknologi ini memungkinkan pengembang untuk memperbaiki *bug*, menambah fitur, atau meningkatkan keamanan perangkat *IoT* secara efisien dan terpusat. Dengan sistem *OTA*, proses pembaruan dapat dilakukan pada skala besar, sehingga sangat bermanfaat bagi perangkat yang tersebar luas di lapangan.

Dalam penerapannya, *OTA* membutuhkan sistem keamanan yang kuat agar proses transfer dan verifikasi *firmware* tidak mudah dimanipulasi. Selain itu, mekanisme manajemen memori dan optimasi ukuran file firmware juga penting agar pembaruan tetap efisien pada perangkat dengan sumber daya terbatas. Pada project ini, *OTA* digunakan untuk memastikan perangkat *IoT* dapat menerima pembaruan secara otomatis melalui platform *Cloud* tanpa perlu koneksi kabel atau intervensi manual (Park et al., 2025; Malumbres et al., 2024; Wei et al., 2024).

2.2.6 Sensor *DHT22*

Sensor *DHT22* digunakan untuk mengukur suhu dan kelembapan udara pada sistem *IoT*. Sensor ini memiliki akurasi yang cukup baik, yaitu sekitar $\pm 0.5^{\circ}\text{C}$ untuk suhu dan $\pm 2\%\text{RH}$ untuk kelembapan, sehingga cocok digunakan untuk pemantauan lingkungan skala menengah. Selain itu, *DHT22* mudah dihubungkan dengan mikrokontroler seperti *ESP32-S3* karena menggunakan komunikasi digital satu jalur (Yulizar et al., 2023).

Beberapa penelitian menunjukkan bahwa *DHT22* memberikan hasil pengukuran yang stabil dan konsisten, walaupun perlu dilakukan kalibrasi agar hasilnya tetap akurat terutama pada lingkungan dengan kelembapan tinggi (Wardani et al., 2024).

Dalam project ini, sensor *DHT22* berfungsi sebagai alat pembaca data suhu dan kelembapan yang dikirimkan secara *real-time* ke platform *ThingsBoard Cloud* melalui protokol *MQTT*.

2.2.7 MQTT sebagai Standar Protokol IoT

Message Queuing Telemetry Transport (MQTT) merupakan salah satu protokol komunikasi yang paling umum digunakan dalam sistem *Internet of Things (IoT)*. Protokol ini dirancang dengan konsep *publish-subscribe*, di mana data dikirim melalui *broker* untuk menjaga efisiensi dan mengurangi beban komunikasi antar perangkat. *MQTT* dikenal karena ringan, mudah diimplementasikan, dan cocok untuk perangkat dengan keterbatasan sumber daya seperti mikrokontroler (Amanlou, 2020).

Dalam konteks keamanan dan performa, beberapa penelitian menunjukkan bahwa *MQTT* memiliki keunggulan dibandingkan protokol lain seperti *CoAP*, terutama dalam hal kestabilan transmisi dan efisiensi *bandwidth* pada jaringan dengan keterbatasan koneksi (Seoane et al., 2021). Oleh karena itu, dalam project ini, protokol *MQTT* digunakan sebagai jalur utama komunikasi antara *ESP32-S3*, sensor *DHT22*, dan platform *ThingsBoard Cloud*, agar proses pengiriman data dan pembaruan sistem dapat berjalan secara andal dan efisien.

2.2.8 Platform ThingsBoard Cloud

Platform *ThingsBoard* adalah salah satu solusi *Cloud IoT open-source* yang sering dipilih untuk mengelola perangkat, menyimpan data, serta memvisualisasikan informasi secara *real-time*. Platform ini mendukung protokol seperti *MQTT*, *HTTP*, dan *CoAP* serta menyediakan *API REST* untuk integrasi aplikasi (Panagou et al., 2025).

Dalam fungsi *OTA*, *ThingsBoard* memungkinkan pengguna untuk mengunggah *packet firmware* ke repositori *OTA*, menetakannya ke profil perangkat tertentu, dan memonitor status *update* melalui *Dashboard*. Sistem ini akan memberi tahu perangkat terkait *update* yang tersedia dan menyediakan *API* spesifik protokol agar perangkat bisa mengunduh *firmware* tersebut.

Selain itu, dalam studi serialisasi data, Shvaika et al. (2024) menggunakan *ThingsBoard* sebagai contoh implementasi ketika mengintegrasikan protokol seperti *Protobuf* dan *JSON* agar data perangkat dapat di-*handle* secara efisien di platform. Ini menunjukkan fleksibilitas *ThingsBoard* dalam menangani berbagai format data dari perangkat *IoT* yang heterogen.

2.2.9 Bahasa Pemrograman untuk Embedded System (Rust/C++)

Dalam pengembangan sistem *IoT* berbasis mikrokontroler seperti *ESP32-S3*, pemilihan bahasa pemrograman berperan penting terhadap efisiensi dan keandalan sistem. Bahasa *C++* masih menjadi pilihan utama karena performanya yang tinggi

dan kompatibilitasnya dengan berbagai platform mikrokontroler. Selain itu, *C++* menawarkan kontrol penuh terhadap memori dan waktu eksekusi, yang sangat dibutuhkan pada sistem *real-time* (Plauska et al., 2023).

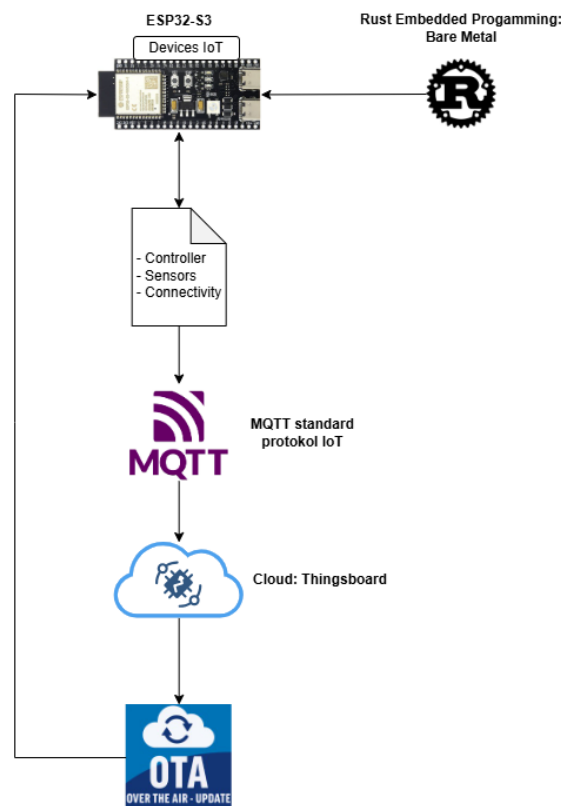
Sementara itu, bahasa *Rust* mulai banyak digunakan untuk pengembangan *embedded system* karena mengutamakan keamanan memori tanpa mengorbankan performa. *Rust* juga mendukung pemrograman paralel yang aman dan efisien, menjadikannya alternatif modern bagi *C++* dalam sistem *IoT* (Culic et al., 2022; Sharma et al., 2023).

Secara keseluruhan, pemilihan antara *C++* dan *Rust* bergantung pada kebutuhan sistem yang dikembangkan. Untuk sistem yang sudah mapan dan membutuhkan kompatibilitas luas, *C++* masih menjadi pilihan utama. Namun, jika fokusnya pada keamanan, stabilitas, dan pengelolaan memori otomatis, *Rust* dapat menjadi solusi yang lebih modern dan efisien dalam pengembangan sistem *IoT* masa kini.

BAB III METODOLOGI

3.1 Desain Sistem dan Arsitektur

Sistem *IoT* yang dikembangkan dalam project ini dirancang untuk melakukan pemantauan suhu dan kelembapan secara *real-time* menggunakan sensor *DHT22* yang terhubung dengan mikrokontroler *ESP32-S3*. Perangkat ini diprogram menggunakan bahasa *Rust Embedded (bare metal)*, yang memberikan efisiensi memori serta keamanan akses terhadap sumber daya perangkat keras. Alur kerja sistem ditunjukkan pada Gambar 3.1, yang menggambarkan arsitektur komunikasi antara perangkat *IoT*, protokol komunikasi, dan platform *Cloud*.



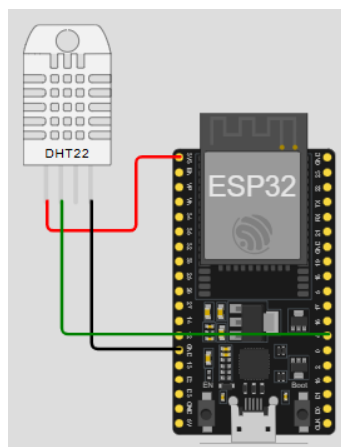
Gambar 3.1 Desain Sistem dan Arsitektur

Pada sistem ini, *ESP32-S3* berperan sebagai pusat pengendali utama yang mengakuisisi data dari sensor *DHT22* secara periodik, kemudian mengolah dan mengirimkan hasil pembacaan tersebut ke platform *ThingsBoard Cloud* melalui protokol *MQTT* sebagai standar komunikasi *IoT*. Data yang dikirim mencakup nilai suhu, kelembapan, serta *timestamp* dari *Real-Time Clock (RTC)* untuk keperluan analisis waktu tunda (*latency*) dan keandalan transmisi data. Protokol *MQTT* dipilih karena memiliki karakteristik ringan, efisien, dan andal dalam mengirim data antar perangkat *IoT* dengan *bandwidth* rendah.

Selain fungsi pemantauan, sistem ini juga dilengkapi fitur *Over-The-Air (OTA) firmware update*, yang memungkinkan pembaruan perangkat lunak dilakukan secara jarak jauh tanpa perlu akses fisik ke perangkat. Mekanisme *OTA* ini memastikan setiap *node IoT* dapat menerima *update* terbaru dengan cepat, sekaligus meminimalkan *downtime* sistem. Untuk menjaga keamanan komunikasi data, koneksi *MQTT* dikonfigurasi menggunakan autentikasi berbasis token dari *ThingsBoard*, sehingga hanya perangkat yang terdaftar yang dapat mengirimkan data ke server *Cloud*. Dengan kombinasi fitur tersebut, sistem menjadi lebih fleksibel, aman, mudah dikelola, dan dapat dioptimalkan secara berkelanjutan sesuai kebutuhan pengguna.

3.2 Perancangan Perangkat Keras (*Hardware Design*)

Perangkat keras yang digunakan dalam sistem ini terdiri dari mikrokontroler *ESP32-S3* sebagai unit utama, sensor *DHT22* untuk mengukur suhu dan kelembapan, serta beberapa komponen pendukung seperti kabel *jumper*, dan catu daya 5V. *ESP32-S3* dipilih karena memiliki kemampuan konektivitas *Wi-Fi* dan *Bluetooth*, serta prosesor *dual-core* yang mendukung pemrosesan data sensor dan komunikasi *IoT* secara efisien. Sensor *DHT22* digunakan karena mampu memberikan data suhu dan kelembapan dengan tingkat akurasi tinggi dan mudah diintegrasikan melalui antarmuka digital tunggal. Data sensor dibaca oleh *ESP32-S3* melalui pin data digital, kemudian dikirim ke platform *ThingsBoard Cloud* menggunakan protokol *MQTT*. Sistem juga dilengkapi modul *Real-Time Clock (RTC)* untuk memberikan penanda waktu (*timestamp*) pada setiap data sensor. Seluruh perangkat dirangkai pada *breadboard* untuk memudahkan proses pengujian, modifikasi, dan pengembangan sistem lebih lanjut. Gambar 3.2 berikut menunjukkan rangkaian koneksi (*wiring*) antara *ESP32-S3* dan *DHT22*.

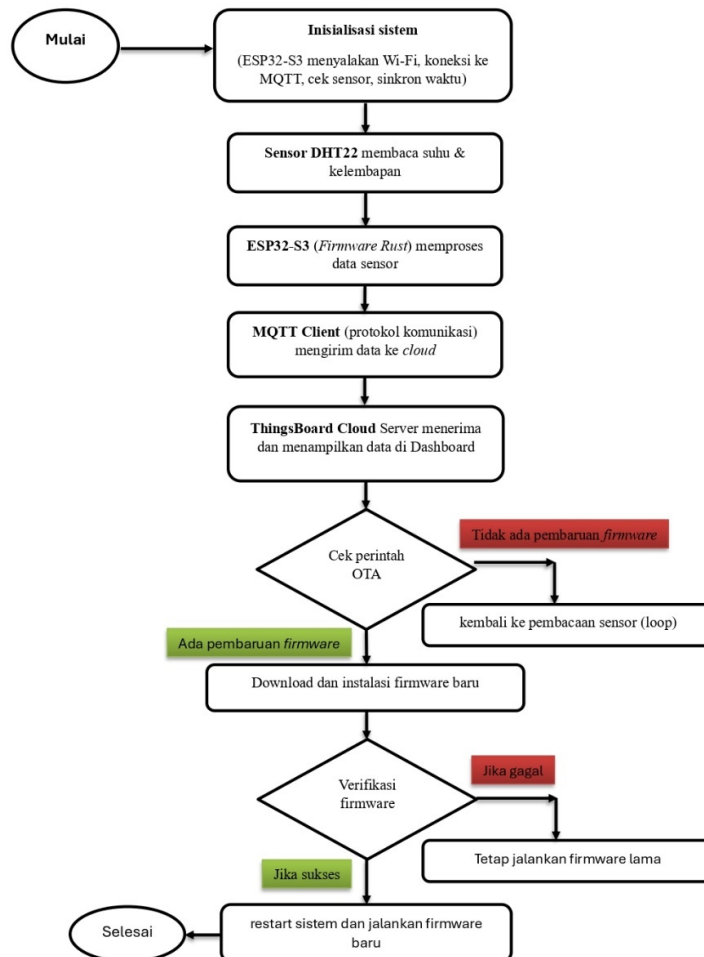


Gambar 3.2 *Wiring* antara *ESP32-S3* dan *DHT22*

Pin VCC *DHT22* dihubungkan ke 3.3V *ESP32-S3*, pin GND ke *ground*, dan pin Data ke pin digital GPIO4 *ESP32-S3*. Konfigurasi ini memastikan sensor dapat membaca suhu dan kelembapan dengan stabil, kemudian data diproses oleh mikrokontroler dan dikirim secara berkala ke server *ThingsBoard*.

3.3 Perangkat Lunak (*Software*)

Perangkat lunak pada sistem ini dirancang menggunakan bahasa pemrograman *Rust Embedded*, yang dijalankan secara *bare metal* pada mikrokontroler *ESP32-S3*. Perancangan ini melibatkan beberapa proses utama, yaitu inisialisasi sistem, akuisisi data sensor, pengiriman data melalui protokol *MQTT*, serta penerapan fitur pembaruan *firmware Over-The-Air (OTA)*.



Gambar 3.3 Alur kerja perangkat lunak pada sistem *IoT* berbasis *ESP32-S3*

Sistem diawali dengan proses inisialisasi, di mana *ESP32-S3* mengaktifkan koneksi *Wi-Fi*, melakukan sinkronisasi waktu, serta memastikan sensor *DHT22* terdeteksi dengan baik. Setelah itu, sensor *DHT22* akan membaca data suhu dan kelembapan, kemudian diproses oleh *ESP32-S3* dan dikirim ke platform *ThingsBoard Cloud* menggunakan *MQTT Client* sebagai protokol komunikasi standar *IoT*.

Data yang diterima oleh *ThingsBoard* ditampilkan secara *real-time* di *dashboard* sebagai hasil pemantauan sistem. Selain itu, perangkat lunak juga dilengkapi dengan mekanisme pemeriksaan perintah *OTA*, yang berfungsi untuk mendeteksi apakah terdapat pembaruan *firmware* baru di server *ThingsBoard*. Jika pembaruan tersedia, perangkat akan melakukan proses *download*, instalasi, dan verifikasi *firmware* secara otomatis. Bila instalasi berhasil, sistem akan *restart* dan menjalankan *firmware* baru; namun jika gagal, perangkat akan tetap menggunakan *firmware* lama agar sistem tetap stabil.

Bagian perangkat lunak sistem *IoT* ini dikembangkan menggunakan *Rust Embedded* dengan dua file utama, yaitu *main.rs* dan *cargo_OTA*. File *main.rs* berisi logika utama sistem, termasuk inisialisasi perangkat, pembacaan sensor *DHT22*, pengiriman data ke *ThingsBoard* melalui protokol *MQTT*, serta pengecekan status koneksi.

Sementara itu, file *cargo_OTA* digunakan untuk mendukung proses *Over-The-Air (OTA) update*, yang memungkinkan pembaruan *firmware* dilakukan secara jarak jauh tanpa intervensi fisik terhadap perangkat. Dalam proses ini, sistem secara otomatis mengunduh dan memverifikasi *firmware* terbaru dari *ThingsBoard Cloud* sebelum menggantikan *firmware* lama. Mekanisme ini memastikan sistem tetap fleksibel, efisien, dan mudah dikelola selama masa operasional.

3.4 Perancangan Tampilan dan Integrasi Sistem

3.4.1 Perancangan Program Utama (*main.rs*)

```
// -----  
// #[no_std] Architecture Implementation  
// Aplikasi ini menggunakan arsitektur 'no_std' dengan memanfaatkan  
modul 'core'  
// dan 'alloc' (untuk alokasi heap) sebagai pengganti Standard  
Library (std).  
// Komponen network/OS disediakan oleh esp-idf-svc, yang berjalan di  
atas  
// FreeRTOS.  
// -----
```

```

// -----
use core::{
    time::Duration,
    str, // Menggantikan std::str
    sync::atomic::{AtomicBool, Ordering}, // Menggantikan
std::sync::atomic
};

// Menggunakan alokasi heap dari modul 'alloc'
extern crate alloc;
use alloc::sync::Arc;

// Di lingkungan ESP-IDF Rust, 'thread' adalah
wrapper ergonomis yang membuat FreeRTOS Task.
// Ini adalah cara idiomatik untuk membuat Task baru
// di embedded Rust dengan ESP-IDF.
use thread;

use anyhow::{Result, Error};
// Chrono tidak memiliki versi no_std murni, tetapi
berfungsi baik di environment ESP-IDF.
use chrono::{Duration as ChronoDuration,
NaiveDateTime, Utc, TimeZone};
use dht_sensor::dht22::Reading;
use dht_sensor::DhtReading;
use esp_idf_svc::{
    eventloop::EspSystemEventLoop,
    hal::{
        delay::Ets,
        // Driver pin yang eksplisit untuk interaksi perangkat keras
langsung
        gpio::PinDriver,
        prelude::*,
    },
    log::EspLogger,

```

```

mqtt::client::,
nvs::EspDefaultNvsPartition,
sntp,
systemtime::EspSystemTime,
wifi::,
ota::EspOta,
http::client::EspHttpConnection,
};
use embedded_svc::{
    mqtt::client::QoS,
    http::client::Client,
    io::Read,
};
use heapless::String; // String yang ramah 'no_std'
use serde_json::json;

// -- Konfigurasi Firmware & Device --
const CURRENT_FIRMWARE_VERSION: &str =
"PaceP-s3-v2.0";
const TB_MQTT_URL: &str =
"mqtt://mqtt.thingsboard.cloud:1883";
const THINGSBOARD_TOKEN: &str =
"3IHhjZZYoPP1P2Mu0V61";

// -- FUNGSI C (Interaksi Sistem Langsung) --
// Deklarasi fungsi C/ESP-IDF untuk me-restart
sistem secara low-level.
extern "C" {
    fn esp_restart();
}

//

=====

// -- MQTT Client State (Global Access) --
// Digunakan untuk memungkinkan Task/Handler lain seperti OTA Task)
mengirim pesan.

```

```

static mut MQTT_CLIENT: Option<EspMqttClient<'static> = None;

fn get_mqtt_client() -> Option<&'static mut EspMqttClient<'static> {
    unsafe {
        MQTT_CLIENT.as_mut().map(|c| {
            // SAFETY: Transmute diperlukan karena callback C tidak tahu tentang
            lifetime Rust
            core::mem::transmute:::<&mut EspMqttClient<'_>, &mut
            EspMqttClient<'static>(c)
        })
    }
}

// -----
// -- PUBLISH FUNGSI (Memanfaatkan Global Client) --

// Fungsi untuk mengirim telemetry fw_state
fn publish_fw_state(state: &str) {
    let payload = format!("{}", "{}", "{}", "fw_state", state);
    log::info!(" Mengirim telemetry fw_state: {}", payload);

    if let Some(client) = get_mqtt_client() {
        if let Err(e) = client.publish(
            "v1/devices/me/telemetry",
            QoS::AtLeastOnce, // QoS 1: Memastikan pengiriman ke ThingsBoard
            false,
            payload.as_bytes(),
        ) {
            log::error!(" Gagal kirim fw_state {}: {:?}", state, e);
        }
    } else {
        log::error!(" MQTT client belum siap untuk kirim fw_state {}",
            state);
    }
}

```

```

}

// Mengirim versi firmware saat ini
fn publish_fw_version() {
    let payload = format!("{}", "", "", "fw_version", CURRENT_FIRMWARE_VERSION);
    log::info!(" Mengirim Current FW Version: {}", payload);

    if let Some(client) = get_mqtt_client() {
        if let Err(e) = client.publish(
            "v1/devices/me/telemetry",
            QoS::AtLeastOnce,
            false,
            payload.as_bytes(),
        ) {
            log::error!(" Gagal kirim fw_version: {:?}", e);
        }
    } else {
        log::error!(" MQTT client belum siap untuk kirim fw_version");
    }
}

// Fungsi untuk mengirim RPC response ke ThingsBoard
fn send_rpc_response(request_id: &str, status: &str) {
    let topic = format!("v1/devices/me/rpc/response/{}", request_id);
    log::info!(" Mengirim RPC response ke: {}", topic);

    let payload = format!("{}", "", "", "status", status);

    if let Some(client) = get_mqtt_client() {
        if let Err(e) = client.publish(
            topic.as_str(),
            QoS::AtLeastOnce,

```

```

false,
payload.as_bytes(),
) {
log::error!(" Gagal kirim RPC response:  {:?}", e);
}
} else {
log::error!(" MQTT client belum siap untuk kirim RPC response");
}
}

// -----
// -- OTA PROCESS FUNCTION (Berjalan di FreeRTOS Task terpisah) --
fn ota_process(url: alloc::string::String) {
    // NOTE: argumen diubah menjadi String (owned) agar dapat dipindah
    (move) ke Task baru
    log::info!(" Mulai OTA dari URL: {}", url);
    publish_fw_state("DOWNLOADING");

    // Jeda singkat untuk memastikan pesan "DOWNLOADING" terkirim oleh
    Task MQTT
    thread::sleep(Duration::from_millis(500));

    match EspOta::new() {
    Ok(mut ota) => {
        let http_config = esp_idf_svc::http::client::Configuration {
            ..Default::default()
        };

        let conn = match EspHttpConnection::new(&http_config) {
            Ok(c) => c,
            Err(e) => {
                log::error!(" Gagal buat koneksi HTTP: {:?}", e);
                publish_fw_state("FAILED");
            }
        };
    }
}

```



```

return;
}
};

let mut client = Client::wrap(conn);
// Menggunakan &url.as_str() karena type data url sekarang adalah
String (alloc)
let request = match client.get(url.as_str()) {
Ok(r) => r,
Err(e) => {
log::error!(" Gagal buat HTTP GET: {:?}", e);
publish_fw_state("FAILED");
return;
}
};

let mut response = match request.submit() {
Ok(r) => r,
Err(e) => {
log::error!(" Gagal submit request: {:?}", e);
publish_fw_state("FAILED");
return;
}
};

if response.status() < 200 || response.status() >= 300 {
log::error!(" HTTP request gagal. Status code: {}",
response.status());
publish_fw_state("FAILED");
return;
}

let mut buf = [0u8; 1024];
let mut update = match ota.initiate_update() {

```

```

Ok(u) => u,
Err(e) => {
  log::error!("Gagal init OTA: {:?}", e);
  publish_fw_state("FAILED");
  return;
}

};

loop {
  match response.read(&mut buf) {
    Ok(0) => break,
    Ok(size) => {
      if let Err(e) = update.write(&buf[..size]) {
        log::error!("Gagal tulis OTA: {:?}", e);
        publish_fw_state("FAILED");
        return;
      }
    }
    Err(e) => {
      log::error!("HTTP read error: {:?}", e);
      publish_fw_state("FAILED");
      return;
    }
  }
}

publish_fw_state("VERIFYING");

if let Err(e) = update.complete() {
  log::error!("OTA complete error: {:?}", e);
  publish_fw_state("FAILED");
  return;
}

```

```

log::info!(" OTA selesai, restart...");
publish_fw_state("SUCCESS");

// Jeda 1 detik agar pesan SUCCESS terkirim sebelum restart
thread::sleep(Duration::from_secs(1));

// Panggilan fungsi C untuk restart sistem
unsafe { esp_restart(); }
}
Err(e) => {
log::error!(" Gagal init OTA: {:?}", e);
publish_fw_state("FAILED");
}
}
}

// -----
// -- MAIN APPLICATION (Berjalan di Main Task/Core 1) --
fn main() -> Result<(), Error> {
// -- Inisialisasi dasar (Sistem & Log) --
esp_idf_svc::sys::link_patches();
EspLogger::initialize_default();
log::info!(" Program dimulai, Versi FW: {} - FIRMWARE AKTIF!",
CURRENT_FIRMWARE_VERSION);

// -- Inisialisasi perangkat --
let peripherals = Peripherals::take().unwrap();
let sysloop = EspSystemEventLoop::take()?;
let nvs = EspDefaultNvsPartition::take().unwrap();

// -- Konfigurasi WiFi --
let mut wifi = EspWifi::new(peripherals.modem, sysloop.clone(),

```

```

Some(nvs.clone()))?;

let mut ssid: String<32> = String::new();
ssid.push_str("No Internet").unwrap(); // Ganti dengan SSID Anda

let mut pass: String<64> = String::new();
pass.push_str("tertolong123").unwrap(); // Ganti dengan Password
Anda

let wifi_config = Configuration::Client(ClientConfiguration {
    ssid,
    password: pass,
    auth_method: AuthMethod::WPA2Personal,
    ..Default::default()
});

log::info!(" Koneksi WiFi dimulai...");
wifi.set_configuration(&wifi_config)?;
wifi.start()?;
wifi.connect()?;

// Tunggu sampai WiFi benar-benar aktif (Proses Blocking)
while !wifi.is_connected().unwrap() {
    log::info!(" Menunggu koneksi WiFi...");
    thread::sleep(Duration::from_secs(1));
}
log::info!(" WiFi terhubung!");

// -- Manajemen Sumber Daya (Pinning Services) --
let _wifi = alloc::boxed::Box::leak(alloc::boxed::Box::new(wifi));
let _sysloop = alloc::boxed::Box::leak(alloc::boxed::Box::new(
sysloop));
let _nvs = alloc::boxed::Box::leak(alloc::boxed::Box::new(nvs));

// -- Sinkronisasi waktu via NTP --

```

```

log::info!(" Sinkronisasi waktu NTP...");
let sntp = sntp::EspSntp::new_default()?;

// Tunggu sinkronisasi NTP (Proses Blocking)
loop {
    if sntp.get_sync_status() == sntp::SyncStatus::Completed {
        log::info!(" Waktu berhasil disinkronkan dari NTP");
        break;
    }
    log::info!(" Menunggu sinkronisasi NTP...");
    thread::sleep(Duration::from_secs(1));
}

// Delay tambahan agar waktu stabil
thread::sleep(Duration::from_secs(5));

// -- Konfigurasi MQTT (ThingsBoard Cloud) --
let mqtt_config = MqttClientConfiguration {
    client_id:  Some("esp32-rust-ota"),
    username:  Some(THINGSBOARD_TOKEN),
    password:  None,
    keep_alive_interval:  Some(Duration::from_secs(30)),
    ..Default::default()
};

let mqtt_connected = Arc::new(AtomicBool::new(false));

// -- MQTT Callback Handler (Event-Driven) --
let mqtt_callback = {
    let mqtt_connected = mqtt_connected.clone();

    move |event:  EspMqttEvent<'_ | {
        use esp_idf_svc::mqtt::client::EventPayload;

```

```

match event.payload() {
  EventPayload::Connected(_) => {
    log::info!(" MQTT connected");
    mqtt_connected.store(true, Ordering::SeqCst);
  }
  EventPayload::Received { topic, data, .. } => {
    // Menggunakan core::str::from_utf8
    let payload_str = str::from_utf8(data).unwrap_or("");
    log::info!(" Payload diterima. Topic:  {?}, Data:  {}", topic,
payload_str);

    if let Some(topic_str) = topic {
      if topic_str.starts_with("v1/devices/me/rpc/request/") {
        let parts:  alloc::vec::Vec<&str> = topic_str.split('/').collect();
        if let Some(request_id) = parts.last() {
          log::info!(" Menerima RPC request_id:  {}", request_id);

          if let Ok(json) = serde_json::from_str::<serde_json::Value>(payload_str)
          {

            let ota_url_owned = json.get("params")
              .and_then(|p| p.get("ota_url"))
              .and_then(|u| u.as_str())
              // Menggunakan alloc::string::ToString untuk cloning
              .map(|s| s.to_string());

            if let Some(url) = ota_url_owned {
              log::info!(" Dapat OTA URL dari RPC: {}", url);
              send_rpc_response(request_id, "success");

              // Pembuatan Task (Thread) dengan stack size eksplisit

```

```

thread::Builder::new()
  .name("ota_task")
  .stack_size(10 * 1024)
  .spawn(move || {
    ota_process(url); // Menerima String yang di-move
  })
  .expect("Gagal membuat FreeRTOS Task OTA");

return;
} else {
  log::warn!(" Payload RPC diterima, tetapi öta_url tidak ditemukan.");
}
} else {
  log::error!(" Gagal mem-parse JSON payload RPC.");
}
send_rpc_response(request_id, "failure");
}
}
}
}
EventPayload::Disconnected => {
  log::warn!(" MQTT Disconnected!");
  mqtt_connected.store(false, Ordering::SeqCst);
}
_ => {}
}
}
};

// -- Inisialisasi MQTT Client --
let client = loop {
  let res = unsafe {

```

```

EspMqttClient::new_nonstatic_cb(
    TB_MQTT_URL,
    &mqtt_config,
    mqtt_callback.clone(),
)
};

match res {
    Ok(c) => {
        unsafe { MQTT_CLIENT = Some(c) };

        if let Some(c_ref) = get_mqtt_client() {
            while !mqtt_connected.load(Ordering::SeqCst) {
                log::info!(" Menunggu MQTT connect...");
                thread::sleep(Duration::from_millis(500));
            }
            log::info!(" MQTT Connected!");

            c_ref.subscribe("v1/devices/me/rpc/request/+",
                QoS::AtLeastOnce).unwrap();

            publish_fw_version();
            publish_fw_state("IDLE");

            break c_ref;
        } else {
            log::error!(" Gagal mendapatkan referensi client setelah koneksi.");
            thread::sleep(Duration::from_secs(5));
            continue;
        }
    }
    Err(e) => {
        log::error!(" MQTT connect gagal:  {?}", e);
    }
}

```



```

thread::sleep(Duration::from_secs(5));
}
}
};

// -- Inisialisasi sensor DHT22 (GPIO4) --
let mut pin = PinDriver::input_output_od(peripherals.pins.gpio4)?;
let mut delay = Ets;

// -- Loop utama kirim data (Core Task) --
loop {
    // Ambil waktu sekarang dari SystemTime
    let systime = EspSystemTime {}.now();
    let secs = systime.as_secs() as i64;
    let nanos = systime.subsec_nanos();

    let naive = NaiveDateTime::from_timestamp_opt(secs, nanos as u32)
        .unwrap_or(NaiveDateTime::from_timestamp_opt(0, 0).unwrap());

    // Konversi ke WIB (UTC + 7 jam)
    let utc_time = Utc.from_utc_datetime(&naive);
    let wib_time = utc_time + ChronoDuration::hours(7);
    let ts_millis = naive.and_utc().timestamp_millis();
    let send_time_str = wib_time.format("%Y-%m-%d
%H:%M:%S").to_string();

    // Baca sensor DHT22
    match Reading::read(&mut delay, &mut pin) {
        Ok(Reading {
            temperature,
            relative_humidity,
        }) => {
            // Siapkan payload JSON

```

```

let payload = json!({
    "send_time": send_time_str,
    "ts": ts_millis,
    "temperature": temperature,
    "humidity": relative_humidity
});

let payload_str = payload.to_string();

// Kirim data Telemetry
match client.publish(
    "v1/devices/me/telemetry",
    QoS::AtLeastOnce,
    false,
    payload_str.as_bytes(),
    ) {
    Ok(_) => log::info!(" Data terkirim (T: {}°C, H: {}%): {}",
temperature, relative_humidity, payload_str),
    Err(e) => log::error!(" Gagal publish ke MQTT: {:?}", e),
    }
}

Err(e) => log::error!(" Gagal baca DHT22: {:?}", e),
}

// Delay 60 detik (mengosongkan CPU untuk Task lain)
thread::sleep(Duration::from_secs(60));
}
}

```

Program utama dikembangkan menggunakan bahasa pemrograman *Rust* dengan pendekatan *bare-metal* (`#[no_std]`), sehingga dapat berjalan langsung pada mikrokontroler *ESP32-S3* tanpa sistem operasi tambahan. Pendekatan ini membuat sistem lebih ringan dan efisien dalam penggunaan memori.

Pada tahap awal, dilakukan inisialisasi perangkat keras seperti `GPIO`, *driver delay*, serta konfigurasi *Wi-Fi* menggunakan pustaka *EspWifi*. Setelah perangkat berhasil terhubung ke jaringan, sistem melakukan sinkronisasi waktu melalui *NTP* (*Network Time Protocol*) agar waktu pengiriman data sesuai dengan waktu aktual.

Selanjutnya, sistem menginisialisasi koneksi ke broker *MQTT* milik *ThingsBoard Cloud* menggunakan token autentikasi perangkat. Setelah koneksi berhasil, perangkat mulai membaca nilai suhu dan kelembapan dari sensor *DHT22* yang terhubung ke pin `GPIO4`. Data hasil pembacaan sensor dikirim ke *ThingsBoard* dalam format *JSON* dengan parameter:

- `send_time`: waktu lokal perangkat saat data dikirim,
- `ts`: timestamp dari sistem dalam milidetik,
- `temperature`: nilai suhu dalam $^{\circ}\text{C}$,
- `humidity`: nilai kelembapan dalam %.

Payload data dikirim ke topik `v1/devices/me/telemetry` setiap 60 detik. Proses ini menghasilkan pesan di terminal seperti “data terkirim”, yang menandakan bahwa proses pengiriman data telah berhasil.

Selain itu, perangkat juga dilengkapi dengan fitur *Over-The-Air* (*OTA*) *Firmware Update*, yang memungkinkan pembaruan *firmware* dilakukan langsung melalui *ThingsBoard* tanpa kabel. Ketika pembaruan tersedia, sistem akan menerima perintah dari *server*, mengunduh *firmware* baru, menuliskannya ke memori *flash*, dan menampilkan pesan “OTA selesai, restart...” sebelum melakukan *restart* otomatis untuk memuat *firmware* versi terbaru.

Dengan penerapan program ini, sistem terbukti mampu melakukan pembacaan sensor, pengiriman data, serta pembaruan *firmware* secara otomatis melalui jaringan.

3.4.2 Konfigurasi Build dan Depensi(*Cargo.toml*)

```
[package]
name = "streamdht"
version = "0.1.0"
authors = ["Aufa"]
edition = "2021"
resolver = "2"
rust-version = "1.77"

[[bin]]
name = "streamdht"
harness = false # Do not use the built-in cargo test harness ->
resolves rust-analyzer errors

[profile.release]
opt-level = "s"
```

```

[profile.dev]
debug = true # Symbols are nice and they don't increase the size on
Flash
opt-level = "z"

[features]
default = []
experimental = ["esp-idf-svc/experimental"]

[dependencies]
log = "0.4"
esp-idf-svc = "0.51"
rand = "0.8"
anyhow = "1.0"
heapless = "0.8"
serde_json = "1.0"
dht-sensor = "0.2"
chrono = { version = "0.4", features = ["clock"] }
embedded-svc = { version = "0.28.1" } # FIX: Disinkronkan dengan
esp-idf-svc 0.51

# -- Optional Embassy Integration --
# esp-idf-svc = { version = "0.51", features = ["critical-section",
"embassy-time-driver", "embassy-sync"] }

# If you enable embassy-time-driver, you MUST also add one of:
# embassy-time = { version = "0.4.0", features = ["generic-queue-8"]
}
# embassy-executor = { version = "0.7", features =
["executor-thread", "arch-std"] }

# -- Temporary workaround for embassy-executor < 0.8 --
# esp-idf-svc = { version = "0.51", features =
["embassy-time-driver", "embassy-sync"] }
# critical-section = { version = "1.1", features = ["std"],
default-features = false }

[build-dependencies]
embuild = "0.33"

```

Bagian ini berisi pengaturan utama proyek *Rust* yang digunakan untuk membangun sistem di mikrokontroler *ESP32-S3*. File `Cargo.toml` mengatur identitas proyek, versi, serta pustaka-pustaka (dependensi) yang dibutuhkan agar program bisa dijalankan dengan baik.

Pada bagian `[package]`, ditulis nama proyek `streamdht`, versi program, nama penulis, serta edisi Rust yang dipakai. Baris `harness = false` digunakan supaya Cargo tidak menjalankan mode pengujian otomatis, karena program ini langsung dijalankan di mikrokontroler, bukan di komputer biasa.

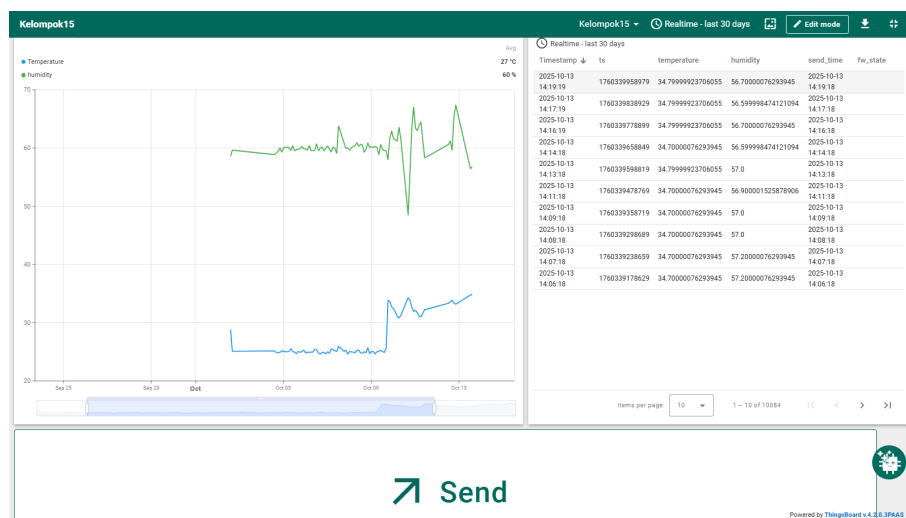
Bagian `[profile]` mengatur mode kompilasi. Mode `release` menggunakan optimasi agar ukuran file hasil *build* lebih kecil, sedangkan mode `dev` digunakan saat pengujian agar lebih mudah melihat log atau error. Kemudian di bagian

[dependencies], terdapat beberapa pustaka penting seperti:

- `esp-idf-svc` untuk koneksi Wi-Fi, MQTT, dan pengelolaan sistem.
- `dht-sensor` untuk membaca data suhu dan kelembapan dari sensor DHT22.
- `serde_json` untuk membentuk data JSON sebelum dikirim ke ThingsBoard.
- `chrono` untuk mengatur waktu dan *timestamp*.
- `anyhow` dan `log` untuk membantu proses *debug* dan pencatatan *log* program.

Selain itu, ada juga `embuild` di bagian [build-dependencies] yang membantu proses kompilasi otomatis ke dalam format yang bisa dijalankan di *ESP32*. Secara keseluruhan, file `Cargo.toml` ini berfungsi untuk memastikan semua pustaka dan pengaturan yang dibutuhkan sudah lengkap, sehingga proses *build* berjalan lancar dan program bisa dikompilasi ke perangkat tanpa *error*.

3.4.3 Rancangan Tampilan *Dashboard ThingsBoard*



Gambar 3.4.3 Rancangan tampilan *Dashboard Monitoring* pada *Thingsboard*

Bagian ini menampilkan rancangan *dashboard* ThingsBoard Cloud yang digunakan untuk memantau data hasil pembacaan sensor DHT22 secara *real-time*. *Dashboard* ini dibuat untuk memudahkan pengguna dalam mengamati perubahan nilai suhu dan kelembapan yang dikirim oleh perangkat ESP32-S3 setiap 60 detik.

Tampilan *dashboard* terdiri dari dua komponen utama, yaitu *grafik time series* dan *tabel data telemetry*. Grafik *time series* digunakan untuk memperlihatkan tren perubahan suhu dan kelembapan terhadap waktu dalam bentuk garis berwarna, di mana garis biru menunjukkan suhu (*temperature*) dan garis hijau menunjukkan kelembapan (*humidity*). Sementara itu, tabel di sisi kanan berisi data lengkap seperti *timestamp*, *temperature*, *humidity*.

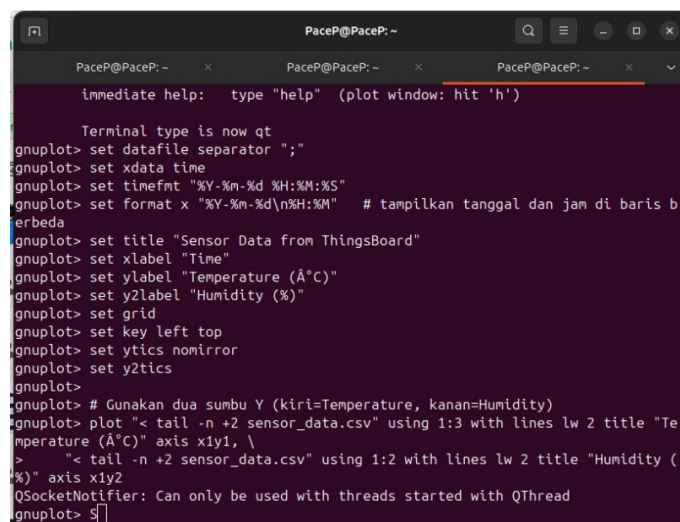
`send_time`, dan `fw_state` yang dikirim melalui protokol *MQTT* ke server *ThingsBoard*.

Dari tampilan tersebut, pengguna dapat melihat data sensor secara langsung tanpa harus mengakses perangkat fisik. Informasi yang dikirim dalam format JSON ini memungkinkan proses *monitoring* berjalan otomatis, sehingga sistem menjadi lebih efisien dan mudah dikembangkan untuk kebutuhan *IoT* berbasis *Cloud*.

3.4.4 Perintah Gnuplot untuk Visualisasi Data Sensor dan Analisis Latensi

Pada tahap ini dilakukan proses visualisasi data menggunakan *Gnuplot* yang dijalankan melalui terminal *Ubuntu*. *Gnuplot* digunakan untuk menampilkan grafik hasil pembacaan suhu (*temperature*) dan kelembapan (*humidity*) yang dikirim dari sensor ke platform *ThingsBoard*, serta untuk menganalisis latensi pengiriman data antara sensor dan server.

Langkah awal yang dilakukan yaitu membuka terminal, lalu menjalankan *Gnuplot* dengan mengetikkan perintah `gnuplot`. Setelah itu, sistem dikonfigurasi agar dapat membaca file data CSV hasil pengukuran sensor dengan memisahkan nilai-nilai menggunakan tanda “,”. Berikut adalah contoh perintah yang digunakan untuk menampilkan grafik suhu dan kelembapan:

A screenshot of a terminal window titled 'PaceP@PaceP: ~'. The terminal shows a series of commands for configuring and running Gnuplot. The commands include setting the datafile separator to semicolon, setting xdata to time, setting timefmt to '%Y-%m-%d %H:%M:%S', setting format x to '%Y-%m-%d\n%H:%M', setting title to 'Sensor Data from ThingsBoard', setting xlabel to 'Time', setting ylabel to 'Temperature (°C)', setting y2label to 'Humidity (%)', setting grid, setting key left top, setting ytics nomirror, setting y2tics, and finally plotting two data series from 'sensor_data.csv' using 'x1y1' and 'x1y2' axes. The plot window is titled 'Sensor Data from ThingsBoard'.

```
PaceP@PaceP: ~
immediate help: type "help" (plot window: hit 'h')

Terminal type is now qt
gnuplot> set datafile separator ";"
gnuplot> set xdata time
gnuplot> set timefmt "%Y-%m-%d %H:%M:%S"
gnuplot> set format x "%Y-%m-%d\n%H:%M" # tampilkan tanggal dan jam di baris b
erbeda
gnuplot> set title "Sensor Data from ThingsBoard"
gnuplot> set xlabel "Time"
gnuplot> set ylabel "Temperature (°C)"
gnuplot> set y2label "Humidity (%)"
gnuplot> set grid
gnuplot> set key left top
gnuplot> set ytics nomirror
gnuplot> set y2tics
gnuplot>
gnuplot> # Gunakan dua sumbu Y (kiri=Temperature, kanan=Humidity)
gnuplot> plot "< tail -n +2 sensor_data.csv" using 1:3 with lines lw 2 title "Te
mperature (°C)" axis x1y1, \
> "< tail -n +2 sensor_data.csv" using 1:2 with lines lw 2 title "Humidity (
%)" axis x1y2
QSocketNotifier: Can only be used with threads started with QThread
gnuplot> S
```

Gambar 3.4.4 Perintah *Gnuplot* untuk Visualisasi Data Sensor

Perintah di atas digunakan untuk menampilkan grafik dengan dua sumbu Y. Hasil grafik menunjukkan perubahan nilai suhu dan kelembapan dari waktu ke waktu berdasarkan data yang diterima oleh *ThingsBoard*.

Selain itu, dilakukan juga analisis latensi untuk mengetahui jeda waktu antara pembacaan data di sensor dan penerimaan data di server. Proses ini juga dijalankan melalui terminal dengan *Gnuplot* menggunakan file `latency_ready.csv`. Berikut perintahnya:

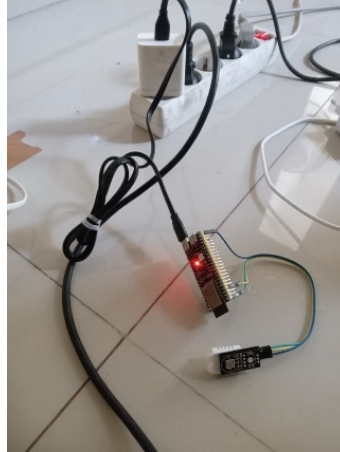
```
PaceP@PaceP: ~  
gnuplot> set key top left  
gnuplot> plot "latency_ready.csv" using 1:2 with linespoints lw 2 lc rgb "orange"  
" title "Latency"  
warning: Skipping data file with no valid points  
^  
x range is invalid  
gnuplot> head -n 5 latency_ready.csv | cat -A  
^  
invalid command  
gnuplot> reset  
gnuplot> set datafile separator ";"  
gnuplot> set decimalsign locale  
decimal_sign in locale is .  
gnuplot> set title "Latency Analysis: Sensor RTC vs ThingsBoard"  
gnuplot> set xlabel "Sample Index"  
gnuplot> set ylabel "Latency (seconds)"  
gnuplot> set grid  
gnuplot> set key top left  
gnuplot> plot "latency_ready.csv" using 1:2 with linespoints lw 2 lc rgb "orange"  
" title "Latency"  
gnuplot> S
```

Gambar 3.4.4.1 Perintah *Gnuplot* untuk Grafik Latensi

Dari hasil grafik latensi yang dihasilkan, dapat diamati variasi waktu tunda (*delay*) dalam satuan detik antara sensor dan server *ThingsBoard*. Nilai latensi yang relatif kecil dan stabil menunjukkan bahwa proses komunikasi data berjalan dengan baik tanpa adanya gangguan jaringan yang berarti.

BAB IV ANALISIS DAN HASIL PEMBAHASAN

4.1 Hasil Implementasi Sistem



Gambar 4.1 *ESPS3-32* terhubung dengan adaptor dan sensor *DHT22*

```
Sep 18 22:30
salmanaufo@salmanaufo-VirtualBox: ~/dht_OTA

I (700) wifi_init: tcp nss: 1440
I (710) wifi_init: WiFi IRAM OP enabled
I (715) wifi_init: WiFi 802.11n enabled
I (720) phy_init: phy_version 780,8582a7fd, Feb 10 2025,20:13:11
I (760) phy_init: Saving new calibration data due to checksum failure or outdated calibration data, mode(0)
I (830) wifi_mode: sta (98:88:e0:03:ae:08)
I (830) wifi_enable: tsf
I (840) dht_sensor: Menunggu koneksi WiFi...
I (840) dht_sensor: Menunggu koneksi WiFi...
I (840) dht_sensor: Menunggu koneksi WiFi...
I (3250) wifi:new: <3,0>, old: <1,0>, ap: <255,255>, sta: <3,0>, prof: 1, snd_ch_cfg: 0x0
I (3250) wifi:state: init -> auth (0x0)
I (3270) wifi:state: auth -> assoc (0x0)
I (3300) wifi:state: assoc -> run (0x0)
I (3300) wifi:connected with vivo, aid = 3, channel 3, BQ70, bssid = 36:77:3c:5a:8f:ad
I (3300) wifi:security: WPA2-PSK, phy: bgn, rssi: -69
I (3300) wifi:pm start, type: 1

I (3300) wifi:dp: 1, bi: 102400, li: 3, scale listen interval from 307200 us to 307200 us
I (3310) wifi:set rx beacon pti, rx_bcn_pti: 0, bcn_timeout: 25000, nt_pti: 0, nt_time: 10000
I (3330) wifi:dp: 2, bi: 102400, li: 4, scale listen interval from 307200 us to 409600 us
I (3330) wifi:dp's beacon interval = 102400 us, DTIM period = 2
I (3330) wifi:ba-add:idx:0 (lfr:0, 36:77:3c:5a:8f:ad), tid:0, ssn:0, winSize:64
I (3840) dht_sensor: Terhubung ke WiFi
I (4320) esp_mqtt_humbers: sta ip: 192.168.43.105, mask: 255.255.255.0, gw: 192.168.43.231
I (4840) dht_sensor: Terhubung ke broker MQTT ThingsBoard Cloud
I (6840) dht_sensor: Event MQTT: BeforeConnect
I (6840) gpio: GPIO[41] (output: 0) (pullup: 0) (mode: 0) (pin: 41) (pullup: 0) (pulldown: 0) (intr: 0)
I (7320) wifi:ba-add:idx:1 (lfr:0, 36:77:3c:5a:8f:ad), tid:1, ssn:0, winSize:64
I (7720) dht_sensor: Event MQTT: Connected(false)
I (7730) dht_sensor: Data terkirim: {"humidity":68.30000305175781,"temperature":31.100000381469727}
I (7750) dht_sensor: Data terkirim: {"humidity":69.00000305175781,"temperature":30.5}
I (7750) dht_sensor: Data terkirim: {"humidity":70.0,"temperature":30.5}
```

Gambar 4.1.1 Tampilan konektivitas di terminal *Ubuntu*

Implementasi sistem dilakukan dengan merangkai *ESP32-S3*, sensor *DHT22*, serta komponen pendukung seperti kabel *jumper*, dan adaptor 5V sebagai sumber daya utama. Mikrokontroler dihubungkan ke *laptop* menggunakan kabel USB untuk proses pemrograman *firmware* berbasis *Rust*.

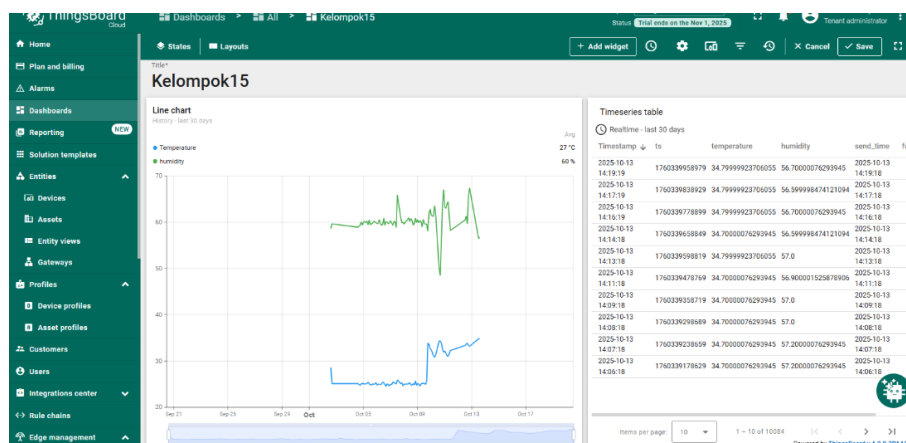
Proses implementasi dimulai dengan *build* proyek menggunakan perintah *cargo build*, kemudian dilakukan *upload* program melalui *cargo flash* agar *firmware* dapat dijalankan langsung pada mikrokontroler. Setelah proses *flashing* berhasil, sistem menampilkan *log*:

eksekusi di terminal Ubuntu. Dari hasil pengujian, terminal menampilkan beberapa status penting seperti:

- “*Connected to Wi-Fi*” → menunjukkan bahwa perangkat berhasil tersambung ke jaringan.
- “*MQTT broker connected*” → sistem telah terhubung dengan server *ThingsBoard Cloud* melalui protokol *MQTT*.
- “*Data terkirim (humidity, temperature)*” → menandakan data sensor telah berhasil dipublikasikan ke *Cloud*.
- “Selain itu, hasil implementasi menunjukkan bahwa perangkat juga mampu melakukan pembaruan firmware melalui mekanisme *OTA (Over-The-Air)*.”

Secara keseluruhan, hasil implementasi menunjukkan bahwa proses komunikasi dan pembacaan sistem berjalan stabil tanpa adanya *error* pada tahap pengujian.

4.2 Hasil Pengujian Pengiriman Data Sensor



Gambar 4.2 Dashboard *ThingsBoard Cloud* menampilkan grafik suhu dan kelembapan

Pengujian pengiriman data dilakukan untuk memastikan bahwa *sensor DHT22* mampu membaca serta mengirimkan nilai suhu dan kelembapan ke platform *ThingsBoard Cloud* secara *real-time* menggunakan protokol *MQTT*. Pada tahap ini, perangkat *ESP32-S3* dijalankan dalam kondisi terhubung ke jaringan *Wi-Fi* dan di konfigurasi agar mengirimkan data secara periodik dengan interval waktu tertentu. Setiap *payload* yang dikirim berisi tiga *parameter* utama:

- Nilai suhu ($^{\circ}\text{C}$).
- Nilai kelembapan (%).

- Timestamp dari *RTC (Real-Time Clock)* internal *ESP32-S3*.

Data yang diterima oleh *ThingsBoard* kemudian divisualisasikan dalam bentuk *dashboard* yang menampilkan grafik perubahan suhu dan kelembapan terhadap waktu. Hasil pengujian menunjukkan bahwa pengiriman data berlangsung stabil, dengan tingkat keterlambatan (*delay*) yang sangat kecil.

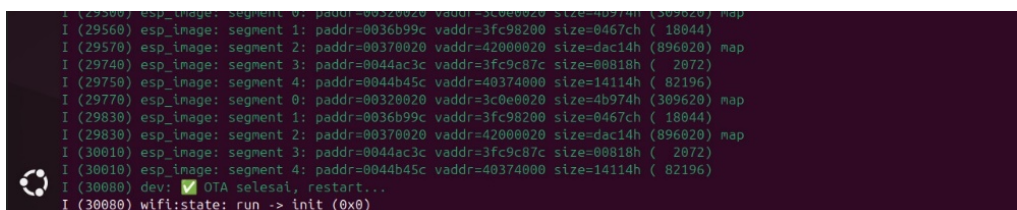
Untuk memastikan keakuratan waktu pengiriman, dilakukan perbandingan antara *timestamp RTC* perangkat dan *timestamp* dari server *ThingsBoard*. Hasil perbandingan ini kemudian digunakan pada tahap analisis *latency* (dibahas pada subbab berikutnya).

Selama periode pengujian yang direncanakan antara 30 September hingga 12 Oktober 2025, terdapat beberapa penyesuaian teknis yang perlu dilakukan:

- Pada 30 September–1 Oktober, sistem belum melakukan *streaming* karena masih dalam tahap pengaturan dan pembelajaran konfigurasi *ThingsBoard Cloud* serta *MQTT payload structure*.
- Pengambilan data berjalan optimal mulai 2 Oktober hingga 13 Oktober, dengan konektivitas stabil dan *payload* JSON berhasil diterima *ThingsBoard* tanpa *error*.
- Pada 3 Oktober–4 Oktober pukul 13.00, data tidak terekam karena masih mencoba untuk pengujian daya eksternal menggunakan adaptor 5V agar bisa *streaming* 24 jam dan laptop bisa dibawa kemana-mana.

Sementara itu, data untuk analisis *latency* baru dapat dikumpulkan mulai 9 Oktober hingga 12 Oktober, karena pada tanggal tersebut sistem sudah berhasil menampilkan parameter “*ts*” (*timestamp server*) dan “*send_time*” (*timestamp* dari *ESP32-S3*) yang diperlukan untuk penghitungan *delay* jaringan. Kondisi tersebut menunjukkan bahwa kendala yang terjadi bersifat teknis dan telah diatasi melalui penyesuaian sistem serta konfigurasi *firmware*, tanpa mempengaruhi keberhasilan utama pengujian pengiriman data.

4.3 Hasil Pengujian *OTA Firmware Update*



Gambar 4.3 Hasil Pengujian *OTA Firmware Update*

<input type="checkbox"/>	Last update time	Key ↑	Value	
<input type="checkbox"/>	2025-10-12 12:10:03	fw_state	IDLE	■
<input type="checkbox"/>	2025-10-12 13:10:32	fw_version	PaceS3-v2.0	■
<input type="checkbox"/>	2025-10-12 14:19:19	humidity	56.7000007629945	■
<input type="checkbox"/>	2025-10-12 14:19:19	send_time	2025-10-12 14:19:18	■
<input type="checkbox"/>	2025-10-09 17:02:23	target_fw_tag	Update-OTA-S3 2.0	■
<input type="checkbox"/>	2025-10-09 17:02:23	target_fw_idle	Update-OTA-S3	■
<input type="checkbox"/>	2025-10-09 17:02:23	target_fw_js	1760004203882	■
<input type="checkbox"/>	2025-10-09 17:02:23	target_fw_version	2.0	■
<input type="checkbox"/>	2025-10-12 14:19:19	temperature	34.79999923706055	■
<input type="checkbox"/>	2025-10-12 14:19:19	ts	176039958979	■

Gambar 4.3.1 Tampilan Parameter *OTA* pada *Thingsboard Cloud*

Pengujian *Over-The-Air (OTA)* dilakukan untuk memastikan bahwa sistem mampu melakukan pembaruan firmware secara jarak jauh tanpa perlu proses *flashing* manual melalui kabel USB. Pada tahap ini, *firmware* terbaru diunggah ke server *ThingsBoard Cloud* dan kemudian dikirimkan ke perangkat *ESP32-S3* menggunakan protokol *MQTT* sesuai konfigurasi atribut *OTA* yang telah diatur sebelumnya.

Ketika pembaruan tersedia, perangkat menerima perintah dari server dan secara otomatis menjalankan proses unduh (*download*), verifikasi *checksum*, serta instalasi firmware baru ke dalam memori *flash*. Selama proses berlangsung, status pembaruan dapat dipantau melalui *parameter* telemetri *OTA* seperti *fw_state*, *fw_version*, *target_fw_version*, dan *update_tag*.

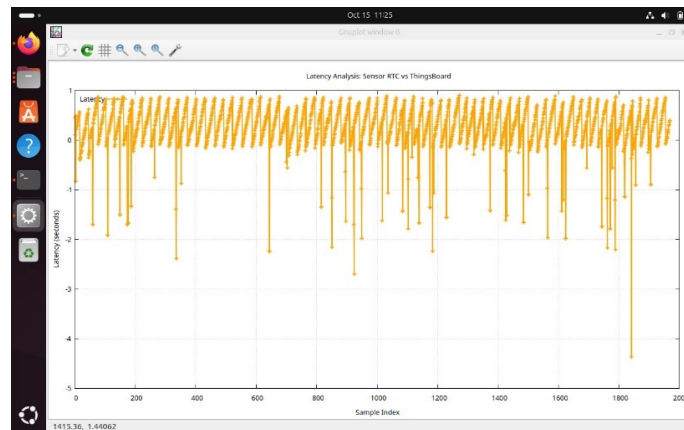
Setelah proses instalasi selesai, perangkat menampilkan pesan “*OTA update completed, restarting...*” pada terminal Ubuntu yang menandakan bahwa pembaruan berhasil dilakukan. Perangkat kemudian melakukan *reboot* dan memuat firmware versi terbaru secara otomatis.

Hasil pengujian menunjukkan bahwa sistem berhasil memperbarui *firmware* dari versi sebelumnya ke versi terbaru (*fw_version* = *PaceS3-v2.0*) dengan status akhir *fw_state* = *IDLE*, yang menandakan tidak terdapat kesalahan selama proses pembaruan. Selain itu, proses *OTA* juga berjalan stabil tanpa terjadi gangguan komunikasi antara *ESP32-S3* dan *ThingsBoard Cloud*.

4.4 Analisis *Latency* dan Visualisasi Data Sensor Menggunakan *Gnuplot*



Gambar 4.4 Grafik *Temperature and Humidity vs Time* menggunakan *Gnuplot*



Gambar 4.4.1 Grafik *Latency Analysis Sensor RTC vs ThingsBoard*

Analisis dilakukan untuk memantau performa sistem dalam melakukan transmisi data sensor dan menilai tingkat *latency* jaringan antara perangkat *ESP32-S3* dan server *ThingsBoard Cloud*. Proses visualisasi dan analisis dilakukan menggunakan *Gnuplot*, yang berfungsi untuk menampilkan tren data sensor secara grafis serta menghitung keterlambatan pengiriman data (*network latency*).

Pada Gambar 4.4, ditampilkan hasil visualisasi data sensor suhu dan kelembapan yang diambil selama periode 2–13 Oktober 2025. Berdasarkan grafik tersebut, terlihat bahwa nilai suhu berada pada rentang 20–36°C, sedangkan kelembapan berada pada rentang 45–70. Data ditampilkan secara *real-time* melalui *ThingsBoard Cloud*, di mana setiap titik mewakili *payload* JSON yang berisi parameter *temperature*, *humidity*, dan *timestamp*.

Terlihat adanya jeda data pada tanggal 3 Oktober, yang disebabkan oleh proses pengujian daya eksternal menggunakan adaptor 5V agar sistem dapat melakukan *streaming* 24 jam tanpa koneksi laptop langsung. Setelah penyesuaian, pengambilan data kembali stabil hingga tanggal 13 Oktober.

Selanjutnya, untuk analisis *latency* ditampilkan pada Gambar 4.4.1. Grafik tersebut dihasilkan dari hasil pengolahan data menggunakan perintah *Gnuplot* yang membandingkan perbedaan waktu antara *send_time* (*timestamp* dari *RTC ESP32-S3*) dan *ts* (*timestamp* dari server *ThingsBoard*). Pengambilan data dilakukan pada 9–12 Oktober 2025, yaitu saat sistem sudah berhasil menampilkan kedua *parameter* tersebut.

Hasil pengamatan menunjukkan bahwa nilai *latency* rata-rata berada di kisaran 1,4 detik, dengan fluktuasi kecil yang diakibatkan oleh kondisi jaringan *Wi-Fi* lokal. Pola data yang relatif stabil menunjukkan bahwa komunikasi *MQTT* antara perangkat dan server berjalan dengan baik tanpa kehilangan paket (*packet loss*). Beberapa *spike delay* yang terlihat disebabkan oleh proses *refresh* koneksi *MQTT* ketika server melakukan pembaruan session.

Secara keseluruhan, hasil ini menunjukkan bahwa sistem stabil, andal, dan responsif dalam melakukan pengiriman data sensor ke cloud, dengan tingkat keterlambatan yang masih dalam batas wajar untuk sistem *IoT* berbasis *Wi-Fi*.

BAB V KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan hasil perancangan, implementasi, dan pengujian sistem monitoring suhu dan kelembapan menggunakan mikrokontroler *ESP32-S3*, sensor *DHT22*, dan platform *ThingsBoard Cloud* dengan protokol *MQTT*, dapat disimpulkan bahwa sistem yang dibuat telah berhasil berfungsi dengan baik. Perangkat mampu mengirimkan data suhu dan kelembapan secara *real-time* ke *ThingsBoard* dalam format *JSON* yang berisi suhu, kelembapan, dan *timestamp* dari *RTC* internal *ESP32-S3*. Hasil pengujian menunjukkan proses komunikasi antara perangkat dan server berjalan stabil tanpa kehilangan paket data, serta mampu mempertahankan konektivitas dalam jangka waktu lama selama periode pengujian dari 2 hingga 13 Oktober 2025. Berdasarkan analisis *Gnuplot* terhadap perbandingan antara *send_time* dan *timestamp Thingsboard*, diperoleh rata-rata *network latency* sekitar 1,4 detik, yang menandakan keterlambatan pengiriman data tergolong sangat kecil dan masih dalam batas normal. Selain itu, fitur *OTA (Over-The-Air) firmware update* juga berhasil diimplementasikan dengan baik, di mana perangkat dapat melakukan pembaruan *firmware* secara otomatis dari *ThingsBoard* tanpa perlu pemrograman manual, dengan status pembaruan yang terpantau melalui parameter *fw_state* dan *fw_version*. Secara keseluruhan, sistem ini telah memenuhi tujuan utama *project*, yaitu menghasilkan sistem *IoT* berbasis *cloud* yang efisien, stabil, dan mudah diperbarui dari jarak jauh.

5.2 Saran

Untuk pengembangan selanjutnya, sistem ini masih bisa ditingkatkan dari sisi efisiensi daya dan keandalan pengiriman data. Saat ini, perangkat masih bergantung pada adaptor 5V sebagai sumber utama, sehingga jika terjadi pemadaman listrik sistem akan berhenti bekerja. Oleh karena itu, dapat ditambahkan modul baterai serta rangkaian manajemen daya agar perangkat tetap aktif sementara waktu ketika sumber daya utama terputus.

Selain itu, sistem juga dapat dilengkapi dengan mekanisme penyimpanan data lokal sementara (*offline buffer*). Fungsinya untuk menampung data sensor ketika koneksi *WiFi* terputus, lalu mengirimkannya kembali ke *ThingsBoard Cloud* setelah koneksi pulih. Dengan cara ini, tidak ada data yang hilang dan hasil pemantauan tetap lengkap.

Dari sisi keamanan, komunikasi *MQTT* juga bisa ditingkatkan menggunakan enkripsi *TLS/SSL* agar proses pertukaran data dan pembaruan *OTA* lebih aman. Terakhir, tampilan *dashboard* di *ThingsBoard* dapat dikembangkan dengan menam-

notifikasi otomatis saat suhu atau kelembapan melebihi batas, serta indikator status *OTA* agar pengguna lebih mudah memantau kondisi perangkat. Dengan pengembangan tersebut, sistem ini berpotensi diterapkan pada skala yang lebih luas seperti *monitoring* lingkungan atau sistem industri kecil yang membutuhkan pemantauan jarak jauh secara *real-time* dan aman.

DAFTAR PUSTAKA

Hercog, D., Lerher, T., Truntiĉ, M., & Teĵak, O. (2023). Design and implementation of ESP32-based IoT devices. *Sensors (Basel)*, 23(15), 6739. <https://doi.org/10.3390/s23156739>

Aghenta, L. O., & Iqbal, M. T. (2020). Design and implementation of a low-cost, open source IoT-based SCADA system using ESP32 with OLED, ThingsBoard and MQTT protocol. *AIMS Electronics and Electrical Engineering*, 4(1), 57–74. <https://doi.org/10.3934/electreng.2020.1.57>

El Jaouhari, S., & Bouvet, E. (2022). Secure over-the-air updates for IoT: Survey, challenges, and discussions. *Internet of Things*, 18, 100508. <https://doi.org/10.1016/j.iot.2022.100508>

Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., & Ayyash, M. (2015). Internet of Things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys & Tutorials*, 17(4), 2347–2376. <https://doi.org/10.1109/COMST.2015.2444095>

Integrating Edge Computing and IoT for Real-Time Air and Water Quality Monitoring Systems. (2025). *International Journal of Environmental Sciences*, 11(17s). <https://doi.org/10.64252/z52kvj10>

Peng, F. (2024). Environment Monitoring based on the Integration of Edge Computing and Multimodal Sensors. *Applied and Computational Engineering*, 113, 77–82. <https://doi.org/10.54254/2755-2721/2024.18307>

Roostaei, J., & co. (2023). IoT-based edge computing (IoTEC) for improved environmental monitoring. *ScienceDirect*.

Montesclaros, R. M. M., Cruz, J. E. B., Parocha, R. C., & Macabebe, E. Q. B. (2021). MQTT Based Power Consumption Monitoring with Usage Pattern Visualization Using Uniform Manifold Approximation and Projection for Smart Buildings. *Proceedings of the Intelligent Computing Conference 2021 (Virtual)*. Springer Nature. https://doi.org/10.1007/978-3-030-80126-7_79

Shvaika, A., Shvaika, D., Landiak, D., et al. (2025). A distributed architecture for MQTT messaging: the case of TBMQ. *Journal of Big Data*, 12, 224. <https://doi.org/10.1186/s40537-025-01271-x>

El-Basioni, B. M. M. (2024). A conceptual modeling approach of MQTT for IoT-based systems. *Journal of Electrical Systems and Information Technology*, 11, Article 62. <https://doi.org/10.1186/s43067-024-00181-x>

Nitol Saha, Md. Masruk Aulia, Md. Mostafizur Rahman, Mohammed Shafiul

Alam Khan. (2024). IoT-Driven Cloud-based Energy and Environment Monitoring System for Manufacturing Industry. *arXiv preprint*.

Ahmad, Y. A., Gunawan, T. S., Mansor, H., Hamida, B. A., Hishamudin, A. F., & Arifin, F. (2021). On the evaluation of DHT22 temperature sensor for IoT application. *Proceedings of the 2021 8th International Conference on Computer and Communication Engineering (ICCCE 2021)*, 131–134. <https://its.id/m/3529142670ntheEvaluationofDHT22TemperatureSensor>

Yulizar, D., Soekirno, S., Ananda, N., Prabowo, M. A., Putra Perdana, I. F., & Aofany, D. (2023). Performance analysis comparison of DHT11, DHT22, and DS18B20 temperature sensors. *Proceedings of the 2023 International Conference on Computing and Applied Electronics*. Atlantis Press. <https://www.atlantis-press.com/article/125989928.pdf>

Putri, P., Saragi, M. A., & Hasibuan, Y. A. E. H. (2024). A prototype IoT temperature and humidity monitoring ESP32-based via ThingSpeak. *INFOKUM: Jurnal Informatika dan Komputer*, 13(1), 15–21. <https://infor.seaninstitute.org/index.php/infokum/article/view/2755>

Wardani, I. K., Ichniarsyah, A. N., Telaumbanua, M., Priyonggo, B., Fil'aini, R., Mufidah, Z., & Dewangga, D. A. (2024). The feasibility study: Accuracy and precision of DHT22 in measuring the temperature and humidity in the greenhouse. *IOP Conference Series: Earth and Environmental Science*, 1290(1), 012032. <https://doi.org/10.1088/1755-1315/1290/1/012032>

Amirkhanov, B., Amirkhanova, G., Kunelbayev, M., Adilzhanova, S., & Tokhtassyn, M. (2025). Evaluating HTTP, MQTT over TCP and MQTT over Web-Socket for digital twin applications: A comparative analysis on latency, stability, and integration. *International Journal of Innovative Research and Scientific Studies*, 8(1), 679–694. <https://doi.org/10.53894/ijirss.v8i1.4414>

Palmese, F., Redondi, A. E. C., & Cesana, M. (2022). Adaptive Quality of Service Control for MQTT-SN. *Sensors*, 22(22), 8852. <https://doi.org/10.3390/s22228852>

Puthiyidam, J. J., & Joseph, S. (2024). Internet of things network performance: Impact of message and client sizes and reliability levels. *ECTI Transactions on Electrical Engineering, Electronics, and Communications*, 22(1). <https://doi.org/10.37936/ecti-eec.2024221.252941>

Park, C.-Y., Lee, S.-J., & Lee, I.-G. (2025). Secure and lightweight Over-the-Air update mechanism for Internet of Things. *Electronics*, 14(8), 1583. <https://doi.org/10.3390/electronics14081583>

- Malumbres, V., Saldana, J., Berné, G., & Modrego, J. (2024). Updates over the air via LoRa: Unicast and broadcast combination for boosting update speed. *Sensors*, 24(7), 2104. <https://doi.org/10.3390/s24072104>
- Wei, W., Banerjee, J., Islam, S., Pan, C., & Xie, M. (2023). Energy-aware Incremental OTA Update for Flash-based Batteryless IoT Devices. *arXiv preprint arXiv:2406.12189*. <https://arxiv.org/abs/2406.12189>
- Atzori, L., Iera, A., & Morabito, G. (2016). Understanding the Internet of Things: Definition, potentials, and societal role of a fast evolving paradigm. *Ad Hoc Networks*, 56, 122–140. <https://doi.org/10.1016/j.adhoc.2016.12.004>
- Ray, P. P. (2016). A survey on Internet of Things architectures. *Journal of King Saud University – Computer and Information Sciences*, 30(3), 291–319. <https://doi.org/10.1016/j.jksuci.2016.10.003>
- Espressif Systems. (2022). ESP32-S3 Series Datasheet. *Espressif Systems*. https://www.espressif.com/sites/default/files/documentation/esp32-s3_datasheet_en.pdf
- Nemlaha, E., Strelec, P., Horák, T., Kováč, S., & Tanuška, P. (2023). Suitability of MQTT and REST Communication Protocols for AIoT or IIoT Devices Based on ESP32-S3. *ResearchGate*. <https://www.researchgate.net/publication/366776204>
- Wang, T., Wang, L., & Yan, P. (2022). Application of embedded Linux in the design of Internet of Things gateway. *Journal of Intelligent Systems*, 31(1), 1014–1023. <https://doi.org/10.1515/jisys-2021-0208>
- Chaudhary, H., Anthony, A., Abiona, O., & Onime, C. (2025). Unix-Based Systems in Embedded and IoT Devices: Exploring the Versatility and Robustness. *International Journal of Communications, Network and System Sciences*, 18, 15–25. <https://doi.org/10.4236/ijcns.2025.182002>
- Farina, M. D. O., Pohren, D. H., Roque, A. dos S., Silva, A., da Costa, J. P. J., Fontoura, L. M., ... & Freitas, E. P. de. (2024). Hardware-Independent Embedded Architecture Framework. *Journal of Internet Services and Applications*, 15(1), 14–24. <https://doi.org/10.5753/jisa.2024.3634>
- Amanlou, S. (2020). Lightweight security mechanism over MQTT protocol for IoT devices. *International Journal of Advanced Computer Science and Applications*, 11(7). <https://doi.org/10.14569/IJACSA.2020.0110726>
- Seoane, V., García-Rubio, C., Almenares, F., & Campo, C. (2021). Performance evaluation of CoAP and MQTT with security support for IoT environments. *Computer Networks*. <https://doi.org/10.1016/j.comnet.2021.108338>

Panagou, I. C., Katsoulis, S., Nannos, E., Zantalis, F., & Koulouras, G. (2025). A Comprehensive Evaluation of IoT Cloud Platforms: A Feature-Driven Review with a Decision-Making Tool. *Sensors*, 25(16), 5124. <https://doi.org/10.3390/s25165124>

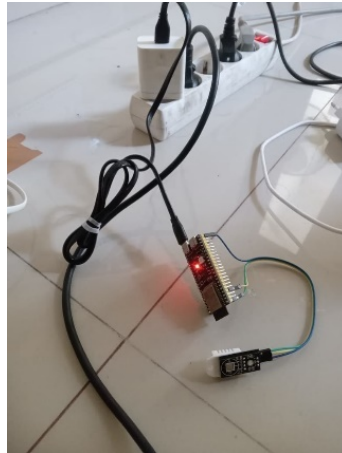
Shvaika, D. I., Shvaika, A. I., & Artemchuk, V. O. (2024). Advancing IoT interoperability: Dynamic data serialization using ThingsBoard. *Journal of Edge Computing*, 3(2), 126–135. <https://doi.org/10.55056/jec.74>

Plauska, I., Liutkevičius, A., & Janavičiūtė, A. (2023). Performance Evaluation of C/C++, MicroPython, Rust and TinyGo Programming Languages on ESP32 Microcontroller. *Electronics*, 12(1), 143. <https://doi.org/10.3390/electronics12010143>

Culic, I., Vochescu, A., & Radovici, A. (2022). A Low-Latency Optimization of a Rust-Based Secure Operating System for Embedded Devices. *Sensors*, 22(22), 8700. <https://doi.org/10.3390/s22228700>

Sharma, A., Sharma, S., Torres-Arias, S., & Machiry, A. (2023). Rust for Embedded Systems: Current State, Challenges and Open Problems (Extended Report). *arXiv*. <https://arxiv.org/abs/2311.05063>

LAMPIRAN



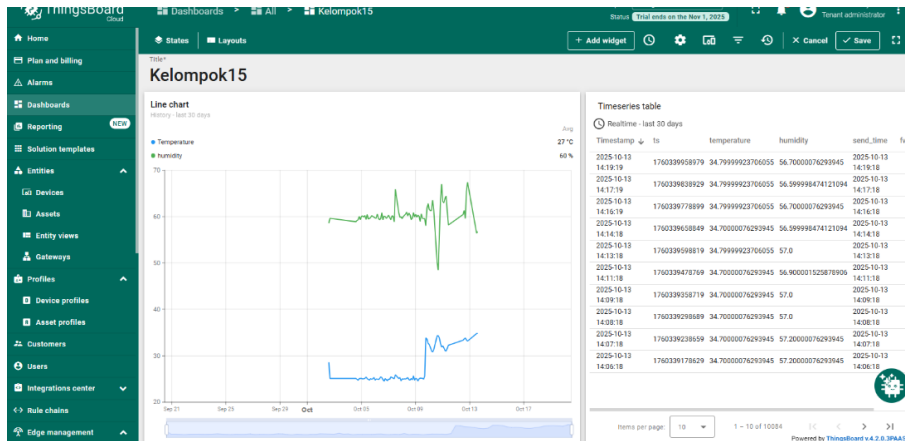
Lampiran 1. *ESPS3-32* terhubung dengan adaptor dan sensor *DHT22*

```
Sep 18 22:30
salmanaufa@salmanaufa-VirtualBox: ~/dht_OTA

I (708) wifi_init: tcp mss: 1440
I (718) wifi_init: WiFi IRAM OP enabled
I (718) wifi_init: WiFi RX IRAM OP enabled
I (728) phy_init: phy_version 700,8582a7fd,Feb 10 2025,20:13:11
I (768) phy_init: Saving new calibration data due to checksum failure or outdated calibration data, mode(0)
I (838) wifi:mode : sta (98:88:e0:03:ae:08)
I (838) wifi:enable tsf
I (848) dht_sensor: ⌚ Menunggu koneksi WiFi...
I (1848) dht_sensor: ⌚ Menunggu koneksi WiFi...
I (2848) dht_sensor: ⌚ Menunggu koneksi WiFi...
I (3258) wifi:new:<3,0>, old:<1,0>, ap:<255,255>, sta:<3,0>, prof:1, snd_ch_cfg:0x0
I (3258) wifi:state: init -> auth (0xb0)
I (3278) wifi:state: auth -> assoc (0x0)
I (3288) wifi:state: assoc -> run (0x10)
I (3308) wifi:connected with vivo, aid = 3, channel 3, BW20, bssid = 36:77:3c:5a:8f:ad
I (3308) wifi:security: WPA2-PSK, phy: bgn, rssi: -69
I (3308) wifi:pm start, type: 1

I (3308) wifi:dp: 1, bi: 102400, li: 3, scale listen interval from 307200 us to 307200 us
I (3318) wifi:set rx beacon pti, rx_bcn_pti: 0, bcn_timeout: 25000, mt_pti: 0, mt_time: 10000
I (3338) wifi:dp: 2, bi: 102400, li: 4, scale listen interval from 307200 us to 409600 us
I (3338) wifi:AP's beacon interval = 102400 us, DTIM period = 2
I (3338) wifi:cba-add:idx:0 (ifx:0, 36:77:3c:5a:8f:ad), tid:0, ssn:0, winSize:64
I (3848) dht_sensor: ✅ Terhubung ke WiFi!
I (4328) esp_netif_handlers: sta ip: 192.168.43.185, mask: 255.255.255.0, gw: 192.168.43.231
I (6848) dht_sensor: 🌐 Terhubung ke broker MQTT ThingsBoard Cloud
I (6848) dht_sensor: 📡 Event MQTT: BeforeConnect
I (6848) gpio: GPIO[4] InputEn: 0 OutputEn: 0 OpenDrain: 0 Pullup: 0 Pulldown: 0 Intr: 0
I (7328) wifi:cba-add:idx:1 (ifx:0, 36:77:3c:5a:8f:ad), tid:1, ssn:0, winSize:64
I (7728) dht_sensor: 📡 Event MQTT: Connected(false)
I (7738) dht_sensor: 📡 Data terkirim: {"humidity":68.30000305175781,"temperature":31.10000381469727}
I (12758) dht_sensor: 📡 Data terkirim: {"humidity":69.80000385175781,"temperature":30.5}
I (12758) dht_sensor: 📡 Data terkirim: {"humidity":70.0,"temperature":30.5}
```

Lampiran 2. Tampilan konektivitas di terminal *Ubuntu*



Lampiran 3. *Dashboard ThingsBoard Cloud* menampilkan grafik suhu dan kelembapan

```
I (29560) esp_image: segment 0: paddr=00320020 vaddr=3c0e0020 size=0b97fch (389620) map
I (29560) esp_image: segment 1: paddr=00320020 vaddr=3c0e0020 size=0b97fch (389620) map
I (29570) esp_image: segment 2: paddr=00370020 vaddr=42000020 size=dac14h (896020) map
I (29740) esp_image: segment 3: paddr=0044ac3c vaddr=3fc9c87c size=00818h ( 2072)
I (29750) esp_image: segment 4: paddr=0044b45c vaddr=40374000 size=14114h ( 82196)
I (29770) esp_image: segment 0: paddr=00320020 vaddr=3c0e0020 size=0b97fch (389620) map
I (29830) esp_image: segment 1: paddr=00320020 vaddr=3c0e0020 size=0b97fch (389620) map
I (29830) esp_image: segment 2: paddr=00370020 vaddr=42000020 size=dac14h (896020) map
I (30010) esp_image: segment 3: paddr=0044ac3c vaddr=3fc9c87c size=00818h ( 2072)
I (30010) esp_image: segment 4: paddr=0044b45c vaddr=40374000 size=14114h ( 82196)
I (30080) dev: [✓] OTA selesai, restart...
I (30080) wifi:state: run -> init (0x0)
```

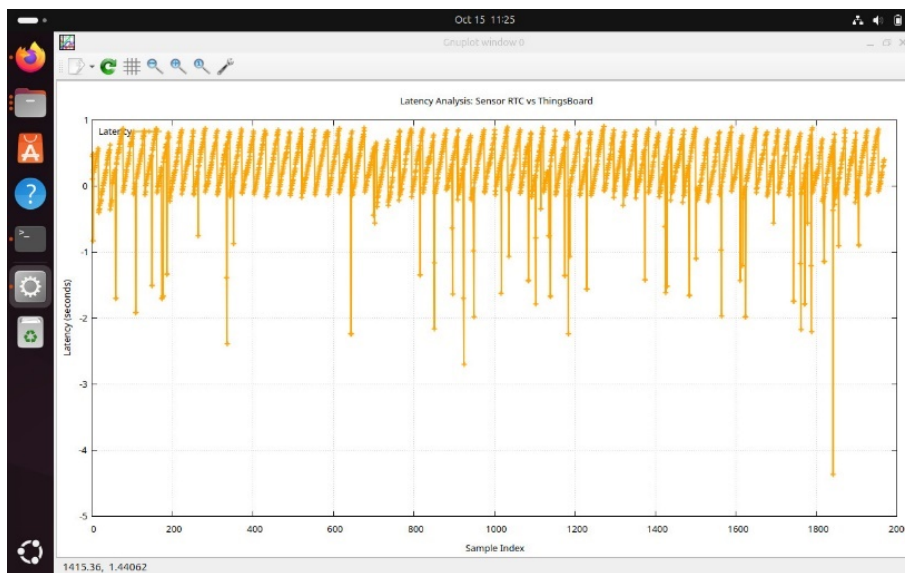
Lampiran 4. Hasil Pengujian *OTA Firmware Update*

<input type="checkbox"/>	Last update time	Key ↑	Value	
<input type="checkbox"/>	2025-10-12 13:10:03	fw_state	IDLE	
<input type="checkbox"/>	2025-10-12 13:10:02	fw_version	PaceP-s3-v2.0	
<input type="checkbox"/>	2025-10-13 14:19:19	humidity	56.70000076293945	
<input type="checkbox"/>	2025-10-13 14:19:19	send_time	2025-10-13 14:19:18	
<input type="checkbox"/>	2025-10-09 17:02:23	target_fw_tag	Update-OTA-G3 2.0	
<input type="checkbox"/>	2025-10-09 17:02:23	target_fw_title	Update-OTA-G3	
<input type="checkbox"/>	2025-10-09 17:02:23	target_fw_ts	1760004209882	
<input type="checkbox"/>	2025-10-09 17:02:23	target_fw_version	2.0	
<input type="checkbox"/>	2025-10-13 14:19:19	temperature	34.79999923706055	
<input type="checkbox"/>	2025-10-13 14:19:19	ts	1760339958979	

Lampiran 5. Tampilan Parameter *OTA* pada *Thingsboard Cloud*



Lampiran 6. Grafik *Temperature and Humidity vs Time* menggunakan *Gnuplot*



Lampiran 7. Grafik *Latency Analysis Sensor RTC vs ThingsBoard*

Link Drive Kelompok 15: <https://its.id/m/Kelompok15IoT>

BIODATA PENULIS

 A portrait of a young man with dark hair, wearing a blue jacket over a green shirt, standing outdoors next to a white wall and some greenery.	<p>Nama : Muhammad Salman Alfarisyi</p> <p>NRP : 2042231006</p> <p>Tempat, tanggal lahir : Surabaya, 4 April 2005</p>
 A portrait of a young man with dark hair, wearing a blue jacket over a patterned shirt, standing indoors against a plain white wall.	<p>Nama : Muhammad Aufa Affandi</p> <p>NRP : 2042231011</p> <p>Tempat, tanggal lahir : Surabaya, 05 Oktober 2004</p>