**SISTEM IOT UNTUK PENGIRIMAN DATA GAMBAR**

**Oleh**

**VIERI FAJAR FIRDAUS**

**NIM : 13521099**

**JURUSAN : TEKNIK INFORMATIKA**

A black and white drawing of an elephant

AI-generated content may be incorrect.

**PROGRAM STUDI TEKNIK INFORMATIKA**

**SEKOLAH TEKNIK ELEKTRO & INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**April 2025**

Pendahuluan

### Latar Belakang

Perkembangan Internet of Things (IoT) yang pesat dalam beberapa tahun terakhir telah membawa perubahan signifikan dalam berbagai sektor, mulai dari rumah pintar (smart homes), pertanian pintar (smart farming), hingga pemantauan kesehatan jarak jauh (telemedicine). IoT memungkinkan perangkat yang terhubung untuk saling bertukar data dan berinteraksi secara otomatis melalui jaringan internet. Salah satu aplikasi penting IoT adalah dalam pengolahan data berbasis gambar, yang dapat digunakan dalam berbagai bidang seperti pengawasan keamanan, analisis citra medis, dan pemantauan lingkungan.

Salah satu tantangan utama dalam aplikasi IoT yang berbasis gambar adalah bagaimana menangani pengiriman data gambar secara efisien dan aman. Data gambar berukuran besar memerlukan bandwidth yang cukup besar, dan pengiriman data secara terus-menerus, seperti dalam streaming video, bisa sangat membebani jaringan jika tidak dikelola dengan baik. Oleh karena itu, pengiriman gambar melalui protokol komunikasi yang efisien, seperti MQTT (Message Queuing Telemetry Transport), sangat penting untuk memastikan bahwa data dapat dikirimkan dengan latensi yang rendah dan menggunakan bandwidth yang efisien.

### Tujuan

* Merancang sistem pengiriman data gambar dari ESP board ke server menggunakan MQTT.
* Menyimpan data di database dan menampilkannya pada dashboard.
* Mengevaluasi kinerja sistem berdasarkan interval pengiriman dan latensi end-to-end.
* (Opsional) Mengimplementasikan enkripsi-dekripsi data untuk keamanan.

Rancangan Sistem

Spesifikasi Sistem

|  |  |
| --- | --- |
| **Komponen** | **Deskripsi** |
| **Sensing Unit** | ESP32-CAM |
| **Resolusi Gambar** | 640×480 piksel (Base64) |
| **Protokol Komunikasi** | MQTT (Wi-Fi) |
| **Interval Pengiriman (T)** | T = 10 (dalam detik) NIM=13521099 |
| **Jumlah Pengiriman (K)** | K ∈ {10, 20, 100} |
| **Data yang Dikirim** | {image\_data, timestamp, capture\_time, publish\_time} |

### Kebutuhan Sistem

Sistem IoT yang dibuat memiliki beberapa fitur dibawah ini :

* Mengirimkan gambar secara real time melalui MQTT
* Melakukan enkripsi terhadap gambar
* Melihat gambar melalui dashboard

Selain itu, Penulis menggunakan ESP32 CAM yang dapat mengirimkan gambar berdasarkan hasil yang ditangkap.

### Diagram Blok Sistem

Arsitektur sistem ini dibangun dengan pendekatan modular, yang terdiri dari beberapa komponen utama yang saling berinteraksi untuk mencapai tujuan pengiriman gambar secara efisien dan aman. Komponen utama dalam arsitektur ini adalah perangkat **ESP32** sebagai unit pengambil data, **MQTT** sebagai protokol komunikasi, **backend server** untuk pengelolaan data, dan **frontend** sebagai antarmuka pengguna untuk menampilkan gambar. Untuk mengelola service yang ada digunakan containerization menggunakan docker.

A screenshot of a computer

AI-generated content may be incorrect.

### Alur Pengiriman Data

1. ESP32 mengambil gambar dan menyimpannya dalam buffer.
2. Data gambar dikonversi ke format Base64 (jika diperlukan).
3. Data dikirim via MQTT dengan format JSON
4. Server MQTT menyimpan data ke database (MySQL).
5. Dashboard menampilkan data gambar dan informasi pengiriman.

### Implementasi Kode di ESP32

|  |
| --- |
| void sendPhoto() {  camera\_fb\_t \* fb = NULL;  start\_time = millis();  fb = esp\_camera\_fb\_get();  if(!fb) {  Serial.println("Camera capture failed");  delay(1000);  ESP.restart();  }  end\_time = millis();  capture\_time = end\_time - start\_time;  Serial.println("Connecting to MQTT server...");  if (client.connect("ESP32Client", mqtt\_user, mqtt\_pass)) {  Serial.println("Connected to MQTT server");  String base64Image = base64::encode(fb->buf, fb->len);    Serial.println(getFormattedTime());  String payloadImage = "{\"id\":" + String(message\_id) +  ",\"image\":\"" + base64Image + "\"" +  ",\"timestamp\":\"" + getFormattedTime() + "\"}";  start\_time = millis();  client.publish(mqtt\_topic\_image, payloadImage.c\_str());  end\_time = millis();  publish\_time = end\_time - start\_time;    String payloadLatency = "{\"id\":" + String(message\_id) +  ",\"capture\_time\":" + String(capture\_time) +  ",\"publish\_time\":" + String(publish\_time) +"}";  client.publish(mqtt\_topic\_latency, payloadLatency.c\_str());  Serial.println("Image sent to MQTT topic.");  esp\_camera\_fb\_return(fb);  success++;  } else {  failed++;  Serial.println("Failed to connect to MQTT broker.");  }  int total = success+failed;  if(total > 0) {  float successRate = (success / float(total)) \* 100;  Serial.print("Success Rate: ");  Serial.print(successRate);  Serial.print("%");  Serial.print("(");  Serial.print(success);  Serial.print("/");  Serial.print(total);  Serial.println(")");  }  client.loop();  } |

### Algoritma Enkripsi

Enkripsi menggunakan algoritma AES, dimana proses enkripsi dilakukan ketika server menerima request berupa base64 lalu hasil enkripsi akan disimpan di database, untuk hasil dekripsi bisa diperoleh ketika user membuka dashboard

|  |
| --- |
| def encrypt\_image(image\_data):      cipher = AES.new(AES\_KEY, AES.MODE\_CBC)      ct\_bytes = cipher.encrypt(pad(image\_data, AES.block\_size))      iv = cipher.iv      return iv + ct\_bytes |

|  |
| --- |
| def decrypt\_image(encrypted\_data):      """Dekripsi gambar menggunakan AES CBC"""      iv = encrypted\_data[:16]      ct = encrypted\_data[16:]      cipher = AES.new(AES\_KEY, AES.MODE\_CBC, iv=iv)      pt = unpad(cipher.decrypt(ct), AES.block\_size)      return pt |

### Evaluasi Kinerja

Untuk latensi yang dihitung pada evaluasi kinerja dari pengambilan gambar, pengiriman melalui MQTT, proses enkripsi, proses input, untuk proses penampilan gambar pada dashboard akan diambil beberapa sampel

|  |  |  |
| --- | --- | --- |
| **K** | **Rata-rata Latency (ms)** | **Error Interval (%)** |
| 10 | 750.80 | 0 |
| 20 | 1628.02 | 0 |
| 100 | 841.65 | 0 |

Berikut adalah sampel dari latency end to end

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Kesimpulan

Sistem IoT yang dirancang berhasil mengirim data gambar dari ESP board ke server menggunakan protokol MQTT dengan interval **10 detik** (sesuai 3 digit terakhir NIM). Hasil pengujian menunjukkan bahwa:

* **Latensi pengiriman** bervariasi tergantung pada jumlah pengiriman (K), dengan rata-rata terendah **750.80 ms** (saat K=10) dan tertinggi **1628.02 ms** (saat K=20). Nilai latensi kembali stabil di **841.65 ms** untuk K=100, menunjukkan bahwa sistem mampu menangani beban pengiriman berulang.
* **Error interval pengiriman** adalah **0%** untuk semua nilai `K\*\*, membuktikan bahwa pengiriman data tepat waktu sesuai konfigurasi.
* **Enkripsi data** diimplementasikan di sisi server (menggunakan Docker untuk containerisasi), memastikan keamanan data tanpa menambah beban komputasi di perangkat ESP.
* **Docker** pada server berperan dalam mempermudah deployment dan isolasi proses, termasuk penerimaan data MQTT, enkripsi, dan penyimpanan ke database.

Secara keseluruhan, sistem memenuhi kriteria Level 1 (pengiriman data real-time) dan Level 2 (enkripsi opsional), dengan kinerja yang stabil dan skalabel. Untuk pengembangan selanjutnya, uji coba dengan **interval dinamis** atau **algoritma enkripsi di sisi ESP** dapat dipertimbangkan untuk optimasi. Untuk enkripsi gambar tidak dilakukan di ESP namun dilakukan pada server backend. Dashboard dapat diakses secara langsung di <http://212.85.26.216:3000/>

Lampiran

### Image Dashboard

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

### Dokumentasi Hardware

A close up of a device

AI-generated content may be incorrect.

### Data Pengujian

Data hasil pengujian 10 kali pengiriman gambar

A screenshot of a computer

AI-generated content may be incorrect.

Data hasil pengujian 20 kali pengiriman gambar

A screen shot of a computer

AI-generated content may be incorrect.

Data hasil pengujian 100 kali pengiriman gambar

A screen shot of a computer

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

A screen shot of a computer

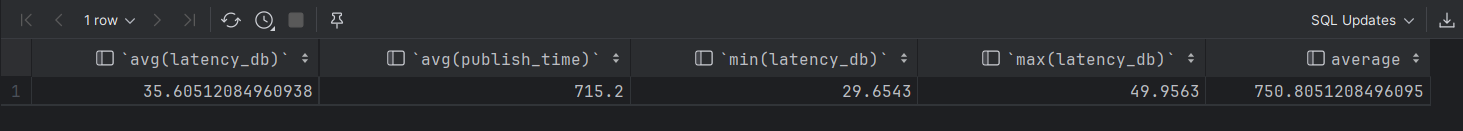
AI-generated content may be incorrect.

A screen shot of a computer screen

AI-generated content may be incorrect.

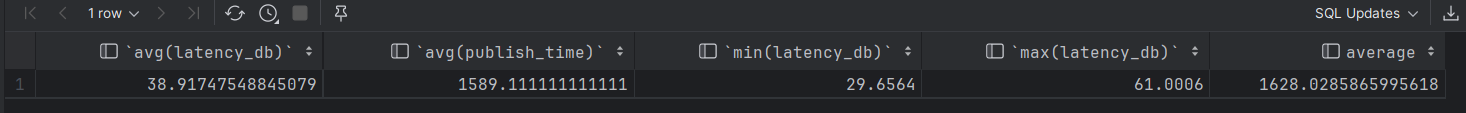
Data penghitungan latency pada 10 kali pengiriman

|  |
| --- |
| select *avg*(latency\_db), *avg*(publish\_time), *min*(latency\_db), *max*(latency\_db), *avg*(latency\_db) + *avg*(publish\_time) as average from images where id >= 2000 and id <= 2009; |



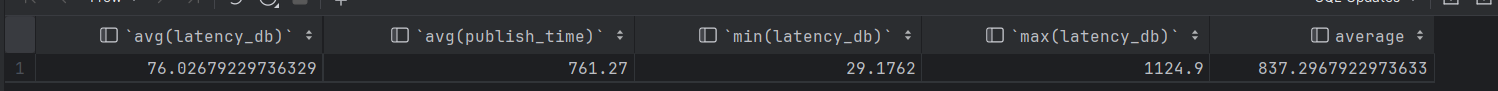
Data penghitungan latency pada 20 kali pengiriman

|  |
| --- |
| select *avg*(latency\_db), *avg*(publish\_time), *min*(latency\_db), *max*(latency\_db), *avg*(latency\_db) + *avg*(publish\_time) as average from images where id >= 2051 and id <= 2070; |



Data penghitungan latency pada 100 kali pengiriman

|  |
| --- |
| select *avg*(latency\_db), *avg*(publish\_time), *min*(latency\_db), *max*(latency\_db), *avg*(latency\_db) + *avg*(publish\_time) as average from images where id >= 2101 and id <= 2200; |



### Skrip Kode

Backend

|  |
| --- |
| from flask import Flask  from flask\_cors import CORS  from dotenv import load\_dotenv  import os  from routes import get\_images, decrypt\_image\_api, get\_information\_image  from mqtt\_handler import on\_connect, on\_message  import paho.mqtt.client as mqtt  load\_dotenv()  app = Flask(\_\_name\_\_)  CORS(app, resources={r"/\*": {"origins": os.getenv("URL\_FRONTEND")}})  app.add\_url\_rule('/','home', lambda: "Welcome to the API", methods=['GET'])  app.add\_url\_rule('/images', 'get\_images', get\_images, methods=['GET'])  app.add\_url\_rule('/images/decrypt/<int:image\_id>', 'decrypt\_image\_api', decrypt\_image\_api, methods=['GET'])  app.add\_url\_rule('/image/<int:image\_id>','get\_information\_image',get\_information\_image, methods=['GET'])  def setup\_mqtt():      """Fungsi untuk setup MQTT"""      mqtt\_client = mqtt.Client()      mqtt\_client.on\_connect = on\_connect      mqtt\_client.on\_message = on\_message      mqtt\_client.username\_pw\_set(os.getenv("MQTT\_USER"), os.getenv("MQTT\_PASS"))      mqtt\_client.connect(os.getenv("MQTT\_BROKER"), int(os.getenv("MQTT\_PORT")), 60)        mqtt\_client.loop\_start()  if \_\_name\_\_ == "\_\_main\_\_":      setup\_mqtt()        app.run(host="0.0.0.0", port=5000, debug=False) |

Frontend

|  |
| --- |
| import './App.css';  import React from 'react';  import { BrowserRouter as Router, Route, Routes } from 'react-router-dom';  import HomePage from './page/Home';  import ImagePage from './page/Image';  function App() {    return (      <Router>        <div className="container mx-auto px-4 py-8">          <h1 className="text-4xl font-bold text-center mb-6 gap-10">Image Dashboard</h1>          <Routes>            <Route exact path="/" element={<HomePage />} />            <Route path="/image/:id" element={<ImagePage />} />          </Routes>        </div>      </Router>    );  }  export default App; |

Docker Config

|  |
| --- |
| version: '3'  services:    backend:      build:        context: ./backend      container\_name: flask-backend      ports:        - "5000:5000"      environment:        - MQTT\_BROKER=${MQTT\_BROKER}        - MQTT\_PORT=${MQTT\_PORT}        - MQTT\_USER=${MQTT\_USER}        - MQTT\_PASS=${MQTT\_PASS}        - MQTT\_IMAGE\_TOPIC=${MQTT\_IMAGE\_TOPIC}        - MQTT\_LATENCY\_TOPIC=${MQTT\_LATENCY\_TOPIC}        - MYSQL\_HOST=${MYSQL\_HOST}        - MYSQL\_PORT=${MYSQL\_PORT}        - MYSQL\_USER=${MYSQL\_USER}        - MYSQL\_PASSWORD=${MYSQL\_PASSWORD}        - MYSQL\_DB=${MYSQL\_DB}        - AES\_KEY=${AES\_KEY}        - URL\_FRONTEND=${URL\_FRONTEND}      networks:        - app-network      restart: always    frontend:      build:        context: ./frontend      container\_name: react-frontend      ports:        - "3000:3000"      depends\_on:        - backend      environment:        - REACT\_APP\_BACKEND\_URL=${URL\_BACKEND}      networks:        - app-network      restart: always  networks:    app-network:      driver: bridge |

### Contoh Payload MQTT

Payload Image

|  |
| --- |
| {"id":4505,"image":<image\_base64>,"timestamp":"2025-04-26 15:50:20"} |

Payload Latency

|  |
| --- |
| {"id":2106,"capture\_time":0,"publish\_time":123} |

Link Github : <https://github.com/vierifirdaus/UTS_IOT>

Link Dashboard : <http://212.85.26.216:3000/>