

**Sistem Pengumpulan Data Pelacakan Transportasi Umum
Menggunakan *Bluetooth Proximity Beacons***

Skripsi

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Teknik

Disusun Oleh :

Muhammad Afifudin Arsyada

NIM: 215150301111001



PROGRAM STUDI TEKNIK KOMPUTER
DEPARTEMEN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2024

LEMBAR PENGESAHAN

Sistem Pengumpulan Data Pelacakan Transportasi Umum Menggunakan
Bluetooth Proximity Beacons

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Teknik

Disusun Oleh :

Muhamamd Afifudin

NIM: 215150301111001

Skripsi ini telah diuji dan dinyatakan lulus pada

Telah diperiksa dan disetujui oleh:

Dosen Pembimbing 1

Dosen Pembimbing 2

Agung Setia Budi, S.T., M.T.,

M.Eng., Ph.D.

NIP: 198704232022031003

Achmad Basuki, S.T., M.MG., Ph.D

NIP: 197411182003121002

Mengetahui

Ketua Departemen Teknik Informatika

Achmad Basuki, S.T., M.MG., Ph.D

NIP: 197411182003121002

PERNYATAAN ORISINILITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar referensi.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur- unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 11 Desember 2024

Muhamamd Afifudin Arsyada

NIM: 215150301111001

PRAKATA

Puji syukur penulis panjatkan kehadirat Allah SWT yang telah melimpahkan berkat, rahmat, dan karunianya sehingga penulis dapat menyelesaikan skripsi yang berjudul “Sistem Pengumpulan Data Pelacakan Transportasi Umum Menggunakan Bluetooth Proximity Beacons” dengan baik, lancar, dan tepat waktu sesuai dengan rencana penulis sebagai syarat dalam memperoleh gelar Sarjana Teknik pada program studi Teknik Komputer Fakultas Ilmu Komputer Universitas Brawijaya.

Pengerjaan penelitian dan penyusunan skripsi ini tidak lepas dari dukungan, bantuan, serta doa dari beberapa pihak. Dengan ini, penulis mengucapkan rasa hormat dan banyak terima kasih kepada:

1. Bapak Prof. Ir. Wayan Firdaus Mahmudy, S.Si., MT., Ph.D. selaku Dekan Fakultas Ilmu Komputer Universitas Brawijaya,
2. Bapak Achmad Basuki, S.T, M.MG., Ph.D. selaku Ketua Departemen Teknik Informatika, Fakultas Ilmu Komputer, Universitas Brawijaya,
3. Bapak Barlian Henryranu Prasetyo, S.T., M.T., Ph.D. selaku Ketua Program Studi Teknik Komputer, Fakultas Ilmu Komputer, Universitas Brawijaya,
4. Bapak Agung Setia Budi, S.T., M.T., M.Eng., Ph.D. selaku dosen pembimbing pertama yang telah memberikan kesempatan untuk ikut serta dalam penelitian *CBR* serta mengusulkan rancangan awal untuk penelitian ini,
5. Bapak Achmad Basuki, S.T, M.MG., Ph.D. selaku dosen pembimbing kedua yang telah memberikan saran, masukan, serta dukungan kepada penulis selama proses penelitian dan penyusunan skripsi,
6. Seluruh jajaran dosen dan tenaga pendidik Program Studi Teknik Komputer, Fakultas Ilmu Komputer, Universitas Brawijaya,
7. Kedua orang tua penulis serta keluarga yang telah memberikan dukungan, memberikan semangat, serta selalu mendoakan penulis sehingga skripsi ini dapat selesai,
8. Alex, Farras, Didu dan Nusa yang telah membantu serta mendukung penulis untuk menyelesaikan skripsi ini,
9. Semua pihak yang tidak dapat dituliskan satu persatu.

Dengan segala kerendahan hati, penulis menyadari bahwa laporan ini jauh dari kesempurnaan, penulis mengharapkan apabila terdapat kritik maupun saran yang bersifat membangun dari para dosen penguji dan pembaca, Penulis berharap laporan ini dapat memberikan manfaat bagi yang membutuhkan.

Malang, 11 Desember 2024

Penulis
udineze1907@student.ub.ac.id

ABSTRAK

Preferensi masyarakat Indonesia terhadap kendaraan pribadi dibandingkan transportasi umum disebabkan oleh masalah seperti layanan yang tidak andal, waktu tempuh yang tidak konsisten, dan kurangnya integrasi teknologi pada transportasi umum. Penelitian ini mengembangkan sistem pelacakan bus sekolah di Kota Malang tanpa menggunakan GPS, dengan memanfaatkan teknologi Bluetooth Low Energy (BLE) dan Internet of Things (IoT). Beacon BLE yang dipasang pada bus memancarkan sinyal yang dideteksi oleh Road Side Unit (RSU) berbasis Raspberry Pi 4 dan smartphone Android, kemudian data dikirimkan ke server cloud untuk memprediksi waktu kedatangan bus secara real-time menggunakan machine learning. Pengujian menunjukkan tingkat keberhasilan deteksi sinyal sebesar 70%-97% untuk Raspberry Pi 4 dan 80%-90% untuk perangkat Android, kecuali di beberapa lokasi dengan kendala tertentu. Akses perangkat secara jarak jauh menggunakan SSH dan AirDroid mencapai keberhasilan 80%-100%, sementara pengiriman data telemetry ke server berhasil dilakukan dengan rata-rata 76%-99%. Sistem ini menunjukkan keandalan yang baik meskipun terdapat tantangan lingkungan dan operasional, sehingga berpotensi meningkatkan layanan transportasi umum secara signifikan.

Kata Kunci: Bluetooth Low Energy, IoT, Transportasi Umum, Pelacakan Bus, Pemantauan Real-Time, Sistem Bus Sekolah.

ABSTRACT

The preference for private vehicles over public transportation in Indonesia is driven by issues such as unreliable services, inconsistent travel times, and lack of technological integration in public transport. To address these issues, this study explores a GPS-less tracking system for school buses in Malang City using Bluetooth Low Energy (BLE) beacons and Internet of Things (IoT). BLE beacons installed on buses emit signals detected by Road Side Units (RSU) equipped with Raspberry Pi 4 and Android smartphones. These units transmit data to a cloud server for real-time bus tracking and arrival prediction using machine learning. Testing revealed detection rates of 70%-97% for Raspberry Pi 4 and 80%-90% for Android devices, except in some challenging locations. Remote device access via SSH and AirDroid achieved an 80%-100% success rate, while telemetry data transmission to the server ranged from 76% to 99%. The system demonstrated overall reliability despite environmental and operational challenges, paving the way for improved public transport tracking.

Keywords : bluetooth low energy, iot, public transportation, bus tracking, real-time Monitoring, school bus system.

Daftar Isi

LEMBAR PENGESAHAN	2
PERNYATAAN ORISINILITAS	3
KATA PENGANTAR.....	Error! Bookmark not defined.
Daftar Isi.....	8
Daftar Gambar	11
BAB I PENDAHULUAN	4
1.1 Latar Belakang.....	4
1.2 Rumusan Masalah.....	7
1.3 Tujuan	7
1.4 Manfaat Penelitian.....	7
1.5 Batasan Masalah	7
1.6 Sistematika Laporan.....	8
BAB II LANDASAN KEPUSTAKAAN	9
2.1 Tinjauan Pustaka	9
2.1.1 Smart Bus Management and Tracking System	10
2.1.2 Transforming urban mobility with internet of things: public bus fleet tracking using proximity-based bluetooth beacons.	12
2.1.3 Vehicle Tracking System Using Greedy Forwarding Algorithms for Public Transportation in Urban Arterial	13
2.1.4 Federated Learning for Intelligent Transportation Systems: Use Cases, Open Challenges, and Opportunities.....	14
2.1.5 Design and Development of a sustainable telemetry system for environmental parameters.....	15
2.1.6 An Indoor Tracking System using iBeacon and Android	16
2.2 Dasar Teori	17
2.2.1. IOT (Internet of Things).....	17
2.2.2. Bluetooth Low Energy Beacon.	18
2.2.3. Raspberry pi 4.	20
2.2.4. Ponsel Android.....	21
2.2.5. Telemetry.....	22
2.2.6. <i>Node Js</i>	22

2.2.7. <i>Bore Client</i>	23
2.2.8. <i>Axios</i>	23
2.2.9. <i>Node Beacon scanner</i>	24
2.2.10. <i>Retrofit</i>	24
2.2.11. <i>AirDroid</i>	25
2.2.12. <i>Android Beacon Library</i>	25
2.2.13. <i>AWS</i>	26
2.2.14. <i>Transportasi Umum</i>	26
2.2.15. <i>Kotlin</i>	27
2.2.16. <i>Box Metal UMG</i>	27
2.2.17. <i>Power Adapter LDNIO A4610C</i>	27
2.2.18 <i>PM2</i>	28
BAB III METODOLOGI PENELITIAN	29
3.1 Tipe Penelitian	30
3.2 Strategi Penelitian	30
3.2.1 Metode Penelitian	31
3.2.2 Objek Penelitian	33
3.2.3 Lokasi Penelitian	34
3.2.4 Teknik Pengumpulan Data	37
3.2.5 Teknik Analisis Data	38
3.2.6 Peralatan Pendukung	39
BAB IV REKAYASA KEBUTUHAN	41
4.1 Kajian Masalah	41
4.2 Identifikasi Stakeholder	41
4.3 Kebutuhan Fungsional	42
4.4 Spesifikasi Sistem	43
4.5 Analisis Kebutuhan Perangkat Keras dan Perangkat Lunak	43
4.5.1. Perangkat Keras	43
4.5.2. Perangkat Lunak	44
BAB V PERANCANGAN DAN IMPLEMENTASI	45
5.1 Perancangan Sistem	45
5.1.1 Perancangan Arsitektur <i>Scanner Raspberry Pi 4</i>	46
5.1.2 Perancangan Arsitektur <i>Scanner Smartphone Android</i>	47

5.1.3 Perancangan Program Scanner BLE	48
5.1.3 Perancangan Telemetry	49
5.1.4 Perancangan Akses Jarak Jauh Pada Raspberry Pi 4 dan Smartphone Android.	50
5.2 Implementasi Sistem.....	51
5.2.1. Implementasi Arsitektur Perangkat Keras.	51
5.2.2. Implementasi Program Scanner.....	55
5.2.3. Implementasi Program Telemetry.	72
5.2.4. Implementasi Akses Jarak Jauh.....	76
BAB VI PENGUJIAN DAN ANALISIS	85
6.1 Hasil Pengujian.....	85
6.1.1. Pengujian Deteksi BLE Beacon Dengan Scanner Raspberry pi 4.....	85
6.1.2. Pengujian Deteksi BLE Beacon Dengan Scanner Smartphone Android.	86
6.1.4. Pengujian Pengiriman Data Ble Scanner dan Telemetry Ke Server.	88
6.1.5. Pengujian Akses Jarak Jauh Pada Raspberry Pi 4.	90
6.1.6. Pengujian Akses Jarak Jauh Pada Smartphone Android.	91
6.2 Analisis Hasil Pengujian.....	92
6.2.1. Hasil Pengujian Deteksi BLE Beacon.	93
6.2.2. Hasil Pengujian Keberhasilan Pengiriman Data Telemetry.....	97
6.2.3. Hasil Pengujian Keberhasilan Akses Jarak Jauh.	100
BAB VII PENUTUP	102
7.1 Kesimpulan.....	102
7.2 Saran	103

Daftar Gambar

Gambar 2.1 Alur Sistem Smart Bus Management and Tracking System	11
Gambar 2.2 Arsitektur pelacakan bus di Johor, Malaysia.....	12
Gambar 2.3 Arsitektur pelacakan transportasi publik	13
Gambar 2.4 Arsitektur Federated Learning	14
Gambar 2.5 Arsitektur Sistem Telemetry Suhu Lingkungan	15
Gambar 2.6 Arsitektur pelacakan Indoor BLE Beacon	16
Gambar 2.7 Bluetooth Low Energy Estimote.....	18
Gambar 2.8 Raspberry pi 4.	20
Gambar 2.9 Smartphone Redmi 12C	21
Gambar 3.1 Perancangan Sistem.....	29
Gambar 3.2 Metode penelitian.....	32
Gambar 3.3 Rute Pagi Bus I.....	34
Gambar 3.4 Rute Sore Bus I	35
Gambar 5.1 Perancangan Sistem RSU	45
Gambar 5.2 Diagram Blok Sistem Pengumpulan Data Menggunakan Raspberry pi 4	46
Gambar 5.3 Diagram Blok Sistem Pengumpulan Data Menggunakan Smartphone Android	47
Gambar 5.3 Flow Chart Scanner BLE.....	48
Gambar 5.4 Flow Chart Telemetry.....	49
Gambar 5.5 Diagram Akses Jarak jauh.....	50
Gambar 5.6 Implementasi Raspberry Pi 4 dan Peripheral.....	55
Gambar 5.7 Halaman Login dan Tampilan PlayStore.....	82
Gambar 5.8 Implementasi Raspberry Pi 4 dan Peripheral.....	83
Gambar 5.9 Implementasi Raspberry Pi 4 dan Peripheral.....	84
Gambar 6.1 Hasil Deteksi BLE Beacon dengan Raspberry pi 4	86
Gambar 6.2 Hasil Deteksi BLE Beacon dengan Raspberry pi 4	87
Gambar 6.3 Percobaan Pengiriman Data ke Server.....	89
Gambar 6.4 Kumpulan Data Deteksi BLE Menggunakan Android	89
Gambar 6.5 Implementasi Raspberry Pi 4 dan Peripheral.....	90
Gambar 6.6 Hasil Akses Jarak Jauh Raspberry pi 4 Melalui PC.....	91
Gambar 6.7 Hasil Akses Jarak Jauh Smartphone Android Melalui PC	92
Gambar 6.8 Grafik Presentase Keberhasilan Scanning Android	95
Gambar 6.9 Grafik Presentase Keberhasilan Scanning Raspberry Pi 4.....	96
Gambar 6.10 Grafik Presentase Keberhasilan Telemetry Tiap RSU.....	100
Gambar 6.11 Grafik Presentase Keberhasilan Akses Jarak Jauh.....	100

Daftar Tabel

Tabel 2.1 Tinjauan Pustaka	9
Tabel 2.2 Data Transmisi BLE Beacon	19
Tabel 3.1 Lokasi RSU	36
Tabel 5.1 Implementasi Pemasangan RSU.....	52
<i>Tabel 6.1 Performa Deteksi BLE Beacon dengan Smartphone</i>	<i>93</i>
<i>Tabel 6.2 Performa Deteksi BLE Beacon dengan Raspberry Pi 4</i>	<i>95</i>
Tabel 6.3 Presentase Keberhasilan Pengiriman Telemetry Tiap RSU	97

BAB I PENDAHULUAN

1.1 Latar Belakang

Sebagian besar masyarakat Indonesia yang memiliki pilihan untuk menggunakan kendaraan pribadi akan memilih untuk menggunakan kendaraan pribadi daripada menggunakan transportasi umum. Hal ini disebabkan oleh banyaknya kekurangan yang ada pada transportasi umum yang ada pada kota-kota di Indonesia. Kondisi transportasi yang tidak layak, biaya yang lebih mahal daripada menggunakan kendaraan pribadi, dan kinerja pelayanan yang masih kurang seperti waktu tunggu dan tempuh yang tidak konsisten menjadi alasan masyarakat Indonesia lebih memilih menggunakan kendaraan pribadi (Mutiawati, 2019). Permasalahan tersebut membuat masyarakat Indonesia lebih memilih transportasi dengan pemesanan *online* yang lebih konsisten dalam pelayanannya. Pada penerapannya, transportasi *online* dapat melihat lokasi kendaraan yang dipesan dan perkiraan waktu tiba di lokasi pemesanan. Transportasi umum dalam kota masih belum bisa menerapkan hal tersebut karena kurangnya penerapan teknologi pada transportasi umum menjadikan transportasi umum kurang digemari oleh masyarakat Indonesia sebagai sarana transportasi.

Penggunaan transportasi publik seperti bus dan angkot dapat menjadi peran penting suatu negara. Pengurangan emisi karbon dari kendaraan bermotor dapat dicapai dengan meningkatkan penggunaan transportasi publik oleh masyarakat. Namun, negara-negara berkembang masih dihadapkan pada berbagai tantangan dalam jaringan transportasi publik mereka, seperti layanan yang tidak teratur, keterlambatan, ketidakakuratan dan ketidakandalan waktu kedatangan, waktu tunggu yang lama, serta terbatasnya informasi *real-time* yang tersedia bagi pengguna (Elijah et al., 2023).

Dengan masalah yang ada, telah dibuat beberapa inovasi untuk mengatasi hal tersebut dengan melakukan pelacakan lokasi kendaraan umum secara *real-time* dengan menggunakan *GPS* yang ditanamkan pada kendaraan umum seperti bus. *Global Positioning System (GPS)* beroperasi dengan menerima sinyal dari jaringan satelit yang mengorbit Bumi. Sebuah penerima *GPS*, seperti yang terdapat pada perangkat *GPS*, dapat menentukan lokasinya dengan memantau sinyal dari setidaknya empat satelit, memungkinkan penerima untuk menentukan posisinya dengan akurasi antara 1 hingga 10 meter (Kassim et al., 2022). Namun pada penggunaan *GPS*, akurasi *GPS* dapat terganggu di lingkungan perkotaan dengan bangunan tinggi atau vegetasi yang lebat, yang dapat menghalangi sinyal satelit.

Serta seiring bertambahnya armada kendaraan umum biaya pembuatan sistem pelacakan bus berbasis *GPS* akan menjadi semakin tinggi.

Penelitian ini berfokus pada pengembangan sistem pelacakan transportasi yang memanfaatkan teknologi *Bluetooth Low Energy (BLE)* berbasis *Internet of Things (IoT)* untuk armada bus sekolah di Kota Malang. Latar belakang dari studi kasus ini adalah adanya kebutuhan untuk memantau pergerakan bus sekolah guna memastikan kedatangan yang tepat waktu serta meningkatkan efisiensi operasional. Pelacakan bus yang efektif sangat penting, terutama di kawasan perkotaan seperti Malang, yang sering kali mengalami kemacetan dan keterlambatan akibat kondisi lalu lintas yang tidak menentu.

Dalam konteks ini, sistem yang diusulkan memanfaatkan *BLE proximity beacons* yang dipasang pada setiap bus sekolah. Beacon tersebut akan memancarkan sinyal *BLE* yang dapat dideteksi oleh perangkat khusus, seperti *Raspberry Pi 4* dan ponsel *smartphone Android*, yang ditempatkan secara strategis di sepanjang rute perjalanan bus. Perangkat deteksi ini berfungsi sebagai *Road Side Unit (RSU)*, yang akan menangkap sinyal *BLE* dari bus yang melintas di dekatnya. Sistem ini dirancang agar mampu mendeteksi sinyal *BLE* hingga jarak 30 meter, bahkan ketika perangkat *Android* dan *Raspberry Pi 4* dilindungi casing logam, sehingga diharapkan mampu beroperasi dalam berbagai kondisi lingkungan.

Sistem ini kemudian mengirimkan data lokasi yang diperoleh ke server berbasis *cloud* untuk memperkirakan waktu kedatangan bus di titik tertentu. Prediksi waktu kedatangan dilakukan menggunakan algoritma *machine learning*, yang memerlukan data historis berupa waktu kedatangan dan keberangkatan bus di setiap titik penempatan *RSU*. Oleh karena itu, proses pengumpulan data dari *RSU* menjadi bagian penting dari penelitian ini, agar dapat memodelkan prediksi dengan akurasi tinggi.

Namun, penerapan teknologi *BLE* untuk pelacakan transportasi tidak lepas dari beberapa tantangan. Salah satu kendala utama adalah keberadaan *BLE beacon* jenis lain yang dapat terdeteksi oleh sistem. Hal ini berpotensi menyebabkan kekeliruan dalam mengidentifikasi bus, terutama jika terdapat *BLE beacon* lain yang melintas di dekat *RSU* yang telah dipasang. Untuk mengatasi masalah ini, perlu diterapkan metode filtering *BLE beacon address* agar sistem dapat membedakan sinyal *beacon* yang relevan dengan armada bus sekolah.

Selain itu, stabilitas jaringan komunikasi menjadi faktor kritis karena data yang dikirimkan dari *RSU* ke server harus sampai dengan lengkap tanpa adanya gangguan. Dalam lingkungan perkotaan yang padat, sinyal radio dari berbagai perangkat lain dapat menyebabkan interferensi, yang pada penerapannya dapat mempengaruhi keakuratan sistem deteksi *BLE*. Oleh karena itu, penelitian ini juga akan mengeksplorasi pendekatan untuk meningkatkan ketahanan sistem terhadap gangguan radiofrekuensi, serta memastikan komunikasi data yang stabil dan konsisten antara perangkat dan *server cloud*.

Selain tantangan teknis yang berkaitan dengan deteksi sinyal *BLE* dan stabilitas komunikasi, terdapat pula beberapa masalah praktis yang muncul selama implementasi sistem ini di lapangan. Salah satu isu yang dihadapi adalah potensi pencurian atau vandalisme terhadap perangkat *RSU*, terutama ketika *Raspberry pi 4*, *smartphone Android* dan perangkat pendukung lainnya ditempatkan di area terbuka sepanjang rute bus. *RSU* yang dipasang di tiang-tiang atau lokasi strategis sering kali berada dalam jangkauan orang yang tidak bertanggung jawab, sehingga rawan dicuri atau dirusak. Oleh karena itu, penelitian ini juga perlu mempertimbangkan strategi pengamanan, seperti pemasangan perangkat di lokasi yang tidak mudah dijangkau atau penggunaan casing yang lebih tahan banting dan terkunci untuk melindungi komponen *RSU*.

Selain ancaman pencurian, masalah lain yang muncul adalah serangan hama, terutama semut. Beberapa *RSU* yang dipasang berdekatan dengan pohon atau area hijau cenderung menjadi sasaran semut yang tertarik dengan panas yang dihasilkan oleh perangkat elektronik, khususnya *Raspberry Pi* dan *smartphone Android*. Kehadiran semut di dalam perangkat dapat menyebabkan kerusakan pada komponen elektronik atau mengakibatkan koneksi menjadi tidak stabil. Oleh karena itu, perlu diterapkan langkah-langkah pencegahan, seperti penggunaan pelindung khusus atau penempatan *RSU* di lokasi yang lebih steril untuk menghindari gangguan dari hama.

Dengan mengidentifikasi dan mengatasi tantangan-tantangan yang telah disebutkan, diharapkan sistem pelacakan transportasi umum berbasis *BLE* yang diusulkan dapat diimplementasikan dengan lebih andal dan efisien, serta memberikan hasil yang akurat meskipun beroperasi di lingkungan perkotaan yang penuh dengan berbagai hambatan.

1.2 Rumusan Masalah

Berdasarkan latar belakang yang dikemukakan, maka dapat dibuat suatu rumusan masalah sebagai berikut:

1. Bagaimana performa *Raspberry pi 4* dalam melakukan pendeteksian *beacon*?
2. Bagaimana performa *smartphone android* dalam melakukan pendeteksian *beacon*?
3. Bagaimana tingkat keberhasilan akses perangkat *RSU* secara jarak jauh?
4. Bagaimana tingkat keberhasilan pengiriman data *telemetry* ke server?

1.3 Tujuan

Sesuai dengan rumusan masalah, maka dapat ditentukan tujuan penelitian sebagai berikut:

1. Menganalisis performa *Raspberry Pi 4* dalam melakukan pendeteksian *beacon*.
2. Menganalisis performa *smartphone android* dalam melakukan pendeteksian *beacon*.
3. Menentukan metode akses jarak jauh paling optimal dalam mengakses *RSU*.
4. Menentukan metode *telemetry* paling efektif dalam memantau kondisi *RSU*.

1.4 Manfaat Penelitian

1. Memberikan dasar yang kuat untuk penelitian lebih lanjut dalam pengembangan dan penyempurnaan sistem pelacakan berbasis BLE dan teknologi terkait.
2. Menyediakan solusi teknologi yang dapat diterapkan pada berbagai jenis armada, berkontribusi pada peningkatan sistem transportasi publik yang lebih efisien.
3. Menyediakan data dan metode untuk mengembangkan model prediksi waktu kedatangan yang lebih akurat, yang dapat digunakan dalam penelitian selanjutnya untuk meningkatkan sistem informasi transportasi.

1.5 Batasan Masalah

Pada penelitian ini dilakukan beberapa batasan masalah, agar nantinya pembahasan yang terkait dengan penelitian dapat lebih terfokus. Berikut adalah batasan masalah yang ada:

1. Berfokus pada pendeteksian *BLE beacon* menggunakan *raspberry pi 4* dan ponsel *android*.

2. Telemetry hanya dilakukan pada *raspberry pi 4* sebagai representasi suhu dalam RSU.
3. Berfokus pada pengumpulan data untuk data latih prediksi waktu tiba.

1.6 Sistematika Laporan

Sistematika pembahasan dari skripsi ini terdiri dari lima bagian utama sebagai berikut:

BAB 1 PENDAHULUAN

Pada bab ini berisi latar belakang, rumusan masalah, tujuan penelitian, manfaat penelitian, batasan masalah, dan sistematika pembahasan. Bagian ini menjadi pendahuluan dalam memahami urgensi dari adanya penelitian ini dan tujuan yang ingin dicapai, serta hasil yang diharapkan dari penelitian.

BAB 2 LANDASAN KEPUSTAKAAN

Bab ini akan menjabarkan mengenai penelitian terdahulu yang berkaitan dengan topik penelitian dan juga teori yang digunakan untuk memahami permasalahan yang dibahas pada penelitian.

BAB 3 METODOLOGI PENELITIAN

Pada bab ini akan menjabarkan arsitektur umum dari penelitian yang dilakukan. Setiap tahap yang dilakukan pada proses perancangan arsitektur dan gambaran umum mengenai sistem yang akan dibuat.

BAB II LANDASAN KEPUSTAKAAN

2.1 Tinjauan Pustaka

Untuk menunjang penelitian ini, diperlukan pengkajian pustaka terdahulu mengenai penelitian yang serupa atau terkait dengan penelitian yang akan dikerjakan. Dengan adanya pengkajian pustaka ini diharapkan dapat mempermudah kami sebagai peneliti untuk mengerjakan penelitian ini. **Tabel 2.1** di bawah ini akan merangkum mengenai penelitian sebelumnya.

Tabel 2.1 Tinjauan Pustaka

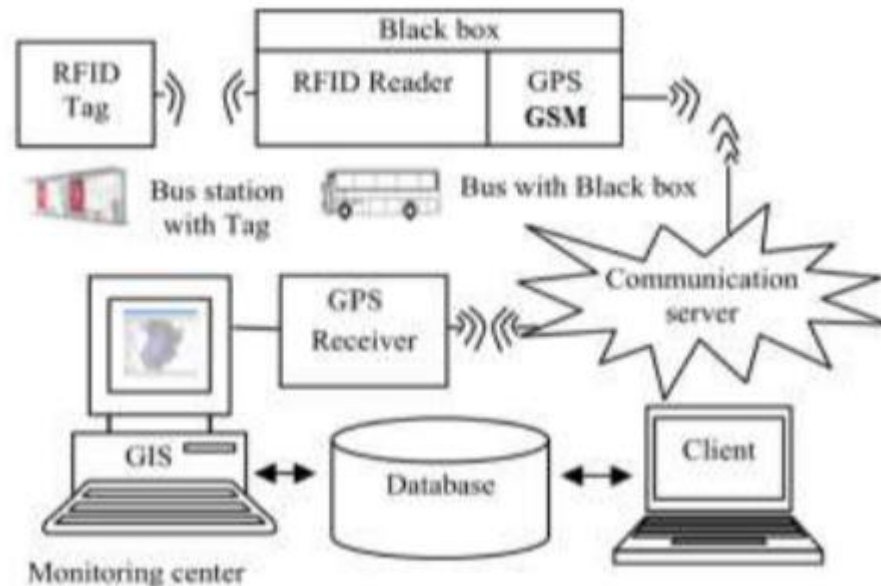
No	(Penulis, Tahun), Judul	Persamaan	Perbedaan	
			Penelitian terdahulu	Rencana penelitian
1	(Godge et al., 2019) <i>Smart Bus Management and Tracking System</i>	<i>Tracking bus.</i>	Menggunakan RFID.	Menggunakan BLE.
2	(Elijah et al., 2023) <i>Transforming urban mobility with internet of things: public bus fleet tracking using proximity-based bluetooth beacons</i>	<i>Tracking bus dengan BLE beacon dan raspberry pi.</i>	Menggunakan <i>Raspberry pi 0</i> untuk <i>scanner BLE</i>	Peningkatan sistem dengan memberikan fitur pemantauan dan penggunaan ponsel android sebagai scanner tambahan
3	(Jimoh et al., 2020) <i>A Vehicle Tracking System Using Greedy Forwarding Algorithms for Public Transportation in Urban Arterial</i>	<i>Tracking bus</i>	Menggunakan <i>GPS</i> .	Menggunakan BLE scanner.

No	(Penulis, Tahun), Judul	Persamaan	Perbedaan	
			Penelitian terdahulu	Rencana penelitian
4	(Chong et al., 2024) <i>Federated Learning for Intelligent Transportation Systems: Use Cases, Open Challenges, and Opportunities</i>	Pengumpulan data transportasi umum	Pengumpulan Inteligent transportation system	Penelitian ini bertujuan untuk pengumpulan data guna meningkatkan model intelegent system
5	(Vargas et al., 2021) <i>Design and Development of a sustainable telemetry system for environmental parameters</i>	Penggunaan <i>Telemetry</i> untuk memantau kondisi lingkungan sekitar	Fokus pada pemantauan kondisi lingkungan	Sistem telemetry memantau kondisi <i>RSU</i>
6	(Moneer et al., 2020) <i>An Indoor Tracking System using iBeacon and Android</i>	Penggunaan <i>BLE beacon</i> dan ponsel android sebagai scanner.	Berfokus pendeteksian <i>BLE</i> pada area dalam ruangan	Berfokus pelacakan bus dengan BLE di area perkotaan

2.1.1 Smart Bus Management and Tracking System

Penelitian yang dilakukan oleh Godge, p adalah mengenai pelacakan bus kota menggunakan *RFID* dan *GPS*. Pada penerapannya sistem ini menggunakan *website* untuk memantau bus dan beberapa aspek lainnya. *Website* yang dibangun mempunyai 2 bagian, yang pertama yaitu admin *login* dan yang kedua adalah *stationmaster*. Admin modul digunakan oleh admin penyedia layanan bus untuk menambahkan unit bus, halte bus, serta bisa melihat informasi dari bus dan informasi dari halte yang ada. Sementara *website* untuk *stationmaster* digunakan

untuk melihat tipe bus, plat bus, waktu kedatangan dan tujuan selanjutnya, serta lokasi bus saat ini.

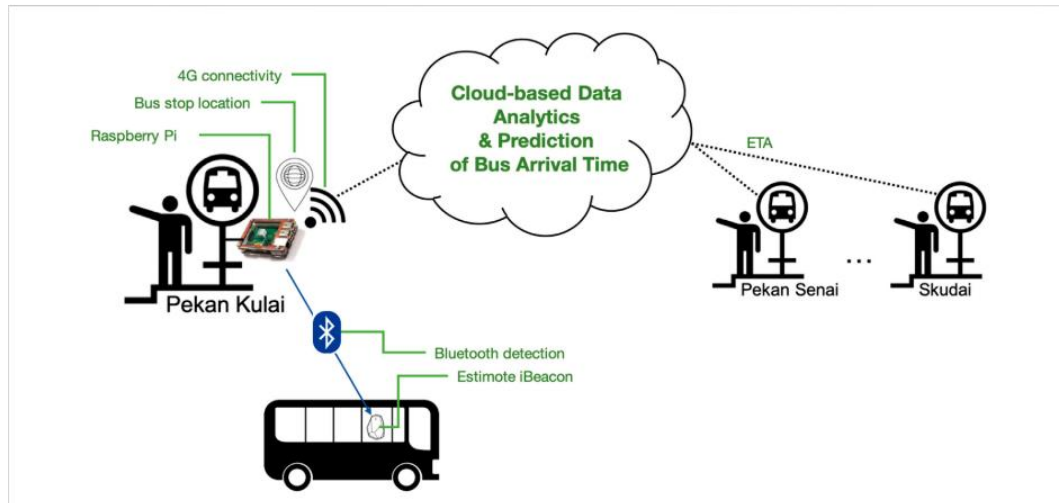


Gambar 2.1 Alur Sistem *Smart Bus Management and Tracking System*

Sumber: (Godge et al., 2019)

RFID pada sistem ini digunakan untuk melakukan *tagging* pada bus yang lewat pada *bus stop* tertentu. Setelah *black box* melakukan *tagging* maka *module gps* dan *gsm* akan melakukan pengiriman data ke *server*. Data yang diterima dikirimkan ke *client* melalui *database*. Pada penelitian ini belum terdapat hasil percobaan sistem, namun sistem yang diajukan oleh Godge, p diharapkan menampilkan perilaku bus saat jam operasional.

2.1.2 Transforming urban mobility with internet of things: public bus fleet tracking using proximity-based bluetooth beacons.



Gambar 2.2 Arsitektur pelacakan bus di Johor, Malaysia

Sumber: (Elijah et al., 2023)

Penelitian yang dilakukan oleh Elijah O. di Johor, Malaysia, memperkenalkan pendekatan inovatif untuk pelacakan bus tanpa menggunakan teknologi GPS. Mereka menggunakan *IoT* berbasis *BLE* sebagai alternatif yang lebih ekonomis. Dalam sistem ini, *BLE beacon* dipasang pada bus, sementara *Raspberry Pi Zero* berfungsi sebagai pemindai yang ditempatkan di halte dan terminal tertentu. Saat bus mendekati atau berhenti di halte, sinyal *BLE* dari beacon akan terdeteksi oleh *Raspberry Pi*, dan data lokasi ini kemudian dikirim ke server untuk menghitung perkiraan waktu tiba bus di halte berikutnya. Dengan menggunakan metode ini, biaya manajemen armada bus dapat ditekan secara signifikan dibandingkan dengan penggunaan sistem *GPS* tradisional.

Hasil pengujian lapangan di beberapa rute bus di Johor, termasuk Johor Bahru, Iskandar Puteri, dan Kulai, menunjukkan bahwa sistem ini efektif dalam mengurangi waktu perjalanan bus, terutama pada hari libur di mana terjadi pengurangan waktu perjalanan antara 5 hingga 10 menit dibandingkan hari kerja biasa. Selain itu, penelitian ini membuktikan bahwa teknologi *BLE* dapat diandalkan untuk melacak bus dengan biaya lebih rendah dan efisiensi tinggi, memberikan solusi yang potensial untuk diadopsi di negara-negara berkembang yang menghadapi tantangan dalam manajemen transportasi umum. Namun pada penerapannya masih banyak kekurangan yang dihadapi, seperti banyaknya gangguan sinyal yang radio dan gangguan jaringan yang menghalangi transmisi data. Serta banyaknya *BLE* pada halte menyebabkan gangguan pada *Raspberry pi*.

2.1.3 Vehicle Tracking System Using Greedy Forwarding Algorithms for Public Transportation in Urban Arterial



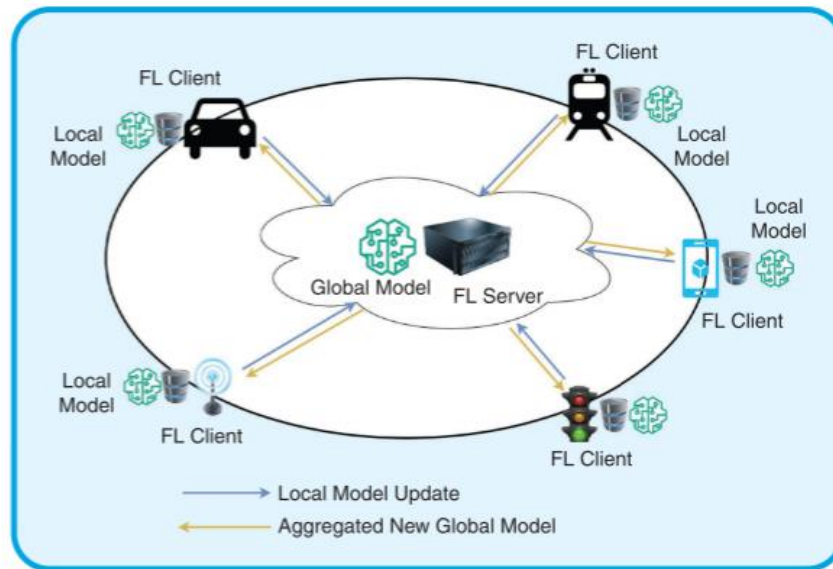
Gambar 2.3 Arsitektur pelacakan transportasi publik

Sumber: (Jimoh et al., 2020)

Penelitian yang dilakukan oleh Onemayin David Jimoh dan timnya di Nigeria mengembangkan sistem pelacakan kendaraan untuk transportasi umum di jalan arteri perkotaan dengan menggunakan algoritma *Greedy Forwarding (GFA)*. Sistem ini menggabungkan teknologi *GPS* dan komunikasi *GSM* untuk melacak posisi bus secara *real-time* dan menampilkan informasi kedatangan kendaraan di terminal secara akurat. Algoritma *GFA* digunakan untuk memilih jalur optimal berdasarkan data geolokasi, dan model matematika diterapkan untuk menghitung jarak dan waktu tempuh dengan presisi tinggi.

Dalam hal biaya, sistem pelacakan ini dirancang untuk menjadi solusi yang ekonomis bagi manajemen transportasi umum. Biaya implementasi sistem ini diperkirakan sekitar \$2.570 per terminal, yang mencakup perangkat keras seperti modul *GPS*, modul *GSM*, mikrokomputer (seperti *Raspberry Pi*), dan layar *LED* untuk menampilkan informasi. Meskipun sistem pelacakan kendaraan berbasis *GPS* dan *GSM* yang dikembangkan oleh Onemayin David Jimoh dan timnya di Nigeria menawarkan keunggulan dalam hal akurasi dan efektivitas, biaya implementasi yang diperkirakan sekitar \$2.570 per terminal memang tergolong lebih tinggi jika dibandingkan dengan pendekatan lain seperti penggunaan *BLE* beacon, *Raspberry Pi* serta *smartphone Android*. Teknologi *BLE* yang dikombinasikan dengan *Raspberry Pi* dan *smartphone Android* memungkinkan pelacakan bus dengan biaya yang jauh lebih rendah, karena perangkat-perangkat tersebut relatif murah dan mudah diimplementasikan.

2.1.4 Federated Learning for Intelligent Transportation Systems: Use Cases, Open Challenges, and Opportunities



Gambar 2.4 Arsitektur Federated Learning

Sumber: (Chong et al., 2024)

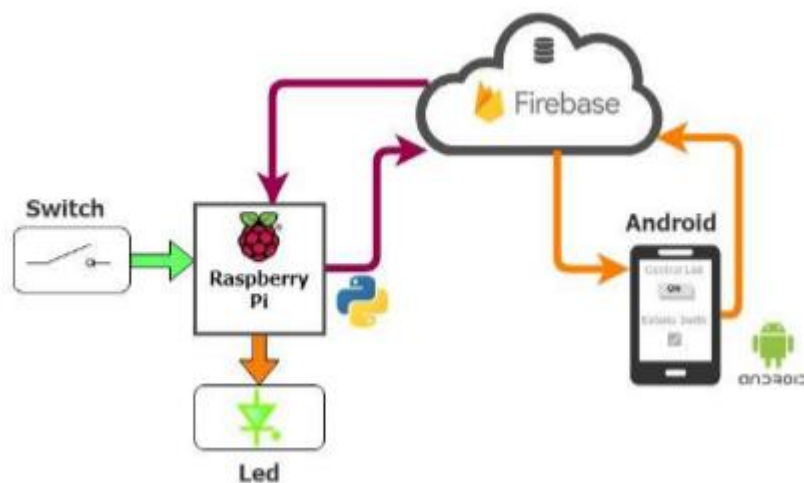
Penelitian ini mengkaji penggunaan *Federated Learning (FL)* dalam sistem transportasi pintar untuk meningkatkan pengelolaan lalu lintas tanpa melanggar privasi pengguna. *FL* memungkinkan perangkat dalam jaringan transportasi, seperti kendaraan dan *RSU*, untuk melatih model *machine learning* secara kolaboratif tanpa berbagi data mentah. Dalam sistem ini, setiap perangkat melatih modelnya sendiri dan mengirim pembaruan model ke *server* pusat yang kemudian mengagregasi hasil pembaruan untuk menghasilkan *model global*. Sistem ini digunakan untuk kasus seperti prediksi arus lalu lintas dan manajemen kemacetan.

Hasil dari penelitian ini menunjukkan bahwa *FL* dapat meningkatkan akurasi prediksi arus lalu lintas serta manajemen kemacetan tanpa harus mengumpulkan data mentah dari setiap perangkat. Dengan demikian, sistem transportasi pintar dapat menjaga privasi pengguna sambil memanfaatkan data dari berbagai sumber secara efisien. *FL* memungkinkan adaptasi sistem terhadap kondisi lalu lintas secara *real-time*, menghasilkan prediksi dan penyesuaian yang tepat waktu.

Kendala utama dalam sistem ini terkait dengan konsumsi *bandwidth* dan latensi dalam komunikasi antara perangkat tepi dan *server* pusat. Transfer model

pembaruan secara teratur dapat membebani jaringan, terutama saat jumlah perangkat yang terlibat dalam pelatihan meningkat. Penggunaan kompresi data dan pengoptimalan frekuensi pembaruan model dapat membantu mengurangi beban jaringan. Selain itu, penelitian lebih lanjut tentang manajemen data heterogen yang dihasilkan oleh perangkat yang berbeda dalam jaringan transportasi juga diperlukan untuk meningkatkan interoperabilitas dan efisiensi sistem secara keseluruhan.

2.1.5 Design and Development of a sustainable telemetry system for environmental parameters



Gambar 2.5 Arsitektur Sistem *Telemetry* Suhu Lingkungan

Sumber: (Vargas et al., 2021)

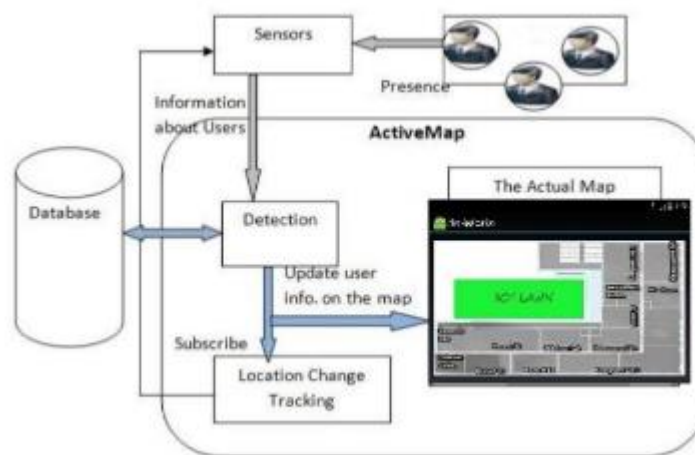
Penelitian ini memfokuskan pada pengembangan sistem telemetri yang berkelanjutan untuk memantau parameter lingkungan seperti suhu dan kelembaban. Sistem ini menggunakan *Raspberry Pi* yang ditenagai oleh panel surya sebagai sumber daya, serta sensor *DHT11* untuk mengukur parameter lingkungan. Data yang dikumpulkan kemudian dikirimkan ke server melalui jaringan nirkabel untuk pemantauan jarak jauh secara real-time. Penggunaan daya surya membuat sistem ini lebih berkelanjutan dan dapat beroperasi di lokasi terpencil tanpa tergantung pada sumber daya listrik konvensional.

Hasil penelitian menunjukkan bahwa sistem telemetri ini dapat beroperasi secara mandiri dengan stabil menggunakan daya dari panel surya. Sistem ini berhasil mengirimkan data parameter lingkungan secara akurat ke server, yang memungkinkan pemantauan kondisi lingkungan di lokasi-lokasi yang sulit dijangkau. Selain itu, penggunaan teknologi energi terbarukan menjadikan sistem

ini sebagai solusi yang hemat energi dan ramah lingkungan, memenuhi kebutuhan pemantauan lingkungan dengan biaya operasional yang rendah.

Beberapa tantangan masih ada dalam sistem ini, terutama terkait kestabilan daya saat hari mendung atau saat intensitas sinar matahari berkurang. Untuk menjaga sistem tetap aktif dalam kondisi cuaca buruk, peningkatan dapat dilakukan dengan menambahkan baterai berkapasitas lebih besar atau sumber energi alternatif sebagai cadangan. Di samping itu, peningkatan efisiensi penggunaan daya pada perangkat *Raspberry Pi* dan sensor juga dapat membantu memperpanjang umur operasional sistem, menjadikannya lebih andal untuk pemantauan jangka panjang.

2.1.6 An Indoor Tracking System using iBeacon and Android



Gambar 2.6 Arsitektur pelacakan *Indoor BLE Beacon*

Sumber: (Moneer et al., 2020)

Penelitian ini mengembangkan sistem pelacakan dalam ruangan berbasis teknologi *iBeacon* dan perangkat *Android*. Sistem ini dirancang untuk melacak posisi pengguna secara akurat di dalam gedung menggunakan sinyal *BLE* yang dikirimkan oleh perangkat *iBeacon*. Teknologi ini dianggap sebagai alternatif yang lebih hemat biaya dan energi dibandingkan teknologi lain seperti *WiFi* atau *GPS* untuk penggunaan di dalam ruangan, di mana *GPS* memiliki keterbatasan. *iBeacon* dipasang di titik-titik strategis dalam gedung untuk mendeteksi pergerakan dan lokasi pengguna, sehingga cocok untuk digunakan di tempat-tempat seperti rumah sakit, pusat perbelanjaan, dan kampus.

Hasil penelitian menunjukkan bahwa sistem berbasis *iBeacon* ini mampu memberikan akurasi pelacakan yang lebih tinggi dibandingkan dengan metode

pelacakan dalam ruangan lainnya. Dengan menggunakan teknologi *BLE*, sistem ini mengonsumsi daya yang rendah sehingga efisien dan tahan lama dalam penggunaannya. Selain itu, hasil uji coba menunjukkan bahwa perangkat *iBeacon* dapat diintegrasikan dengan perangkat *Android* untuk mencapai pelacakan yang *real-time*, memudahkan pengguna untuk mendapatkan informasi lokasi dengan cepat.

Meskipun memberikan hasil yang positif, sistem ini masih memiliki beberapa kendala, terutama terkait fluktuasi sinyal *BLE* akibat interferensi dari objek fisik atau lingkungan sekitar. Fluktuasi ini dapat menurunkan akurasi dalam beberapa kondisi tertentu, terutama di area dengan banyak penghalang. Untuk meningkatkan stabilitas dan akurasi, penelitian di masa mendatang dapat fokus pada pengembangan algoritma *filtering* yang lebih baik atau penggunaan perangkat tambahan yang dapat menstabilkan sinyal BLE dalam lingkungan yang kompleks.

2.2 Dasar Teori

Untuk menunjang penelitian ini, diperlukan beberapa dasar teori untuk beberapa hal yang akan digunakan dalam penelitian ini. Hal-hal tersebut akan dijelaskan pada sub-bab ini.

2.2.1. IOT (Internet of Things).

Internet of things (IoT) merupakan konsep yang sudah berkembang pesat dalam beberapa tahun terakhir. Perangkat atau “things” dalam IoT terhubung ke dalam suatu jaringan internet dan memungkinkan untuk berbagai informasi dan berkomunikasi satu sama lain (Sadhu et al., 2022). IoT memungkinkan terciptanya ekosistem yang saling terhubung, di mana data dari berbagai sumber dapat dianalisis untuk menghasilkan wawasan yang lebih mendalam dan memungkinkan pengambilan keputusan yang lebih tepat.

Sistem Pengumpulan Data Transportasi Umum Menggunakan *Bluetooth Proximity Beacons* termasuk dalam kategori *IoT* karena memiliki karakteristik yang sesuai dengan konsep *IoT*. Pertama, *IoT* melibatkan perangkat fisik yang saling terhubung melalui internet, dan dalam sistem ini, *beacons* yang dipasang di bus serta *RSU* sebagai pusat pemrosesan data berkomunikasi melalui sinyal *bluetooth*. Data yang dikumpulkan oleh *RSU* dapat diteruskan ke *server* atau *platform cloud* untuk dianalisis dan dipantau. Selain itu, sistem ini memungkinkan pengumpulan dan pertukaran data secara otomatis tanpa perlu interaksi manusia, karena

beacons secara otomatis memancarkan sinyal yang dideteksi oleh RSU, yang kemudian mengirimkan data tersebut ke pusat pemantauan.

2.2.2. Bluetooth Low Energy Beacon.

Bluetooth adalah komunikasi antara 2 device atau lebih menggunakan frekuensi radio pada jarak dekat. Seiring perkembangan zaman, teknologi bluetooth sudah berkembang sehingga dapat melakukan komunikasi dengan konsumsi daya yang sangat minim. Teknologi ini disebut dengan Bluetooth Low Energy.

Bluetooth Low Energy beacon adalah device yang memancarkan sinyal radio dengan frekuensi dan interval tertentu. *Beacon* ini hanya dapat memancarkan sinyal radio yang akan ditangkap oleh device lain. Masa hidup dari beacon ini relatif cukup lama seminimal mungkin 1-2 tahun.

Penggunaan *BLE beacon* dibandingkan dengan *GPS* memiliki beberapa keunggulan yang menjadikannya lebih cocok untuk aplikasi seperti pengumpulan data transportasi umum. Salah satu keunggulan utamanya adalah efisiensi daya, di mana BLE beacon dirancang untuk menggunakan energi yang sangat rendah, memungkinkan perangkat ini bertahan hingga 1-2 tahun dengan baterai kecil. Sebaliknya, *GPS* mengonsumsi lebih banyak daya karena memerlukan pemrosesan data lokasi secara terus-menerus. Selain itu, *BLE beacon* lebih akurat di area terbatas seperti lingkungan sekolah atau halte bus, sedangkan keakuratan *GPS* bisa berkurang di area tertutup atau wilayah dengan bangunan tinggi. Dari segi biaya, *BLE beacon* lebih murah dan lebih mudah diimplementasikan karena tidak memerlukan infrastruktur kompleks seperti *GPS* yang membutuhkan perangkat dengan kemampuan pemrosesan lokasi yang lebih tinggi.



Gambar 2.7 *Bluetooth Low Energy Estimote*

Sumber: (<https://estimote.com>)

Salah satu contoh BLE beacon yang banyak digunakan adalah *Estimote Beacon*, yang dikenal karena keandalannya dalam berbagai aplikasi. *Beacon* ini dilengkapi dengan prosesor *ARM Cortex M0* dan memiliki memori *256KB flash*

serta *RAM 32KB*, yang cukup untuk menjalankan berbagai fitur pemancaran data. *Estimote Beacon* menggunakan baterai A2 berkapasitas 10000mAh, yang memungkinkan daya tahan hingga 3-5 tahun tergantung pada frekuensi dan kekuatan sinyal yang dipancarkan. *Beacon* ini dapat mengirimkan berbagai data, termasuk *UUID (Universal Unique Identifier)*, *Major* dan *Minor values* untuk identifikasi, serta informasi tambahan seperti *BLE Address*, *RSSI(Received Signal Strength Indicator)*, *Tx Power* tergantung pada modelnya. *Estimote Beacon* juga mendukung konfigurasi interval sinyal dan kekuatan transmisi, sehingga dapat dioptimalkan untuk berbagai kebutuhan aplikasi dengan efisiensi daya yang maksimal. Spesifikasi ini membuat *Estimote Beacon* menjadi solusi yang ekonomis dan fleksibel untuk sistem berbasis *BLE*.

Data yang dikirimkan oleh *estimote BLE beacon* dapat bervariasi tergantung jenis yang digunakan. Namun ada beberapa data yang umum digunakan adalah sebagai berikut:

Tabel 2.2 Data Transmisi BLE Beacon

Data	Deskripsi	Ukuran Data
<i>UUID</i>	<i>Universally Unique Identifier (16-byte ID)</i> yang membedakan grup <i>beacon</i> tertentu.	<i>16 Byte</i>
<i>Major</i>	<i>Subset ID</i> dalam grup <i>UUID</i> untuk mengidentifikasi lokasi atau area tertentu.	<i>2 Byte</i>
<i>Minor</i>	Identifikasi individu <i>beacon</i> dalam <i>subset Major</i> , misalnya <i>beacon</i> tertentu dalam sebuah ruangan.	<i>2 Byte</i>
<i>Tx Power</i>	Tingkat kekuatan transmisi yang digunakan untuk menghitung jarak berdasarkan intensitas sinyal.	<i>1 Byte</i>
<i>Ble Address</i>	Alamat unik perangkat BLE untuk membedakan setiap <i>beacon</i> .	<i>6 Byte</i>
<i>RSSI</i>	<i>Received Signal Strength Indicator</i> , memberikan intensitas sinyal yang diterima perangkat penerima.	<i>1 Byte</i>

2.2.3. Raspberry pi 4.



Gambar 2.8 Raspberry pi 4.

Raspberry Pi 4 adalah versi terbaru dari komputer kecil yang dirancang untuk berbagai aplikasi, mulai dari pendidikan dan pengembangan teknologi. Dibandingkan dengan pendahulunya *Raspberry Pi 3*, *Raspberry Pi 4* menawarkan peningkatan performa yang signifikan dengan prosesor *quad-core ARM Cortex-A72 64-bit* yang berjalan pada kecepatan 1.5 GHz. Dengan pilihan kapasitas *RAM* yang bervariasi, mulai dari 2GB hingga 8GB, *Raspberry Pi 4* mampu menangani tugas-tugas yang lebih kompleks dan menjalankan aplikasi berat dengan lebih efisien.

Keunggulan utama dari *Raspberry Pi* adalah fleksibilitasnya. *Raspberry Pi* dapat digunakan dalam berbagai proyek, seperti pengembangan sistem *IoT*, sistem keamanan, pengendalian perangkat rumah, bahkan hingga menjadi perangkat untuk pengajaran pemrograman di sekolah. Selain itu, komunitas yang besar dan ekosistem yang luas menjadikan *Raspberry Pi* sebagai alat yang sangat populer di kalangan penggemar teknologi, pengembang, dan pendidik di seluruh dunia.

Raspberry Pi 4 juga dilengkapi dengan dukungan konektivitas yang lebih baik, termasuk *Port USB* terdiri dari dua *port USB 3.0* dan dua *port USB 2.0*, yang memberikan kecepatan transfer data lebih tinggi untuk perangkat eksternal. *Raspberry Pi 4* memiliki konektivitas jaringan yang lebih cepat dengan *Ethernet gigabit* serta modul *Wi-Fi dual-band 802.11ac* dan *Bluetooth 5.0* untuk konektivitas nirkabel. Dengan slot kartu *microSD* untuk penyimpanan utama dan *port GPIO 40-pin* untuk ekspansi perangkat keras, *Raspberry Pi 4* menawarkan fleksibilitas dan kinerja yang memadai untuk berbagai kebutuhan.

2.2.4. Ponsel Android.



Gambar 2.9 Smartphone Redmi 12C

Ponsel *Android* merupakan perangkat mobile berbasis sistem operasi *Android* yang memiliki fleksibilitas tinggi dalam mendukung aplikasi yang dirancang untuk beragam kebutuhan. Salah satu keunggulan utama dari perangkat ini adalah ketersediaan *Android Beacon Library*, pustaka yang memungkinkan deteksi dan pengelolaan perangkat *BLE beacon* dengan mudah. Dalam penelitian ini, ponsel *Android* digunakan sebagai *scanner* tambahan untuk mendeteksi sinyal *BLE* yang dipancarkan oleh *beacon* yang terpasang pada bus sekolah. Perangkat ini bertindak sebagai pelengkap sistem utama, membantu dalam pengumpulan data di lokasi yang lebih dinamis, seperti halte bus atau persimpangan jalan tertentu.

Redmi 12C hadir dengan spesifikasi yang menarik di kelas *entry-level*. Ponsel ini ditenagai oleh prosesor *MediaTek Helio G85*, yang menawarkan kinerja optimal untuk aktivitas harian dan aplikasi ringan hingga menengah, termasuk aplikasi berbasis *BLE*. Terdapat beberapa opsi *RAM* dan penyimpanan, mulai dari 3GB/32GB hingga 6GB/128GB, dengan dukungan untuk ekspansi melalui kartu *microSD* hingga 1TB, sehingga memberikan fleksibilitas dalam penyimpanan data.

Dengan baterai besar berkapasitas 5000mAh, perangkat ini mampu bertahan hingga seharian penuh dengan pemakaian normal, serta mendukung pengisian daya 10W melalui *port micro-USB*. Fitur lainnya termasuk konektivitas *Bluetooth 5.0*, *Wi-Fi 802.11 b/g/n*, dan dukungan *Dual SIM*. Spesifikasi ini menjadikan *Redmi 12C* tidak hanya ekonomis, tetapi juga cukup mumpuni untuk memenuhi kebutuhan berbagai aplikasi, termasuk sistem berbasis *BLE* dalam proyek atau penelitian tertentu.

Selain itu, ponsel Android dipilih karena sifatnya yang portabel dan kemampuannya untuk menjalankan berbagai aplikasi berbasis *API*. Dalam konteks penelitian, aplikasi yang dirancang untuk ponsel *Android* dapat dikonfigurasi untuk mengirimkan data secara langsung ke *server cloud*. Ponsel ini juga memiliki antarmuka pengguna yang intuitif, sehingga memungkinkan operator atau pengguna untuk dengan mudah mengakses informasi kondisi aplikasi yang dijalankan. Dengan kombinasi fitur-fitur tersebut, perangkat *Android* menjadi komponen yang efektif dan ekonomis dalam membangun sistem pelacakan transportasi umum berbasis BLE.

2.2.5. Telemetry.

Telemetry adalah proses pengumpulan, pengukuran, dan pengiriman data dari lokasi yang jauh ke sistem pusat untuk dianalisis dan dimanfaatkan lebih lanjut. Dalam penelitian ini, telemetry digunakan untuk memantau kondisi perangkat keras yang terpasang, seperti Raspberry Pi, serta memastikan keberlanjutan operasional sistem. Proses telemetry melibatkan pengukuran parameter penting, seperti suhu prosesor Raspberry Pi, kekuatan sinyal BLE, dan status koneksi internet. Data ini dikirimkan ke server secara berkala untuk memantau performa sistem secara real-time.

Penggunaan telemetry sangat penting untuk memastikan keandalan sistem dalam kondisi lingkungan yang beragam. Sebagai contoh, jika perangkat Raspberry Pi mengalami panas berlebih atau kehilangan koneksi internet, sistem dapat segera mengambil tindakan, seperti mengirimkan notifikasi ke operator atau melakukan reset perangkat secara otomatis. Dengan demikian, telemetry tidak hanya berfungsi sebagai alat pemantauan, tetapi juga sebagai bagian integral dari sistem pemeliharaan prediktif yang membantu mengurangi potensi kerusakan dan gangguan operasional.

2.2.6. Node Js.

Node.js adalah lingkungan *runtime* berbasis *JavaScript* yang dirancang untuk membangun aplikasi jaringan yang ringan dan skalabel. Dalam penelitian ini, *Node.js* digunakan untuk menjalankan program yang menangani pemrosesan data *BLE beacon* serta komunikasi *raspberry pi 4* dengan *server cloud*. *Node.js* memiliki keunggulan dalam menangani banyak koneksi secara simultan melalui model pemrograman berbasis *event-driven* dan *non-blocking I/O*, menjadikannya pilihan ideal untuk aplikasi *real-time* seperti sistem pelacakan berbasis BLE.

Keunggulan lain dari *Node.js* adalah ekosistem pustakanya yang luas, yang memungkinkan pengembang untuk dengan mudah mengintegrasikan fitur-fitur tambahan seperti pengelolaan *MQTT* untuk komunikasi antara perangkat *IoT* dan *server*. Selain itu, kompatibilitasnya dengan berbagai *platform* dan kemampuannya untuk berjalan pada perangkat keras dengan sumber daya terbatas, seperti *Raspberry Pi*, menjadikan *Node.js* sebagai pilihan bagus dalam implementasi sistem *IoT* yang kompleks namun efisien.

2.2.7. Bore Client.

Bore Client adalah perangkat lunak berbasis *CLI (Command Line Interface)* yang dirancang untuk membuat terowongan (*tunnel*) jaringan, termasuk untuk akses jarak jauh melalui *SSH(Secure Socket Shell)*. Dengan menggunakan *Bore*, port lokal pada perangkat dapat diekspos ke *server* jarak jauh melalui koneksi *TCP*, memungkinkan akses ke layanan seperti *SSH* tanpa konfigurasi tambahan pada *firewall* atau *NAT(Network Address Translation)*.

Dalam konteks remote access melalui *SSH*, *Bore* dapat digunakan untuk membuat terowongan dengan cara mengekspos *port SSH* atau port 22 ke internet publik.

Keunggulan *Bore* dalam akses jarak jauh adalah kemudahan penggunaannya, desainnya yang ringan, serta kemampuan untuk menghindari keterbatasan jaringan seperti *firewall NAT* tanpa memerlukan konfigurasi yang rumit. Selain itu, pengguna juga dapat menjalankan *server Bore* secara mandiri untuk meningkatkan keamanan dan kontrol atas data yang ditransfer. Informasi lengkap dari *bore client server* dapat diakses melalui *link github* berikut. <https://github.com/ekzhang/bore>

2.2.8. Axios.

Axios adalah pustaka *JavaScript* berbasis *HTTP* yang memfasilitasi pengiriman permintaan ke server, seperti *GET*, *POST*, *PUT*, dan *DELETE*, secara asinkron. Dalam penelitian ini, *Axios* digunakan untuk mengirimkan data seperti informasi sinyal *BLE beacon* dan *telemetry* dari *Raspberry Pi 4* ke server cloud. *Axios* menawarkan berbagai fitur, termasuk dukungan untuk transformasi data secara otomatis ke dalam bentuk *JSON (JavaScript Object Notation)*, penanganan *timeout*, dan intersepsi permintaan dan *respons*.

Fleksibilitas *Axios* dalam menangani berbagai metode *HTTP* dan integrasi mudah dengan pustaka lain menjadikannya pilihan utama untuk aplikasi berbasis jaringan. Dalam konteks sistem ini, *Axios* membantu menjaga aliran data *real-time* yang bagus dari perangkat *IoT* ke *server cloud*, memastikan data diterima dengan cepat untuk analisis atau pengolahan lebih lanjut. Informasi lengkap dari *Axios* dapat diakses melalui *link github* berikut <https://github.com/axios/axios>.

2.2.9. Node Beacon scanner.

Node Beacon Scanner adalah modul berbasis *Node.js* yang dirancang untuk memindai sinyal dari perangkat *BLE beacon* dan menguraikan datanya. Modul ini mendukung format *beacon* populer seperti *iBeacon*, *Eddystone*, dan *Estimote*, sehingga cocok untuk berbagai aplikasi dalam *IoT*. Dengan menggunakan pustaka *noble*, *Node Beacon Scanner* dapat menangkap data dari *advertising packets* yang dipancarkan oleh *beacon*, termasuk informasi seperti *UUID*, *RSSI*, dan *TxPower*.

Modul ini menggunakan pendekatan berbasis *event* untuk pemrosesan data, di mana setiap kali *beacon* terdeteksi, *event handler* akan menerima objek data yang mencakup detail teknis *beacon*. Pengguna hanya perlu membuat objek *BeaconScanner*, memulai proses pemindaian dengan metode *startScan()*, dan menangani data melalui *event handler onadvertisement*. Modul ini tidak hanya mempermudah pengelolaan sinyal BLE tetapi juga memungkinkan pengguna untuk melakukan pemilahan dan mengoptimalkan data yang diterima, sehingga lebih akurat dalam lingkungan yang penuh interferensi sinyal.

Dalam sistem pelacakan bus sekolah, *Node Beacon Scanner* dapat digunakan untuk mengidentifikasi *beacon* yang dipasang pada kendaraan. Data seperti *UUID* dan *RSSI* dari *beacon* dikirimkan ke server untuk memperkirakan waktu kedatangan bus. Dengan kemampuan modul ini untuk memilah sinyal yang relevan, efisiensi dan akurasi sistem pelacakan dapat ditingkatkan secara signifikan. Dokumentasi lebih lengkap tentang modul ini dapat ditemukan pada <https://github.com/futomi/node-beacon-scanner>.

2.2.10. Retrofit.

Retrofit adalah pustaka *HTTP* berbasis *Java* untuk aplikasi *Android*, dirancang untuk menyederhanakan proses komunikasi dengan *server API RESTful*. Dengan *Retrofit*, pengembang dapat memetakan *endpoint API* menjadi *interface Java*, membuat kode lebih bersih dan mudah dikelola. Dalam penelitian ini,

Retrofit digunakan untuk mengelola transfer data antara aplikasi *Android* dan *server*, seperti pengiriman dan pengambilan informasi estimasi waktu kedatangan bus.

Keunggulan utama *Retrofit* adalah kemampuannya untuk memetakan respons API langsung ke objek *Java* menggunakan pustaka seperti *Gson* untuk *parsing data JSON*. Pustaka ini juga mendukung berbagai fitur seperti *interceptors*, *retry logic*, dan dukungan *asinkron* melalui *callback* atau *Kotlin coroutines*, menjadikannya alat yang fleksibel untuk aplikasi *real-time*.

Penggunaan *Retrofit* dalam aplikasi *Android* memungkinkan pengembang menyajikan informasi lokasi dan waktu kedatangan bus secara akurat dan *real-time* kepada pengguna. Dokumentasi dan kode sumber *Retrofit* dapat ditemukan di <https://square.github.io/retrofit/>.

2.2.11. AirDroid.

AirDroid adalah aplikasi untuk mengelola perangkat *Android* secara jarak jauh melalui koneksi internet. Dalam penelitian ini, *AirDroid* digunakan untuk memantau dan mengontrol ponsel *Android* yang berfungsi sebagai *scanner BLE beacon*. Aplikasi ini memungkinkan operator untuk memeriksa status perangkat, memperbarui aplikasi, atau menyelesaikan masalah teknis tanpa perlu interaksi fisik dengan perangkat.

Keunggulan *AirDroid* adalah kemampuannya untuk menyediakan antarmuka yang mudah digunakan dan dukungan yang luas untuk berbagai model perangkat *Android*. Dengan alat ini, proses pemeliharaan perangkat menjadi lebih efisien, mengurangi waktu dan biaya yang dibutuhkan untuk manajemen perangkat secara manual.

2.2.12. Android Beacon Library.

Android Beacon Library adalah pustaka *open-source* yang digunakan untuk mendeteksi dan memproses sinyal *BLE beacon* pada perangkat *Android*. Dalam penelitian ini, pustaka ini digunakan untuk membangun aplikasi yang dapat mengidentifikasi *beacon* yang relevan dan mengirimkan data ke *server cloud*. *Android Beacon Library* mendukung berbagai fungsi lanjutan seperti *filter* sinyal, pengelompokan *beacon*, dan pengelolaan *event* berbasis *proximity*.

Dengan pustaka ini, aplikasi Android dapat mendeteksi *beacon* secara efisien bahkan di lingkungan yang penuh dengan interferensi. Fitur-fitur tersebut menjadikannya alat yang sangat berguna dalam implementasi sistem pelacakan berbasis *BLE* yang kompleks.

2.2.13. AWS.

Amazon Web Services (AWS) adalah platform komputasi awan yang menyediakan berbagai layanan seperti penyimpanan data, analitik, dan pemrosesan. Dalam penelitian ini, AWS digunakan sebagai infrastruktur utama untuk menyimpan dan menganalisis data yang dikumpulkan oleh sistem. Dengan memanfaatkan layanan seperti *AWS IoT Core* dan *AWS Lambda*, sistem dapat memproses data secara real-time untuk memberikan estimasi waktu kedatangan bus kepada pengguna.

Keandalan dan skalabilitas AWS menjadikannya solusi yang ideal untuk sistem pelacakan berbasis *IoT*. Selain itu, kemampuan AWS untuk menyediakan analitik data yang mendalam membantu dalam pengembangan model prediksi yang lebih akurat, sehingga meningkatkan kualitas layanan yang diberikan kepada pengguna.

2.2.14. Transportasi Umum.

Transportasi umum merupakan salah satu sarana mobilitas yang vital bagi masyarakat, terutama di kawasan perkotaan seperti Malang. Keberadaan transportasi umum membantu mengurangi ketergantungan pada kendaraan pribadi, yang secara langsung berkontribusi pada pengurangan kemacetan lalu lintas dan emisi karbon. Di kota-kota besar, termasuk Malang, transportasi umum menjadi solusi yang ideal untuk mendukung mobilitas harian, baik bagi pekerja, pelajar, maupun masyarakat umum. Namun, di Indonesia, termasuk di Kota Malang, sistem transportasi umum masih menghadapi berbagai tantangan, seperti ketidakandalan waktu kedatangan, kurangnya integrasi rute, dan layanan yang tidak merata, sehingga pengguna sering kali lebih memilih kendaraan pribadi atau transportasi berbasis aplikasi.

Bus sekolah di Malang adalah salah satu jenis transportasi umum yang dirancang khusus untuk mendukung mobilitas pelajar. Bus ini menyediakan layanan yang menghubungkan kawasan permukiman dengan berbagai sekolah di kota, seperti SMP, SMA, dan madrasah. Dengan rute yang terencana dan jadwal

keberangkatan yang tetap, bus sekolah berperan penting dalam memastikan siswa dapat tiba di sekolah tepat waktu tanpa perlu menggunakan kendaraan pribadi. Selain mengurangi beban lalu lintas, keberadaan bus sekolah juga membantu meringankan biaya transportasi bagi keluarga, terutama di kawasan dengan akses transportasi terbatas. Namun demikian, layanan bus sekolah di Malang masih memerlukan peningkatan, baik dari segi cakupan rute, kenyamanan, maupun keandalan jadwal, untuk memastikan transportasi ini dapat menjadi pilihan utama bagi siswa dan masyarakat.

2.2.15. Kotlin.

Kotlin adalah bahasa pemrograman modern yang dikembangkan oleh *JetBrains* dan dirancang untuk kompatibel sepenuhnya dengan *Java*. *Kotlin* sering digunakan dalam pengembangan aplikasi *Android* karena sintaksnya yang ringkas, aman, dan mendukung fitur-fitur canggih seperti *null safety* dan *coroutine* untuk pemrograman asinkron. Dalam sistem ini, *Kotlin* digunakan untuk mengembangkan aplikasi *mobile* yang memungkinkan pengumpulan, pemrosesan dari perangkat *BLE beacon*. Dengan kemampuannya yang kuat dalam pemrosesan data dan integrasi yang mudah dengan berbagai pustaka *Android*, *Kotlin* menjadi pilihan ideal untuk membangun aplikasi yang stabil dan responsif.

2.2.16. Box Metal UMG

Box Metal UMG adalah kotak pelindung berbahan logam dengan dimensi 20x30x12 cm yang dirancang khusus untuk melindungi perangkat keras dari kondisi lingkungan yang ekstrem. Dengan konstruksi logam yang kokoh, kotak ini mampu melindungi perangkat dari paparan hujan, debu, suhu tinggi, dan benturan fisik, sehingga sangat cocok untuk penggunaan di area luar ruangan. Ukuran 20x30x12 cm memberikan ruang yang cukup untuk menampung perangkat seperti *Raspberry Pi 4*, adaptor daya, dan komponen lainnya, dengan tetap mempertahankan portabilitas dan efisiensi ruang. *Box* ini juga dilengkapi dengan mekanisme kunci untuk meningkatkan keamanan perangkat keras, sehingga mengurangi risiko akses tidak sah atau kerusakan.

2.2.17. Power Adapter LDNIO A4610C.

Power Adapter LDNIO A4610C adalah adaptor daya *multi-port* yang mendukung berbagai perangkat elektronik, termasuk *Raspberry Pi 4* dan *smartphone*. Adaptor ini memiliki kombinasi port *USB-C* dan *USB-A*, dengan *output*

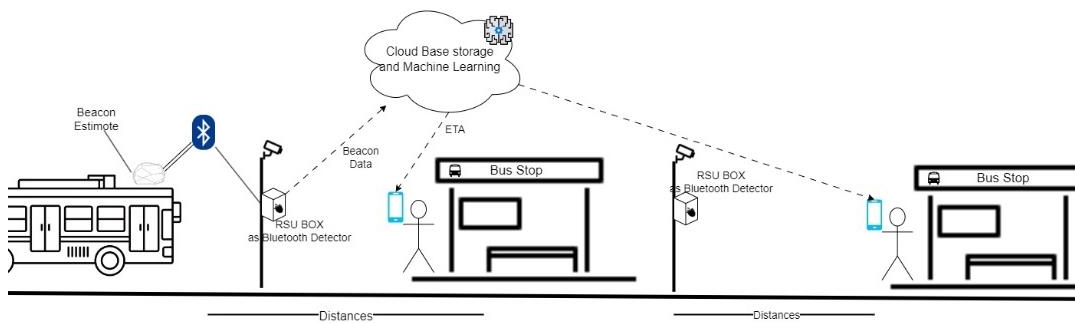
daya hingga 5V/3A yang cocok untuk memenuhi kebutuhan daya perangkat seperti *Raspberry Pi*. Teknologi pengisian cepat seperti *Power Delivery* (PD) dan *Quick Charge* (QC) memastikan perangkat dapat diisi dengan efisien tanpa risiko *overvoltage* atau *overcurrent*. Desainnya yang ringkas dan multifungsi memungkinkan adaptor ini digunakan secara fleksibel di berbagai kebutuhan operasional, termasuk dalam sistem pengumpulan data transportasi umum berbasis *BLE beacon*.

2.2.18 PM2

PM2 (Process Manager 2) adalah manajer proses yang digunakan untuk menjalankan, memantau, dan mengelola aplikasi *JavaScript*. *PM2* sangat cocok digunakan pada *Raspberry Pi 4* untuk memastikan aplikasi berjalan otomatis saat perangkat dihidupkan kembali. Dengan fitur seperti manajemen *log*, *PM2* memungkinkan pengguna untuk melihat *output* dan *error* dari setiap program secara *real-time*. *PM2* juga mendukung *restart otomatis* jika aplikasi mengalami *crash*, sehingga memastikan sistem tetap berjalan stabil. Dalam konteks sistem berbasis *BLE beacon*, *PM2* dapat digunakan untuk menjaga proses pengumpulan data tetap berjalan secara konsisten.

BAB III METODOLOGI PENELITIAN

Bab ini akan menjelaskan mengenai rencana kerja penelitian. Tipe penelitian serta rancangan yang akan dilakukan pada penelitian “Sistem Pengumpulan Data Transportasi Umum Menggunakan Bluetooth Proximity Beacons.”



Gambar 3.1 Perancangan Sistem

Gambar 3.1 menunjukkan arsitektur sistem pelacakan transportasi berbasis *BLE* yang digunakan dalam penelitian ini. Sistem ini terdiri dari beberapa komponen utama, yaitu beacon *BLE* yang dipasang pada bus, perangkat *RSU* yang berfungsi sebagai pendeteksi sinyal, dan server berbasis cloud yang memanfaatkan *machine learning*.

Beacon BLE yang dipasang pada bus memancarkan sinyal unik yang dapat terdeteksi oleh *RSU*. *RSU* dipasang di tiang strategis, seperti tiang CCTV di dekat halte bus, dan dilengkapi dengan perangkat *scanner* seperti *Raspberry Pi 4*, *Smartphoen Android*, dan perangkat pendukung lainnya untuk mendeteksi sinyal *BLE* dari bus yang melintas. Informasi yang diperoleh dari deteksi sinyal *BLE*, seperti identitas bus dan waktu deteksi, kemudian dikirimkan ke server cloud untuk diproses lebih lanjut. Di server, data ini digunakan untuk menghitung estimasi waktu kedatangan di halte atau *RSU* berikutnya.

Hasil prediksi waktu kedatangan akan dikirimkan kembali ke aplikasi pengguna, sehingga penumpang dapat mengetahui waktu kedatangan bus secara *real-time*. Arsitektur ini dirancang untuk memastikan sistem pelacakan bus yang efisien, ekonomis, dan dapat diandalkan dalam berbagai kondisi lingkungan perkotaan. Dengan pengintegrasian teknologi *cloud* dan *machine learning*, sistem ini mampu memprediksi kedatangan bus dengan akurasi yang lebih baik, membantu pengguna merencanakan perjalanan mereka secara lebih efektif.

Penelitian ini berfokus pada tahap pengumpulan data yang akan digunakan sebagai data latih untuk pengembangan model *machine learning* dalam memprediksi estimasi waktu kedatangan bus. Data yang dikumpulkan meliputi waktu deteksi sinyal *BLE*, identitas unik *beacon* pada bus, serta parameter lingkungan seperti intensitas sinyal. Proses pengumpulan data dilakukan melalui perangkat *RSU* yang dipasang di sepanjang rute bus sekolah. Data ini akan menjadi *input* penting untuk membangun model prediktif berbasis *machine learning* yang bertujuan meningkatkan akurasi prediksi waktu kedatangan. Penelitian ini tidak mencakup implementasi penuh dari sistem prediksi waktu kedatangan, melainkan hanya berfokus pada pengumpulan dataset yang relevan untuk tahap pengembangan dan pelatihan model di masa mendatang.

3.1 Tipe Penelitian

Penelitian ini merupakan jenis penelitian implementatif pengembangan lanjut yang bertujuan untuk mengaplikasikan teori yang telah diselidiki sebelumnya ke dalam sistem nyata. Fokus utama dari penelitian ini adalah mengembangkan dan mengimplementasikan sistem pelacakan transportasi berbasis *BLE* pada bus sekolah di Kota Malang.

3.2 Strategi Penelitian

Strategi penelitian ini dirancang untuk memastikan implementasi sistem pelacakan transportasi berbasis *BLE* dapat berjalan secara efektif dan efisien. Penelitian diawali dengan penentuan rute bus sekolah di Kota Malang, yang dipilih berdasarkan tingkat kepadatan lalu lintas, distribusi halte, dan area strategis untuk pemasangan perangkat *RSU*. Selanjutnya, *RSU* akan dipasang pada lokasi strategis, seperti tiang CCTV di sepanjang rute bus sekolah, dengan mempertimbangkan jarak antar-titik optimal, akses listrik, dan minimnya hambatan fisik yang dapat mengganggu deteksi sinyal *BLE*.

Sebelum data aktual dikumpulkan, dilakukan simulasi dan pengujian sistem di lapangan untuk memastikan perangkat keras seperti Raspberry Pi, *smartphone android* dan perangkat lunak dapat berfungsi sesuai desain. Setelah itu, pengumpulan data real-time dimulai dengan memantau aktivitas *RSU* dalam mendeteksi sinyal *BLE* yang dipancarkan oleh *beacon* pada bus sekolah. Data yang diperoleh, seperti waktu kedatangan, *RSSI*, dan identitas *BLE beacon*, dikirimkan ke server untuk dianalisis. Selama proses pengumpulan data, evaluasi terhadap interferensi lingkungan dilakukan untuk memastikan sistem dapat berfungsi di

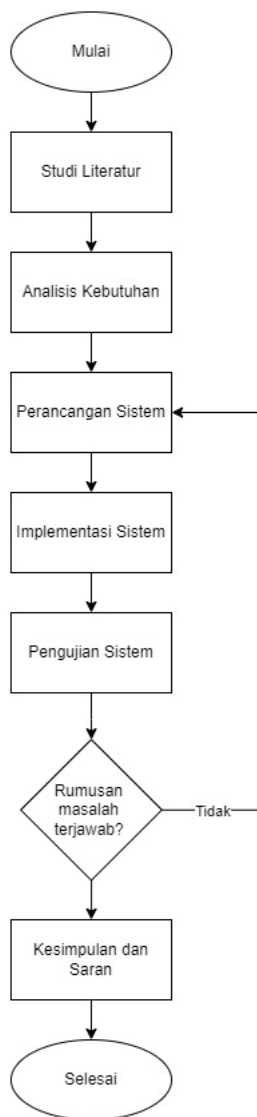
berbagai kondisi, seperti lalu lintas padat, bangunan tinggi, atau area dengan banyak sinyal *BLE* lainnya.

Tahapan selanjutnya adalah pengolahan data historis waktu kedatangan dan keberangkatan bus di setiap titik *RSU* menggunakan metode prediksi berbasis algoritma *machine learning*. Analisis ini bertujuan untuk memodelkan estimasi waktu kedatangan yang lebih akurat di halte atau *RSU* berikutnya. Sistem ini juga diuji keandalannya dalam berbagai kondisi, seperti variasi jarak antar-titik *RSU* dengan bus sekolah, cuaca, dan stabilitas koneksi internet. Berdasarkan hasil evaluasi, dilakukan perbaikan pada perangkat lunak, seperti algoritma *filter* dan pemrosesan data, serta perangkat keras, seperti peningkatan stabilitas sinyal dan perlindungan perangkat dari vandalisme maupun gangguan lingkungan. Dengan strategi ini, diharapkan penelitian dapat menghasilkan sistem pelacakan berbasis *BLE* yang andal, efisien, dan dapat diimplementasikan secara nyata, sekaligus mendukung terciptanya model prediksi waktu kedatangan bus yang akurat.

Strategi penelitian akan dijelaskan lebih lengkap pada sub bab selanjutnya guna memperjelas strategi penelitian.

3.2.1 Metode Penelitian

Penelitian ini melibatkan serangkaian langkah yang dilakukan secara berurutan dan terstruktur. Selama pelaksanaan penelitian, revisi atau perbaikan dapat dilakukan jika ditemukan ketidaksesuaian atau kebutuhan untuk meningkatkan proses yang telah dilakukan sebelumnya. Proses-proses ini dapat digambarkan dengan menggunakan diagram alir yang tercantum dalam **Gambar 3.2.**



Gambar 3.2 Metode penelitian

Gambar tersebut menunjukkan sebuah diagram alir proses perancangan sistem, dimulai dari studi literatur dan analisis kebutuhan. Setelah itu, dilakukan perancangan sistem berdasarkan hasil analisis kebutuhan. Selanjutnya sistem diimplementasikan sesuai rancangan yang telah dibuat. Setelah implementasi selesai, dilakukan pengujian sistem untuk memastikan bahwa sistem yang dirancang telah berfungsi sesuai dengan yang diharapkan. Jika dalam pengujian ditemukan adanya masalah atau ketidaksesuaian, maka proses akan kembali ke tahap analisis kebutuhan untuk dilakukan perbaikan. Proses ini terus berulang hingga sistem yang dirancang benar-benar sesuai dengan kebutuhan dan menyelesaikan rumusan masalah yang telah dibuat.

3.2.2 Objek Penelitian

Objek penelitian dari Sistem untuk Sistem Pengumpulan Data Pelacakan Transportasi Umum Menggunakan *Bluetooth Proximity Beacons* adalah bis sekolah yang berkeliling kota Malang pada pagi dan sore hari. Dengan penelitian ini diharapkan para siswa yang menaiki bus sekolah dapat melihat keberadaan bus yang akan dia tumpangi.

Bus yang digunakan dalam penelitian ini adalah BUS I, yang memiliki rincian sebagai berikut:

Titik Pergerakan Awal: SPBU Tlogomas atau halaman Baiduri Sepah dengan rute Perlintasan Jl. MT Haryono – Jl. Soekarno-Hatta – Bundaran Pesawat – Taman Krida Budaya Jatim (TKBJ) – Jl. DI Panjaitan – Jl. Bogor – Jl. Veteran – Bundaran Diknas/UB – Jl. Veteran – Jl. Bandung – Jl. Ijen – Jl. Semeru – Jl. Kahuripan – Bundaran Tugu – Balai Kota Malang.

Lokasi Halte/*Shelter*:

- Pasar Dinoyo
- Griyashanta
- Taman Krida Budaya
- Ijen
- Semeru
- Stadion Gajayana.

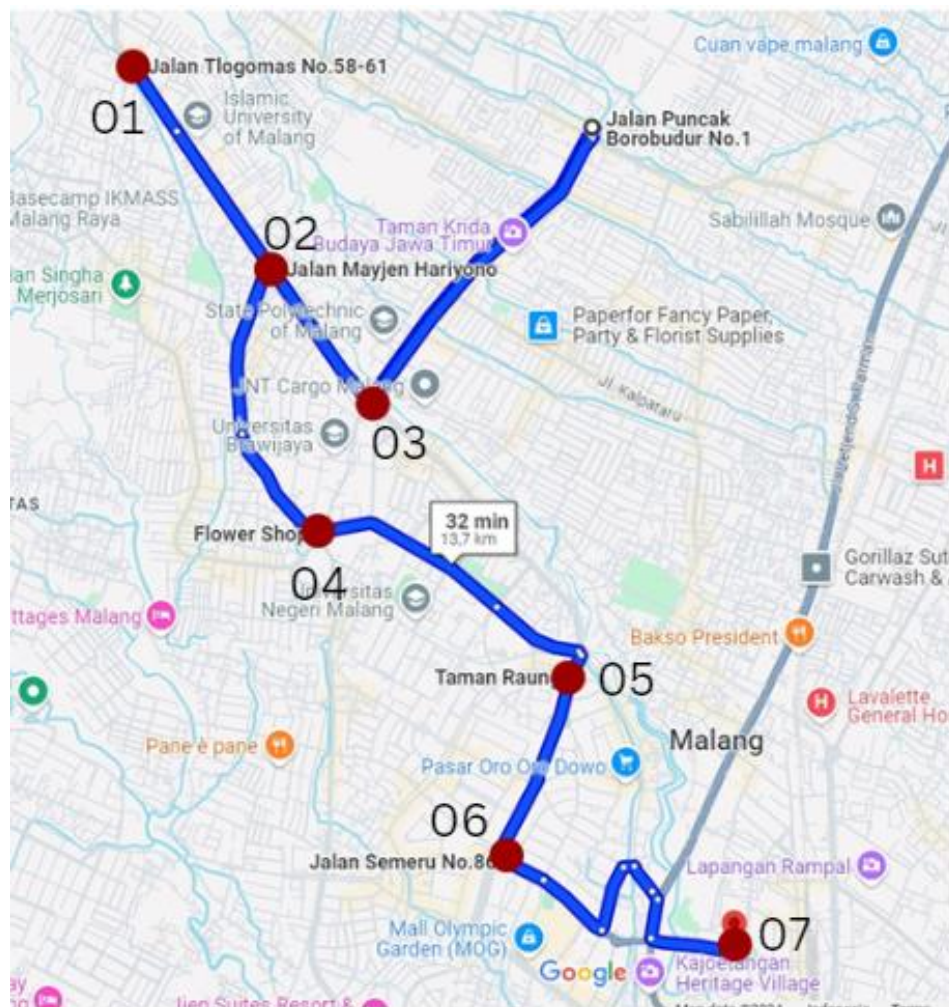
Sekolah Tujuan:

- SMPN 18
- SMAN 9
- MAN 3 Malang
- MTsN Malang I
- SMAN 8
- SMPN 4
- SMKN 2
- SMPN 1
- SMPN 8 (turun di Bundaran Bromo Semeru)
- SMPN 6 (turun di Bundaran Bromo Semeru)
- SMAN 1
- SMAN 3
- SMAN 4.

Bus sudah berada di titik pergerakan awal mulai pukul 05.45 WIB untuk memastikan siswa dapat sampai di sekolah tepat waktu. Lokasi penelitian ini dipilih karena rute dan jadwal BUS I memberikan representasi yang baik untuk menguji sistem pelacakan dalam kondisi nyata di lingkungan perkotaan. Rute yang melibatkan halte strategis dan berbagai institusi pendidikan memungkinkan pengumpulan data yang relevan untuk mengevaluasi kinerja sistem.

3.2.3 Lokasi Penelitian

Penelitian ini dilakukan di Kota Malang dengan fokus pada rute bus sekolah yang telah ditentukan. Pengambilan data dilakukan sepanjang rute bus untuk memastikan sistem pelacakan berbasis *BLE* dapat berjalan dengan baik.



Gambar 3.3 Rute Pagi Bus I

Gambar 3.3 menunjukkan rute pagi bus sekolah yang berangkat pada pukul 05.45 pagi. Rute ini dimulai dari Jalan Tlogomas No. 58-61 dan berakhir di Tugu (Kajoeangan Heritage Village). Setiap titik merah dalam gambar tersebut

menandai lokasi pemasangan *RSU*, yang dipasang pada tiang CCTV sepanjang rute bus sekolah.



Gambar 3.4 Rute Sore Bus I

Gambar di atas menunjukkan rute sore atau rute kepulangan bus sekolah. Rute ini dimulai dari Tugu (dekat Alun-Alun Malang) dan berakhir di Jalan Tlogomas.

Ada sedikit perbedaan antara rute sore dan pagi bus I sehingga menyebabkan beberapa *RSU* yang ada tidak akan mendeteksi BLE beacon pada rute sore. Berikut adalah rincian lokasi dan *RSU* yang akan dilewati pada saat rute pagi dan sore.

Tabel 3.1 Lokasi *RSU*

No	Lokasi	Latitude	Longitude
1	Tlogomas	-7,933342	112,603617
2	MT Haryono	-7,943272	112,610494
3	Suhat / MT Haryono Barat	-7,949783	112,615514
4	Veteran Timur	-7,956525	112,612789
5	Ijen Utara/Jl Jakarta	-7,963403	112,625306
6	Ijen/Jl Ijen	-7,972269	112,621822
7	Tugu	-7,976903	112,633419

Pada rute pagi urutan *RSU* yang dilewati adalah sebagai berikut:

1. Tlogomas
2. MT Haryono
3. Suhat
4. Suhat
5. MT Haryono
6. Veteran Timur
7. Ijen Utara
8. Ijen
9. Tugu

Sementara pada rute sore urutan *RSU* yang dilewati adalah sebagai berikut:

1. Tugu
2. Ijen
3. Suhat
4. MT Haryono
5. Tlogomas

RSU ini dirancang untuk mendeteksi sinyal *BLE* dari bus sekolah yang melintas di dekatnya. Lokasi pemasangan *RSU* dipilih secara strategis untuk memaksimalkan jangkauan deteksi, dengan mempertimbangkan faktor-faktor seperti kepadatan lalu lintas dan kebutuhan operasional sistem.

3.2.4 Teknik Pengumpulan Data

Pengumpulan data dilakukan dengan memantau aktivitas setiap *RSU* yang dipasang pada rute bus sekolah yang dilalui. Pemasangan *RSU* bertujuan untuk menangkap sinyal *BLE* yang dipancarkan oleh perangkat *Estimote* yang terdapat dalam bus tersebut. Dengan demikian, diharapkan dapat dikumpulkan data yang akurat dan relevan terkait pergerakan dan deteksi *beacon BLE* dalam lingkungan bus sekolah di Kota Malang. Selain deteksi antara *beacon BLE*, dihitung juga keberhasilan pengiriman paket *telemetry* ke *server* serta keberhasilan akses jarak jauh ke *RSU*.

Pada setiap rute pagi dan sore, diharapkan minimal terjadi satu deteksi pada setiap *RSU* yang dipasang dengan *BLE* setiap harinya. Namun, ada beberapa *RSU* yang memiliki minimal dua deteksi dengan waktu yang berbeda, misalnya karena lokasi *RSU* yang memungkinkan pendeteksian lebih dari satu kali dalam satu rute seperti pada . Pada rute sore, terdapat beberapa *RSU* yang mungkin tidak akan mendeteksi dengan *BLE*. Hal ini dapat terjadi karena rute sore tidak sepenuhnya sama dengan rute pagi, sehingga beberapa *RSU* tidak dilewati oleh bus atau tidak mendeteksi sinyal *BLE* akibat kondisi lalu lintas atau faktor lainnya. Data dari deteksi dan interval waktu antar *RSU* yang berdekatan akan digunakan sebagai dataset untuk membuat prediksi waktu tiba bus di antara *RSU* tersebut.

Pengumpulan data *telemetry* dilakukan setiap hari mulai pukul 03:59 hingga 19:01. Data *telemetry* yang dikirimkan meliputi waktu pengiriman dan suhu *CPU* dari perangkat *Raspberry Pi 4* yang terpasang pada *RSU*. Data tersebut dikirim secara berkala dengan jeda waktu pengiriman setiap 10 detik. Tujuan dari pengumpulan data *telemetry* ini adalah untuk memastikan kinerja *RSU* tetap stabil serta memantau kondisi perangkat keras selama operasional berlangsung. Dengan pengiriman data yang konsisten, setiap anomali, seperti lonjakan suhu *CPU* yang tidak normal, dapat segera diidentifikasi dan ditangani untuk mencegah kerusakan perangkat.

Selain itu, pengumpulan data akses jarak jauh dilakukan dengan mencoba mengakses setiap *RSU* pada tiga waktu berbeda dalam sehari: pagi, siang, dan sore. Pengujian ini dilakukan selama *RSU* masih dalam keadaan aktif untuk memastikan konektivitas jarak jauh berfungsi dengan baik. Akses jarak jauh ini penting untuk mendukung pemeliharaan sistem, seperti pembaruan perangkat lunak, pemantauan kinerja, atau penanganan masalah teknis yang mungkin terjadi tanpa perlu langsung berada di lokasi. Keberhasilan akses jarak jauh diukur dengan respons perangkat terhadap permintaan akses yang dilakukan, baik dalam hal waktu respon maupun stabilitas koneksi.

Secara keseluruhan, pengumpulan data ini bertujuan untuk menghasilkan dataset yang komprehensif terkait operasional *RSU* dan deteksinya dengan perangkat *BLE* di bus sekolah. Dataset ini mencakup data deteksi *BLE* dengan *RSU* pada rute pagi dan sore, data *telemetry* yang dikirimkan secara periodik, serta keberhasilan akses jarak jauh ke *RSU*. Dengan data yang terkumpul, sistem prediksi waktu tiba bus dapat dikembangkan dan dioptimalkan untuk memberikan informasi waktu tiba yang lebih akurat. Pendekatan ini juga memungkinkan evaluasi menyeluruh terhadap kinerja sistem dalam kondisi operasional nyata, termasuk tantangan yang mungkin muncul dalam pengelolaan *RSU* di Kota Malang.

3.2.5 Teknik Analisis Data

Teknik analisis data diperoleh melalui evaluasi langsung hasil pengujian sistem. Tujuan dari analisis ini adalah untuk mendapatkan pemahaman yang lebih mendalam tentang kemampuan sistem dan batasan-batasan yang perlu diakui ketika sistem diuji. Pengujian melibatkan pemantauan beberapa aspek yang memerlukan perhatian khusus, termasuk:

1. Menghitung persentase keberhasilan deteksi antara *BLE beacon* dengan *RSU scanner* setiap minggunya.

Analisis ini bertujuan untuk mengevaluasi kemampuan *RSU* dalam mendeteksi sinyal *BLE* yang dipancarkan oleh perangkat *Estimote* pada bus sekolah. Persentase keberhasilan dihitung berdasarkan jumlah deteksi yang berhasil dibandingkan dengan jumlah total potensi deteksi yang seharusnya terjadi setiap minggunya. Presentase keberhasilan dapat dihitung menggunakan rumus berikut:

$$Presentase = \frac{\text{Interaksi setiap } RSU}{\text{Jumlah interaksi yang diharapkan}} \times 100\% \quad (1)$$

2. Menghitung presentase keberhasilan dalam pengiriman data pemantuan.

Dalam tahap ini, analisis fokus pada keberhasilan sistem dalam mengirimkan data *telemetry* dari *RSU* ke *server*. Persentase keberhasilan pengiriman dihitung dengan membandingkan jumlah data *telemetry* yang berhasil dikirimkan terhadap jumlah data yang direncanakan untuk dikirimkan dalam periode waktu tertentu. Perhitungan presentase keberhasilan akan dihitung menggunakan rumus dibawah ini:

$$Presentase = \frac{\text{Jumlah data telemetry}}{\text{Jumlah data telemetry yang diharapkan}} \times 100\% \quad (2)$$

3. Menghitung keberhasilan sistem dalam melakukan akses jarak jauh.

Analisis ini dilakukan untuk mengevaluasi stabilitas dan konsistensi koneksi jarak jauh ke *RSU* yang memungkinkan pengelolaan sistem dari lokasi terpencil. Persentase keberhasilan akses jarak jauh dihitung berdasarkan jumlah upaya akses yang berhasil dibandingkan dengan total upaya yang dilakukan pada waktu pagi, siang, dan sore. Presentase keberhasilan deteksi akan dihitung menggunakan rumus di bawah ini:

$$Presentase = \frac{\text{jumlah upaya berhasil}}{\text{jumlah upaya yang dilakukan}} \times 100\% \quad (3)$$

Setelah semua data dikumpulkan dan dianalisis, hasilnya akan dirangkum untuk memberikan gambaran komprehensif tentang kinerja sistem. Dataset yang dihasilkan dari analisis ini tidak hanya membantu dalam mengevaluasi keberhasilan sistem tetapi juga berfungsi sebagai bahan masukan untuk pengembangan lebih lanjut. Dengan mengidentifikasi pola kegagalan atau ketidakefisienan, sistem dapat dioptimalkan untuk meningkatkan keberhasilan deteksi *BLE*, pengiriman data *telemetry*, dan akses jarak jauh, sehingga menghasilkan solusi yang lebih andal dan efektif dalam pengelolaan bus sekolah berbasis teknologi di Kota Malang.

3.2.6 Peralatan Pendukung

Pada penelitian ini, diperlukan beberapa peralatan pendukung untuk menyelesaikan permasalahan penelitian. Kebutuhan peralatan pendukung dibagi menjadi dua yaitu perangkat lunak dan perangkat keras. Perangkat keras yang diperlukan adalah sebagai berikut:

1. Laptop
2. *Raspberry pi 4*
3. *Smartphone Android Redmi 12c*
4. *Estimote beacon*

Sementara kebutuhan perangkat lunak adalah sebagai berikut:

1. *Amazon web service*
2. *Thingsboard*
3. *Airdroid*

4. *Android Studio*

BAB IV REKAYASA KEBUTUHAN

4.1 Kajian Masalah

Transportasi umum, khususnya bus, memiliki peran penting dalam mendukung mobilitas masyarakat dan mengurangi dampak negatif dari penggunaan kendaraan pribadi, seperti kemacetan dan emisi karbon. Namun, di negara-negara berkembang, seperti Indonesia, sistem transportasi umum sering kali menghadapi berbagai tantangan yang membuatnya kurang diminati oleh masyarakat. Kajian masalah berikut merangkum poin-poin utama yang menjadi fokus penelitian ini untuk meningkatkan kualitas sistem transportasi umum, khususnya dalam aspek pelacakan bus secara *real-time*.

1. Ketidakpuasan pengguna transportasi umum disebabkan oleh ketidakakuratan dan ketidakandalan waktu kedatangan bus di halte.
2. Kurangnya data aktual tentang lokasi dan pergerakan bus menghalangi operator bus untuk memberikan layanan yang lebih baik.
3. Banyak operator bus di negara-negara berkembang menghadapi biaya yang tinggi untuk menerapkan sistem pelacakan bus, terutama dengan teknologi *GPS*.
4. Gangguan sinyal radio dan jaringan yang dapat mengganggu transmisi data *real-time* dari sistem pelacakan bus
5. Sistem *GPS* sulit diterapkan pada armada besar karena biaya dan kompleksitas yang meningkat.

4.2 Identifikasi Stakeholder

Beberapa stakeholder yang terlibat dalam sistem pengumpulan data transportasi umum yang menggunakan *Bluetooth Proximity Beacons* ini adalah:

1. Operator Bus Sekolah
Untuk merencanakan perjalanan dengan lebih baik dan mengharapkan layanan transportasi yang lebih andal dan konsisten, Anda membutuhkan informasi yang akurat tentang waktu kedatangan bus.
2. Peneliti selanjutnya
Untuk meningkatkan fungsionalitas prediksi waktu kedatangan berdasarkan data yang dikumpulkan dan mengevaluasi penggunaan sistem pengumpulan data kedatangan bus sekolah menggunakan *BLE proximity beacon*.
3. Siswa Sekolah

Untuk membantu siswa sekolah sebagai pengguna bus mendapatkan kualitas pelayanan bus sekolah yang lebih layak dengan adanya prediksi kedatangan bus di halte terdekat.

4.3 Kebutuhan Fungsional

Kebutuhan fungsional akan membahas mengenai bagian-bagian sistem yang akan diterapkan serta tujuan dari setiap bagian tersebut. Berikut adalah kebutuhan fungsional dari Sistem Pengumpulan Data Transportasi Umum Menggunakan *Bluetooth Proximity Beacons*:

1. Sistem dapat mendeteksi kedatangan bus sekolah berdasarkan pendeteksian *BLE proximity beacon*.

Sistem akan menggunakan *BLE proximity beacon* yang dipasang pada bus sekolah untuk mendeteksi keberadaan dan kedatangan bus. Ketika bus dengan beacon *BLE* mendekati *RSU*, perangkat *scanner* akan mendeteksi sinyal *BLE* tersebut dan mencatat waktu pendeteksian. Hal ini memungkinkan identifikasi deteksi bus dan *RSU* secara *real-time* untuk meningkatkan efisiensi dan kenyamanan pengguna.

2. Sistem dapat melakukan pengiriman data pendeteksian dan data *telemetry* kemudian dikumpulkan pada *server*.

Setelah pendeteksian dilakukan, data seperti lokasi bus, waktu kedatangan, serta informasi *telemetry* lainnya seperti suhu akan dikirimkan secara otomatis ke *server* pusat. *Server* ini bertanggung jawab untuk menyimpan, dan memproses data tersebut dalam format yang dapat digunakan oleh pengguna atau pengelola sistem untuk keperluan analisis maupun pelaporan.

3. Sistem memiliki fungsi akses secara jarak jauh.

Sistem dirancang agar dapat diakses oleh peneliti melalui jaringan internet. Dengan fungsi ini diharapkan peneliti dapat memantau dan memperbaiki sistem secara jarak jauh apabila terjadi *error* pada perangkat *RSU*.

4. Sistem dapat melakukan restart secara otomatis pada waktu yang telah ditentukan.

Sistem dirancang untuk dinonaktifkan pada jam non-operasional. Dengan ini diharapkan dapat mengurangi *error* dan meningkatkan stabilitas sistem pada saat jam operasional.

4.4 Spesifikasi Sistem

Spesifikasi sistem menjelaskan elemen yang lebih mendalam berkaitan dengan fungsionalitas utama sistem. Berikut adalah beberapa spesifikasi penting untuk sistem pengumpulan data transportasi umum menggunakan Bluetooth Proximity Beacons terutama pada bis sekolah kota Malang:

1. Sistem dapat mendeteksi *beacon* dengan jarak maksimum 30 meter dari *RSU*
2. Setiap deteksi yang terjadi pada *scanner* akan ada jeda waktu 10 detik antara deteksi dan pendeteksian selanjutnya untuk kestabilan *RSU* dan server
3. Sinyal yang dipancarkan oleh *BLE beacon* memiliki *delay* 1 detik guna mengantisipasi kecepatan bis yang tinggi saat melintasi *RSU* dan mengoptimalkan *lifespan* baterai pada *BLE beacon*.
4. Sistem *telemetry* mengirimkan suhu *CPU raspberry pi 4* setiap 10 detik ke server.
5. Sistem akses jarak jauh dapat diakses setiap saat selama jam operasional mulai dari jam 04:00 hingga 19:00.
6. Sistem akan dimatikan pada jam 19:00 dan dinyalakan kembali pada jam 04:00 setiap harinya

4.5 Analisis Kebutuhan Perangkat Keras dan Perangkat Lunak

Dari kebutuhan fungsional dan spesifikasi sistem maka dapat disusun kebutuhan perangkat keras dan lunak guna menunjang kebutuhan sistem. Berikut adalah kebutuhan perangkat keras dan lunak Sistem Pengumpulan Data Transportasi umum Menggunakan Bluetooth Proximity Beacons:

4.5.1. Perangkat Keras.

Berdasarkan spesifikasi sistem dan kebutuhan fungsional dari Sistem Pengumpulan Data Pelacakan Transportasi Umum Menggunakan Bluetooth Proximity Beacons, diperlukan perangkat keras sebagai berikut:

- **Raspberry Pi 4** digunakan untuk memindai sinyal *BLE* dari *Estimote BLE Beacon*, mengirimkan data deteksi dan *telemetry* suhu *CPU* ke server pusat.
- **Smartphone Redmi 12C** digunakan untuk memindai sinyal *BLE* secara paralel dengan *Raspberry Pi 4*, serta mengirimkan data deteksi ke server. Selain itu, berfungsi sebagai penyedia koneksi *hotspot* untuk menyediakan koneksi internet bagi *Raspberry Pi 4*.

- **Estimote BLE Beacon** dipasang pada bus sekolah untuk mengirimkan sinyal *BLE* yang dipindai oleh *RSU*.
- **Box Metal UMG** digunakan untuk melindungi perangkat keras dari kondisi lingkungan yang ekstrem, memastikan perangkat tetap aman dan berfungsi dengan baik.
- **Power Adapter LDNIO A4610C** digunakan untuk memastikan suplai daya yang stabil bagi perangkat, menjaga kestabilan operasional sistem.

4.5.2. Perangkat Lunak.

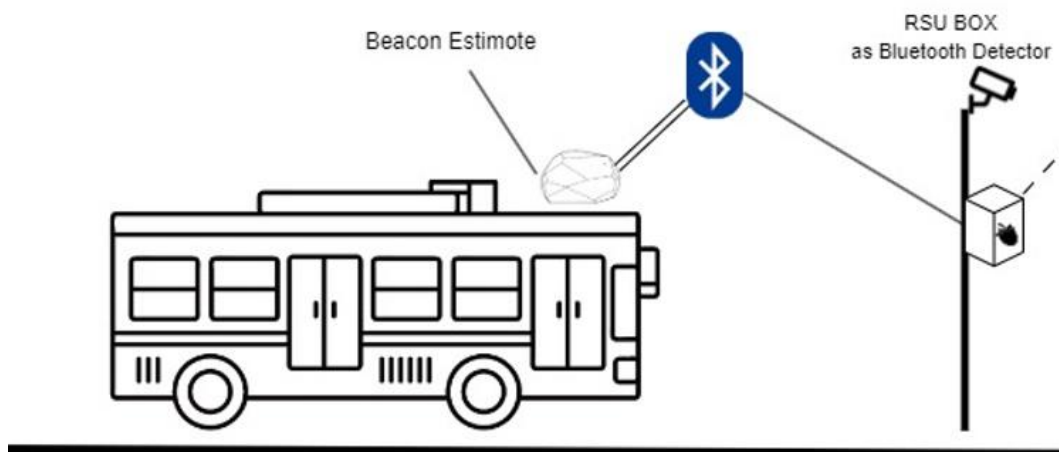
Berdasarkan spesifikasi sistem dan kebutuhan fungsional dari Sistem Pengumpulan Data Pelacakan Transportasi Umum Menggunakan *Bluetooth Proximity Beacons*, diperlukan perangkat lunak sebagai berikut:

- **Kotlin** digunakan untuk pengembangan aplikasi *Android* yang menerima data *BLE beacon* secara *real-time*, dengan fitur seperti *null safety* dan sintaksis bersih untuk mempermudah pengembangan.
- **PM2** digunakan untuk manajemen proses aplikasi *Node.js*, memastikan raspberry pi 4 berjalan stabil dengan fitur *restart* otomatis, *monitoring*, dan *log management*.
- **Node.js** digunakan untuk mengembangkan aplikasi scanner dan menjalankan operasi lain pada raspberry pi 4.
- **AirDroid** digunakan untuk mengelola perangkat secara jarak jauh melalui internet, memungkinkan pengawasan dan pengontrolan perangkat seperti *smartphone* tanpa kehadiran fisik.
- **ThingsBoard** digunakan untuk mengelola, memvisualisasikan, dan menganalisis serta menyediakan *dashboard* interaktif untuk data *telemetry*.
- **Bore Client** digunakan untuk membuka koneksi ke server lokal melalui jaringan publik menggunakan *tunneling*, memungkinkan akses jarak jauh pada raspberry pi 4.

BAB V PERANCANGAN DAN IMPLEMENTASI

5.1 Perancangan Sistem

Dalam perangan sistem ini akan dibahas mengenai perancangan guna memastikan sistem dapat berjalan dengan baik pada saat implementasi. Perancangan ini berupa diagram alir dan blok yang akan kemudian diimplementasikan. Perancangan ini mencakup perancangan perangkat keras atau *hardware* dan perangkat lunak atau *software*. Pada gambar 5.1 menunjukkan keseluruhan sistem.

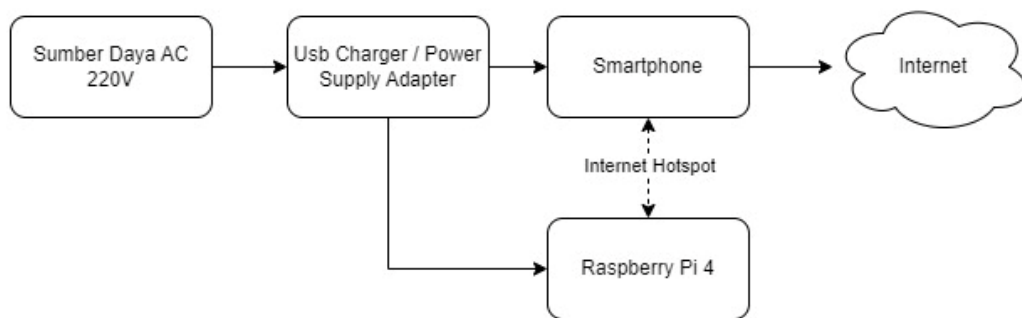


Gambar 5.1 Perancangan Sistem RSU

Pada gambar 5.1, menggambarkan mengenai konsep dasar dari sistem deteksi bus menggunakan RSU yang memanfaatkan teknologi *BLE*. Dalam sistem ini, setiap bus dilengkapi dengan *Beacon Estimote*, yang berfungsi memancarkan sinyal *BLE* dengan identitas unik. Beacon ini dipasang di bagian dalam dekat pintu masuk bus agar sinyalnya dapat dideteksi dengan lebih mudah oleh *RSU*, yang ditempatkan di tiang di sisi jalan. *RSU* bertugas untuk menangkap sinyal tersebut menggunakan detektor *BLE* yang terpasang di dalam sistemnya. Lokasi *RSU* diletakkan di titik strategis, misalnya tiang yang juga digunakan untuk pemasangan kamera *CCTV*, agar instalasi dapat dimaksimalkan dengan memanfaatkan sumber listrik yang sudah terpasang pada tiang *CCTV*.

5.1.1 Perancangan Arsitektur *Scanner Raspberry Pi 4*

Ketika sinyal *BLE* dari *beacon* terdeteksi oleh *RSU*, informasi seperti identitas bus dan waktu pendeteksian segera dikirimkan ke *server* berbasis *cloud* untuk diproses lebih lanjut. Server ini menggunakan data tersebut untuk menghitung estimasi waktu kedatangan bus di halte berikutnya. Informasi prediksi waktu ini kemudian dikirimkan ke pengguna di halte bus melalui aplikasi pada *smartphone*. Dengan cara ini, *RSU* memainkan peran penting dalam menciptakan sistem transportasi yang pintar dan responsif terhadap kebutuhan pengguna, membantu mereka merencanakan perjalanan dengan lebih baik. Untuk itu dibuatlah perancangan arsitektur *scanner raspberry pi 4* seperti gambar berikut.



Gambar 5.2 Diagram Blok Sistem Pengumpulan Data Menggunakan *Raspberry pi 4*

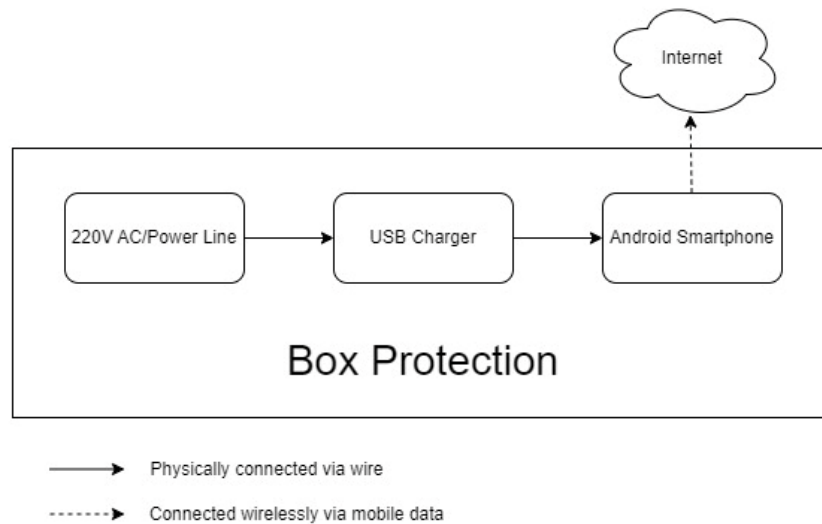
Gambar ini menggambarkan desain *RSU* yang menggunakan *Raspberry Pi 4* sebagai pusat pemrosesan data. Dalam desain ini, sumber daya masih berasal dari AC 220V yang disalurkan melalui *USB charger* atau *power supply adapter*. Daya yang disalurkan digunakan untuk menjalankan *Raspberry Pi 4*, yang bertanggung jawab untuk memproses data yang diterima dari *BLE beacon* dan mengelola jaringan lokal melalui koneksi *WiFi*.

Sebagai penghubung ke internet, sistem ini menggunakan *smartphone* yang berfungsi sebagai *hotspot*. *Raspberry Pi 4* terhubung ke *smartphone* melalui *WiFi*, dan *smartphone* menyediakan akses internet yang dibutuhkan untuk mengirimkan data yang telah diproses ke *server cloud*.

Pada diagram tersebut, garis lurus menggambarkan koneksi antar perangkat yang dilakukan secara kabel, seperti antara sumber daya listrik dengan *USB charger* dan *Raspberry Pi 4*. Sementara itu, garis putus-putus menunjukkan koneksi yang dilakukan secara nirkabel, seperti koneksi *WiFi* antara *Raspberry Pi 4* dan *smartphone* atau antara *smartphone* dengan internet. Hal ini menunjukkan

pemisahan yang jelas antara koneksi fisik dan koneksi nirkabel, sehingga mempermudah pemahaman alur komunikasi dan penyediaan daya dalam sistem.

5.1.2 Perancangan Arsitektur Scanner Smartphone Android

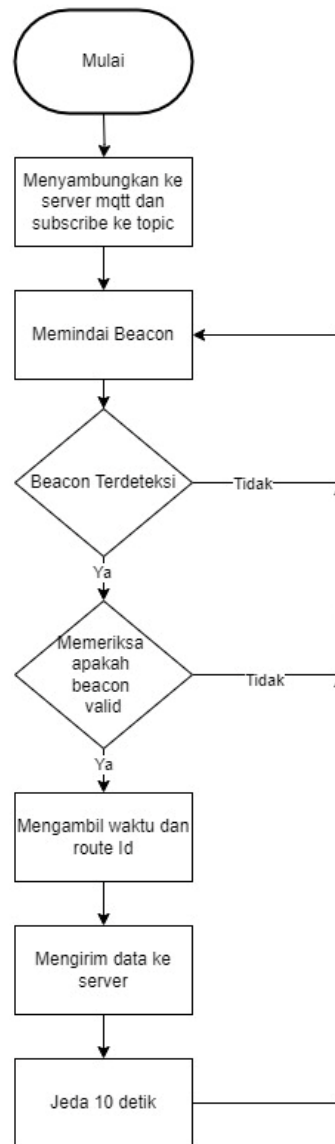


Gambar 5.3 Diagram Blok Sistem Pengumpulan Data Menggunakan *Smartphone Android*

Gambar 5.3 menjelaskan struktur *RSU* yang menggunakan *smartphone* sebagai komponen inti dalam memproses dan mengirim data. Sistem *RSU* ini dirancang untuk menggunakan *smartphone Android* yang dilindungi oleh *Box Protection*, yaitu sebuah kotak pelindung yang dirancang untuk melindungi perangkat dari kondisi lingkungan, seperti panas, hujan, dan debu. *Smartphone berfungsi* sebagai pusat pemrosesan data yang dikumpulkan dari beacon BLE dan juga sebagai perangkat pengirim data ke *server cloud* menggunakan koneksi internet.

Sumber daya untuk *smartphone* ini berasal dari jalur listrik AC 220V yang dihubungkan ke *USB charger*. *USB charger* ini memastikan daya yang stabil untuk menjaga *smartphone* tetap beroperasi. Dengan memanfaatkan *mobile data* yang tersedia di *smartphone*, *RSU* dapat mengirimkan data secara *real-time* ke server untuk diproses lebih lanjut. Sistem ini memiliki keunggulan berupa biaya yang relatif rendah dan kemudahan instalasi karena hanya memerlukan beberapa komponen utama. Solusi ini cocok untuk lokasi dengan infrastruktur terbatas namun tetap membutuhkan sistem pendeteksian yang stabil.

5.1.3 Perancangan Program Scanner BLE



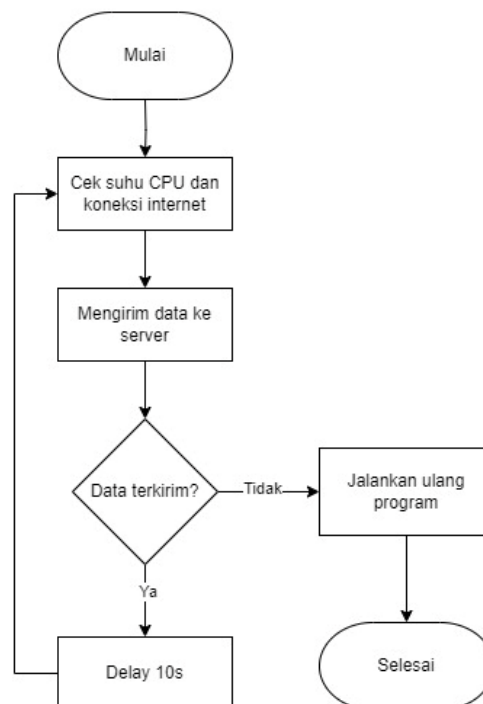
Gambar 5.3 Flow Chart Scanner BLE

Gambar di atas merupakan diagram alur dari program pendeteksi BLE beacon. Program ini dimulai dengan melakukan koneksi ke *server MQTT* dan melakukan *subscribe* ke topik tertentu untuk dapat berkomunikasi dengan *server*. Setelah koneksi berhasil, program akan secara terus menerus melakukan pemindaian untuk mencari *beacon BLE* yang berada di sekitar area. Ketika program mendeteksi adanya *beacon BLE* di sekitar, maka akan dilakukan verifikasi untuk memastikan bahwa *beacon BLE* tersebut terdaftar dalam sistem. Jika *beacon* terdeteksi namun tidak terdaftar, program akan kembali melakukan pemindaian. Namun jika *beacon* terdeteksi dan terdaftar, program akan

melanjutkan proses dengan memeriksa waktu dan menentukan *id* rute berdasarkan *beacon* yang terdeteksi. Setelah informasi terkumpul, data tersebut akan dikirimkan ke *server*. Kemudian untuk menjaga stabilitas sistem, terdapat *delay* waktu selama 10 detik sebelum program kembali melakukan pemindaian *beacon*. Proses ini akan terus berulang untuk memastikan pemantauan *beacon* berjalan secara berkelanjutan.

5.1.3 Perancangan Telemetry

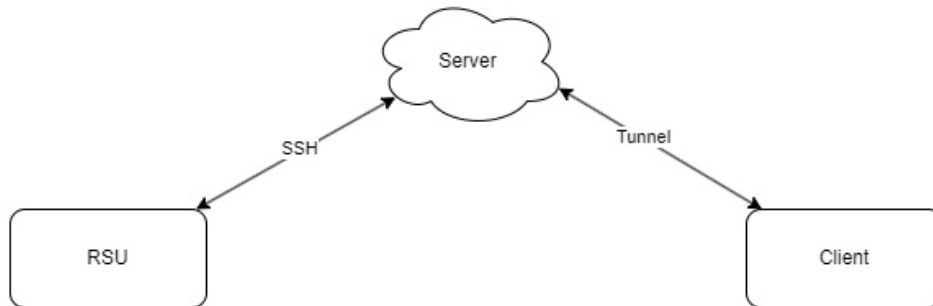
Pada Sistem Pengumpulan Data Transportasi Umum Menggunakan *Bluetooth Proximity Beacons* dibutuhkan juga program untuk melakukan pemantauan kondisi dari setiap RSU yang ada. Pada gambar 5.4 dijelaskan mengenai program *telemetry* untuk melakukan pemantauan.



Gambar 5.4 Flow Chart Telemetry

Gambar di atas merupakan diagram alur dari program telemetry *Raspberry pi 4*. Proses dimulai dengan pemeriksaan suhu CPU dari *Raspberry pi* bersamaan dengan pengecekan koneksi internet untuk memastikan data dapat dikirim ke server. Setelah pembacaan suhu berhasil dan koneksi internet tersedia, data tersebut langsung dikirimkan ke *server* untuk disimpan atau diproses lebih lanjut. Sistem kemudian akan menunggu selama 10 detik sebelum melakukan pembacaan suhu berikutnya. Proses ini akan terus berulang untuk memberikan pemantauan suhu yang berkelanjutan.

5.1.4 Perancangan Akses Jarak Jauh Pada Raspberry Pi 4 dan Smartphone Android.



Gambar 5.5 Diagram Akses Jarak jauh

Pada gambar 5.5 menunjukkan diagram yang menggambarkan sistem akses jarak jauh yang dirancang untuk memungkinkan pemantauan dan pengelolaan perangkat *Raspberry Pi* serta *Android* dari lokasi yang berbeda. Sistem ini menggunakan dua pendekatan berbeda untuk masing-masing perangkat. *Bore* untuk *Raspberry Pi* dan *AirDroid* untuk perangkat *Android*. Peran *server* dalam sistem ini adalah sebagai penghubung antara perangkat yang ingin diakses dan klien yang melakukan akses.

Pada perangkat *Raspberry Pi*, teknologi *tunneling Bore* digunakan untuk memungkinkan akses jarak jauh. *Bore* bekerja dengan membuat *port Raspberry Pi* dapat diakses dari jaringan luar, meskipun *Raspberry Pi* berada di balik *NAT (Network Address Translation)* atau menggunakan *IP* dinamis. *Raspberry Pi* terhubung ke *server* melalui protokol *SSH (Secure Shell)* yang menjamin keamanan komunikasi. *Server* kemudian menjadi perantara yang mengarahkan akses dari klien ke *Raspberry Pi*, memungkinkan pengguna untuk memantau atau mengelola perangkat *Raspberry Pi* dari jarak jauh. Dalam kasus ini, aplikasi seperti monitoring suhu *Raspberry Pi* dapat dilakukan dengan mudah melalui koneksi aman ini.

Untuk perangkat *Android*, akses jarak jauh dilakukan menggunakan aplikasi *AirDroid*. *AirDroid* memberikan antarmuka pengguna yang intuitif untuk mengakses dan mengelola perangkat *Android* dari jarak jauh. Perangkat *Android* langsung terhubung ke internet melalui aplikasi *AirDroid*, yang memungkinkan klien untuk mengontrol perangkat tanpa memerlukan server khusus. *AirDroid* mempermudah pengelolaan file, pengiriman pesan, serta pengoperasian perangkat *Android* dari komputer atau smartphone lain. Ini membuat *Android* dapat diakses secara langsung oleh klien dengan cara yang mudah dan aman.

Server dalam sistem ini berfungsi sebagai perantara utama yang memastikan komunikasi yang efisien antara klien dan perangkat yang ingin diakses, terutama untuk *Raspberry Pi*. *Server* menerima data dari *Raspberry Pi* melalui koneksi SSH dan memungkinkan klien untuk memanfaatkan koneksi tunneling yang dibuat oleh Bore. Dengan demikian, klien dapat mengakses *Raspberry Pi* tanpa perlu konfigurasi kompleks pada jaringan lokal tempat *Raspberry Pi* berada. Sementara itu, perangkat Android yang menggunakan AirDroid tidak membutuhkan server khusus, karena AirDroid sudah dilengkapi dengan layanan cloud yang memungkinkan akses jarak jauh secara langsung.

Secara keseluruhan, diagram ini mencerminkan arsitektur hybrid untuk akses jarak jauh yang memanfaatkan dua teknologi berbeda. Bore untuk *Raspberry Pi* dan AirDroid untuk perangkat Android. Kombinasi ini memberikan fleksibilitas bagi pengguna untuk mengakses dan mengelola kedua perangkat dari lokasi yang berbeda dengan cara yang aman, efisien, dan mudah digunakan. Dengan integrasi yang baik antara server, *Raspberry Pi*, dan Android, sistem ini mampu memberikan solusi akses jarak jauh yang tangguh, baik untuk kebutuhan pengawasan maupun pengelolaan perangkat secara real-time.



5.2 Implementasi Sistem


5.2.1. Implementasi Arsitektur Perangkat Keras.

Implementasi perangkat keras sistem pengumpulan data transportasi umum dilakukan dengan memerhatikan aspek keamanan, ketahanan, dan efisiensi operasional. Box pelindung dipasang pada tiang CCTV yang telah dipilih berdasarkan analisis lokasi strategis sepanjang rute bus sekolah. Pemasangan dilakukan pada ketinggian 2-3 meter dari permukaan tanah untuk memastikan keamanan perangkat sekaligus mempertahankan kualitas penerimaan sinyal Bluetooth yang optimal. Box pelindung diposisikan menghadap ke arah jalan untuk memaksimalkan area deteksi BLE beacon dari bus yang melintas.

Tabel 5.1 Implementasi Pemasangan *RSU*

Lokasi	Gambar Implementasi
Tlogomas	
MT Haryono	

Lokasi	Gambar Implementasi
Suhat/MT Haryono Barat	
Ijen Utara	
Ijen	

Lokasi	Gambar Implementasi
Tugu	

Dalam implementasi internal box, komponen-komponen ditempatkan dengan mempertimbangkan aspek thermal management dan kemudahan maintenance. Power adapter LDNIO A4610C ditempatkan di sisi kanan box dengan jarak aman dari Raspberry Pi untuk menghindari interferensi elektromagnetik. Hasil implementasi internal box dapat dilihat pada gambar 5.6.



Gambar 5.6 Implementasi Raspberry Pi 4 dan *Peripheral*

Pada Gambar di atas, Raspberry Pi 4 ditempatkan secara strategis agar memiliki akses yang mudah ke port-port utama, seperti port USB dan power, tanpa perlu membongkar keseluruhan perangkat. Hal ini sangat penting untuk kemudahan dalam pemeliharaan dan troubleshooting, sehingga peneliti dapat dengan cepat mengganti atau memperbaiki kabel yang terhubung tanpa memindahkan Raspberry Pi dari dudukannya. Penempatan yang sejajar juga membantu dalam pengelolaan panas, karena memberikan ruang yang cukup untuk aliran udara di sekitar Raspberry Pi, sehingga mencegah overheating yang dapat menurunkan kinerja perangkat.

Selain itu, pemasangan Raspberry Pi 4 di posisi tengah box memungkinkan pembagian jarak yang optimal dari perangkat lain, seperti power adapter dan smartphone yang berfungsi sebagai hotspot. Dengan penempatan yang tepat, hal ini bertujuan untuk mengurangi interferensi elektromagnetik serta memastikan bahwa Raspberry Pi menerima suplai daya yang stabil tanpa gangguan dari perangkat lain. Smartphone yang digunakan sebagai hotspot juga berfungsi sebagai koneksi internet utama untuk Raspberry Pi, menghubungkan perangkat tersebut ke internet untuk mengirimkan data yang diproses ke server cloud, memberikan fleksibilitas dan kemudahan dalam pengelolaan sistem secara keseluruhan.

5.2.2. Implementasi Program Scanner.

Program Scanner BLE berfungsi untuk mendeteksi beacon-beacon BLE yang berada di sekitar area. Beacon ini sering digunakan dalam berbagai aplikasi seperti pemantauan lokasi, pengumpulan data transportasi umum, serta berbagai aplikasi IoT lainnya. Dalam konteks sistem pengumpulan data transportasi umum, beacon BLE digunakan untuk mendeteksi bus sekolah tertentu yang dapat memberikan informasi penting terkait transportasi umum.

Berikutnya, implementasi kode yang digunakan untuk menjalankan program scanner ini akan dibahas lebih lanjut, termasuk penggunaan JavaScript pada Raspberry Pi untuk pemindaian beacon, serta implementasi Kotlin untuk scanner BLE pada perangkat Android.

1	<code>import BeaconScanner from 'node-beacon-scanner';</code>
2	<code>const scanner = new BeaconScanner();</code>
3	<code>import axios from 'axios';</code>


```

4 import mqtt from 'mqtt';
5 import fs from 'fs';
6 import dotenv from 'dotenv';
7 dotenv.config();
8 import isOnline from 'is-online';
9 import shell from 'shelljs';
10 import { Console } from 'console';
11
12
13 const ca = fs.readFileSync(process.env.CA, 'utf8');
14 const cert = fs.readFileSync(process.env.CERT, 'utf8');
15 const key = fs.readFileSync(process.env.KEY, 'utf8');
16 const endpoint = process.env.ENDPOINT
17 const topic = process.env.TOPIC;
18 const bus = fs.readFileSync(process.env.BUS, 'utf8');
19 const busObj = JSON.parse(bus);
20 const busStop =
21 fs.readFileSync(process.env.BUS_STOP, 'utf8');
22 const busStopObj = JSON.parse(busStop);
23 const rsuID =
24 fs.readFileSync(process.env.NODE_ID, 'utf8');
25 const rsuIDObj = JSON.parse(rsuID);
26 const API_getETA = process.env.API_GET_ETA;
27 const API_busInsertLocation =
28 process.env.API_INSERT_BUS_LOCATION;
29 const thresholdHour = Number(process.env.TRESHOLD_HOUR);
30 const nodeID = rsuIDObj.nodeID;
31 const heartBeatInterval = process.env.HEARTBEAT_INTERVAL;
32 const autorestartPeriod = process.env.AUTORESTART_PERIOD;
33 const updateTopic = process.env.UPDATE_TOPIC;
34
35 function run() {
36     const mqttOptions = {
37         host: endpoint,
38         protocol: "mqtt",
39         clientId: "sdk-nodejs-v2",
40         clean: true,
41         key: key,
42         cert: cert,
43         ca: ca,
44         reconnectPeriod: 0,
45         debug:true
46     };
47
48     const client = mqtt.connect(mqttOptions);
49
50     client.on('connect', function () {
51         console.log("Connected to AWS IoT Core!");
52         client.subscribe([updateTopic], () => {
53             console.log(Subscribe to topic '${updateTopic}');
54         });
55     });
56

```

```

57     async function busInsertLocation(postData) {
58         const options = {
59             method: 'POST',
60             headers: {
61                 'Content-Type': 'application/json',
62                 'Content-Length': Buffer.byteLength(postData)
63             }
64         };
65         try {
66             const response = await
67 axios.post(API_busInsertLocation, postData, options);
68             // console.log('Response:', response.data);
69             console.log("SUCCESS");
70         } catch (error) {
71             console.log(error);
72             // console.log(error.response["data"]);
73         }
74         console.log("Delay 30s");
75         setTimeout(() => {
76
77             scanner.startScan().then(() => {
78                 console.log('Started to scan after insert.');
```

```

79             }).catch((error) => {
80                 console.error(error);
81             });
82         }, 10000);
83     }
84
85     async function getETA(busID, serviceNo, routeID,
86 busStopID) {
87         const postData = JSON.stringify({
88             bus_id: busID,
89             service_no: serviceNo,
90             route_id: routeID,
91             bus_stop_id: busStopID
92         })
93         console.log('getETA:' + postData);
94         const options = {
95             method: 'POST',
96             headers: {
97                 'Content-Type': 'application/json',
98                 'Content-Length':
99 Buffer.byteLength(postData)
100             },
101         };
102         try {
103             const response = await axios.post(API_getETA,
104 postData, options);
105             try {
106                 var etaStatus = response.status;
107                 var etaData = response.data;
108             } catch (error) {
109                 console.log(error);

```

```

110         var etaStatus = 400;
111         var etaData = 'no data';
112     }
113
114     } catch (error) {
115         try {
116             // var etaStatus = error.response["status"];
117             var etaData = error.response["data"];
118         } catch (error) {
119             console.log(error);
120             var etaStatus = 400;
121             var etaData = 'no data';
122         }
123     }
124
125     // console.log(etaStatus);
126     // console.log(etaData);
127
128     const sData = JSON.stringify(etaData);
129     var cleanData = '';
130     for(let i = 0; i < sData.length; i++){
131         if((sData[i] !== "[" && (sData[i] !== "]"))){
132             cleanData += sData[i]
133         }
134     }
135     var objData = JSON.parse(cleanData);
136     if(objData.route_id === routeID) return true;
137     else if(etaData === 'No bus service found')
138 return false;
139     else return false;
140 }
141
142     async function checkAndSendData(scannedBLE,
143 bleAddress, proximityUUID, rssi, txPower){
144
145         var temp =
146 fs.readFileSync("/sys/class/thermal/thermal_zone0/temp");
147         var temp_c = temp/1000;
148
149         // var temp_c = randomInt(25, 40);
150
151         const date = new Date();
152         var sampleTime = date.getTime();
153         //publish AWS for beacon data
154         const msg = {
155             timestamp: sampleTime,
156             deviceId:
157 busStopObj.busStopID[nodeID.toString()].go.toString(),
158             bleAddress: bleAddress,
159             proximityUUID: proximityUUID,
160             rssi: rssi,
161             txPower: txPower,
162             raspiTemp: temp_c

```

```

163         };
164         const json = JSON.stringify(msg);
165         if (client){
166             client.publish(topic, json, { qos: 0, retain:
167 false }, (error) => {
168                 if (error){
169                     console.log(error)
170                 }
171             })
172         }
173
174
175         if(await getETA(busObj.bleID[scannedBLE].busID,
176 busStopObj.serviceNo, busStopObj.busRoutes.go,
177 busStopObj.busStopID[nodeID.toString()].go)){
178             const postData = JSON.stringify({
179                 bus_id: busObj.bleID[scannedBLE].busID,
180                 route_id: busStopObj.busRoutes.go,
181                 imei: bleAddress,
182                 latlong:
183 busStopObj.coordinate[nodeID.toString()],
184                 speed: 10
185             });
186             console.log('busInserLocation: ' + postData);
187             busInsertLocation(postData);
188         }
189         else{
190             if(await getETA(busObj.bleID[scannedBLE].busID,
191 busStopObj.serviceNo, busStopObj.busRoutes.back,
192 busStopObj.busStopID[nodeID.toString()].back)){
193                 const postData = JSON.stringify({
194                     bus_id: busObj.bleID[scannedBLE].busID,
195                     route_id: busStopObj.busRoutes.back,
196                     imei: bleAddress,
197                     latlong:
198 busStopObj.coordinate[nodeID.toString()],
199                     speed: 10
200                 });
201                 console.log('busInserLocation: ' + postData);
202                 busInsertLocation(postData);
203             }
204             else{
205                 const time = new Date();
206                 let hour = time.getHours();
207                 if(hour <= treshholdHour){
208                     const postData = JSON.stringify({
209                         bus_id: busObj.bleID[scannedBLE].busID,
210                         route_id: busStopObj.busRoutes.go,
211                         imei: bleAddress,
212                         latlong:
213 busStopObj.coordinate[nodeID.toString()],
214                         speed: 10
215                     });

```

```

216         console.log('busInserLocation: ' +
217 postData);
218         busInsertLocation(postData);
219     }
220     else{
221         const postData = JSON.stringify({
222             bus_id: busObj.bleID[scannedBLE].busID,
223             route_id: busStopObj.busRoutes.back,
224             imei: bleAddress,
225             latlong:
226 busStopObj.coordinate[nodeID.toString()],
227             speed: 10
228         });
229         console.log('busInserLocation: ' +
230 postData);
231         busInsertLocation(postData);
232     }
233 }
234 }
235 }
236 scanner.onadvertisement = (ad) => {
237     scanner.stopScan();
238     console.log(ad);
239
240     const scannedBLE = ad["id"];
241
242     if
243 (Object.keys(busObj.bleID).includes(scannedBLE)) {
244         const bleAddress = ad["address"];
245         const proximityUUID =
246 ad["iBeacon"]["uuid"];
247         const rssi = ad["rssi"];
248         const txPower = ad["iBeacon"]["txPower"];
249         console.log('iBeacon of the bus is
250 found!');
251         console.log("Ble Address Beacon: ",
252 bleAddress);
253         checkAndSendData(scannedBLE, bleAddress,
254 proximityUUID, rssi, txPower);
255     } else {
256         console.log('Beacon detected, but not the
257 Bus!');
258         scanner.startScan().then(() => {
259             console.log('Started to scan after bus
260 not detected');
261         }).catch((error) => {
262             console.error(error);
263         });
264     }
265 };
266
267 // Start scanning
268 scanner.startScan().then(() => {

```

269	console.log('Started to scan.');
270	}).catch((error) => {
271	console.error(error);
272	});
273	
274	
275	}
276	
277	run();
278	

- **Pada kode baris 1-10** mengimpor *library* penting seperti *node-beacon-scanner* untuk mendeteksi perangkat BLE, *axios* untuk komunikasi *HTTP*, dan *mqtt* untuk berinteraksi dengan *AWS IoT Core*. Modul *fs* digunakan untuk membaca file, sementara *dotenv* membaca variabel lingkungan dari file *.env* ke runtime. Pustaka *is-online* memeriksa koneksi internet, *shelljs* menjalankan perintah *shell*, dan *Console* digunakan untuk *logging*. *Scanner BLE* diinisialisasi menggunakan *const scanner = new BeaconScanner()* untuk memulai proses pemindaian.
- **Pada kode baris 13-33** program melakukan pemanfaatan modul *fs* untuk membaca berbagai file konfigurasi yang berisi informasi penting, termasuk sertifikat keamanan seperti *Certificate Authority (CA)*, *Client Certificate (CERT)*, dan *Private Key (KEY)*, yang diperlukan untuk autentikasi dan koneksi aman ke *AWS IoT Core*. File-file ini dibaca berdasarkan lokasi yang diatur dalam variabel lingkungan melalui file *.env*. Selain sertifikat, program juga memuat data JSON dari file konfigurasi lain, seperti informasi tentang bus, halte bus (*busStop*), dan node *RSU*, yang kemudian diuraikan menjadi objek JavaScript menggunakan *JSON.parse*. Data ini mencakup ID bus, ID halte, ID node, dan informasi lain yang digunakan untuk menentukan rute dan lokasi terkait dalam logika sistem. Endpoint API juga diatur melalui file *.env* untuk keperluan utama, yaitu mendapatkan estimasi waktu kedatangan bus dan mengirimkan data lokasi bus ke server. Program ini mengandalkan variabel tambahan seperti *threshholdHour*, yang digunakan untuk menentukan arah perjalanan bus berdasarkan waktu, serta *heartBeatInterval* dan *autorestartPeriod*, yang mengatur interval sinyal heartbeat dan periode restart otomatis sistem. Semua parameter ini dirancang agar program dapat berfungsi secara fleksibel dan sesuai dengan kebutuhan operasionalnya.
- **Pada kode baris 35-55** adalah fungsi *run* yang merupakan inti dari program yang bertanggung jawab untuk mengatur koneksi ke *AWS IoT Core* dan memulai komunikasi melalui *MQTT*. Pada awalnya, opsi koneksi *MQTT* dikonfigurasi dalam objek *mqttOptions*, yang mencakup detail seperti

alamat *endpoint*, protokol, ID klien, serta sertifikat keamanan yang diperlukan untuk autentikasi perangkat. Fungsi ini kemudian membuka koneksi dengan menggunakan *mqtt.connect*, dan setelah berhasil terhubung, sistem akan menampilkan pesan di konsol untuk mengonfirmasi koneksi serta berlangganan ke topik tertentu yang diatur dalam *updateTopic*. Berlangganan ini memungkinkan perangkat untuk menerima pembaruan atau pesan dari server *AWS IoT Core*. Fungsi ini memastikan bahwa semua parameter komunikasi dan otentikasi sudah disiapkan dengan benar sebelum melanjutkan ke proses lain dalam program.

- **Pada kode baris 57-83** adalah fungsi *busInsertLocation* digunakan untuk mengirim data lokasi bus ke *server API* melalui *HTTP POST*. Data yang akan dikirim terlebih dahulu diatur dalam format JSON, dan opsi *HTTP* disiapkan untuk memastikan data memiliki jenis konten yang sesuai dan panjang data dihitung dengan benar. Fungsi ini menggunakan modul *axios* untuk melakukan pengiriman data ke *endpoint* yang ditentukan dalam *variabel API_busInsertLocation*. Jika pengiriman berhasil, pesan sukses ditampilkan; jika gagal, pesan kesalahan akan ditangani dan dicatat di konsol. Setelah pengiriman, pemindaian BLE dilanjutkan dengan memberikan jeda waktu selama 10 detik menggunakan *setTimeout*. Fungsi ini memastikan bahwa sistem terus memindai perangkat BLE setelah data berhasil dikirim, menjaga kesinambungan proses pemantauan.
- **Pada kode baris 85-139** adalah fungsi *getETA* yang bertujuan untuk mendapatkan estimasi waktu kedatangan (ETA) bus pada halte tertentu dengan mengirimkan data ke *API endpoint API_getETA*. Data yang mencakup ID bus, nomor layanan, ID rute, dan ID halte disiapkan dalam *format JSON* sebelum dikirim menggunakan *axios* melalui metode *HTTP POST*. Setelah mendapatkan respons dari API, fungsi memeriksa status respons dan data yang diterima. Data ini diuraikan untuk menentukan apakah bus sesuai dengan rute yang diinginkan. Jika ID rute dalam respons cocok dengan ID rute yang diminta, fungsi akan mengembalikan nilai *true*, menunjukkan bahwa bus ada di jalur yang benar. Sebaliknya, fungsi akan mengembalikan *false* jika bus tidak ditemukan pada jalur yang sesuai atau jika terjadi kesalahan selama proses pengambilan data.
- **Pada kode baris 142-235** adalah fungsi *checkAndSendData* yang bertanggung jawab untuk memproses data *BLE* yang terdeteksi dan mengirimkannya ke *AWS IoT Core* menggunakan protokol *MQTT*. Pada awalnya, fungsi membaca suhu prosesor *Raspberry Pi* dari file sistem untuk memantau kondisi perangkat, lalu mengambil timestamp dari waktu saat

data diproses. Informasi seperti alamat *BLE*, *UUID*, kekuatan sinyal (*RSSI*), daya transmisi (*TxPower*), suhu perangkat, dan ID perangkat kemudian disusun menjadi pesan *JSON*. Setelah pesan siap, fungsi mengirimkannya ke *AWS IoT Core* melalui metode *client.publish* menggunakan topik yang telah ditentukan. Selain itu, fungsi ini juga memanfaatkan fungsi *busInsertLocation* untuk mengirimkan data lokasi bus yang telah diproses ke *server API*, yang berfungsi untuk memperbarui informasi lokasi bus secara *real-time*. Dengan demikian, selain mengirimkan data *BLE* langsung ke *AWS IoT Core*, fungsi ini memastikan bahwa data lokasi bus yang relevan dapat disimpan di *server API* menggunakan metode *HTTP POST*. Setelah pengiriman data selesai, pemindaian *BLE* dilanjutkan kembali untuk menjaga kesinambungan proses deteksi perangkat di sekitar. Fungsi ini dirancang untuk mengintegrasikan proses pengiriman data *BLE*, data perangkat, dan pembaruan lokasi bus dalam satu alur kerja yang efisien.

- **Pada kode baris 236-265** adalah *Event onadvertisement* yang dijalankan setiap kali *scanner BLE* mendeteksi perangkat *BLE* di sekitarnya. Saat perangkat terdeteksi, pemindaian dihentikan sementara untuk memproses data perangkat yang ditemukan. Fungsi ini kemudian memeriksa apakah perangkat *BLE* yang terdeteksi sesuai dengan salah satu *ID beacon* bus yang telah diatur dalam konfigurasi. Jika perangkat cocok, data seperti alamat *BLE*, *UUID*, *RSSI*, dan *TxPower* diambil untuk diproses lebih lanjut menggunakan fungsi *checkAndSendData*, yang akan mengirimkan data tersebut ke *AWS IoT Core*. Namun, jika perangkat yang terdeteksi bukan *beacon bus*, sistem akan memulai kembali pemindaian setelah mencatat bahwa perangkat tersebut tidak relevan. *Event* ini dirancang untuk memprioritaskan pemrosesan *beacon bus* sambil terus memindai perangkat lain di area sekitarnya.
- **Pada kode baris 267-272** digunakan untuk pemindaian dimulai dengan memanggil metode *scanner.startScan()*, yang memulai proses deteksi perangkat *BLE* di sekitar. Jika pemindaian berhasil dimulai, pesan konfirmasi akan dicetak ke konsol untuk memberi tahu bahwa sistem telah memulai pemindaian. Namun, jika terjadi kesalahan, pesan kesalahan akan ditampilkan, memberikan informasi tentang kegagalan proses. Fungsi ini memastikan bahwa *scanner BLE* aktif untuk mendeteksi perangkat yang relevan di lingkungan operasional.

Secara keseluruhan, program ini mengintegrasikan berbagai komponen untuk membangun sistem pemantauan transportasi berbasis *BLE*. Dimulai dengan mengimpor pustaka-pustaka penting seperti *node-beacon-scanner* untuk

pemindaian perangkat BLE, *axios* untuk komunikasi *HTTP*, dan *mqtt* untuk interaksi dengan *AWS IoT Core*, program ini mengelola koneksi dan komunikasi data secara efektif. Sertifikat keamanan untuk autentikasi dan koneksi aman dibaca melalui modul *fs*, sementara *file* konfigurasi lainnya yang berisi informasi tentang bus, halte, dan *node RSU* juga dimuat untuk digunakan dalam proses pemindaian dan pengolahan data. Fungsi utama program, seperti *run*, *busInsertLocation*, dan *getETA*, bertanggung jawab untuk mengatur koneksi *MQTT*, mengirimkan data lokasi bus, serta mendapatkan estimasi waktu kedatangan bus di halte tertentu. Selanjutnya, fungsi *checkAndSendData* memproses data *BLE* yang terdeteksi dan mengirimkannya ke *AWS IoT Core*, sementara event *onadvertisement* memastikan bahwa hanya *beacon* bus yang relevan yang diproses, dan pemindaian terus berlanjut. Dengan interval waktu yang telah ditentukan, sistem ini menjaga kontinuitas proses pemantauan dan pengiriman data, sekaligus memastikan sistem tetap berjalan dengan stabil melalui pemantauan kondisi perangkat dan *restart* otomatis bila diperlukan. Secara keseluruhan, program ini dirancang untuk mengoptimalkan pemantauan dan pengumpulan data transportasi umum secara real-time menggunakan teknologi BLE, memastikan komunikasi yang efisien dengan server, serta integrasi data yang berkelanjutan.

1	<code>private fun startScanning() {</code>
2	<code> beaconManager.startRangingBeacons(region)</code>
3	<code> Log.d("BeaconService", "Scanning started")</code>
4	<code>}</code>
5	
6	<code>// Function to stop scanning</code>
7	<code>private fun stopScanning() {</code>
8	<code> beaconManager.stopRangingBeacons(region)</code>
9	<code> Log.d("BeaconService", "Scanning stopped for</code>
10	<code>cooldown")</code>
11	<code>}</code>

- **Pada kode baris 1-4** fungsi *startScanning* digunakan untuk memulai pemindaian beacon. Pada baris pertama, *beaconManager.startRangingBeacons(region)* digunakan untuk memulai pencarian beacon dalam area yang telah ditentukan oleh objek *region*. Kemudian, pada baris kedua, *Log.d("BeaconService", "Scanning started")* digunakan untuk mencatat *log* dengan tag "*BeaconService*", yang memberi tahu bahwa pemindaian beacon telah dimulai.
- **Pada kode baris 7-10** fungsi *stopScanning* digunakan untuk menghentikan pemindaian beacon. *beaconManager.stopRangingBeacons(region)* menghentikan pencarian beacon dalam *region*. Pada baris kesembilan, *Log.d("BeaconService", "Scanning stopped for cooldown")* mencatat *log*

yang memberi tahu bahwa pemindaian telah dihentikan dan memberikan jeda sebelum pemindaian dapat dimulai kembali.

Fungsi `startScanning` memulai pemindaian beacon dalam area yang ditentukan, sementara `stopScanning` menghentikan pemindaian dan memberikan jeda sebelum pemindaian dapat dimulai kembali. Kedua fungsi ini juga mencatat log untuk memberikan informasi tentang status pemindaian, memastikan proses berjalan dengan efisien dan stabil.

```
1 private fun initializeBeaconManager() {
2     beaconManager = BeaconManager.
3     getInstanceForApplication(this)
4     val iBeaconLayout =
5         "m:2-3=0215,i:4-19,i:20-21,i:22-23,p:24-
6 24,d:25-25"
7     beaconManager.beaconParsers.add(BeaconParser().
8     setBeaconLayout(iBeaconLayout))
9
10    beaconManager.
11
12    enableForegroundServiceScanning(createNotification(), 2)
13
14    // Configure the scanning periods
15    beaconManager.foregroundBetweenScanPeriod = 0L
16    beaconManager.foregroundScanPeriod = 1000L
17    beaconManager.backgroundBetweenScanPeriod = 0L
18    beaconManager.backgroundScanPeriod = 1000L
19    region = Region("all-beacons-region", null, null,
20    null)
21    startScanning()
```

- Pada kode baris 1-3 fungsi `initializeBeaconManager` digunakan untuk menginisialisasi objek `beaconManager`. Pada baris pertama, `beaconManager = BeaconManager.getInstanceForApplication(this)` menginisialisasi instance dari `BeaconManager` untuk aplikasi yang sedang berjalan. Ini memungkinkan aplikasi untuk berinteraksi dengan beacon yang terdeteksi.
- Pada kode baris 1-3 fungsi `initializeBeaconManager` digunakan untuk menginisialisasi objek `beaconManager`. Pada baris pertama, `beaconManager = BeaconManager.getInstanceForApplication(this)` menginisialisasi instance dari `BeaconManager` untuk aplikasi yang sedang berjalan. Ini memungkinkan aplikasi untuk berinteraksi dengan beacon yang terdeteksi.
- Pada kode baris 4, `val iBeaconLayout = "m:2-3=0215,i:4-19,i:20-21,i:22-23,p:24-24,d:25-25"` mendefinisikan format atau layout dari `beacon` yang akan dipindai. `Layout` ini menggunakan format `iBeacon` untuk

menggambarkan struktur data *beacon*, seperti identifikasi, tipe *beacon*, dan data lainnya. *Layout* ini penting agar aplikasi dapat mengenali dan memproses *beacon* yang sesuai dengan struktur yang diharapkan.

- **Pada kode baris 5**

beaconManager.beaconParsers.add(BeaconParser().setBeaconLayout(iBeaconLayout)) menambahkan *parser* untuk *beacon* dengan *layout* yang telah ditentukan sebelumnya. *BeaconParser* bertugas untuk memformat data *beacon* yang diterima sesuai dengan *layout* tersebut, memungkinkan aplikasi untuk mengekstrak informasi yang relevan dari *beacon* yang terdeteksi.

- **Pada kode baris 6**

beaconManager.enableForegroundServiceScanning(createNotification(), 2) mengaktifkan pemindaian *beacon* di latar depan dengan menampilkan notifikasi yang dibuat oleh *createNotification()*. Angka 2 menunjukkan prioritas layanan pemindaian dalam konteks latar depan, memastikan bahwa pemindaian *beacon* mendapatkan prioritas tinggi saat aplikasi aktif dan berjalan di layar pengguna.

- **Pada kode baris 15-18**

konfigurasi periode pemindaian *beacon* diatur. *beaconManager.foregroundBetweenScanPeriod = 0L* dan *beaconManager.foregroundScanPeriod = 1000L* mengatur interval waktu antara dua pemindaian berturut-turut dan durasi pemindaian saat aplikasi berada di latar depan. Begitu pula, *beaconManager.backgroundBetweenScanPeriod = 0L* dan *beaconManager.backgroundScanPeriod = 1000L* mengonfigurasi periode pemindaian ketika aplikasi berjalan di latar belakang.

- **Pada kode baris 19-20** *region = Region("all-beacons-region", null, null, null)* mendefinisikan sebuah objek *region* untuk memindai semua *beacon* tanpa membatasi pada *ID* atau *UUID* tertentu. *region* ini memastikan bahwa aplikasi dapat memindai *beacon* di seluruh area yang dapat dijangkau tanpa adanya *filter* spesifik.

- **Pada kode baris 21** *startScanning()* dipanggil untuk memulai proses pemindaian *beacon* setelah inisialisasi dan konfigurasi selesai. Fungsi ini mengaktifkan pemindaian *beacon* sesuai dengan pengaturan yang telah dilakukan sebelumnya, baik di latar depan maupun latar belakang.

Secara keseluruhan, kode ini mengonfigurasi pemindaian *beacon* menggunakan *BeaconManager*. Dimulai dengan inisialisasi *beaconManager* untuk aplikasi, diikuti dengan mendefinisikan *layout beacon* sesuai format *iBeacon*.

BeaconParser ditambahkan untuk memformat data yang diterima, sementara pemindaian diaktifkan di latar depan dengan notifikasi dan prioritas tinggi. Interval pemindaian diatur untuk latar depan dan latar belakang, dan sebuah *region* dibuat untuk memindai semua *beacon* tanpa filter. Terakhir, pemindaian dimulai dengan memanggil *startScanning()*, memungkinkan deteksi beacon di seluruh area yang dapat dijangkau.

```
1 beaconManager.addRangeNotifier { beacons, _ ->
2   if (beacons.isNotEmpty()) {
3     // Get the device's unique ID (UUID)
4     val deviceId =
5     Secure.getString(applicationContext.contentResolver,
6     Secure.ANDROID_ID)
7
8     for (beacon in beacons) {
9       // Check if the beacon's Bluetooth address matches the
10      desired IDs
11      if (beacon.bluetoothAddress == "FB:FD:2A:A8:E2:BE" ||
12      beacon.bluetoothAddress == "F2:AB:73:19:59:79") {
13        val timestamp = getCurrentUnixTimestamp()
14
15        // Create the payload object
16        val beaconPayload = BeaconData(
17        operation = "post-beacon", payload = Payload(
18        timestamp = timestamp,
19        deviceId = deviceId, // Use the device's UUID
20        bleAddress = beacon.bluetoothAddress,
21        distance = beacon.distance,
22        isRead = 0,
23        proximityUUID = beacon.id1.toString(),
24        rssi = beacon.rssi,
25        txPower = beacon.txPower
26        )
27        )
28
29        // Logging the data class object
30        Log.d("BeaconPayload", beaconPayload.toString())
31
32        // Save to CSV or other handling
33        saveToCsv(
34        beacon.id1.toString(),
35        beacon.id2.toString(),
36        beacon.id3.toString(),
37        beacon.rssi.toString(),
38        timestamp
39        )
40
41        // Send the object directly if your logic allows
42        sendBroadcast(beaconPayload.toString())
43        sendBeaconDataToServer(beaconPayload)
44      }
```

```

45 // Stop scanning and start cooldown period of 10 seconds
46 stopScanning()
47
48 handler.postDelayed({
49     startScanning() // Resume scanning after 10 seconds
50 }, 10000)
51 }
52 }
53 }
54 }

```

- **Pada kode baris 1-6** `beaconManager.addRangeNotifier { beacons, _ -> ... }` digunakan untuk menambahkan *notifier* yang akan dipanggil setiap kali ada *beacon* yang terdeteksi dalam jangkauan. Fungsi ini menerima daftar *beacon* yang terdeteksi, dan kemudian memeriksa apakah ada *beacon* yang terdeteksi dengan `if (beacons.isNotEmpty())`. Jika ada *beacon*, aplikasi melanjutkan untuk memproses setiap *beacon* yang terdeteksi. Pada baris 6, `val deviceId = Secure.getString(applicationContext.contentResolver, Secure.ANDROID_ID)` digunakan untuk mendapatkan *ID* unik perangkat *Android* yang tetap, yang berfungsi sebagai identifikasi perangkat dalam *payload beacon*.
- **Pada kode baris 7-27** aplikasi melakukan iterasi terhadap setiap *beacon* yang terdeteksi dengan `for (beacon in beacons)`. Di dalam loop, aplikasi memeriksa apakah alamat *Bluetooth beacon* (`beacon.bluetoothAddress`) cocok dengan dua alamat yang telah ditentukan yakni "FB:FD:2A:A8:E2:BE" dan "F2:AB:73:19:59:79". Jika kecocokan ditemukan, aplikasi melanjutkan untuk mengambil *timestamp* saat ini menggunakan `val timestamp = getCurrentUnixTimestamp()`, yang memberikan tanda waktu dalam format *Unix*. Kemudian, objek *BeaconData* dibuat dengan data lengkap *beacon*, termasuk informasi seperti *timestamp*, *ID* perangkat (*deviceId*), alamat *Bluetooth beacon*, jarak *beacon* (*distance*), status pembacaan *beacon* (*isRead*), *UUID beacon* (*proximityUUID*), *RSSI* (kekuatan sinyal), dan *TxPower* (daya transmisi). Data ini kemudian dibungkus dalam *payload* yang siap untuk diproses lebih lanjut.
- **Pada kode baris 42-43** aplikasi menggunakan `sendBroadcast(beaconPayload.toString())` untuk mengirimkan data *beacon* melalui *broadcast intent*, memungkinkan komponen lain dalam aplikasi untuk menerima data *beacon* yang terdeteksi. Selain itu, `sendBeaconDataToServer(beaconPayload)` digunakan untuk mengirimkan data *beacon* langsung ke *server*, yang memungkinkan pengolahan atau penyimpanan data *beacon* di *server* untuk analisis lebih lanjut.
- **Pada kode baris 46-50** `stopScanning()` menghentikan pemindaian *beacon* saat ini, dan kemudian menggunakan `handler.postDelayed({`

startScanning() }, 10000) untuk memulai pemindaian *beacon* kembali setelah periode *cooldown* selama 10 detik. Fungsi ini memastikan pemindaian *beacon* dilakukan secara teratur, dengan jeda 10 detik di antara setiap sesi pemindaian.

Secara keseluruhan, kode ini menangani proses deteksi beacon dengan menambahkan *rangeNotifier* yang akan dipanggil setiap kali *beacon* terdeteksi dalam jangkauan. Aplikasi memeriksa apakah ada *beacon* yang terdeteksi dan, jika ada, memproses setiap *beacon* dengan memeriksa kecocokan alamat Bluetooth beacon tertentu. Setelah itu, data beacon, termasuk informasi seperti *timestamp*, ID perangkat, alamat *Bluetooth*, jarak, *UUID*, *RSSI*, dan *TxPower*, dikumpulkan dan dibungkus dalam *payload*. Data ini kemudian dikirimkan melalui *broadcast intent* dan langsung dikirim ke server untuk diproses lebih lanjut. Setelah pemindaian selesai, pemindaian dihentikan sementara dan dimulai kembali setelah jeda 10 detik untuk memastikan pemindaian berlangsung secara teratur.

```
1 private fun getCurrentUnixTimestamp(): Long {
2     return System.currentTimeMillis()
3 }
4
5 private fun sendBeaconDataToServer(beaconInfo: BeaconData)
6 {
7     val call =
8     ApiClient.apiService.sendBeaconData(beaconInfo)
9     call.enqueue(object : retrofit2.Callback<Void> {
10         override fun onResponse(call: Call<Void>,
11 response: retrofit2.Response<Void>) {
12             if (response.isSuccessful) {
13                 Log.d("BeaconService", "Data sent
14 successfully")
15             } else {
16                 Log.e("BeaconService", "Failed to send
17 data: ${response.code()}")
18             }
19         }
20
21         override fun onFailure(call: Call<Void>, t:
22 Throwable) {
23             Log.e("BeaconService", "Error sending data",
24 t)
25         }
26     })
27 }
```

- Pada kode baris 1-3 adalah fungsi *getCurrentUnixTimestamp()* digunakan untuk mendapatkan *timestamp* saat ini dalam *format Unix*. Fungsi ini

memanfaatkan *System.currentTimeMillis()*, yang mengembalikan jumlah milidetik yang telah berlalu sejak *epoch* (1 Januari 1970, 00:00:00 UTC). Nilai yang dikembalikan berupa waktu dalam milidetik, yang sering digunakan dalam aplikasi untuk mencatat waktu atau untuk membandingkan waktu kejadian tertentu. Fungsi ini umumnya digunakan dalam kasus di mana pengukuran waktu yang akurat dibutuhkan, seperti saat mencatat waktu deteksi *beacon* atau untuk keperluan *log* lainnya.

- **Pada kode baris 5-27** merupakan fungsi *sendBeaconDataToServer()* yang bertujuan untuk mengirim data *beacon* ke *server* menggunakan *Retrofit*. Data *beacon*, yang dikemas dalam objek *beaconInfo*, dikirimkan ke *server* melalui *endpoint sendBeaconData*. Permintaan ini dieksekusi secara asinkron menggunakan *enqueue*, sehingga tidak menghalangi *UI thread*. Ketika *server* merespons, jika respons berhasil, aplikasi mencatat bahwa data berhasil dikirim menggunakan *Log.d*. Sebaliknya, jika respons gagal, aplikasi akan mencatat pesan kesalahan dengan kode status dari respons tersebut. Selain itu, jika terjadi kesalahan selama pengiriman data, seperti masalah jaringan, *callback onFailure* akan dipanggil, dan aplikasi akan mencatat kesalahan tersebut bersama dengan informasi lebih lanjut. Fungsi ini memastikan bahwa data *beacon* dikirimkan dengan benar dan memungkinkan aplikasi untuk menangani kasus sukses maupun kegagalan pengiriman.

Secara keseluruhan, kode ini menggunakan fungsi *getCurrentUnixTimestamp()* untuk mendapatkan *timestamp* saat ini dalam format *Unix*, yang berguna untuk mencatat waktu kejadian. Fungsi *sendBeaconDataToServer()* mengirim data *beacon* ke *server* menggunakan *Retrofit* secara asinkron, memastikan *UI thread* tidak terhalang. Jika pengiriman berhasil, aplikasi mencatatnya, dan jika gagal, pesan kesalahan serta informasi tambahan dicatat untuk penanganan lebih lanjut. Fungsi ini memastikan pengiriman data *beacon* yang tepat dan menangani respons server dengan baik.

```
1  override fun onCreate(savedInstanceState: Bundle?) {
2      super.onCreate(savedInstanceState)
3      setContentView(R.layout.activity_main)
4
5      val toggleScannerButton =
6  findViewById<Button>(R.id.button_toggle_scanner)
7      beaconDataTextView =
8  findViewById(R.id.beacon_data_text_view)
9
10     // Register to receive broadcast messages from the
11
```

```

12 BeaconService
13     val filter = IntentFilter
14     ("com.example.beaconproximitysensor.BEACON_DATA")
15     registerReceiver(beaconDataReceiver, filter)
16
17     toggleScannerButton.setOnClickListener {
18         if (isScanning) {
19             stopService(Intent(this,
20 BeaconService::class.java))
21             toggleScannerButton.text = "Start Scanning"
22         } else {
23             if (arePermissionsGranted()) {
24                 startScanning()
25                 toggleScannerButton.text = "Stop Scanning"
26             } else {
27                 requestPermissions()
28             }
29         }
30         isScanning = !isScanning
31     }
32
33     // Automatically start scanning if permissions are
34 granted
35     if (arePermissionsGranted()) {
36         startScanning()
37         toggleScannerButton.text = "Stop Scanning"
38     } else {
39         requestPermissions()
40     }
41 }

```

- Pada kode baris 1-14 terdapat fungsi *onCreate()* yang digunakan untuk menginisialisasi aktivitas saat pertama kali dijalankan. Fungsi ini memanggil *super.onCreate(savedInstanceState)* untuk memastikan bahwa siklus hidup aktivitas dipatuhi, kemudian menetapkan *layout XML activity_main* dengan *setContentView()*. Selanjutnya, dua elemen UI, yaitu tombol untuk mengaktifkan dan menonaktifkan pemindaian *beacon* (*button_toggle_scanner*) dan *TextView* untuk menampilkan data *beacon* yang diterima (*beacon_data_text_view*), diinisialisasi menggunakan *findViewById()*. Di sini juga, aplikasi mendaftarkan penerima siaran (*broadcast receiver*) yang akan mendengarkan pesan yang dikirim oleh *BeaconService*. Penerima ini menggunakan *IntentFilter* dengan aksi "*com.example.beaconproximitysensor.BEACON_DATA*" untuk menerima data *beacon* yang dipancarkan.
- Pada kode baris 16-30 terdapat logika untuk menangani klik pada tombol *toggleScannerButton* yang berfungsi untuk memulai atau menghentikan pemindaian *beacon*. Ketika tombol diklik, kode memeriksa apakah

pemindaian sedang aktif dengan memeriksa nilai *boolean isScanning*. Jika *isScanning* bernilai *true*, pemindaian dihentikan dengan memanggil *stopService()* dan teks pada tombol diubah menjadi "*Start Scanning*". Sebaliknya, jika pemindaian tidak aktif, aplikasi akan memeriksa apakah izin yang dibutuhkan telah diberikan dengan fungsi *arePermissionsGranted()*. Jika izin sudah diberikan, pemindaian dimulai dengan memanggil *startScanning()* dan teks tombol diubah menjadi "*Stop Scanning*". Jika izin belum diberikan, maka aplikasi akan meminta izin menggunakan fungsi *requestPermissions()*. Setelah itu, status *isScanning* dibalik untuk mencatat status pemindaian yang baru.

- **Pada kode baris 34-40** terdapat bagian yang memastikan pemindaian *beacon* dimulai secara otomatis jika izin sudah diberikan. Jika aplikasi telah memiliki izin yang diperlukan untuk memindai *beacon*, maka pemindaian akan langsung dimulai dengan memanggil *startScanning()* dan tombol diubah menjadi "*Stop Scanning*". Jika izin belum diberikan, aplikasi akan meminta izin melalui *requestPermissions()*. Fungsi ini memastikan bahwa aplikasi dapat mulai memindai *beacon* secara otomatis saat aplikasi dibuka, tanpa perlu interaksi lebih lanjut dari pengguna.

Secara keseluruhan, kode ini mengatur inisialisasi aktivitas dan pengelolaan pemindaian *beacon*. Pada fungsi *onCreate()*, elemen UI seperti tombol dan *TextView* diinisialisasi, dan penerima siaran untuk menerima data *beacon* dari *BeaconService* didaftarkan. Ketika tombol diklik, kode memeriksa status pemindaian dan mengubah teks tombol sesuai dengan kondisi, baik memulai atau menghentikan pemindaian. Jika izin untuk memindai *beacon* telah diberikan, pemindaian dimulai otomatis, jika belum, aplikasi akan meminta izin terlebih dahulu.

5.2.3. Implementasi Program Telemetry.

```

1 import BeaconScanner from 'node-beacon-scanner';
2 import axios from 'axios';
3 import fs from 'fs';
4 import dotenv from 'dotenv';
5 import shell from 'shelljs';
6 dotenv.config();
7
8 const bus = fs.readFileSync(process.env.BUS, 'utf8');
9 const busStop =
10 fs.readFileSync(process.env.BUS_STOP, 'utf8');
11 const busStopObj = JSON.parse(busStop);
12 const rsuID = fs.readFileSync(process.env.NODE_ID, 'utf8');
```

```

13 const rsuIDObj = JSON.parse(rsuID);
14 const nodeID = rsuIDObj.nodeID;
15 const heartBeatInterval = process.env.HEARTBEAT_INTERVAL;
16
17 const telemetryToken =
18 busStopObj.telemetryToken[nodeID.toString()].toString();
19 const telemetryURL = process.env.API_TELEMETRY;
20
21 function run(){
22     async function sendTelemetry(telemetryData){
23         console.log('telemetryToken: ', telemetryToken);
24         if(telemetryToken === "-"){
25             console.log('Telemetry data is not sent -> RSU
26 does not has ACCESS_TOKEN!')
27         }
28         else{
29             const API_telemetry =
30 telemetryURL.replace("$ACCESS_TOKEN", telemetryToken);
31             const options = {
32                 method: 'POST',
33                 headers: {
34                     'Content-Type': 'application/json'
35                 }
36             };
37             try {
38                 const response = await
39 axios.post(API_telemetry, telemetryData, options);
40                 console.log('Send telemetry data to
41 thingsboard: ', telemetryData);
42             } catch (error) {
43                 console.log(error);
44                 const result = shell.exec('pm2 restart all');
45                 if (result.code !== 0) {
46                     console.log('Failed to restart process:',
47 result.stderr);
48                 } else {
49                     console.log('Process restarted
50 successfully');
51                 }
52             }
53         }
54     }
55 }
56
57 var temp =
58 fs.readFileSync("/sys/class/thermal/thermal_zone0/temp");
59 var temp_c = temp/1000;
60
61 // var temp_c = randomInt(25, 40);
62
63 const date = new Date();
64 var sampleTime = date.getTime();
65
66 const msg1 = {

```

```

67     internet: 1,
68     node: 1,
69     temperature: temp_c
70   };
71   const telemetryData = JSON.stringify(msg1);
72   sendTelemetry(telemetryData);
73 }
74
75 setInterval(run, heartBeatInterval);

```

- **Pada kode baris 1-6** berbagai modul diimpor untuk digunakan dalam skrip. Modul *BeaconScanner* digunakan untuk memindai sinyal *beacon*, meskipun tidak digunakan di bagian kode ini. Modul *axios* digunakan untuk mengirimkan permintaan *HTTP POST* ke server *API* untuk mengirim data telemetry. Modul *fs* digunakan untuk membaca *file* dari sistem *file*, memungkinkan skrip untuk mengakses konfigurasi yang disimpan dalam *file*. *dotenv* digunakan untuk memuat variabel lingkungan dari *file* *.env* sehingga informasi sensitif seperti token dan *URL API* dapat disimpan dengan aman. Sedangkan *shelljs* digunakan untuk menjalankan perintah *shell* dalam *Node.js*, di sini digunakan untuk memulai kembali aplikasi jika terjadi kesalahan. *dotenv.config()* digunakan untuk memuat variabel lingkungan dari *file* *.env*.
- **Pada kode baris 8-15** beberapa file konfigurasi dibaca dan diolah. *fs.readFileSync(process.env.BUS, 'utf8')* membaca file yang ditentukan oleh variabel lingkungan *BUS* dan menyimpannya dalam variabel *bus*. Demikian juga, *fs.readFileSync(process.env.BUS_STOP, 'utf8')* membaca konfigurasi pemberhentian bus dari file yang ditentukan oleh *BUS_STOP* dan mengonversinya menjadi objek *JavaScript* menggunakan *JSON.parse()*. File konfigurasi *NODE_ID* yang berisi informasi ID node juga dibaca dengan *fs.readFileSync(process.env.NODE_ID, 'utf8')* dan diproses menjadi objek *rsulDObj*. Dari objek ini, nilai *nodeID* diambil. Terakhir, variabel *heartBeatInterval* diambil dari *file* *.env*, yang menyimpan interval waktu untuk mengirimkan data *telemetry*.
- **Pada kode baris 17-19** token autentikasi dan *URL API* disiapkan untuk digunakan dalam pengiriman data *telemetry*. *telemetryToken* diambil dari objek *busStopObj*, yang berisi token autentikasi untuk setiap *node*. Nilai token ini ditentukan berdasarkan *nodeID* yang telah dibaca sebelumnya. Kemudian, *telemetryURL* diambil dari variabel lingkungan *API_TELEMETRY*, yang berisi *URL endpoint* untuk mengirimkan data *telemetry*. Token ini akan digunakan dalam permintaan *API* untuk memastikan bahwa hanya perangkat yang terautentikasi yang dapat mengirimkan data.
- **Pada kode baris 21-73** Di bagian ini, fungsi *run()* didefinisikan, yang berfungsi untuk mengumpulkan data dan mengirimkannya secara berkala.

Di dalam *run()*, fungsi *sendTelemetry()* didefinisikan sebagai fungsi asinkron yang bertanggung jawab untuk mengirimkan data *telemetry* ke server API. Pertama, kode memeriksa apakah *telemetryToken* valid. Jika token adalah "-", artinya perangkat tidak memiliki akses token, dan data tidak akan dikirim, disertai pesan kesalahan. Jika token valid, URL API dibangun dengan mengganti *placeholder \$ACCESS_TOKEN* dengan token yang sesuai. Permintaan HTTP POST kemudian dikirimkan menggunakan *axios.post()* dengan data *telemetry* yang dikirim dalam format JSON. Jika pengiriman data berhasil, pesan konfirmasi ditampilkan. Namun, jika terjadi kesalahan, aplikasi akan mencoba untuk memulai kembali proses dengan menjalankan perintah *pm2 restart all* menggunakan *shell.exec()*. Hasil dari perintah *restart* dicetak, apakah berhasil atau gagal. Selain itu, suhu perangkat dibaca dari file sistem */sys/class/thermal/thermal_zone0/temp*, yang mengembalikan suhu dalam satuan milidesial (1000) derajat Celsius. Nilai suhu ini dibagi 1000 untuk mendapatkan suhu dalam derajat Celsius dan disimpan dalam variabel *temp_c*. Waktu saat pengambilan sampel juga dicatat dengan menggunakan *new Date()* untuk mendapatkan *timestamp* saat ini. Data *telemetry* yang akan dikirim berisi status koneksi internet (internet: 1), ID node (node: 1), dan nilai suhu perangkat. Data tersebut kemudian dikonversi menjadi format JSON dan dikirimkan ke fungsi *sendTelemetry()*.

- **Pada kode baris 75** fungsi *run()* dipanggil setiap interval yang telah ditentukan oleh *heartBeatInterval*. Dengan menggunakan *setInterval()*, fungsi *run()* akan dijalankan secara berkala untuk mengumpulkan data *telemetry* dan mengirimkannya ke server API. Interval ini bergantung pada nilai yang ditentukan dalam variabel lingkungan *HEARTBEAT_INTERVAL*, yang memastikan data dikirimkan secara periodik sesuai dengan kebutuhan aplikasi.

Kode ini menginisialisasi proses pengumpulan dan pengiriman data *telemetry* menggunakan modul seperti *BeaconScanner*, *axios*, dan *dotenv*. File konfigurasi dibaca untuk mendapatkan informasi penting, seperti bus, pemberhentian bus, dan *node ID*. Fungsi *run()* mengumpulkan data seperti suhu perangkat dan status koneksi internet, kemudian mengirimkannya ke server API secara berkala menggunakan interval yang ditentukan. Jika terjadi kesalahan, aplikasi mencoba untuk melakukan *restart* otomatis agar pengiriman data tetap berjalan.

5.2.4. Implementasi Akses Jarak Jauh.

```
1 import fs from 'fs';
2 import dotenv from 'dotenv';
3 dotenv.config();
4
5
6
7 const rsuID = fs.readFileSync(process.env.NODE_ID, 'utf8');
8 const rsuIDObj = JSON.parse(rsuID);
9 const nodeID = rsuIDObj.nodeID;
10
11
12 import { exec } from 'child_process';
13
14
15 var maincommand = 'bore local 22 --to 54.169.36.133 --port
16 900';
17 var startCommand = maincommand + nodeID.toString();
18
19 console.log(startCommand);
20
21 exec(startCommand, (error, stdout, stderr) => {
22   if (error) {
23     console.error(Error starting the service: ${error});
24     return;
25   }
26   console.log(Service started successfully:\n${stdout});
27 });
```

- **Pada kode baris 1-3** kode ini mengimpor modul *fs* dan *dotenv*. Modul *fs* digunakan untuk membaca *file* sistem secara sinkronus, sementara *dotenv* digunakan untuk memuat variabel lingkungan yang disimpan dalam *file .env* ke dalam *process.env*, sehingga variabel-variabel tersebut dapat digunakan dalam kode. Fungsi *dotenv.config()* digunakan untuk memuat *file .env* yang berisi konfigurasi lingkungan.
- **Pada kode baris 7-9** kode ini membaca *file* yang berada pada *path* yang ditentukan oleh variabel *process.env.NODE_ID*, yang biasanya berisi *path file* yang menyimpan informasi konfigurasi, seperti *ID node*. Fungsi *fs.readFileSync()* digunakan untuk membaca file tersebut dengan *encoding 'utf8'* dan kemudian isinya disimpan dalam variabel *rsuID*. Setelah itu, fungsi *JSON.parse()* digunakan untuk mengonversi *string JSON* yang dibaca dari file menjadi objek *JavaScript* yang disimpan dalam *rsuIDObj*. Selanjutnya, variabel *nodeID* diambil dari properti *nodeID* dalam objek *rsuIDObj*.

- **Pada kode baris 12** modul `exec` dari paket `child_process` diimpor. Fungsi `exec` digunakan untuk menjalankan perintah sistem atau *shell* dari dalam aplikasi *Node.js*.
- **Pada kode baris 15-17** variabel `maincommand` didefinisikan dengan nilai string `'bore local 22 --to 54.169.36.133 --port 900'`, yang berfungsi sebagai perintah dasar untuk menjalankan sebuah layanan atau aplikasi (dalam hal ini sepertinya terkait dengan *tunneling* atau *forward port*). Variabel `startCommand` kemudian didefinisikan dengan menggabungkan `maincommand` dan nilai `nodeID` yang diperoleh sebelumnya, menghasilkan sebuah perintah *shell* yang unik berdasarkan *ID node*.
- **Pada kode baris 21-27** fungsi `exec` dipanggil dengan perintah `startCommand` sebagai argumen. Fungsi `exec` menjalankan perintah *shell* dan menangani hasilnya dalam *callback*. *Callback* ini menerima tiga argumen: `error`, `stdout`, dan `stderr`. Jika terjadi kesalahan saat menjalankan perintah (misalnya perintah tidak ditemukan atau terjadi masalah lain), maka objek `error` akan berisi informasi kesalahan tersebut dan akan dicetak ke konsol. Jika perintah berhasil dijalankan, hasil keluaran dari perintah akan dikembalikan melalui `stdout`, yang kemudian dicetak ke konsol, menunjukkan bahwa layanan dimulai dengan sukses. Jika ada pesan kesalahan dari perintah, itu akan dicetak melalui `stderr`.

Kode ini mengimpor modul `'fs'` dan `'dotenv'` untuk membaca *file* konfigurasi dan memuat variabel lingkungan. File yang berisi informasi *ID node* dibaca dan diproses menjadi objek *JavaScript*. Selanjutnya, perintah *shell* untuk menjalankan layanan atau aplikasi dikonstruksi dengan menggunakan *ID node* dan dijalankan melalui fungsi `'exec'` dari modul `'child_process'`. Hasil eksekusi perintah dicetak ke konsol, baik itu sukses atau kesalahan.

```

1 import { exec } from 'child_process';
2
3 const CHECK_INTERVAL = 60000; // 1 minute
4 const RESTART_INTERVAL = 300000; // 5 minutes
5
6 const serviceStatusCommand = 'systemctl status
7 boreClient.service';
8 const reloadCommand = 'sudo systemctl start
9 boreClient.service';
10 const restartCommand = 'sudo systemctl restart
11 boreClient.service';
12
13 function checkBore() {
14   exec(serviceStatusCommand, (error, stdout, stderr) => {
15     if (error) {

```

```

16     exec(reloadCommand, (error, stdout) => {
17         if (error) {
18             console.log(Failed to reload BoreClient, Error:
19 ${error});
20             return;
21         }
22         console.log(Success to start BoreClient:
23 ${stdout});
24     });
25     return;
26 }
27 console.log(Service status:\n${stdout});
28 });
29 }
30
31 function restartBore() {
32     exec(restartCommand, (error, stdout, stderr) => {
33         if (error) {
34             console.log(Failed to restart BoreClient, Error:
35 ${error});
36             return;
37         }
38         console.log(Successfully restarted BoreClient:
39 ${stdout});
40     });
41 }
42
44 setInterval(checkBore, CHECK_INTERVAL);
45 setInterval(restartBore, RESTART_INTERVAL);

```

- **Pada kode baris 1-11** terdapat deklarasi dua konstanta *CHECK_INTERVAL* dan *RESTART_INTERVAL* yang masing-masing menentukan interval waktu pengecekan status dan restart layanan dalam milidetik. *CHECK_INTERVAL* diatur ke 60000 milidetik (1 menit), sedangkan *RESTART_INTERVAL* diatur ke 300000 milidetik (5 menit). Selanjutnya, terdapat tiga perintah untuk mengelola layanan *boreClient*: *serviceStatusCommand*, *reloadCommand*, dan *restartCommand*, yang masing-masing digunakan untuk memeriksa status layanan, memulai layanan jika tidak berjalan, dan merestart layanan tersebut.
- **Pada kode baris 13-29** digunakan untuk memeriksa status layanan *boreClient*. Pada baris ini, fungsi *exec()* dipanggil dengan perintah *systemctl status boreClient.service* untuk memeriksa status layanan. Jika terjadi kesalahan dalam menjalankan perintah (misalnya layanan tidak aktif), maka perintah *reloadCommand* (untuk memulai kembali layanan) akan dijalankan sebagai pengganti. Jika layanan berhasil dimulai, pesan berhasil akan ditampilkan, dan jika gagal, pesan kesalahan akan dicetak.

- **Pada kode baris 31-41** bertugas untuk memulai ulang layanan *boreClient* setiap interval 5 menit, menggunakan perintah *systemctl restart boreClient.service*. Jika perintah *restart* gagal dijalankan, maka akan ditampilkan pesan kesalahan. Jika berhasil, pesan keberhasilan akan dicetak.
- **Pada kode baris 31-41** bertugas untuk memulai ulang layanan *boreClient* setiap interval 5 menit, menggunakan perintah *systemctl restart boreClient.service*. Jika perintah *restart* gagal dijalankan, maka akan ditampilkan pesan kesalahan. Jika berhasil, pesan keberhasilan akan dicetak.
- **Pada kode baris 44-45** adalah *setInterval* digunakan untuk memanggil fungsi *checkBore* setiap 1 menit dan fungsi *restartBore* setiap 5 menit. Hal ini memungkinkan pemantauan status layanan secara otomatis, dengan pengecekan status dan restart layanan yang dilakukan secara berkala sesuai dengan interval yang telah ditentukan.

Kode ini adalah skrip *Node.js* untuk memantau dan mengelola layanan *'boreClient.service'* menggunakan perintah *'systemctl'*. Fungsi utama skrip ini adalah memastikan layanan tetap aktif, dengan memeriksa status dan merestart layanan jika diperlukan, menggunakan fungsi *'checkBore'* dan *'restartBore'*, serta penjadwalan otomatis melalui *'setInterval'* untuk menjaga ketersediaan layanan.

```

1 import isOnline from 'is-online';
2 import shell from 'shelljs';
3 let CHECKINTERVAL = 10000;
4 var RASPIONLINE = true;
5
6 async function onlineCheck(){
7     if(await isOnline()){
8         if (!RASPIONLINE) {
9             try {
10                 const restartResult = shell.exec('sudo
11 systemctl restart boreClient.service')
12                 if (restartResult.code === 0) {
13                     console.log('boreClient.service
14 restarted successfully.');
```

```

15                 } else {
16                     console.error('Failed to restart
17 boreClient.service');
```

```

18                 }
19                 console.log('Raspi is Online and ready!');
20                 console.log("BoreClient is restarted");
21
22             } catch (error) {
23                 console.error("ERROR RESTARTING BORECLIENT
24 SERVICE ", error.message);
```


25	}
26	
27	RASPIONLINE = true;
28	}
29	}
30	else{
31	console.error('Raspi is offline');
32	RASPIONLINE = false;
33	}
34	};
35	setInterval(onlineCheck, CHECKINTERVAL);

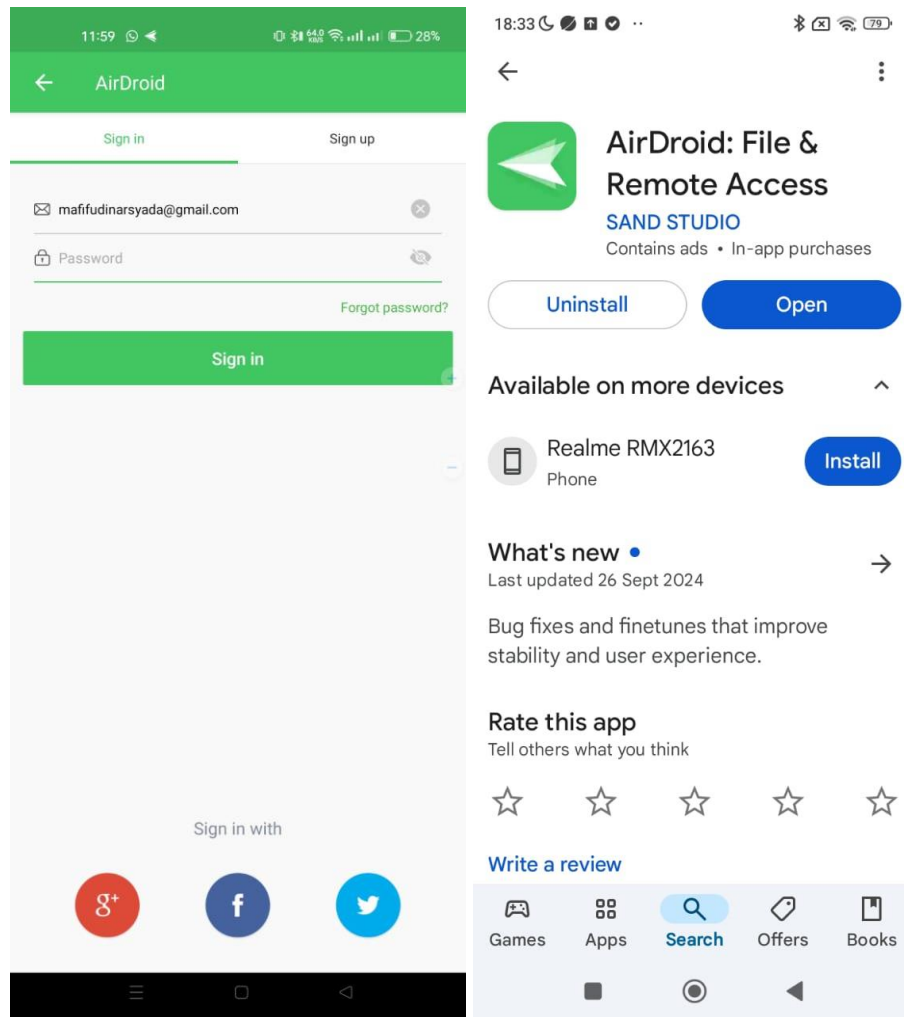
- **Pada kode baris 1-2** dua modul diimpor yakni *isOnline* untuk memeriksa konektivitas internet dan *shelljs* untuk menjalankan perintah *shell* dalam *Node.js*.
- **Pada kode baris 3-4** mendeklarasikan dua variabel yakni variabel *CHECKINTERVAL* diatur dengan nilai 10000 milidetik (10 detik), yang menunjukkan interval waktu antara setiap pengecekan koneksi internet yang dilakukan oleh skrip. Dan variabel *RASPIONLINE* digunakan untuk menyimpan status konektivitas perangkat. Nilai awalnya adalah *true*, yang berarti perangkat dianggap online pada awalnya.
- **Pada kode baris 6-34** fungsi *onlineCheck()* didefinisikan untuk memeriksa status koneksi internet perangkat. Fungsi ini pertama-tama memanggil await *isOnline()* untuk memeriksa apakah perangkat terhubung ke internet. Jika perangkat online dan sebelumnya terdeteksi *offline* (status *RASPIONLINE* adalah *false*), maka skrip akan mencoba untuk memulai ulang layanan *boreClient.service* menggunakan perintah *shell.exec('sudo systemctl restart boreClient.service')*. Jika perintah berhasil dijalankan, pesan keberhasilan akan ditampilkan; jika gagal, pesan kesalahan akan muncul. Setelah berhasil memulai ulang layanan, status *RASPIONLINE* diubah menjadi *true*, yang menandakan bahwa perangkat sekarang terhubung ke internet. Namun, jika perangkat tidak terhubung ke internet, maka pesan kesalahan "*Raspi is offline*" akan dicetak, dan status *RASPIONLINE* diatur menjadi *false*.
- **Pada kode baris 35** terdapat fungsi *onlineCheck()* dipanggil berulang kali setiap 10 detik sesuai dengan nilai *CHECKINTERVAL*. Dengan cara ini, skrip akan terus memeriksa status konektivitas perangkat secara berkala.

Kode ini adalah skrip untuk memeriksa konektivitas internet perangkat secara berkala. Setiap 10 detik, fungsi `onlineCheck()` memeriksa apakah perangkat terhubung ke internet menggunakan modul `isOnline()`. Jika perangkat online dan sebelumnya *offline*, skrip akan mencoba merestart layanan

`boreClient.service`. Status koneksi perangkat disimpan dalam variabel `RASPIONLINE`, yang diperbarui berdasarkan hasil pengecekan koneksi.

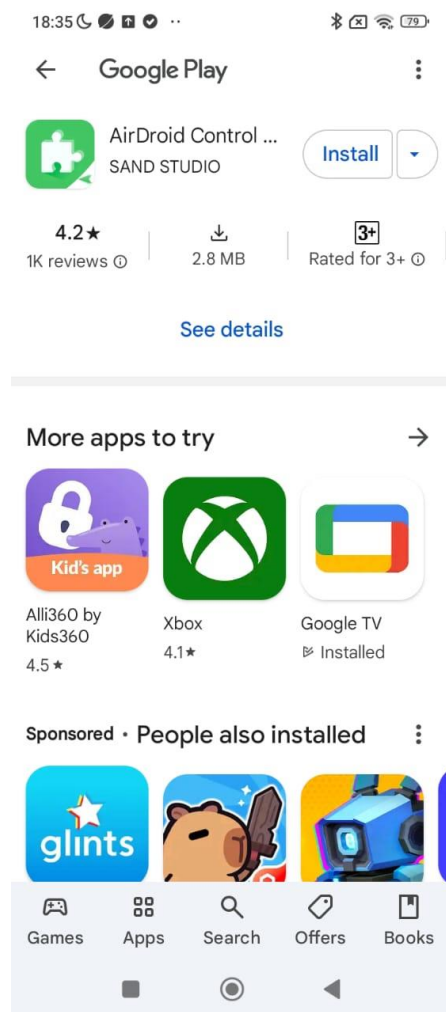
Sementara itu, Untuk mengimplementasikan akses jarak jauh pada smartphone Android, Anda dapat menggunakan aplikasi AirDroid, yang tersedia di Google Play Store. Aplikasi ini memungkinkan pengguna untuk mengakses dan mengontrol smartphone mereka melalui perangkat lain, seperti laptop atau smartphone lainnya yang juga memiliki aplikasi serupa. AirDroid mempermudah manajemen perangkat tanpa perlu berada di lokasi fisik, sehingga sangat berguna untuk pengelolaan perangkat yang terhubung ke sistem secara efisien.

Langkah pertama dalam menggunakan AirDroid adalah mengunduh dan instal aplikasi dari Google Play Store. Setelah aplikasi terpasang, buka AirDroid dan lakukan login menggunakan akun AirDroid Anda. Pada tahap ini, Anda akan diminta untuk memasukkan informasi akun yang telah terdaftar sebelumnya seperti pada gambar 5.7, yang memungkinkan Anda untuk mengakses fitur-fitur aplikasi secara penuh.

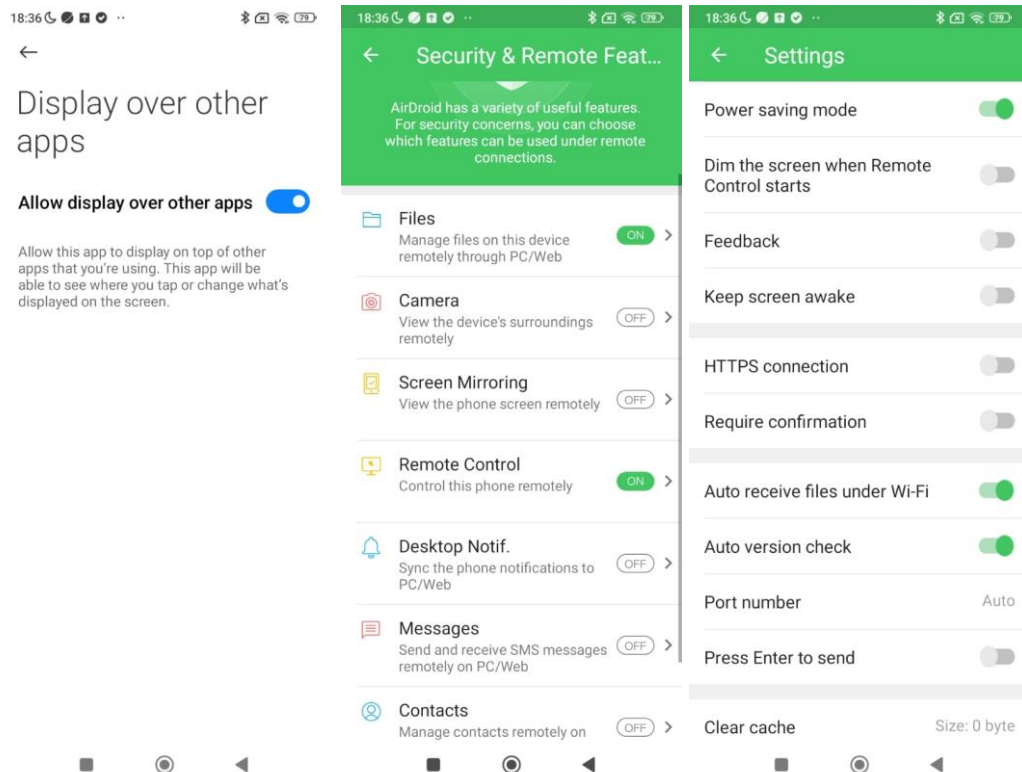


Gambar 5.7 Halaman *Login* dan Tampilan *PlayStore*

Setelah berhasil login, langkah berikutnya adalah menginstal AirDroid Control Add-On Seperti pada gambar 5.8 Add-on ini memungkinkan pengaturan kontrol jarak jauh tanpa memerlukan akses root pada perangkat Android. Proses ini penting untuk memastikan bahwa perangkat Android dapat dikendalikan secara penuh dari jarak jauh oleh pengguna lain.



Gambar 5.8 Implementasi Raspberry Pi 4 dan *Peripheral*



Gambar 5.9 Implementasi Raspberry Pi 4 dan *Peripheral*

Selanjutnya, buka pengaturan aplikasi AirDroid dan aktifkan pengaturan remote control. Pastikan untuk mematikan opsi require confirmation, yang biasanya meminta izin setiap kali perangkat terhubung, agar proses kontrol jarak jauh bisa berjalan lancar tanpa gangguan. Terakhir, berikan izin yang diminta oleh aplikasi untuk memastikan fungsi kontrol jarak jauh dapat berjalan dengan sempurna seperti pada gambar 5.9

Dengan langkah-langkah ini, Anda akan dapat mengakses dan mengontrol smartphone Android dari jarak jauh menggunakan aplikasi AirDroid, mempermudah pengelolaan dan pemantauan perangkat di lapangan.

BAB VI PENGUJIAN DAN ANALISIS

Bab ini berisikan pengujian terhadap sistem untuk menguji kemampuan dari sistem. Sistem diharapkan dapat menjalankan tiap fungsinya sebagaimana mestinya. Pada bagian ini akan dilakukan serangkaian tes untuk memastikan hal tersebut. Pengujian terbagi menjadi beberapa bagian sesuai tahap implementasi yang telah dilakukan sebelumnya.

6.1 Hasil Pengujian

Bagian sub-bab pengujian ini menjabarkan penemuan-penemuan yang ditemukan dari pengujian terhadap sistem. Alasan dan tujuan dari tiap pengujian yang dilakukan. Serta prosedur dari pengujian tersebut.

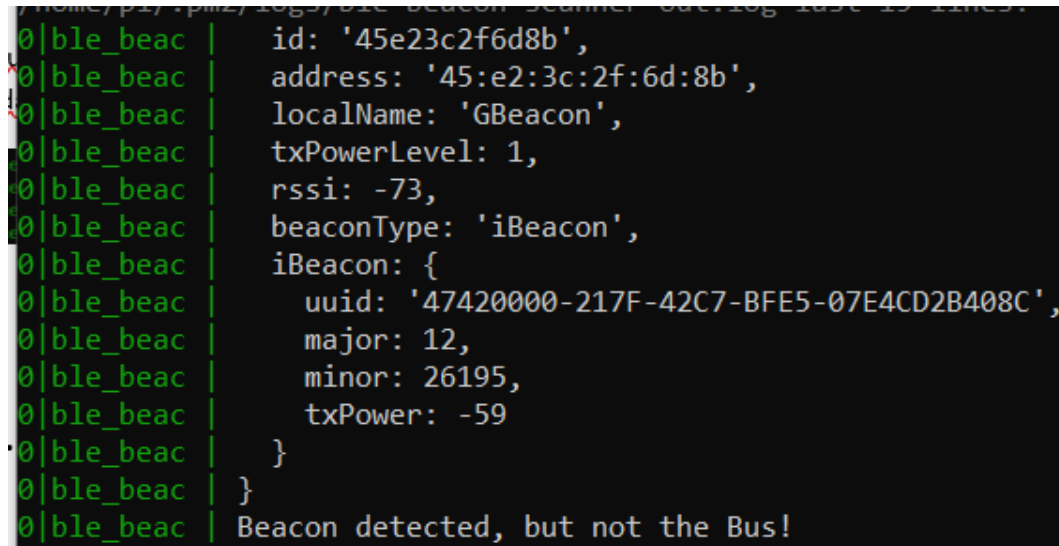
6.1.1. Pengujian Deteksi BLE Beacon Dengan Scanner Raspberry pi 4.

Pengujian ini bertujuan untuk menguji sejauh mana kemampuan Raspberry Pi 4 dalam menangkap sinyal BLE beacon. Proses ini dilakukan dengan mendekatkan BLE beacon ke perangkat scanner Raspberry Pi 4. Raspberry Pi 4 yang dilengkapi dengan kemampuan Bluetooth Low Energy (BLE) akan mendeteksi sinyal yang dipancarkan oleh beacon dan kemudian memproses serta menampilkan informasi terkait sinyal tersebut pada Command Line. Jika sinyal berhasil diterima, informasi seperti waktu penangkapan sinyal, BLE Address, UUID, Major, Minor, dan RSSI akan muncul di layar, memberikan data yang relevan untuk menganalisis jangkauan dan kekuatan sinyal beacon yang diterima. Penjelasan lebih detail dapat dilihat pada proses berikut:

- Pastikan Raspberry Pi 4 sudah terpasang dan berfungsi dengan baik.
- Hubungkan Raspberry Pi 4 dengan monitor dan keyboard untuk memantau output melalui Command Line.
- Siapkan BLE beacon yang akan diuji dan pastikan beacon dalam keadaan aktif.
- Nyalakan Raspberry Pi 4 dan pastikan Bluetooth telah diaktifkan.
- Jalankan kode program untuk melakukan scan BLE beacon
- Tempatkan BLE beacon di dekat Raspberry Pi 4 pada jarak yang diinginkan untuk pengujian.
- Pastikan bahwa beacon memancarkan sinyal BLE yang dapat terdeteksi oleh Raspberry Pi.

- Apabila sinyal BLE ditangkap oleh Raspberry pi 4 akan ditampilkan data sinyal seperti BLE Address, UUID, Major, Minor, dan RSSI (Received Signal Strength Indicator), akan diproses dan ditampilkan di Command Line.

Hasil dari deteksi BLE beacon dan Scanner Raspberry pi 4 dapat dilihat pada gambar 6.1 di bawah ini.



```

0|ble_beac | id: '45e23c2f6d8b',
0|ble_beac | address: '45:e2:3c:2f:6d:8b',
0|ble_beac | localName: 'GBeacon',
0|ble_beac | txPowerLevel: 1,
0|ble_beac | rssi: -73,
0|ble_beac | beaconType: 'iBeacon',
0|ble_beac | iBeacon: {
0|ble_beac |   uuid: '47420000-217F-42C7-BFE5-07E4CD2B408C',
0|ble_beac |   major: 12,
0|ble_beac |   minor: 26195,
0|ble_beac |   txPower: -59
0|ble_beac | }
0|ble_beac | }
0|ble_beac | Beacon detected, but not the Bus!

```

Gambar 6.1 Hasil Deteksi BLE Beacon dengan Raspberry pi 4

Dengan pengujian ini, diharapkan dapat diketahui seberapa efektif Raspberry Pi 4 dalam menangkap sinyal BLE beacon pada jarak dan kondisi tertentu. Pengujian juga dapat memberikan wawasan terkait dengan kinerja sistem dalam kondisi yang berbeda, seperti saat beacon berada pada jarak yang lebih jauh atau lebih dekat, serta bagaimana Raspberry Pi 4 memproses data yang diterima dalam situasi nyata.

6.1.2. Pengujian Deteksi BLE Beacon Dengan Scanner Smartphone Android.

Pengujian deteksi BLE beacon dengan smartphone Android bertujuan untuk memastikan bahwa smartphone dapat mendeteksi dan menerima sinyal BLE beacon dengan benar. Proses ini melibatkan beberapa langkah untuk memastikan bahwa perangkat berfungsi dengan baik dalam menangkap dan menampilkan informasi sinyal dari beacon. Dalam pengujian ini, aplikasi beacon scanner yang sudah dibuat pada smartphone Android digunakan untuk menangkap dan menampilkan data dari beacon yang terdeteksi, seperti BLE

Address, UUID, Major, Minor, dan RSSI. Prosedur pengujian adalah sebagai berikut:

- Pastikan smartphone menyala dan berfungsi dengan baik, serta sistem operasi dalam kondisi optimal.
- Nyalakan Bluetooth pada smartphone dan pastikan terhubung dengan perangkat BLE beacon.
- Pastikan aplikasi beacon scanner terinstal dan dapat mengakses Bluetooth untuk mendeteksi beacon.
- Dekatkan beacon ke smartphone dalam jarak yang diinginkan untuk pengujian.
- Tunggu aplikasi untuk menampilkan informasi beacon, seperti BLE Address, UUID, Major, Minor, dan RSSI.

Pengujian ini juga bertujuan untuk memastikan bahwa smartphone Android dapat menerima sinyal BLE beacon pada berbagai jarak dan kondisi, serta untuk memverifikasi kinerja aplikasi beacon scanner yang telah dikembangkan. Dengan pengujian ini, diharapkan bisa dipastikan bahwa aplikasi dan perangkat dapat bekerja dengan lancar dalam lingkungan yang realistis. Hasil dari pengujian di atas dapat dilihat pada gambar 6.2 di bawah ini.



Gambar 6.2 Hasil Deteksi BLE Beacon dengan Raspberry pi 4

6.1.4. Pengujian Pengiriman Data Ble Scanner dan Telemetry Ke Server.

Pengujian pengiriman data BLE scanner dan telemetry ke server bertujuan untuk memastikan bahwa sistem yang dibangun dapat mendeteksi sinyal dari BLE beacon, memproses data yang diterima, dan mengirimkan data tersebut ke server untuk dianalisis. Proses pengujian ini juga melibatkan pengambilan data suhu dari Raspberry Pi 4 dan pengirimannya ke server ThingsBoard menggunakan API yang sudah disiapkan. Pengujian ini tidak hanya bertujuan untuk memastikan bahwa data beacon diterima dengan benar, tetapi juga untuk menguji kestabilan pengiriman data telemetry secara berkala, serta kemampuan sistem dalam menangani kesalahan pengiriman data dan melakukan pemulihan otomatis melalui perintah restart. Langkah pengujian adalah sebagai berikut:

- Pastikan perangkat yang digunakan untuk pengujian sudah dilengkapi dengan Raspberry Pi 4 yang terhubung ke BLE beacon dan internet melalui koneksi Wi-Fi atau hotspot.
- Periksa bahwa Raspberry Pi 4 memiliki akses internet, baik melalui koneksi Wi-Fi atau hotspot smartphone yang berfungsi.
- Jalankan aplikasi scanner BLE dan telemetry yang memanfaatkan pm2 untuk mendeteksi beacon BLE di sekitar Raspberry Pi 4 dan mengambil data suhu CPU.
- Periksa apakah perangkat dapat berhasil mendeteksi beacon BLE dan menampilkan informasi terkait (misalnya BLE Address, UUID, RSSI) di Command Line.
- Pastikan bahwa data suhu (temperature) Raspberry Pi diambil dengan benar dari sistem.
- Pastikan kode program dapat mengirimkan data telemetry (seperti suhu dan status internet) ke server melalui ThingsBoard API.

Pengujian pengiriman data BLE scanner dan telemetry ke server bertujuan memastikan bahwa data dari BLE beacon dapat diterima dan diproses dengan baik oleh sistem, serta dikirimkan ke server ThingsBoard sesuai interval yang ditentukan. Data yang dikirimkan meliputi informasi suhu dan status internet. Hasil dari pengiriman data dapat dilihat pada gambar 6.3 di bawah ini.

```
2|telemetr | Send telemetry data to thingsboard: {"internet":1,"node":1,"temperature":65.244}
2|telemetr | telemetryToken: qaXEI4SezjJYfxL1dx3d
2|telemetr | Send telemetry data to thingsboard: {"internet":1,"node":1,"temperature":64.27}
2|telemetr | telemetryToken: qaXEI4SezjJYfxL1dx3d
```

```

b|ble_beac | iBeacon of the bus is found!
b|ble_beac | Ble Address Beacon: f2:ab:73:19:59:79
b|ble_beac | getETA:{"bus_id":33,"service_no":"Pool A","route_id":10,"bus_stop_id":1100}
b|ble_beac | getETA:{"bus_id":33,"service_no":"Pool A","route_id":11,"bus_stop_id":1112}
b|ble_beac | busInsertLocation: {"bus_id":33,"route_id":10,"imei":"f2:ab:73:19:59:79","latlong":"-7.943272,112.610494","s

```

Gambar 6.3 Percobaan Pengiriman Data ke Server

Pada gambar 6.4 tersebut, terlihat kumpulan data pendeteksian yang terkumpul di AWS oleh *smartphone Android* yang memindai sinyal BLE setiap harinya. Setiap entri mencakup informasi seperti deviceId, distance, txpower, timestamp dalam format Unix, rssi, proximity UUID, BleAddress, dan isRead. Data ini dikumpulkan secara real-time dan disimpan di AWS, memungkinkan analisis dan pengelolaan data secara efisien, serta memudahkan pemantauan lokasi beacon dan perangkat BLE dalam sistem.

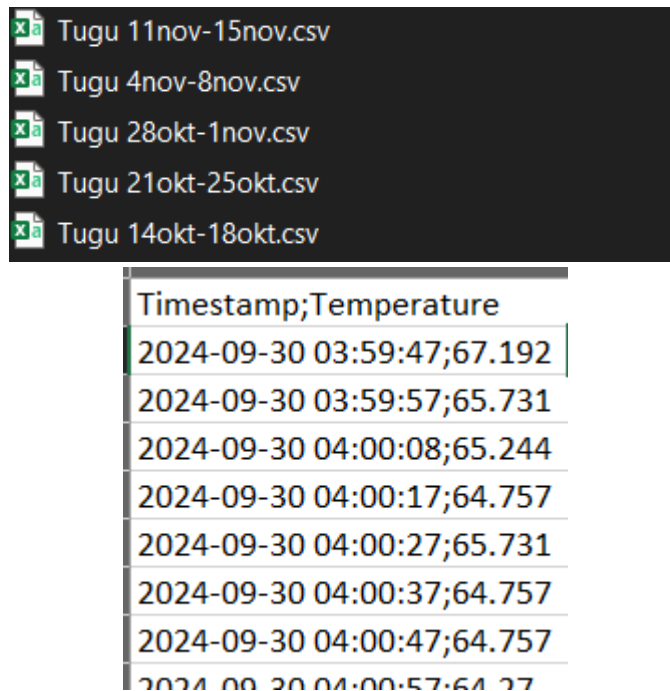
Items returned (2334) [Download](#)

< 1 2 3 4 5 6 ;

distance	deviceId	txPower	timestamp	rssi	proximityUUID	bleAddress	isRead
20.728642898...	57f713574...	-55	173197278...	-96	b9407f30-f5f8-...	F2:AB:73:19:...	0
23.836641075...	2e4c6deefa...	-55	172923937...	-98	b9407f30-f5f8-...	F2:AB:73:19:...	0
15.563296198...	57f713574...	-55	172768535...	-92	b9407f30-f5f8-...	F2:AB:73:19:...	0
25.538474578...	57f713574...	-55	172766414...	-99	b9407f30-f5f8-...	F2:AB:73:19:...	0
20.728642898...	57f713574...	-55	173197275...	-96	b9407f30-f5f8-...	F2:AB:73:19:...	0
22.235016484...	71afdee58f...	-55	173128099...	-97	b9407f30-f5f8-...	F2:AB:73:19:...	0
16.734379164...	1dde47fc7a...	-55	172963958...	-93	b9407f30-f5f8-...	F2:AB:73:19:...	0
22.235016484...	2e4c6deefa...	-55	173019267...	-97	b9407f30-f5f8-...	F2:AB:73:19:...	0
16.734379164...	2e4c6deefa...	-55	173197140...	-93	b9407f30-f5f8-...	F2:AB:73:19:...	0
23.836641075...	57f713574...	-55	172923937...	-98	b9407f30-f5f8-...	F2:AB:73:19:...	0

Gambar 6.4 Kumpulan Data Deteksi BLE Menggunakan Android

Pada gambar 6.5 terlihat bahwa data telemetry yang sudah diunduh mencakup informasi penting seperti timestamp dan temperature. Data ini dikumpulkan dan disimpan dalam format CSV (Comma-Separated Values), di mana setiap entri dipisahkan oleh koma. Data ini diperoleh setiap minggunya dari setiap RSU, yang terhubung dengan sistem monitoring untuk memantau kondisi perangkat atau lingkungan di lokasi tertentu.

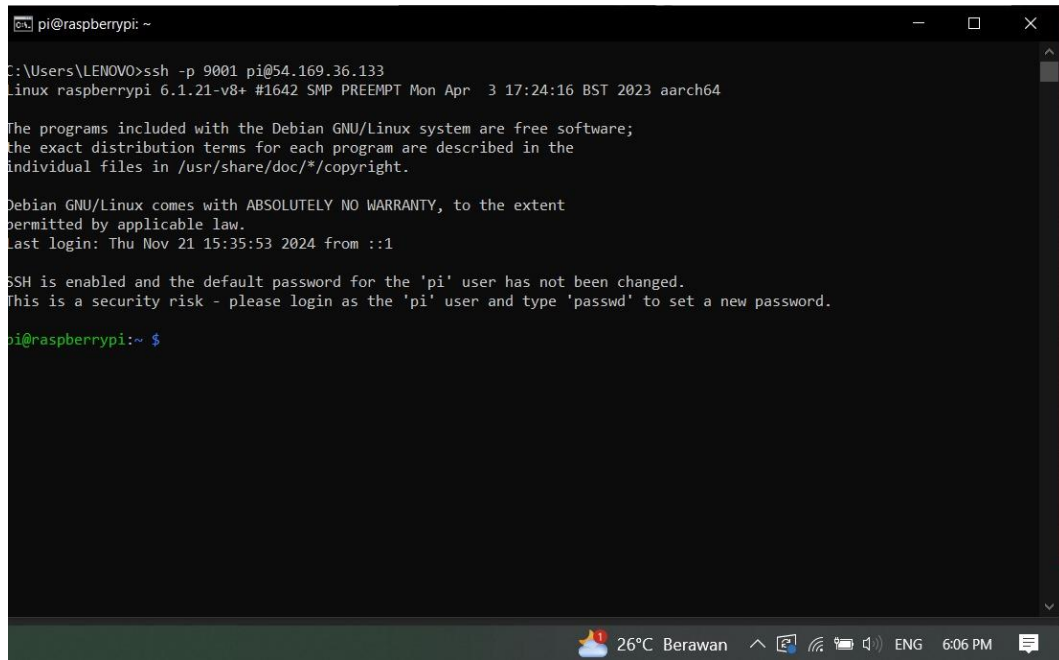


Gambar 6.5 Implementasi Raspberry Pi 4 dan *Peripheral*

6.1.5. Pengujian Akses Jarak Jauh Pada Raspberry Pi 4.

Pengujian akses jarak jauh pada Raspberry Pi 4 dilakukan untuk memastikan perangkat dapat dikelola dari jarak jauh menggunakan SSH dengan Bore Client. Berikut adalah langkah-langkah untuk melakukan pengujian ini:

- Pastikan Raspberry Pi 4 terpasang dan terhubung dengan internet, baik melalui koneksi Wi-Fi atau kabel Ethernet.
- Pastikan Raspberry Pi 4 sudah mengaktifkan layanan SSH.
- Pastikan Bore Client sudah terinstal pada Raspberry Pi 4. Anda bisa mengunduh dan menginstalnya menggunakan perintah yang sesuai untuk Raspberry Pi
- Setelah Bore Client terinstal, lakukan konfigurasi untuk membuat tunnel atau jalur akses ke Raspberry Pi 4 melalui internet. Pastikan konfigurasi sudah benar dan koneksi dapat dibuat.
- Gunakan SSH dari perangkat lain untuk mengakses Raspberry Pi 4 melalui Bore Client. Masukkan alamat IP atau hostname Raspberry Pi dan pastikan dapat melakukan login secara remote.
- Setelah login berhasil, lakukan beberapa operasi dasar seperti menjalankan perintah sistem ntuk memastikan akses jarak jauh berjalan dengan baik.



Gambar 6.6 Hasil Akses Jarak Jauh Raspberry pi 4 Melalui PC

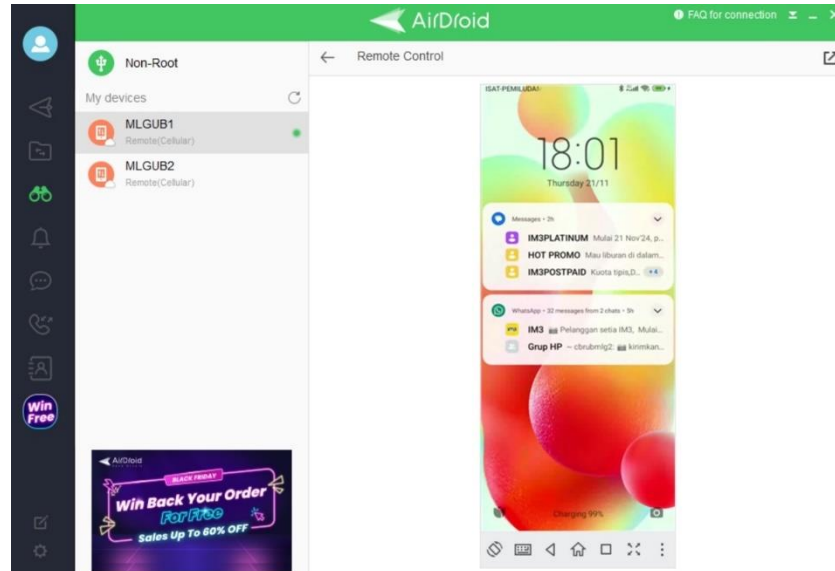
Gambar 6.6 ini menunjukkan hasil akses jarak jauh ke Raspberry Pi 4 melalui Bore Client dan SSH. Tampilan terminal menunjukkan bahwa perintah sistem dapat dijalankan, menandakan koneksi remote berhasil dan Raspberry Pi 4 dapat dikelola dari jarak jauh. Hal ini memastikan perangkat dapat dioperasikan dan dipelihara tanpa perlu berada di lokasi fisik.

6.1.6. Pengujian Akses Jarak Jauh Pada Smartphone Android.

Pengujian akses jarak jauh pada smartphone Android dilakukan untuk memastikan perangkat dapat dikelola secara remote melalui AirDroid. Berikut adalah langkah-langkah untuk melakukan pengujian ini:

- Pastikan smartphone Android dalam keadaan menyala dan terhubung ke internet melalui Wi-Fi atau data seluler.
- Unduh dan instal AirDroid dari Google Play Store di perangkat Android yang akan diuji.
- Buka aplikasi AirDroid dan buat akun jika diperlukan. Pastikan aplikasi terhubung dengan jaringan yang sama untuk pengaturan awal dan lakukan konfigurasi yang diperlukan, seperti pengaturan akses jarak jauh.
- Gunakan perangkat lain (misalnya, komputer atau smartphone lain) untuk mengakses perangkat Android melalui aplikasi AirDroid.

- Setelah terhubung, uji berbagai fitur yang disediakan oleh AirDroid, seperti mengakses file, mengirim pemberitahuan, atau mengontrol aplikasi. Periksa apakah kontrol perangkat berjalan lancar dan responsif.



Gambar 6.7 Hasil Akses Jarak Jauh Smartphone Android Melalui PC

Gambar 6.7 ini menunjukkan hasil akses jarak jauh ke smartphone Android menggunakan aplikasi AirDroid. Pada tampilan ini, pengguna dapat melihat dan mengontrol perangkat Android dari jarak jauh melalui antarmuka aplikasi AirDroid di perangkat lain. Gambar ini menunjukkan pengelolaan perangkat Android secara remote, termasuk kemampuan untuk mengakses file, aplikasi, serta melakukan pengaturan tanpa harus berada di dekat perangkat fisik. Ini mempermudah pengguna untuk melakukan troubleshooting atau pemantauan perangkat Android secara efisien dari jarak jauh.

6.2 Analisis Hasil Pengujian

Pada bagian ini, dilakukan analisis hasil dari serangkaian pengujian yang telah dilakukan untuk menilai kinerja sistem secara keseluruhan. Pengujian yang dilakukan mencakup deteksi antara BLE beacon dengan scanner, pengiriman data telemetry ke server, dan pengujian akses jarak jauh pada perangkat Raspberry Pi 4 dan smartphone Android. Analisis ini bertujuan untuk mengidentifikasi apakah sistem berjalan sesuai dengan ekspektasi dan menemukan potensi masalah yang perlu diperbaiki.

6.2.1. Hasil Pengujian Deteksi BLE Beacon.

Pengujian ini bertujuan untuk mengevaluasi efektivitas perangkat *smartphone android* dan *raspberry pi 4* sebagai scanner dalam mendeteksi sinyal BLE (Bluetooth Low Energy) yang dipancarkan oleh perangkat Estimote BLE beacon. Beacon ini terpasang pada bus sekolah, sementara *smartphone* berperan sebagai alat deteksi dalam skenario simulasi dan pengujian.

Analisis difokuskan pada persentase keberhasilan deteksi, yang dihitung setiap minggu dengan membandingkan jumlah deteksi yang berhasil dilakukan oleh scanner dengan jumlah deteksi yang diharapkan. Jumlah deteksi yang diharapkan dihitung berdasarkan jarak dan waktu yang telah ditentukan untuk setiap skenario pengujian. Keberhasilan deteksi dihitung dengan menggunakan rumus nomer 1 berikut:

Rumus ini diterapkan untuk setiap *RSU*, seperti Ijen, MT Haryono, Ijen Utara, Suhat, Tlogomas, Tugu, dan Veteran. Setiap *RSU* memiliki target jumlah deteksi setiap minggu yang berbeda berdasarkan skenario, yaitu:

- Ijen = 10
- MT Haryono = 15
- Ijen Utara = 5
- Suhat = 15
- Tlogomas = 10
- Tugu = 10
- Veteran = 5

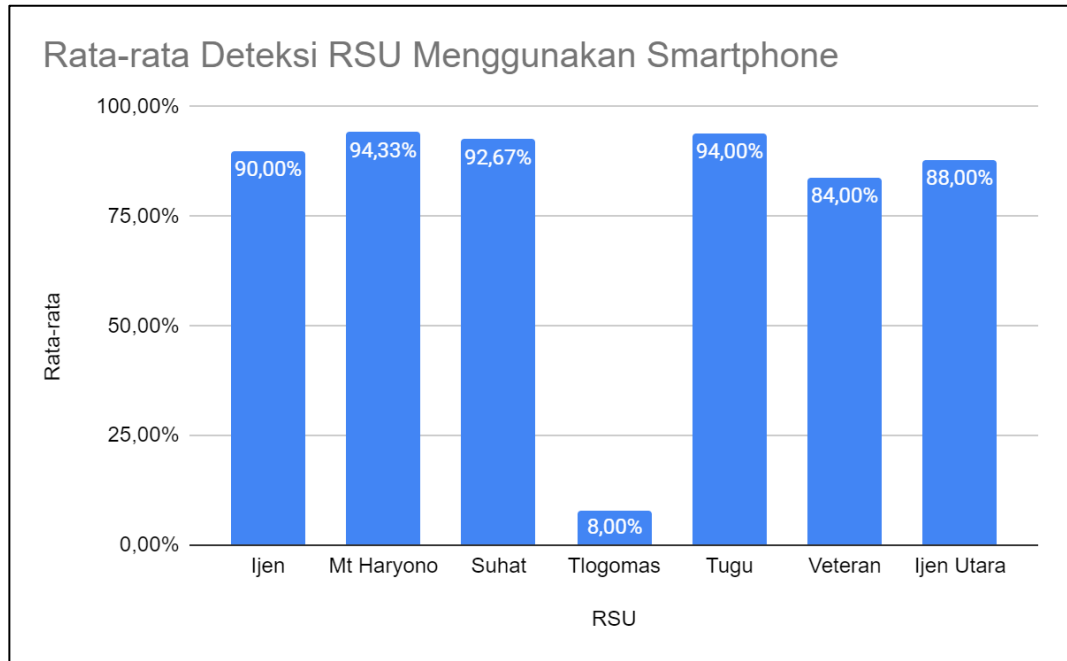
Dengan menggunakan rumus di atas dan nilai jumlah deteksi yang diharapkan untuk masing-masing *scanner smartphone android* dan *raspberry pi 4*, tabel berikut menyajikan persentase pencapaian deteksi beacon pada setiap lokasi *RSU* selama periode waktu tertentu, yaitu 9 September - 4 Oktober 2024 dan hingga 28 Oktober- 31 Oktober 2024.

Setiap nilai persentase dalam tabel menunjukkan tingkat pencapaian deteksi yang tercatat dibandingkan dengan jumlah deteksi yang diharapkan untuk masing-masing *RSU*. Nilai 100% berarti target deteksi telah tercapai sepenuhnya, sedangkan nilai di bawah 100% menunjukkan bahwa target belum terpenuhi.

Tabel 6.1 memberikan gambaran rinci mengenai performa scanner *smartphone Android* dalam mendeteksi sinyal BLE beacon, sekaligus menunjukkan efektivitas sistem deteksi selama jangka waktu pengujian di atas.

Tabel 6.1 Performa Deteksi *BLE Beacon* dengan *Smartphone*

Tanggal Pengujian	Ijen	Mt Haryono	Suhat	Tlogomas	Tugu	Veteran	Ijen Utara
30 September - 4 Oktober 2024	100,00%	100,00%	100,00%	0,00%	100,00%	60,00%	100,00%
7 Oktober - 11 Oktober 2024	100,00%	100,00%	80,00%	20,00%	100,00%	100,00%	100,00%
14 Oktober - 18 Oktober 2024	80,00%	80,00%	100,00%	20,00%	100,00%	80,00%	80,00%
21 Oktober - 25 Oktober 2024	100,00%	100,00%	100,00%	0,00%	100,00%	100,00%	100,00%
28 Oktober - 31 Oktober 2024	70,00%	91,67%	83,33%	0,00%	70,00%	80,00%	60,00%
Rata-rata	90,00%	94,33%	92,67%	8,00%	94,00%	84,00%	88,00%



Gambar 6.8 Grafik Presentase Keberhasilan Scanning Android

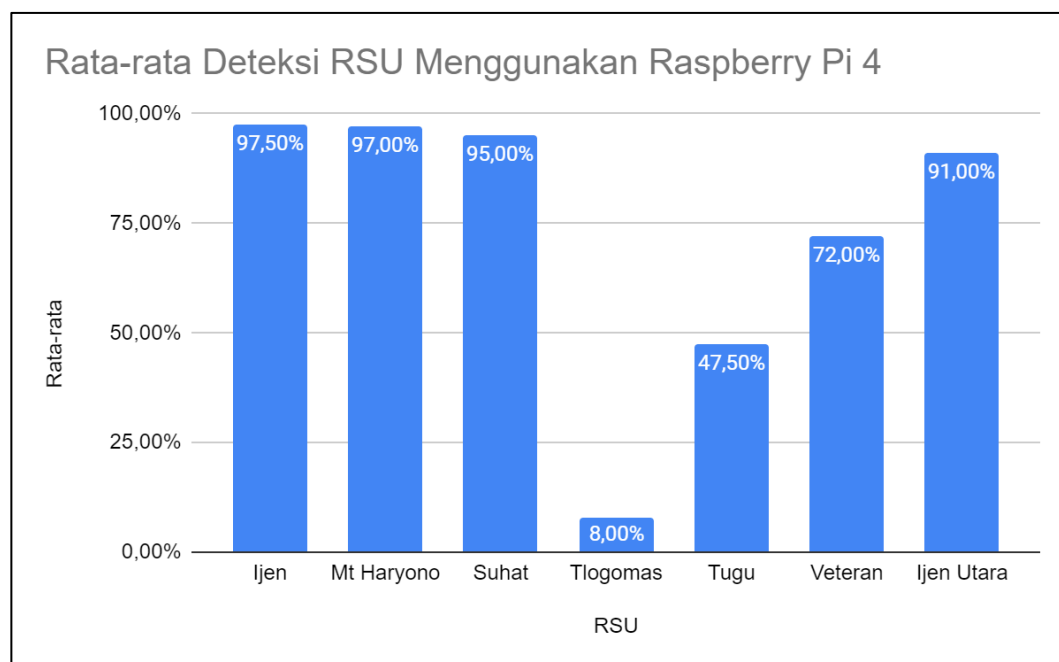
Tabel 6.1 dan gambar 6.8 memberikan kita informasi mengenai presentase deteksi *BLE beacon* menggunakan *smartphone android* selama 1 bulan. Secara keseluruhan deteksi *BLE beacon* sudah di atas 80% sesuai dengan harapan. Namun pada *RSU* Tlogomas performa pendeteksian *BLE beacon* masih sangat jauh di bawah harapan.

Selain performa deteksi *BLE beacon* menggunakan *smartphone* yang telah disajikan pada gambar dan tabel sebelumnya, berikut ini adalah performa deteksi *BLE beacon* menggunakan *Raspberry Pi* dalam rentang waktu yang sama.

Tabel 6.2 Performa Deteksi BLE Beacon dengan Raspberry Pi 4

Tanggal Pengujian	Ijen	Mt Haryono	Suhat	Tlogomas	Tugu	Veteran	Ijen Utara
30 September - 4 Oktober 2024	100,00%	100,00%	100,00%	0,00%	50,00%	100,00%	100,00%
7 Oktober - 11 Oktober 2024	100,00%	93,33%	100,00%	20,00%	50,00%	60,00%	80,00%

Tanggal Pengujian	Ijen	Mt Haryono	Suhat	Tlogomas	Tugu	Veteran	Ijen Utara
14 Oktober - 18 Oktober 2024	100,00%	100,00%	100,00%	20,00%	50,00%	0,00%	100,00%
21 Oktober - 25 Oktober 2024	100,00%	100,00%	100,00%	0,00%	50,00%	100,00%	100,00%
28 Oktober - 31 Oktober 2024	87,50%	91,67%	75,00%	0,00%	37,50%	100,00%	75,00%
Rata-rata	97,50%	97,00%	95,00%	8,00%	47,50%	72,00%	91,00%



Gambar 6.9 Grafik Presentase Keberhasilan Scanning Raspberry Pi 4

Tabel 6.2 dan gambar 6.9 memberikan kita informasi mengenai presentase deteksi *BLE beacon* menggunakan *raspberry pi 4* selama 1 bulan. Secara keseluruhan deteksi *BLE beacon* sudah di atas 75% sesuai dengan harapan. Namun pada *RSU Tlogomas* performa pendeteksian *BLE beacon* masih sangat jauh di

bawah harapan. Dan pada RSU Tugu presentase keberhasilan deteksi hanya dapat mencapai rata-rata 47,5% dikarenakan jarak antara *RSU* dengan bis pada sore hari lebih jauh dibandingkan dengan jarak *RSU* dengan bis pada pagi harinya.

Berdasarkan kedua grafik dan tabel di atas dapat disimpulkan bahwa *RSU* pada Sistem Pengumpulan Data Pelacakan Transportasi Umum Menggunakan *Bluetooth Proximity Beacons* sudah dapat mendeteksi *BLE beacon* yang terletak pada bus sekolah malang dengan baik dan memiliki akurasi deteksi yang tinggi. Namun masih ada beberapa penempatan *RSU* yang masih dapat ditingkatkan dengan beberapa cara seperti menambahkan komponen yang memperluas area deteksi *BLE beacon* atau dengan memindahkan *RSU* ke tempat yang lebih strategis untuk menjangkau *BLE beacon*.

6.2.2. Hasil Pengujian Keberhasilan Pengiriman Data Telemetry.

Pengujian keberhasilan pengiriman data telemetry bertujuan untuk mengukur seberapa efektif sistem dalam mengirimkan data dari RSU (Road Side Unit) ke server. Dalam analisis ini, persentase keberhasilan pengiriman dihitung dengan membandingkan jumlah data telemetry yang berhasil dikirimkan terhadap jumlah data yang direncanakan untuk dikirimkan dalam periode waktu tertentu.

Hasil pengujian menunjukkan bahwa persentase keberhasilan pengiriman data telemetry umumnya sangat baik, dengan tingkat keberhasilan pengiriman yang tinggi setiap hari. Namun, terdapat beberapa kendala yang memengaruhi keberhasilan pengiriman, seperti gangguan pada koneksi internet atau kesalahan autentikasi API yang menyebabkan beberapa pengiriman data gagal.

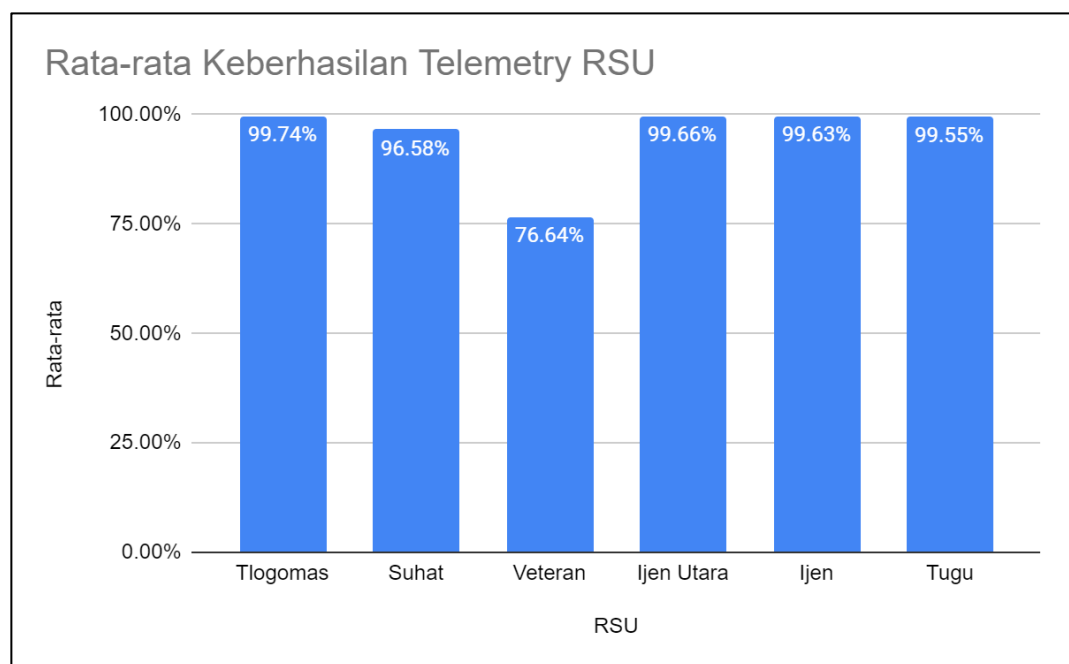
Untuk menghitung persentase pencapaian pengiriman setiap RSU, rumus yang digunakan adalah rumus 2: u

Dalam konteks ini, jika jumlah data yang diharapkan adalah 5412 setiap harinya. Maka akan di bawah menyajikan persentase keberhasilan pengiriman data telemetry dari masing-masing RSU ke server untuk beberapa periode waktu satu hari mulai dari jam 3:59 hingga 19:01 dengan delay 10 detik setiap pengiriman. Setiap nilai persentase dihitung berdasarkan jumlah data telemetry yang berhasil dikirimkan dalam periode tersebut dibandingkan dengan jumlah data yang direncanakan untuk dikirimkan.

Tabel 6.3 Presentase Keberhasilan Pengiriman *Telemetry* Tiap *RSU*

Tanggal	Tlogomas	Suhat	Veteran	Ijen Utara	Ijen	Tugu
30/9/2024	99.87%	99.76%	93.26%	99.83%	99.91%	99.83%
1/10/2024	99.02%	99.69%	92.41%	99.70%	99.76%	99.76%
2/10/2024	99.94%	99.93%	93.22%	99.89%	99.89%	99.87%
3/10/2024	99.82%	99.91%	93.14%	99.83%	99.78%	99.74%
4/10/2024	99.94%	99.93%	93.22%	99.89%	98.04%	99.76%
7/10/2024	99.69%	99.93%	93.13%	99.63%	99.67%	99.65%
8/10/2024	99.87%	99.76%	93.24%	99.89%	99.87%	99.33%
9/10/2024	99.91%	99.93%	93.18%	99.96%	99.93%	99.83%
10/10/2024	99.89%	99.87%	0.00%	99.32%	99.89%	99.91%
11/10/2024	99.83%	99.83%	0.00%	99.78%	99.87%	99.82%
14/10/2024	99.94%	99.91%	0.00%	99.87%	99.89%	99.89%
15/10/2024	98.97%	98.97%	0.00%	98.89%	99.06%	98.78%
16/10/2024	99.91%	99.89%	0.00%	99.94%	99.93%	98.76%
17/10/2024	99.91%	99.91%	0.00%	99.96%	99.91%	99.85%
18/10/2024	99.93%	99.11%	76.77%	99.89%	99.89%	99.87%
21/10/2024	99.87%	99.82%	93.16%	99.72%	99.82%	99.83%
22/10/2024	99.91%	99.91%	93.20%	99.85%	99.91%	99.85%
23/10/2024	99.85%	99.83%	93.22%	99.91%	99.89%	99.87%
24/10/2024	98.97%	99.06%	92.61%	98.43%	99.02%	99.13%
25/10/2024	99.94%	99.89%	93.24%	99.91%	99.91%	99.08%
28/10/2024	99.91%	99.83%	93.26%	99.85%	99.30%	99.87%
29/10/2024	99.93%	89.89%	93.29%	99.35%	99.89%	99.87%
30/10/2024	99.91%	99.85%	93.26%	99.85%	99.91%	99.87%

Tanggal	Tlogomas	Suhat	Veteran	Ijen Utara	Ijen	Tugu
31/10/2024	99.89%	99.93%	93.22%	99.08%	98.13%	95.33%
1/11/2024	99.91%	99.93%	93.16%	99.41%	99.85%	99.87%
4/11/2024	98.82%	0.00%	92.11%	100.02%	99.91%	99.89%
5/11/2024	99.91%	99.89%	93.22%	100.02%	99.91%	99.56%
6/11/2024	99.93%	98.84%	91.50%	98.91%	98.76%	98.74%
7/11/2024	99.91%	99.91%	93.24%	100.02%	99.93%	99.87%
8/11/2024	99.72%	99.61%	93.14%	99.78%	99.85%	99.80%
11/11/2024	99.91%	99.89%	93.24%	99.89%	99.89%	99.98%
12/11/2024	99.54%	99.63%	93.14%	99.52%	99.63%	99.54%
13/11/2024	99.89%	99.83%	93.22%	99.89%	99.87%	99.87%
14/11/2024	98.76%	99.82%	93.24%	98.73%	99.76%	99.76%
15/11/2024	99.91%	98.73%	93.22%	99.83%	98.76%	99.85%
Rata-rata	99.74%	96.58%	76.64%	99.66%	99.63%	99.55%

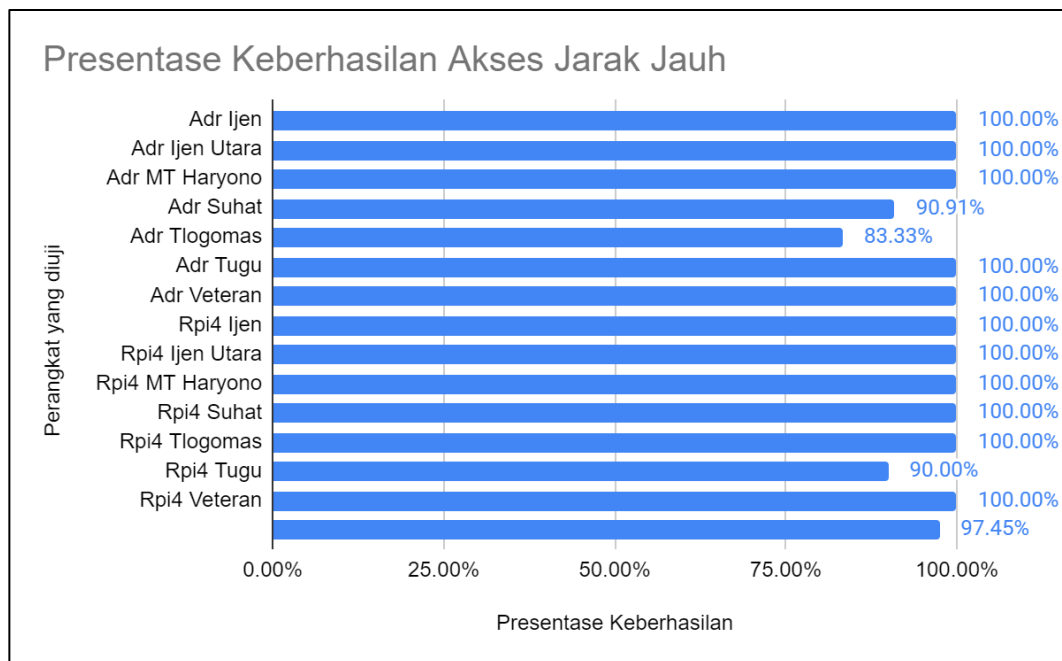


Gambar 6.10 Grafik Presentase Keberhasilan Telemetry Tiap RSU

Tabel 6.3 dan gambar 6.10 memperlihatkan data presentase keberhasilan terhadap telemetry yang dikirimkan dari tiap-tiap *RSU* ke *server*. Data tersebut memperlihatkan bahwa presentase kesuksesan hampir semua *RSU* mencapai lebih dari 96%. Namun pada *RSU Veteran* masih terdapat beberapa kendala yang menyebabkan keberhasilan telemetry hanya mencapai 76,6%. Kendala itu kemungkinan adalah putusnya internet dari *smartphone android* sehingga menyebabkan *raspberry pi* tidak dapat mengirimkan *telemetry* ke *server*.

6.2.3. Hasil Pengujian Keberhasilan Akses Jarak Jauh.

Pengujian keberhasilan akses jarak jauh dilakukan untuk mengevaluasi stabilitas dan keandalan koneksi jarak jauh yang memungkinkan pengelolaan *RSU* dari lokasi terpencil. Persentase keberhasilan akses jarak jauh dihitung berdasarkan jumlah upaya akses yang berhasil dibandingkan dengan total upaya yang dilakukan pada periode waktu tertentu (pagi, siang, dan sore). Upaya dihitung dengan rumus nomer 3.



Gambar 6.11 Grafik Presentase Keberhasilan Akses Jarak Jauh

Berdasarkan gambar 6.11, hasil pengujian menunjukkan bahwa tingkat keberhasilan akses jarak jauh sangat tinggi pada sebagian besar perangkat yang diuji. Perangkat seperti *Android Ijen*, *Android Ijen Utara*, *Android MT Haryono*, *Android Tugu*, *Android Veteran*, serta hampir semua perangkat *Raspberry Pi 4*

(Rpi4) mencapai keberhasilan akses 100%. Namun, ada beberapa pengecualian, seperti *Android* Tlogomas (83,33%), *Android* Suhat (90,91%), dan Rpi4 Tlogomas (90%), yang menunjukkan tingkat keberhasilan sedikit lebih rendah.

Akses jarak jauh dilakukan menggunakan Bore Client pada Raspberry Pi 4 melalui SSH dan AirDroid pada smartphone Android. Meskipun terdapat gangguan pada jaringan internet selama pengujian, koneksi tetap stabil, mendukung keberhasilan akses yang dominan di semua perangkat.

BAB VII PENUTUP

Pada bab ini akan dijelaskan kesimpulan dari pengujian yang sudah dilakukan pada bab 6. Kesimpulan yang diberikan akan menjawab pertanyaan-pertanyaan yang disampaikan pada rumusan masalah penelitian ini. Akan dibahas juga mengenai saran untuk penelitian kedepan beserta kemungkinan solusi untuk permasalahan baru yang ditemui pada pengujian penelitian ini.

7.1 Kesimpulan

Dari penelitian yang telah dilakukan sebelumnya dapat ditarik beberapa kesimpulan berdasarkan rumusan masalah yang sebelumnya telah ditentukan. Kesimpulan yang didapatkan adalah,

- Berdasarkan rumusan masalah pertama, mengenai performa Raspberry Pi 4 dalam melakukan pendeteksian beacon, hasil pengujian menunjukkan bahwa Raspberry Pi 4 berhasil mendeteksi beacon dengan persentase 70% - 97% pada lokasi-lokasi RSU yang diuji kecuali pada RSU Tlogomas dan Veteran. Hal ini menunjukkan pendeteksian sudah bekerja dengan optimal di sebagian besar tempat. Namun masih ada beberapa RSU yang memiliki performa yang tidak sesuai harapan. Meskipun demikian, Raspberry pi tetap memberikan kontribusi signifikan dalam meningkatkan keandalan deteksi beacon di lokasi-lokasi lain yang diuji, serta berfungsi sebagai alat deteksi tambahan yang mendukung sistem secara keseluruhan.
- Berdasarkan rumusan masalah kedua, mengenai performa smartphone Android dalam melakukan pendeteksian beacon, hasil pengujian menunjukkan bahwa smartphone Android berhasil mendeteksi beacon dengan persentase keberhasilan 80% - 90% kecuali pada RSU Tlogomas. Rendahnya tingkat deteksi di lokasi RSU Tlogomas kemungkinan disebabkan oleh perubahan rute bus yang tidak sesuai dengan area pemindaian beacon yang diharapkan. Namun performa ini masih lebih baik apabila dibandingkan dengan scanner raspberry pi 4.
- Berdasarkan rumusan masalah ketiga, mengenai tingkat keberhasilan akses perangkat RSU secara jarak jauh, pengujian menunjukkan tingkat keberhasilan akses perangkat melalui SSH dan AirDroid mencapai 80% - 100%. Seluruh perangkat RSU dapat diakses dan dikelola secara jarak jauh tanpa hambatan yang berarti, Beberapa RSU tidak dapat dijangkau pada percobaan pertama namun berhasil saat percobaan kedua atau ketiga.
- Berdasarkan rumusan masalah keempat, mengenai tingkat keberhasilan pengiriman data telemetry ke server, pengujian menunjukkan bahwa

pengiriman data telemetry berhasil dilakukan dengan persentase rata-rata mencapai 76 hingga 99 persen. Meskipun terdapat beberapa periode di mana data telemetry tidak terkirim selama sehari penuh, sistem tetap dapat melakukan perbaikan diri dan melanjutkan pengiriman data dengan normal.

7.2 Saran

Berdasarkan hasil pengujian, disarankan untuk melakukan penyesuaian pada skenario pemindaian beacon, terutama di lokasi dengan perubahan rute bus seperti Tlogomas. Hal ini dapat dilakukan dengan memperbarui area pemindaian beacon dan menggunakan algoritma yang lebih efisien dalam mengukur jarak dan kekuatan sinyal, untuk meningkatkan akurasi deteksi. Selain itu, untuk mengatasi gangguan dalam pengiriman data telemetry, disarankan untuk menambahkan mekanisme pemulihan otomatis atau notifikasi kegagalan pengiriman, sehingga masalah dapat segera ditangani tanpa mengganggu proses sistem secara keseluruhan.

Untuk meningkatkan keamanan dan stabilitas akses jarak jauh, penting untuk memperkuat sistem autentikasi dan enkripsi komunikasi, misalnya dengan menggunakan VPN atau autentikasi dua faktor. Selain itu, pemeliharaan dan pemantauan sistem secara rutin sangat penting untuk mendeteksi masalah lebih dini. Penerapan sistem pelaporan otomatis terkait status perangkat dan hasil deteksi beacon juga akan mempermudah pengelolaan dan pengawasan sistem, sehingga dapat menjaga kinerja dan keandalan perangkat dalam jangka panjang.

Daftar Referensi

- Chong, Y. W., Yau, K. L. A., Ibrahim, N. F., Rahim, S. K. A., Keoh, S. L., & Basuki, A. (2024). Federated Learning for Intelligent Transportation Systems: Use Cases, Open Challenges, and Opportunities. *IEEE Intelligent Transportation Systems Magazine*. <https://doi.org/10.1109/MITS.2024.3451479>
- Elijah, O., Keoh, S. L., bin Abdul Rahim, S. K., Seow, C. K., Cao, Q., bin Sarijari, M. A., Ibrahim, N. F., & Basuki, A. (2023). Transforming urban mobility with internet of things: public bus fleet tracking using proximity-based bluetooth beacons. *Frontiers in the Internet of Things*, 2. <https://doi.org/10.3389/friot.2023.1255995>
- Godge, P., Gore, K., Gore, A., Jadhav, A., & Nawathe, A. (2019). *Smart Bus Management and Tracking System*. www.ijlesjournal.org
- Jimoh, O. D., Ajao, L. A., Adeleke, O. O., & Kolo, S. S. (2020). A Vehicle Tracking System Using Greedy Forwarding Algorithms for Public Transportation in Urban Arterial. *IEEE Access*, 8, 191706–191725. <https://doi.org/10.1109/ACCESS.2020.3031488>
- Kassim, M., Salleh, A. S., Shahbudin, S., Yusoff, M., & Kamaluddin, N. A. (2022). IoT Bus Tracking System Localization via GPS-RFID. *2022 IEEE International Conference in Power Engineering Application, ICPEA 2022 - Proceedings*. <https://doi.org/10.1109/ICPEA53519.2022.9744710>
- Moneer, M., Aljawarneh, M. M., Das Dhomeja, L., Laghari, G., & Memon, B. R. (2020). *An Indoor Tracking System using iBeacon and Android*. 4(2).
- Mutiawati, C. (2019). *Kinerja Pelayanan Angkutan Umum Jalan Raya*.
- Sadhu, P. K., Yanambaka, V. P., & Abdelgawad, A. (2022). Internet of Things: Security and Solutions Survey. In *Sensors* (Vol. 22, Issue 19). MDPI. <https://doi.org/10.3390/s22197433>
- Vargas, A. P., Valdez-Martinez, J. S., Beltran-Escobar, A. M., Villanueva-Tavira, J., Lopez-Vega, L. J., Alcala-Barojas, I., Calderon, E. C., & Arias, H. M. B. (2021). Design and Development of a sustainable telemetry system for environmental parameters. *Proceedings - 2021 International Conference on Mechatronics, Electronics and Automotive Engineering, ICMEAE 2021*, 262–267. <https://doi.org/10.1109/ICMEAE55138.2021.00049>