BAB 2
TINJAUAN PUSTAKA

2.1 Landasan Teori

Pada penelitian ini, landasan teori digunakan sebagai dasar dalam memahami konsep *Internet of Things* (IoT), protokol MQTT, broker MQTT, *Virtual Machine*, serta pengaruh spesifikasi perangkat keras seperti CPU dan RAM terhadap kinerja sistem komunikasi data. Teori-teori yang dibahas diharapkan dapat mendukung proses analisis dan pengujian yang dilakukan dalam penelitian ini.

2.1.1 Internet of Things (IoT)

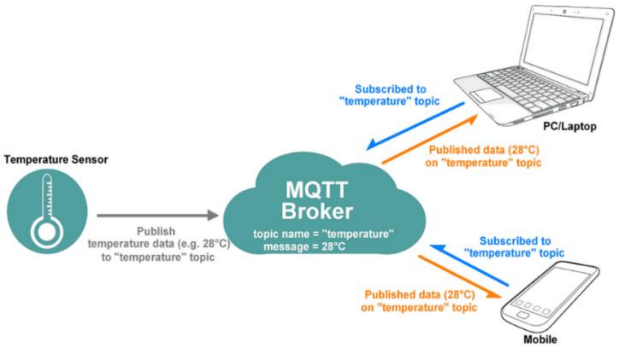
Internet of Things (IoT) adalah konsep yang memungkinkan berbagai perangkat fisik terhubung dan berkomunikasi melalui jaringan internet. Setiap perangkat yang terhubung memiliki kemampuan untuk mengumpulkan, mengirim, dan menerima data, sehingga memungkinkan otomatisasi dan kontrol jarak jauh dalam berbagai bidang seperti industri, kesehatan, dan lingkungan. IoT bertujuan memperluas manfaat konektivitas internet dengan menghubungkan mesin, peralatan, dan objek fisik lainnya melalui sensor jaringan dan aktuator, memungkinkan perangkat untuk mengelola kinerjanya sendiri dan bertindak berdasarkan informasi baru secara independen. Perangkat IoT biasanya terdiri dari sensor untuk mengumpulkan data, koneksi internet sebagai media komunikasi, dan *server* untuk menyimpan serta menganalisis informasi yang diterima dari sensor. Gagasan IoT pertama kali diperkenalkan oleh Kevin Ashton pada tahun 1999, dan kini banyak perusahaan besar seperti Intel, Microsoft, dan Oracle mengembangkannya (Efendi, 2018).

Gambar 2. 1 *Internet ofThings*
Sumber: (SolutionsHub, 2021)

Gambar 2. 1 menampilkan konsep Internet of Things yang menghubungkan berbagai perangkat fisik seperti sensor, aktuator, dan sistem komputasi melalui jaringan internet untuk mengumpulkan dan bertukar data secara otomatis.

2.1.2 Protokol Message Queuing Telemetry Transport (MQTT)

MQTT adalah protokol komunikasi yang dirancang untuk konektivitas *machine-to-machine* (M2M) dan sering digunakan dalam *Internet of Things* (IoT). Protokol ini menggunakan model komunikasi *publish-subscribe*, di mana perangkat yang berperan sebagai *Publisher* mengirimkan pesan ke broker, dan perangkat *Subscriber* menerima pesan dari broker berdasarkan topik yang diminati. Keunggulan MQTT terletak pada efisiensinya dalam penggunaan *bandwidth* dan kemampuannya beroperasi dengan konsumsi daya yang rendah, sehingga cocok untuk perangkat dengan *resource* terbatas (Susanto et al., 2018).



Gambar 2. 2 MQTT
Sumber:(ELMECH, 2024)

Gambar 2. 2 menunjukkan cara kerja protokol MQTT dengan model komunikasi publish-subscribe antara publisher, broker, dan subscriber.

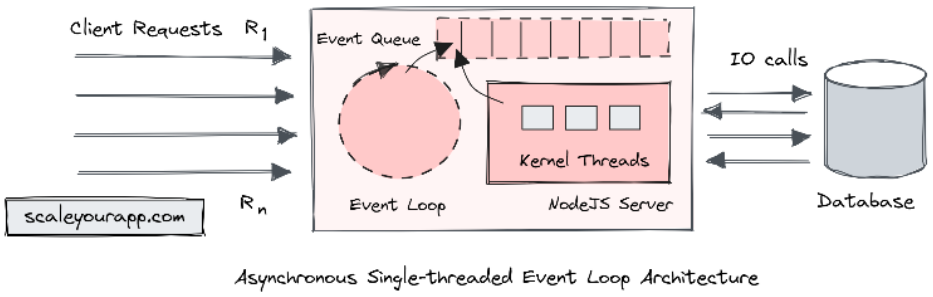
2.1.3 Broker MQTT

Broker MQTT (*Message Queuing Telemetry Transport*) merupakan komponen kunci dalam arsitektur komunikasi berbasis protokol MQTT yang berfungsi sebagai penghubung antara perangkat (*Publisher* dan *Subscriber*) dalam sistem *Internet of Things* (IoT). Broker bertanggung jawab untuk menerima, memfilter, dan mendistribusikan pesan yang dikirimkan oleh *Publisher* kepada *Subscriber* berdasarkan topik yang telah ditentukan. Contoh broker MQTT yang umum digunakan adalah Mosquitto. Protokol ini dirancang untuk beroperasi dengan *bandwidth* rendah dan *resource* terbatas, menjadikannya ideal untuk aplikasi IoT yang memerlukan pengiriman data secara *real time* dengan *latency* minimal (Kurnianto et al., 2022).

Untuk mendukung implementasi arsitektur komunikasi tersebut, saat ini terdapat berbagai pilihan perangkat lunak broker MQTT yang dapat digunakan. Berikut adalah beberapa broker MQTT berbasis *open-source* yang populer dan sering diterapkan dalam sistem IoT:

2.1.3.1 Eclipse Mosquitto

Eclipse Mosquitto adalah perangkat lunak *open-source* yang dikembangkan di bawah naungan Eclipse Foundation menggunakan bahasa pemrograman C. Broker ini dikenal luas sebagai solusi *lightweight messaging* karena mengimplementasikan standar protokol MQTT secara efisien untuk perangkat bersumber daya rendah.



Gambar 2. 3 Diagram Arsitektur *Single-Threaded* atau *Event Loop*.
Sumber:(shivang, n.d.)

Berdasarkan Gambar 2.3 sisi arsitektur, Mosquitto dirancang menggunakan pendekatan *single-threaded* berbasis *event loop*. Mekanisme *event loop* ini bekerja

dengan cara memantau seluruh koneksi secara terus-menerus di dalam satu siklus utas (*thread*) utama. Seluruh proses krusial, mulai dari penanganan koneksi client, autentikasi, perutean (*routing*), hingga distribusi pesan, dieksekusi secara sekuensial (berurutan). Keuntungan utama dari arsitektur *single-threaded* ini adalah ketiadaan beban komputasi tambahan akibat perpindahan tugas (*context switching*) antar-*thread* maupun mekanisme sinkronisasi data yang rumit. Hal ini menjadikan konsumsi CPU dan RAM pada Mosquitto sangat minim dan stabil pada lalu lintas data ringan hingga menengah.

Namun, secara teoritis, arsitektur tunggal ini memiliki keterbatasan (*bottleneck*) ketika dihadapkan pada lalu lintas pesan yang sangat tinggi (*high load*). Karena semua pesan baru harus mengantre untuk diproses di satu jalur yang sama, penumpukan beban pemrosesan dapat terjadi. Antrean panjang inilah yang berpotensi menyebabkan lonjakan waktu tunda (*latency*) secara signifikan, meskipun secara kapasitas, sumber daya perangkat keras (terutama multicore CPU) masih tersedia dan belum terpakai (Light, 2017).

2.1.3.2 EMQX

EMQX (Erlang MQTT Broker) merupakan broker MQTT *open-source* kelas *enterprise* yang dibangun menggunakan bahasa pemrograman Erlang/OTP. Erlang diciptakan secara spesifik untuk membangun sistem telekomunikasi terdistribusi yang menuntut konkurensi tingkat tinggi (*massive concurrency*) dan toleransi kesalahan (*fault tolerance*).

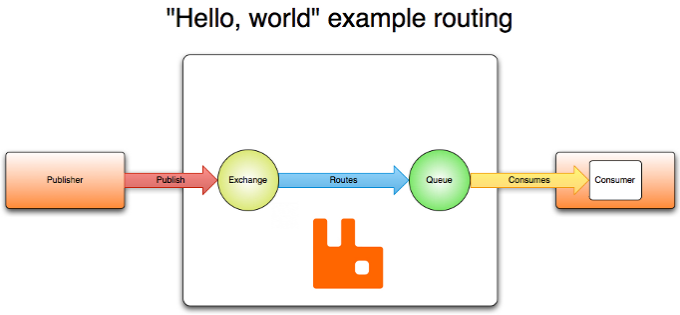
EMQX memanfaatkan mekanisme *lightweight* processes dan *multi-scheduler* dari Erlang, yang dipadukan dengan operasi I/O asinkron (*Asynchronous Input/Output*) secara bawaan. Pendekatan I/O asinkron ini memungkinkan EMQX untuk memproses operasi jaringan seperti menerima dan mengirim pesan dari ribuan perangkat klien tanpa harus memblokir (*non-blocking*) proses komputasi lainnya yang sedang berjalan. Kemampuan mendistribusikan beban I/O secara asinkron ke seluruh core CPU ini membuat EMQX mampu menangani jutaan koneksi secara bersamaan. Berdasarkan hasil evaluasi (Bender et al., 2021), EMQX juga memiliki tingkat kepatuhan standar yang sangat tinggi, terbukti dengan keberhasilannya memenuhi seluruh pengujian pada MQTT 3.1.1 *Conformance Test*.

Sebagai solusi *enterprise*, EMQX tidak hanya berfungsi sebagai perute pesan, tetapi juga dilengkapi dengan berbagai subsistem bawaan yang beroperasi secara otomatis di latar belakang (*background processes*). Subsistem ini mencakup pangkalan data internal terdistribusi (*Mnesia*), agen metrik pengukuran, manajemen sesi yang kompleks, hingga *dashboard monitoring* berbasis web. Keberadaan sistem internal ini menuntut alokasi memori dasar (RAM) yang jauh lebih besar dibandingkan broker ringan sejak pertama kali dijalankan. Meskipun konsumsi sumber daya awalnya tinggi, kompensasinya adalah EMQX mampu menyajikan kinerja *throughput* dan *latency* yang sangat stabil berkat manajemen I/O asinkronnya ketika memproses lalu lintas pesan berskala masif.

2.1.3.3 RabbitMQ

RabbitMQ pada dasarnya adalah message broker multiguna berbasis Erlang/OTP yang arsitektur intinya dibangun berlandaskan protokol AMQP (*Advanced Message Queuing Protocol*). Untuk dapat melayani komunikasi perangkat IoT yang menggunakan MQTT, RabbitMQ tidak memprosesnya secara langsung (*native*), melainkan menerjemahkan pesan MQTT tersebut ke dalam struktur internal AMQP melalui sebuah modul *plugin* tambahan. Secara teoritis, proses adaptasi dan penerjemahan protokol (*protocol translation overhead*) ini membebankan siklus komputasi ekstra pada CPU. Akibatnya, pada skenario pengujian dengan lalu lintas data yang sangat ekstrem, RabbitMQ cenderung lebih cepat mencapai ambang batas kemampuannya (*crashout*) dibandingkan broker MQTT murni.

Di balik *overhead* tersebut, arsitektur AMQP memberikan keunggulan teknis yang signifikan melalui model perutean berbasis *Exchange* dan *Message Queue*. Dalam model ini, RabbitMQ secara tegas memisahkan komponen yang bertugas menerima data dari pengirim (*producer*) dan komponen yang bertugas menyalurkan antrean data ke penerima (*consumer*). Alur kerja perutean dari *Publisher* hingga *Consumer* ini diilustrasikan secara detail pada Gambar 2.4.



Gambar 2. 4 Diagram Arsitektur AMQP (*Exchange ke Message Queue*).
Sumber:(RabbitMQ, 2026)

Pemisahan alur kerja inilah yang membuat RabbitMQ sangat tangguh dalam menangani pemrosesan pesan secara asinkron. Pendekatan ini efektif mencegah terjadinya tabrakan data (data *collision*) pada saat lalu lintas jaringan sedang padat, sekaligus menjaga nilai latency berada di tingkat yang sangat rendah, menjadikannya pilihan ideal untuk aplikasi real-time yang responsif (Madhu M P, 2019)

2.1.3.4 VerneMQ

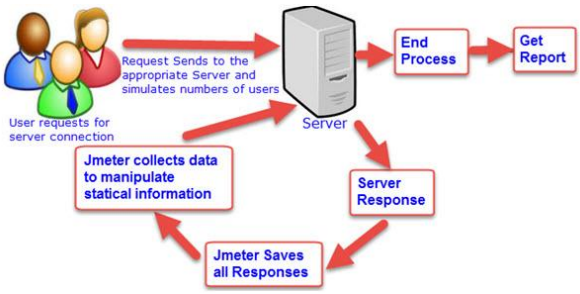
VerneMQ merupakan broker MQTT *open-source* yang fondasi arsitekturnya dikembangkan menggunakan bahasa pemrograman Erlang/OTP. Perbedaan mendasarnya dengan EMQX terletak pada filosofi desain, VerneMQ dirancang jauh lebih minimalis dan ramping dengan fokus utama memecahkan masalah skalabilitas horizontal serta ketersediaan layanan (*high availability*). Melalui pendekatan arsitektur terdistribusi yang efisien, VerneMQ sangat mendukung penerapan clustering (penggabungan beberapa server menjadi satu kesatuan broker). Menurut evaluasi yang ada (Koziolek et al., n.d.), desain spesifik ini menjadikan VerneMQ mampu menangani komunikasi jutaan perangkat secara paralel dengan ketahanan (*reliability*) yang superior dibandingkan protokol perpesanan standar.

Karena berjalan di atas runtime mesin virtual Erlang (BEAM), alokasi dan de-alokasi memori pada VerneMQ diatur secara dinamis oleh sistem mekanis yang disebut *Garbage Collection* (GC). *Garbage collection* adalah algoritma otomatis yang bertugas memindai dan membersihkan ruang RAM dari objek atau data sisa pemrosesan pesan yang tidak lagi memiliki referensi aktif. Secara teoritis, apabila

VerneMQ dioperasikan pada *server* dengan kapasitas RAM yang sangat terbatas, mesin virtual akan dipaksa melakukan proses *garbage collection* dengan frekuensi yang jauh lebih tinggi demi mencegah sistem kehabisan memori (*Out of Memory*). Siklus pembersihan yang terlalu agresif ini dapat menyita utilisasi CPU secara signifikan dan menyebabkan penundaan sementara pada perutean pesan, yang pada akhirnya memicu penurunan throughput. Namun, apabila disediakan sumber daya yang memadai, broker ini akan menunjukkan performa konektivitas yang luar biasa stabil.

2.1.4 Apache JMeter

Apache JMeter merupakan perangkat lunak pengujian kinerja (*load testing*) yang bersifat *open-source* dan dikembangkan oleh Apache Software Foundation. JMeter digunakan untuk mensimulasikan sejumlah pengguna atau permintaan secara bersamaan guna mengevaluasi kinerja, stabilitas, dan respons sistem yang diuji. Selain mendukung protokol HTTP untuk aplikasi web, JMeter juga dapat digunakan untuk menguji berbagai layanan dan protokol lainnya, seperti HTTPS, MQTT, TCP, JMS, dan *Web Services*. Pengaturan skenario pengujian dilakukan melalui konfigurasi *Thread Group* yang memungkinkan penentuan jumlah pengguna *virtual*, pola akses, serta durasi pengujian. Dengan kemampuan tersebut, JMeter dapat menghasilkan berbagai metrik kinerja, seperti waktu respons (*latency*), *throughput*, tingkat kesalahan (*error rate*), serta penggunaan *resource* sistem selama proses pengujian berlangsung (Indrianto, 2023).



Gambar 2. 5 Arsitektur Apache JMeter
Sumber:(Software Testing Class, 2023)

Berdasarkan Gambar 2.3, diagram tersebut menampilkan alur kerja Apache JMeter dalam melakukan pengujian beban (load testing) atau kinerja pada sebuah *server*. Secara umum, JMeter bertindak sebagai simulator yang meniru perilaku pengguna nyata untuk menguji bagaimana *server* menangani beban tertentu.

2.2 Penelitian Terdahulu

Dalam melakukan penelitian mengenai pengujian kinerja broker MQTT sesuai spesifikasi *resource*, beberapa penelitian terdahulu dijadikan sebagai referensi. Penelitian-penelitian ini menjadi dasar dalam merancang pengujian, menentukan parameter kinerja, serta menilai efisiensi broker MQTT berdasarkan *resource* yang tersedia. Berikut adalah beberapa penelitian relevan yang dijadikan acuan.

Penelitian dengan judul "Performance Evaluation of MQTT Brokers *Server*s" Penelitian ini mengevaluasi kinerja beberapa broker MQTT, seperti Mosquitto, HiveMQ, dan EMQX, dalam lingkungan simulasi IoT. Penilaian dilakukan berdasarkan parameter seperti *throughput*, *latency*, konsumsi CPU, dan penggunaan RAMpada berbagai tingkat beban kerja. Hasil penelitian menunjukkan bahwa HiveMQ memiliki kinerja terbaik dalam lingkungan beban tinggi, sementara Mosquitto lebih efisien untuk beban ringan. Kelemahan dari penelitian ini adalah kurangnya pengujian pada infrastruktur dengan spesifikasi rendah, sehingga tidak menggambarkan kinerja broker pada perangkat berdaya terbatas (Mishra, 2018).

Penelitian lain berjudul "Stress-Testing MQTT Brokers: A Comparative Analysis of Performance Measurements" bertujuan untuk mengevaluasi kinerja berbagai implementasi broker MQTT dalam konteks komunikasi *Internet of Things* (IoT). Melakukan pengujian stres pada beberapa broker MQTT untuk menganalisis hubungan antara desain sistem broker dan kinerjanya di bawah kondisi beban tinggi. Metode yang digunakan meliputi pengukuran penggunaan CPU, *latency*, dan laju pengiriman pesan dalam dua lingkungan pengujian yang berbeda: lokal dan berbasis *Cloud*. Hasil penelitian menunjukkan bahwa broker seperti Mosquitto dan Bevywise memiliki kinerja yang baik dalam lingkungan *resource* terbatas, sementara ActiveMQ menunjukkan kinerja terbaik dalam lingkungan *multi-core*. Temuan ini memberikan wawasan penting bagi pengembang sistem *real time* dalam merancang solusi IoT yang efisien dan berkelanjutan, serta menyoroti pentingnya pengujian berbasis eksperimen untuk memahami kinerja broker MQTT dalam berbagai skenario (Mishra et al., 2021).

Penelitian lain berjudul "Performance Evaluation of MQTT Broker *Server*s Deployed in the *Cloud*" bertujuan untuk mengevaluasi kinerja delapan broker

MQTT yang diimplementasikan di *Cloud* dalam mendukung komunikasi antara nanosatelit tipe Cubesat dan terminal darat. Penelitian ini menggunakan koneksi WiFi dan aplikasi berbasis RAD Studio untuk mengukur waktu pengiriman pesan serta kinerja komunikasi. Eksperimen dilakukan dalam tiga tahap, yang mencakup pengujian *latency*, keandalan transmisi, dan regularitas pengiriman pesan pada berbagai interval waktu dan tingkat QoS (Quality of Service). Hasil penelitian menunjukkan bahwa Mosquitto dan NanoMQ unggul dalam perangkat dengan *resource* terbatas karena mampu menangani pesan dengan cepat dan stabil, sementara EMQX memerlukan spesifikasi perangkat lebih tinggi untuk mencapai kinerja optimal. Temuan ini memberikan wawasan penting dalam pemilihan broker MQTT untuk aplikasi komunikasi berbasis IoT di lingkungan satelit dan terminal darat (Pazos, n.d.)

Penelitian berjudul "A Comparative Analysis of MQTT and CoAP for IoT Applications" membandingkan kinerja protokol MQTT dan CoAP pada berbagai parameter, seperti *latency*, efisiensi *bandwidth*, dan konsumsi daya. Meski penelitian ini fokus pada perbandingan protokol, hasilnya memberikan pandangan penting mengenai keunggulan MQTT dalam efisiensi komunikasi data dan stabilitas koneksi pada aplikasi IoT. Penelitian ini juga memberikan panduan dalam memilih protokol yang sesuai dengan kebutuhan aplikasi tertentu (Safitri & Seto Priambodo, 2023).

Tabel 2. 1 Perbandingan penelitian

Peneliti	Judul Penelitian	Fokus penelitian	Parameter pengujian	Hasil penelitian
Mishra (2018)	Performance Evaluation of MQTT Brokers for IoT Applications	Evaluasi kinerja broker MQTT	*Throughput*, latency, CPU, RAM	HiveMQ memiliki kinerja terbaik dalam lingkungan beban tinggi, Mosquitto lebih efisien untuk beban ringan
Mishra et al., (2021)	Stress-Testing MQTT Brokers: A Comparative Analysis of	Pengujian stres broker MQTT	CPU, latency, *throughput*	Mosquitto dan Bevywise cocok untuk

Peneliti	Judul Penelitian	Fokus penelitian	Parameter pengujian	Hasil penelitian
	Performance Measurements			*resource* terbatas, sementara ActiveMQ memiliki kinerja terbaik di lingkungan *multi-core*.
(Pazos, n.d.)	Performance Evaluation of MQTT Broker *Server*s Deployed in the *Cloud*	Evaluasi broker MQTT berbasis *Cloud*	Latency, keandalan transmisi	Mosquitto dan NanoMQ stabil
Safitri et al, (2023).	A Comparative Analysis of MQTT and CoAP for IoT Applications	Perbandingan protokol MQTT dan CoAP.	Latency, *bandwidth*, konsumsi daya.	MQTT lebih stabil dan efisien
Diah Nurhasna (2026)	Analisis kinerja Broker MQTT pada Internet of Things.	Pengujian kinerja broker MQTT berdasarkan spesifikasi *hardware*.	Latency, *throughput*, Penggunaan CPU dan RAM, kapasitas koneksi.	Analisis kinerja broker berdasarkan *resource*

Berdasarkan kajian terhadap beberapa penelitian terdahulu pada Tabel 2.1 diketahui bahwa penelitian-penelitian tersebut telah membahas topik yang relevan mengenai kinerja broker MQTT pada sistem *Internet of Things* (IoT), seperti pengukuran *latency*, *throughput*, penggunaan CPU, serta efisiensi komunikasi data pada berbagai kondisi pengujian. Namun, beberapa penelitian sebelumnya masih memiliki keterbatasan, seperti kurangnya pengujian pada perangkat dengan spesifikasi rendah, terbatas pada lingkungan *Cloud* atau simulasi tertentu, serta belum membandingkan kinerja broker MQTT secara menyeluruh berdasarkan variasi spesifikasi perangkat keras seperti kapasitas RAM dan jumlah vCore CPU. Penelitian ini memiliki perbedaan dengan penelitian sebelumnya karena berfokus pada pengujian kinerja beberapa broker MQTT, yaitu Mosquitto, EMQX, RabbitMQ, dan VerneMQ, pada lingkungan virtual dengan variasi spesifikasi perangkat keras yang berbeda. Pengujian dilakukan dengan menganalisis parameter *latency*, *throughput*, dan kapasitas koneksi pada berbagai skenario beban kerja.

Dengan demikian, penelitian ini diharapkan dapat memberikan kontribusi dalam menentukan konfigurasi perangkat keras yang optimal untuk implementasi broker MQTT pada sistem IoT, serta melengkapi hasil-hasil penelitian sebelumnya terkait efisiensi dan skalabilitas komunikasi data IoT.