SKRIPSI

SISTEM MONITORING KUALITAS AIR TAMBAK UDANG MENGGUNAKAN *REAL TIME OPERATING SYSTEM*

Disusun dan diajukan oleh

SABTIAN JULIANA

D42115008

DEPARTEMEN TEKNIK INFORMATIKA

FAKULTAS TEKNIK

UNIVERSITAS HASANUDDIN

MAKASSAR

2021

LEMBAR PENGESAHAN SKRIPSI

SISTEM MONITORING KUALITAS AIR TAMBAK UDANG MENGGUNAKAN *REAL TIME OPERATING SYSTEM*

Disusun dan diajukan oleh

SABTIAN JULIANA
D42115008

Telah dipertahankan di hadapan Panitia Ujian yang dibentuk dalam rangka Penyelesaian Studi Sarjana Departemen Teknik Informatika Fakultas Teknik Universitas Hasanuddin pada tanggal 29 Januari 2021 dan dinyatakan memenuhi syarat kelulusan

Menyetujui,

Pembimbing Utama,	Pembimbing Pendamping
Adnan, S.T., M.T., Ph.D	Ir. Christoforus Yohannes, M.T.
Nip. 19740426 200501 1 002	Nip. 19600716 198702 1 002

Ketua Program Studi,

Dr. Amil Ahmad Ilham, S.T., M.I.T.
Nip. 19741010 199802 1 001

PERNYATAAN KEASLIAN

Yang bertanda tangan di bawah ini :

Nama : SABTIAN JULIANA

NIM : D421 15 008

Departemen : S1 Teknik Informatika

Menyatakan dengan sebenar-benarnya bahwa skripsi yang berjudul :

SISTEM MONITORING KUALITAS AIR TAMBAK UDANG MENGGUNAKAN *REAL TIME OPERATING SYSTEM*

Adalah karya ilmiah saya sendiri dan sepanjang pengetahuan saya di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan/ditulis/diterbitkan sebelumnya. Kecuali yang secara tertulis dikutip dalam naskah ini dan disebutkan dalam sumber kutipan dan daftar pustaka.

Apabila dikemudian hari ternyata di dalam naskah skripsi ini terdapat unsur-unsur jiplakan, saya bersedia menerima sanksi atas perbuatan tersebut dan diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2000, pasal 25 ayat 2 dan pasal 70)

Gowa, 30 Januari 2021

Yang Membuat Pernyataan

SABTIAN JULIANA

KATA PENGANTAR

Bismillahirrahmanirrahim

Assalamualaikum Warahmatullahi Wabarakatuh

Puji dan syukur penulis panjatkan kepada Allah SWT karena berkat rahmat dan karunia-Nya sehingga penulis dapat menyelesaikan Tugas Akhir dengan judul "SISTEM MONITORING KUALITAS AIR TAMBAK UDANG MENGGUNAKAN REALTIME OPERATING SYSTEM", sebagai salah satu syarat untuk menyelesaikan studi jenjang Strata-1 di Departemen Teknik Informatika Fakultas Teknik Universitas Hasanuddin.

Selama pengerjaan Tugas Akhir terdapat beberapa masalah yang ditemui, namun itu tidak menyurutkan semangat penulis dalam menyelesaikan masalah tersebut. Penulis juga menyadari bahwa dalam penyusunan dan penulisan laporan Tugas Akhir ini tidak lepas dari bantuan, bimbingan serta dukungan dari berbagai pihak, dari masa perkuliahan sampai dengan masa penyusunan Tugas Akhir. Oleh karena itu, penulis dengan senang hati menyampaikan terima kasih kepada:

1. Kedua orang tua penulis, Bapak Ade Mumad dan Ibu Rostika Ningsih yang selalu memberikan dukungan, doa, dan semangat serta selalu sabar dalam mendidik penulis sejak kecil;
2. Bapak Adnan S.T., M.T., Ph.D selaku pembimbing I dan Bapak Ir. Christoforus Yohannes, M.T. selaku pembimbing II yang selalu menyediakan waktu, tenaga, pikiran, dan perhatian yang luar biasa untuk mengarahkan penulis dalam penyusunan Tugas Akhir;

3. Bapak Dr. Amil Ahmad Ilham, S.T., M.IT. selaku Ketua Departemen Teknik Informatika Fakultas Teknik Universitas Hasanuddin yang senantiasa memberikan bimbingan selama masa perkuliahan penulis;
4. Para sahabat, teman-teman dan kakak-kakak di Lab Riset IOTPC FT UH yang telah memberikan begitu banyak bantuan selama penelitian, pengambilan data, dan diskusi *progress* penyusunan Tugas Akhir;
5. Teman-teman HYPERV15OR yang telah memberikan semangat dan dukungan selama masa perkuliahan hingga penyusunan Tugas Akhir ini;
6. Para kru Sapulidi yang senantiasa mendukung, memberikan masukan, dan nasehat dalam kehidupan perkuliahan selama ini;
7. Para penghuni D24 dan Villa Bujang yang telah menemani dan membantu penulis selama masa perkuliahan;
8. Para anggota Lorentz yang selalu mendukung dan menyemangati penulis;
9. Segenap Staf dan Dosen Departemen Informatika Fakultas Teknik Universitas Hasanuddin yang telah membantu penulis;
10. Orang-orang berpengaruh lainnya yang tanpa sadar telah menjadi inspirasi penulis.

Akhir kata, penulis berharap semoga Allah SWT. berkenan membalas segala kebaikan dari semua pihak yang telah membantu. Semoga Tugas Akhir ini dapat memberikan manfaat bagi pengembangan ilmu pengetahuan. Aamiin.

Wassalamualaikum Warahmatullahi Wabarakatuh

Makassar, 19 Agustus 2020

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

ABSTRAK

Dalam pembudidayaan udang, kualitas air tambak merupakan faktor yang sangat penting untuk diperhatikan oleh petambak. Beberapa parameter kualitas air yang perlu diperhatikan diantaranya adalah suhu, salinitas, pH, dan ketinggian air tambak. Sistem pembacaan kualitas air tambak secara manual atau konvensional membuat penanganan menjadi lambat padahal udang merupakan hewan yang sensitif terhadap perubahan kualitas air. Oleh karena itu, dalam penelitian ini dibuatlah sebuah sistem yang dapat memonitoring kualitas air tambak dengan menggunakan ESP32. ESP32 merupakan mikrokontroller penerus ESP8266 yang sudah memiliki dual core prosesor. Akan tetapi, walaupun sudah memiliki prosesor dual core, hanya satu core yang akan digunakan secara default. Oleh karena itu, dalam sistem yang dibuat digunakanlah *Real Time Operating System* untuk memaksimalkan prosesor yang dimiliki. Pengujian dilakukan dengan cara mengirim data setiap menit dan menghitung *data loss* yang terjadi pada sistem yang dibuat. Pengujian dilakukan sebanyak 5 kali dengan jarak antar node sebesar 10 meter, 20 meter, 25 meter, 30 meter, dan 35 meter. Hasil dari pengujan yang dilakukan *data loss* yang terjadi pada jarak 10 meter adalah 0,0%, pada jarak 20 meter 0,0%, pada jarak 25 meter 0,0%, pada jarak 30 meter 0,4%, dan pada jarak 35 meter 38,3%.

Kata kunci : *Data Loss, Esp Mesh, Real Time Operating System*

BAB I

PENDAHULUAN

1.1. Latar Belakang

Internet of Things adalah sebuah konsep dimana sebuah objek yang memiliki kemampuan untuk mentransfer data melalui jaringan tanpa interaksi manusia ke manusia ataupun manusia ke komputer. Sederhananya, *Internet of Things* adalah alat yang terhubung dengan internet dan saling terintegrasi.

Fungsi utama *Internet of Things* pada dasarnya sebagai *data miner*. *Internet of Things* bekerja mencari dan mengumpulkan berbagai data yang nantinya akan diolah menjadi informasi yang lebih bermanfaat.

Internet of Things sangat erat hubungannya dengan revolusi industri 4.0 karena *IoT* adalah unsur utamana dalam revolusi tersebut. *IoT* berpengaruh dalam berbagai macamindustri seperti manufaktur, logistik, tata kota, rumah pertanian, dan lain sebagainya.

Tambak merupakan salah satu jenis habitat yang digunakan untuk kegiatan budidaya air payau yang berada di pesisir, dimana kegiatan budidaya yang dilakukan secara terus menerus dapat menyebabkan terjadinya degradasi terhadap lingkungan dan ditandai dengan menurunnya kualitas air. (Alimuddin, 2015)

Kualitas air pada tambak udang merupakan salah satu faktor yang sangat penting untuk diperhatikan dalam membudidayakan udang. Beberapa parameter kualitas air tambak yang perlu diperhatikan diantaranya adalah salinitas, pH, suhu, dan ketinggian/kedalaman air tambak.

Salinitas air tambak udang sangat berpengaruh pada pertumbuhan udang. Nilai salinitas sangat mudah berfluktuasi. Oleh karena itu, untuk menghindari stress pada udang petambak diharuskan untuk selalu memeriksa tingkat salinitas air dengan rutin.

pH merupakan derajat keasaman yang digunakan untuk menyatakan tingkat keasaman suatu larutan. Perlunya menjaga stabilitas pH di kisaran aman agar tidak berpengaruh pada metabolisme dan kondisi fiologi udang. Stress akibat fluktuasi pH yang besar akan menurunkan laju makan udang, sehingga mengakibatkan pertumbuhan udang melambat dan rawan terinfeksi penyakit.

Suhu air juga mempengaruhi kondisi fisiolgi pada udang. Suhu yang rendah dapat mengakibatkan sistem metabolisme udang menjadi lambat dan sebaliknya suhu yang tinggi mengakibatkan metabolisme udang menjadi cepat, hubungannya kemudian pada nafsu makan udang. Suhu yang tiba-tiba turun akan mengakibatkan udang stress hingga menyebabkan kematian.

Ketinggian atau kedalaman air berpengaruh pada perubahan suhu air tambak. Jika kedalaman air tinggi makan suhu air akan semakin rendah. Begitupun sebaliknya jika kedalaman air rendah mengakibatkan suhu yang cenderung lebih tinggi.

Sistem pembacaan kualitas air tambak secara manual atau konvensional membuat penanganan menjadi lambat padahal udang adalah hewan yang sensitif terhadap perubahan kualitas air. Proses pengecekan yang dilakukan secara periodik kebanyakan menyita waktu petambak. Oleh karena itu, maka dalam penelitian ini

dibuat sebuah sistem untuk memonitoring kualitas air tambak udang menggunakan *real time operating system*.

Real Time Operating System (RTOS) adalah sistem operasi yang digunakan untuk memenuhi kebutuhan aplikasi yang bersifat *realtime. Realtime* disini berarti ia membutuhkan kinerja setiap saat dimana ia dibutuhkan saat itu juga. Salah satu kunci keberhasilan RTOS adalah kemampuannya untuk melakukan kerja secara konsisten baik secara waktu yang ia butuhkan maupun task yang mampu ia kerjakan.

ESP32 adalah mikrokontroller yang dikenalkan oleh Espressif System dan merupakan penerus dari ESP8266. ESP32 ini sudah memiliki modul WIFI dan Bluetooth didalamnya serta mempunyai CPU Xtensa dual core LX6 – 160MHz.

Meskipun memiliki prosesor dual core tapi secara default hanya satu core digunakan. Maka untuk memaksimalkan kinerja dari ESP32 penulis akan menggunakan RTOS (*Realtime Operating System*) agar kedua core tersebut dapat bekerja mengeksekusi perintah sehingga program akan berjalan lebih efektif.

1.2. Rumusan Masalah

Rumusan masalah pada tugas akhir ini adalah:

1. Bagaimana merancang alat monitoring suhu, salinitas, pH, dan ketinggian air tambak udang?
2. Berapa persentase *data loss* yang terjadi pada saat pengiriman data?

1.3. Tujuan Penelitian

Tujuan akhir dari penelitian ini adalah:

1. Untuk membuat sistem yang dapat memonitoring suhu, salinitas, pH, dan ketinggian air tambak udang.
2. Untuk mengetahui persentase *data loss* yang terjadi pada saat pengiriman data.

1.4. Manfaat Penelitian

Dengan dilakukannya penelitian ini, diharapkan manfaat yang didapatkan adalah:

1. Bagi petambak, dapat memiliki sistem yang bisa memonitoring kualitas air tambak udangnya sehingga waktu petambak tidak banyak tersita untuk melakukan pengecekan secara periodik.
2. Bagi peneliti, dapat digunakan untuk menambah pengetahuan dan sebagai referensi mengenai *multitasking*.
3. Bagi institusi pendidikan, dapat digunakan sebagai referensi dalam pengembangan penelitian mengenai *multitasking*.

1.5. Batasan Masalah Penelitian

Yang menjadi batasan masalah dalam tugas akhir ini adalah:

1. Mikrokontroller yang digunakan adalah ESP32.
2. Objek penelitian adalah tambak udang.
3. Parameter yang diukur adalah salinitas, tinggi, pH, dan suhu air tambak.

1.6. Sistematika Penulisan

Untuk memberikan gambaran singkat mengenai isi tulisan secara keseluruhan, maka akan diuraikan beberapa tahapan dari penulisan secara sistematis, yaitu:

BAB I PENDAHULUAN

Bab ini menguraikan secara umum mengenai hal yang menyangkut latar belakang, perumusan masalah, Batasan masalah tujuan penelitian, manfaat penelitian, dan sistematika penulisan.

BAB II TINJAUAN PUSTAKA

Bab ini berisi teori-teori tentang hal-hal yang berhubungan dengan *Real Time Operating System*, konsep dasar *Real Time Operating System*, API reference RTOS *routing, Esp Mesh*, mikrokontroller, dan Esp32.

BAB III METODOLOGI PENELITIAN

Bab ini berisi tentang tahapan penelitian, waktu dan lokasi penelituan, instrumen penelitian, pengambilan data, desain perangkat keras, perancangan sistem, dan evaluasi kinerja.

BAB IV HASIL DAN PEMBAHASAN

Bab ini berisi tentang hasil penelitian serta pembahasan yang disertai tabel hasil penelitian.

BAB V PENUTUP

Bab ini berisi tentang kesimpulan yang didapatkan berdasarkan hasil penelitian yang telah dilakukan serta saran-saran untuk pengembangan lebih lanjut.

BAB II

TINJAUAN PUSTAKA

2.1. *Real Time Operating System*

Real Time Operating System (RTOS) adalah sebuah sistem operasi yang dikembangkan untuk aplikasi *real time embedded system* yang berkembang di sekitar prosesor atau *controller*. Sebelum adanya RTOS, *embedded system* developer menggunakan *primitive interrupt* untuk menjalankan proses multitasking. Akan tetapi, terdapat beberapa kelemahan pada system interrupt konvensional, terutama dalam penjadwalan task yang diberikan kepada *embedded system*. (Wisnu, 2004)

Real Time Operating System (RTOS) mengeksekusi program-program dalam sebuah pola yang teratur. RTOS tidak seperti sistem operasi yang pada umumnya digunakan pada komputer. Sistem operasi pada komputer akan bekerja sesaat ketika power masuk ke komputer, kemudian program komputer dijalankan. Sedangkan pada RTOS, sistem operasi ini dijalankan oleh sebuah program otomatis. Program tersebut merupakan sebuah *kernel*.

Ketika sistem dinyalakan, maka *kernel* akan menyala terlebih dahulu kemudian menyalakan *Real Time Operating Sistem*. Tugas utama dari RTOS adalah mengatur sumber daya yang ada meliputi prosesor, memori/register, serta hak akses menuju dua sumber daya terebut. Selain itu, RTOS bertugas melakukan komunikasi dan sinkronisasi. Selain berjalan menggunakan kernel RTOS dapat terdiri dari kombinasi beberapa module *kernel, file system, networking protocol stack,* dan komponen lain.

Kernel RTOS terdiri dari *kernel object. Kernel object* adalah blok-blok yang digunakan untuk membangun *real time embedded system.* RTOS *kernel object* tersebut yaitu:

- *Task*

Task merupakan *thread* dari suatu proses yang akan dieksekusi oleh prosesor. *Task* bersifat konkuren dan independen terhadap *task* lain. Desain proses untuk *real time application* yaitu dengan membagi pekerjaan yang akan di kerjakan menjadi beberapa *task* dan setiap task diberikan prioritas masing-masing (Chandane, 2016). Setiap *task* biasanya merupakan *infinite loop* dan terdiri dari 4 keadaan yaitu :

1. *Running State*

   Running State merupakan keadaan dimana *Task* sedang berjalan atau dieksekusi. Ketika berada pada keadaan ini suatu *task* memanfaatkan prosesor dan sumber daya lain seperti *register* dan memori. Jumlah *task* yang dapat berada dalam keadaan ini sesuai dengan jumlah prosesor yang dimiliki oleh komputer atau mikrokontroller. Ketika berada pada keadaan *running state, task* dapat kembali ke *ready state* apabila muncul *task* baru yang memiliki prioritas yang lebih tinggi. Pada keadaan ini *task* dapat pula berpindah ke *suspended state atau blocked state* apabila melakukan hal berikut:

   - Meminta *resource* yang tidak tersedia di dalam sistem.
   - Meminta menunggu *event* tertentu muncul.
   - Meminta *delay.*

2. *Ready State*

Sebuah *task* akan berada pada *ready state* setelah *task* tersebut dibuat. *Ready state* adalah keadaan dimana sebuah *task* siap untuk dieksekusi, *task* tersebut akan menunggu *task* lain yang memiliki prioritas sama atau lebih tinggi yang sedang dieksekusi pada *running state.* Pada keadaan ini *task* dapat berpindah ke *running state* atau *suspended state.*

3. *Blocked State*

Task akan berada pada *blocked state* apabila *task* tersebut sedang menunggu *event* lain terjadi. *Event* tersebut meliputi *temporal event* atau *external event. Temporal event* adalah *delay* yang diminta oleh *task* itu sendiri. *Task* akan di-*unblock* dijalankan kembali setelah periode *delay* habis. Sedangkan *external event* berupa *queue* atau *semaphore event*. *Real time system* membutuhkan keadaan ini karena jika *state* ini tidak ada maka *task* dengan prioritas rendah tidak memiliki kesempatan untuk dieksekusi atau dikenal dengan *starvation.*

4. *Suspended State*

Suspended state adalah keadaan dimana sebuah *task* ditunda eksekusinya untuk waktu yang tidak ditentukan. Penundaan tersebut dapat terjadi ketika *task* menunggu *resources* untuk dikosongkan terlebih dahulu. *Task* akan berada di keadaan ini apabila dipanggil fungsi *vTaskSuspend(),* dan akan keluar dari keadaan ini ketika dipanggil *vTaskResume().*

- *Semaphore*

Semaphore merupakan objek berbentuk token yang nilainya dapat ditambah dan dikurangi oleh *task* untuk keperluan sinkronisasi ataupun *mutual exclusion. Mutual exclusion* adalah kondisi dimana terdapat sumber daya yang tidak dapat dipakai bersama dalam waktu yang bersamaan. Jadi, *mutual exclusion* terjadi ketika hanya ada satu proses yang boleh menggunakan sumber daya, dan proses lain yang ingin menggunakan sumber daya tersebut harus menunggu hingga sumber daya tadi dilepaskan atau tidak ada proses yang menggunakan sumber daya tersebut.

Semaphore digunakan sebagai penanda bahwa *task* yang mengakuisisinya memiliki hak penuh untuk menjalankan operasi tertentu dan juga menggunakan *recources* yang ada. Akuisisi *semaphore* dibatasi oleh waktu dalam periode tertentu. *Semaphore* dapat diibaratkan sebagai kunci duplikat rumah yang dipegang oleh kepala keluarga. Anggota keluarga lain dapat meminjam kunci duplikat tersebut. Akan tetapi, kunci duplikat tersebut tersedia dalam jumlah terbatas sehingga tidak bisa dipinjam oleh sekaligus secara bersamaan. (Wisnu, 2004)

- *Message Queues*

Message Queues merupakan struktur data yang digunakan untuk sinkronisasi, *mutual exclusion*, dan pertukaran data dengan mengantarkan pesan antar *task.*

Keunggulan dari *Real Time Operating System* yaitu:

- Dapat berjalan pada perangkat keras dengan sumber daya yang terbatas.
- Memiliki tingkat modularitas yang tinggi.
- Dapat dikembangkan dan digunakan ulang.
- Memiliki beberapa fitur yang terdokumentasi lengkap, diantaranya

- Strategi *scheduling.*
- Maksimal waktu eksekusi untuk setiap proses.
- Mekanisme internal untuk mengatur *latency.*

- Memiliki sistem pengaturan *interrupt* untuk proses-proses penting.
- Konfigurasi *kernel* dapat diatur secara detail oleh developer.

Real Time Operating System (RTOS) memiliki sejumlah karakteristik diantaranya *reliability, predictable, performance, compactness,* dan *scalability.*

1. *Reliability*

Reliability adalah keandalan atau konsistensi sistem dalam menjalankan tugasnya. *Embedded system* biasanya dibuat untuk bekerja dalam waktu yang lama tanpa adanya intervensi manusia. Maka dari itu *embedded system* harus bersifat *reliable.*

2. *Predictability*

RTOS menjamin setiap *real time system* dapat mengoperasikan tugasnya dalam waktu yang sudah ditentukan. Jadi perilaku RTOS itu sendiri dapat diprediksi atau dengan kata lain *system call* pada RTOS akan muncul dalam rentang waktu yang diketahui.

3. *Performance*

Pada umumnya *embedded system* harus dapat beroperasi dalam waktu yang cepat. Jumlah proses yang harus dikerjakan srta waktu antar proses yang singkat memberikan dampak pada kebutuhan CPU yang cepat. Performa dari prosesor dapat dideskripsikan dalam *million instruction per second* (MIPS). Performa tersebut dapat dipengaruhi oleh *hardware* ataupun *software* yang digunakan.

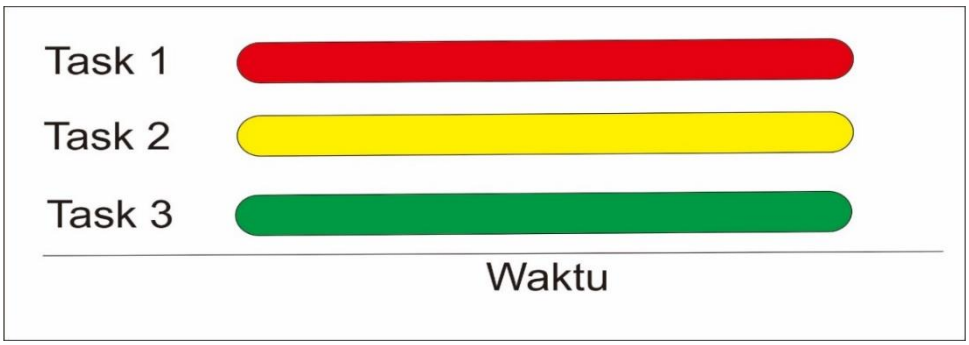
Selain mengukur MIPS, performa sistem juga dapat diukur dengan menggunakan *throughput*. *Throughput* adalaha rata-rata sebuah sistem dapat menjalankan keluaran dari masukan yang diterima. Definisi lainnya adalah bersaran yang menunjukkan ukuran data yang dapat dialirkan berdasarkan satuan waktu. Performa RTOS juga dapat diukur dengan menampilkan *timestamp* ketika sistem dijalankan sampai sistem selesai menjalankan tugasnya. (Wisnu, 2004)

2.2. Konsep Dasar *Real Time Operating System*

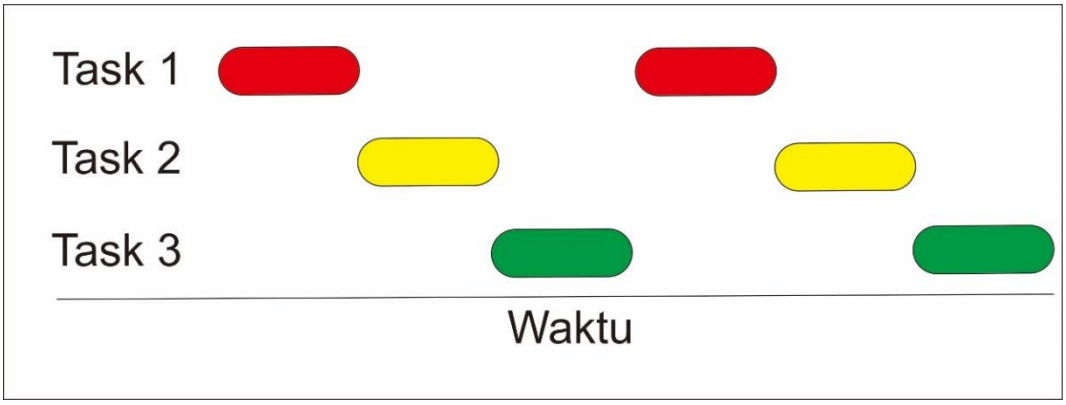
Dalam *Real Time Operating System* memiliki beberapa konsep dasar diantaranya:

2.2.1. *Multitasking*

Multitasking adalah sebuah metode dimana beberapa proses atau *task* dikerjakan dalam rentang waktu yang bersamaan. Akan tetapi, pada prosesor konvensional, konsep multitasking belum dapat dijalankan. Prosesor belum memungkinkan untuk mengeksekusi lebih dari satu *task* dalam suatu waktu. Namun *task-task* tersebut dijalankan secara bergantian. Hal tersebut dinamakan dengan konkurensi. Perbedaan multitasking dan konkurensi dapat dilihat dalam gambar berikut:



Gambar 2.1 Konsep Multitasking



Gambar 2.2 Konsep Konkurensi

Jadi ditunjukkan pada **gambar 2.1** dimana tiga *task* dieksekusi dalam rentang waktu yang sama. Sedangkan pada **gambar 2.2** *task* dieksekusi secara bergantian, sistem operasi membagi rentang waktu yang ada untuk mengeksekusi *task* yang akan berjalan. Hanya satu *task* yang dieksekusi pada satu waktu. Sistem operasi mengatur kapan waktu eksekusi setiap *task* dan menyimpan *state* dari setiap *task* pada memori, sehingga ketika prosesor menjalankan kembali *task* sudah berjalan sebelumnya maka prosesor akan mengambil *state* terakhir dari *task* tersebut kemudian melanjutkan mengeksekusinya. Proses perpindahan eksekusi antar *task* dilakukan dengan sangat cepat dan konkuren sehingga seolah-olah prosesor seperti mengeksekusi beberapa *task* dalam rentang waktu bersamaan. (Setiawan, 2016)

2.2.2. *Scheduling*

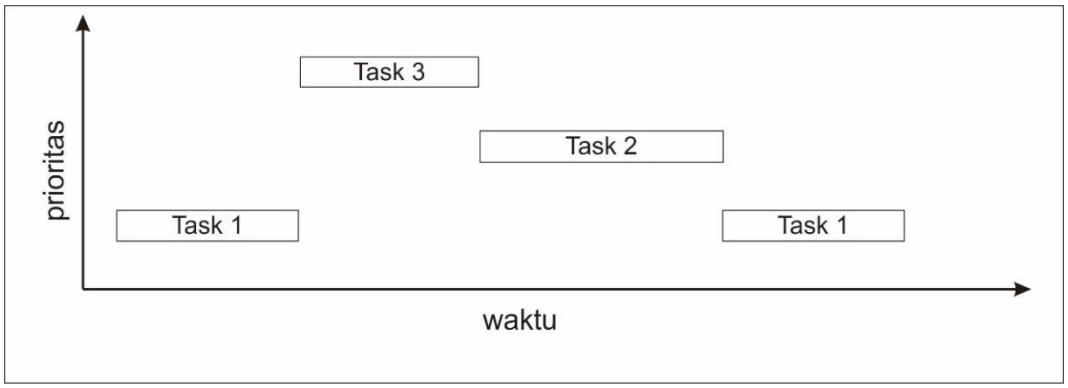
Scheduling atau penjadwalan adalah konsep pembagian waktu eksekusi *task* yang akan dikerjakan (Wisnu, 2004). Untuk melakukan penjadwalan diperlukan adanya sebuah *scheduler*. *Scheduler* berfungsi sebagai pengatur waktu eksekusi suatu *task*, lama eksekusi *task*, waktu tunda suatu *task (suspend)*, dan waktu suatu *task* dikerjakan kembali *(resume)*.

Dalam *Real Time Operating System*, digunakan dua jenis penjadwalan yaitu :

- *Round-Robin scheduling*

Round-Robin scheduling adalah salah satu penjadwalan proses atau *task*, dimana algoritma ini menggilir proses yang ada di antrian. Algoritma *Round-Robin* biasa juga disebut dengan *Fair Time Scheduling*, memiliki konsep dimana semua sumber antrian dianggap sama sehingga diberi waktu eksekusi yang sama yang dinamakan *time quantum*. Jika *time quantum* habis atau *task* selesai, maka *task* yang selanjutnya akan dieksekusi. *Round-Robin scheduling* menggunakan *time slicing* untuk mendapatkan alokasi waktu yang sama pada setiap *task*. Algoritma ini pada dasarnya kurang cocok untuk digunakan pada *real time system* karena tidak memberikan variasi prioritas pada setiap *task.* Padahal pada *real time system* setiap task akan dieksekusi berdasarkan prioritas yang ada.

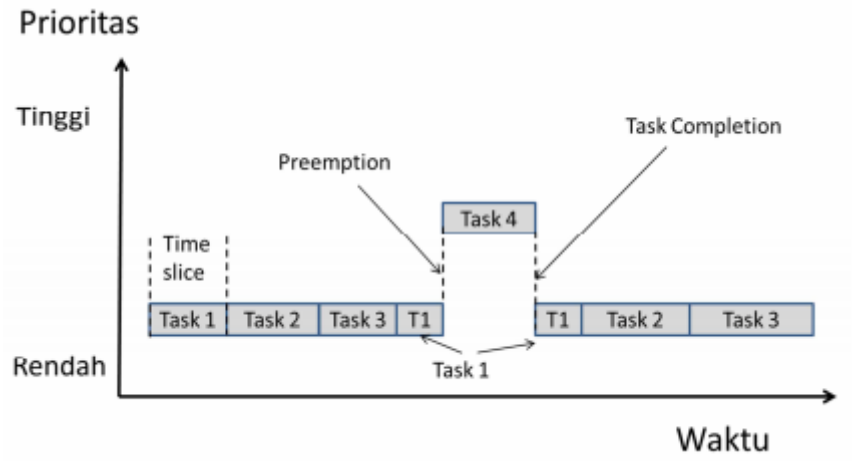
- *Preemptive Priority-Based Scheduling*



Gambar 2.3 Algoritma *Preemptive Priority-Based*

Algoritma *preemptive priority-based scheduling* adalah salah satu penjadwalan proses atau *task*, dimana setiap *task* akan diberi prioritas dan *task* yang memiliki prioritas paling tinggi akan dieksekusi terlebih dahulu. **Gambar 2.3** menunjukkan bagaimana algoritma *preemptive priority-based scheduling* bekerja. *Task* dengan prioritas tinggi akan dieksekusi, sedangkan *task* lain akan ditunda hingga *task* yang memiliki prioritas yang lebih tinggi selesai dieksekusi. Pada RTOS setiap *task* diberi prioritas pada saat pertama kali dibuat. Namun prioritas tersebut dapat diubah secara dinamis menggunakan *kernel-provided calls.* Kemampuan tersebut menambah fleksibilitas sebuah sistem terhadap eksternal event serta menciptakan sistem yang lebih *real time* dan responsif.

Real Time Operating System mengkombinasikan antara *round-robin scheduling* dan *preemptive priority-based scheduling*, seperti gambar berikut:



Gambar 2.4 Kombinasi Algoritma *Round-Robin* dan *Preemptive Priority*

Seperti pada **gambar 2.4** algoritma *round-robin* akan membagi eksekusi *task* menggunakan *time slice*. Terdapat *counter* yang akan menghitung waktu eksekusi setiap *task*. Ketika waktu sudah habis, maka *task* selanjutnya akan dieksekusi. Jika

terdapat *task* yang lain dengan prioritas lebih tinggi tiba-tiba muncul, maka *counter* yang menghitung *task* yang sedang dieksekusi akan disimpan (Task 1 di-*interrupt* oleh Task 4). Kemudian *task* dengan prioritas tinggi akan dieksekusi, setelah selesai eksekusinya, kernel akan kembali mengeksekusi *task* yang sebelumnya telah di-*interrupt* melanjutkan waktu yang sudah dicatat oleh *counter*.

Scheduler sudah mengatur waktu suatu *task* akan ditunda, namun dalam *scheduling* juga memungkinkan setiap *task* untuk mengajukan waktu penundaannya sendiri. Setiap *task* dapat meminta *delay* pada rentang waktu tertentu. Ketika *delay*, *task* akan berada dalam *blocked state* hingga waktu delay habis. Selain melalui permintaan *delay*, suatu *task* dapat meminta penundaan eksekusi ketika *task* tersebut menunggu *resources* dari luar seperti masukan *port* atau terjadinya event tertentu. Karena adanya penundaan oleh suatu *task*, maka *scheduler* dapat mengalokasikan rentang waktu kosong untuk mengeksekusi *task* lain.

2.2.3. *Context Swithing*

Ketika sebuah *task* dijalankan, *task* tersebut menggunakan register prosesor atau mikrokontroller dan mengakses RAM dan ROM sama seperti program lainnya. Semua *recources* ini (register prosesor, *stack*, dan lainnya) digunakan secara bersama-sama oleh beberapa *task.*

Task merupakan bagian dari kode yang tidak tahu kapan akan di ditangguhkan *(suspended)* atau dilanjutkan *(resume)* oleh kernel. Hal tersebut dapat mengakibatkan kesalahan pada instruksi suatu *task* yang melibatkan *register* dan memori. Misalkan sebuah *task* ditangguhkan *(suspended)* segera sebelum

menyelesaikan instruksi penjumlahan yang melibatkan dua *register* prosesor. Ketika *task* berada pada *suspended state*, *task* lain akan dieksekusi dan bisa mengubah nilai di *register* prosesor. Ketika *task* yang sebelumnya dilanjutkan eksekusinya *task* tersebut tidak mengetahui jika nilai di *register* prosesor telah diubah. Hal ini menyebabkan hasil dari instruksi penjumlahan yang dilakukan tidak tepat.

Untuk mencegah kesalahan seperti ini digunakan mekanisme yang menyimpan kondisi atau nilai suatu *register*. Kondisi tersebut yang disebut dengan context. Setiap *task* masing-masing memiliki *context* yang identik. Kernel sistem operasi bertanggung jawab untuk memastikan hal ini terjadi dan melakukannya dengan menyimpan context dari *task* yang di-*suspended.* Ketika *task* akan dieksekusi kembali, *context* tersebut dikembalikan oleh kernel sistem operasi. Proses menyimpan context dari sebuah *task* yang di-*suspended* dan mengembalikan *context* yang di-*resume* dinamakan dengan *context switching.*

2.3. API Reference RTOS

Application Programming Interface (API) merupakan *software interface* yang terdiri atas kumpulan instruksi yang disimpan dalam bentuk *library* dan menjelaskan bagaimana agar suati *software* dapat berinteraksi dengan *software* lain. API dapat dianalogikan jika seseorang ingin membangun rumah dan pemilik rumah tersebut menyewa kontraktor yang dapat menangani bagian yang berbeda. Pemilik rumah dapat memberikan tugas yang perlu dilakukan oleh kontraktor tanpa harus mengetahui bagaimana cara kontraktor menyelesaikan pekerjaan tersebut.

Dalam *real time operating system*, terdapat beberapa API yang umumnya digunakan, antara lain:

2.3.1. Pembuatan Task

1. xTaskCreate dan xTaskCreateStatic

Berfungsi untuk membuat task baru dan menambahkannya ke dalam daftar task yang siap untuk dijalankan. Setiap task membutuhkan RAM yang digunakan untuk memberikan keadaan task *(task state)*, dan digunakan oleh task sebagai *stack*. Jika task dibuat menggunakan xTaskCreate() maka RAM yang dibutuhkan dialokasikan dari FreeRTOS heap. Jika task dibuat menggunakan xTaskCreateStatic() maka RAM disediakan oleh penulis aplikasi, sehingga dapat dialokasikan secara statis pada waktu kompilasi.

2. vTaskDelete

Berfungsi untuk menghapus task dari manajemen kernel RTOS. Task yang dihapus akan dihilangkan dari keadaan *ready, blocked, suspended*, dan *event list.*

Idle task bertanggung jawab untuk membebaskan memori yang dialokasikan kernel RTOS untuk task yang telah dihapus. Oleh karena itu penting agar *idle task* tidak mengurangi waktu pemrosesan mikrokontroller ketika memanggil vTaskDelete(). Memori yang dialokasikan oleh task tidak dibebaskan secara otomatis, dan harus dibebaskan sebelum tugas dihapus.

2.3.2. *Task Control*

1. vTaskDelay

Berfungsi untuk menunda task untuk setiap detik tertentu. Task akan berada dalam *blocked state* selama waktu yang telah ditentukan. Konstanta portTICK_PERIOD_MS dapat digunakan untuk menghitung waktu dalam bentuk *millisecond (ms)*.

vTaskDelay() menetapkan waktu saat task ingin di-*unblock* relatif terhadap waktu saat vTaskDelay() dipanggil. Misalnya, menentukan periode *block* 100 detik akan menyebabkan task akan di-*unblock* 100 detik setelah vTaskDelay() dipanggil.

2. vTaskDelayUntil

Berfungsi untuk menunda task hingga waktu tertentu. Fungsi ini dapat digunakan oleh task yang berkala untuk memastikan frekuensi eksekusi yang konstan.

Fungsi ini berbeda dari vTaskDelay() dalam satu aspek penting. vTaskDelay() menentukan waktu task akan di-*unblock* relatif terhadap waktu pemanggilan vTaskDelay(), sedangkan vTaskDelayUntil() menentukan waktu absolut saat task akan di-*unblock*.

3. vTaskSuspend

Berfungsi untuk melakukan *suspend* ke suatu task. Ketika di-*suspend* task tidak akan mendapat waktu proses dari mikrokontroller, tidak peduli apapun prioritasnya.

Pemanggilan vTaskSuspend tidak akumulatif, artinya memanggil vTaskSuspend() dua kali pada task yang sama hanya memerlukan satu pemanggilan fungsi vTaskResume() agar task Kembali *ready*.

4. vTaskResume

Berfungsi untuk melanjutkan task yang telah di-*suspend*. Sebuah task yang di-*suspend* oleh satu atau lebih vTaskSuspend() akan tersedia untuk dijalankan Kembali dengan satu pemanggilan vTaskResume().

5. xTaskAbortDelay

Berfungsi untuk memindahkan task dari *blocked state* ke *ready state* secara paksa, bahkan jika *event* yang ditunggu oleh task yang berada pada *blocked state* tidak terjadi dan waktu tunda belum habis.

2.3.3. *Kernel Control*

1. vTaskStartScheduler

Berfungsi untuk memulai scheduler RTOS. Setelah dipanggil kernel RTOS memiliki kendali penuh atas task mana yang akan dijalankan dan kapan akan dijalankan.

2. vTaskEndScheduler

Berfungsi untuk menghentikan kernel RTOS. Semua task yang dibuat akan otomatis dihapus dan multitasking akan berhenti.

2.4. *Routing*

Routing merupakan suatu protokol yang digunakan untuk mendapatkan rute dari suatu jaringan ke jaringan yang lain. Agar *router* dapat mengetahui bagaimana meneruskan paket-paket yang dikirim ke alamat yang dituju dengan menggunakan jalur terbaik, router menggunakan peta *routing* atau *routing table*.

2.4.1. *Static Routing*

Static routing adalah salah satu jenis *routing* yang dimana pembuatan dan pembaharuan *routing table* dilakukan secara manual. *Static routing* tidak akan mengubah informasi yang ada pada *table routing* secara otomatis, sehingga admin harus mengubahnya secara manual apabila topologi jaringan berubah.

Beberapa kelebihan dari *static routing* yaitu:

- Pemeliharaan *bandwidth* jaringan, karena pembaharuan informasi router membutuhkan broadcast terus menerus.
- Keamanan jaringan, karena *static routing* hanya mengandung informasi yang telah dimasukkan secara manual.

Sedangkan kekurangan dari *static routing* yaitu:

- Tidak adanya toleransi kesalahan, jika suatu *router down* maka *static routing* tidak akan memperbaharui informasi dan tidak akan menginformasikan ke *router* yang lainnya
- Pengembangan jaringan, jika suatu jaringan ditambah atau dipindahkan maka harus diperbaharui oleh administrator.

2.4.2. *Dynamic Routing*

Dynamic Routing adalah jenis *routing* yang dimana pembuatan jalur dilakukan secara otomatis oleh router itu sendiri sesuai dengan konfigurasi yang dibuat. Jika ada perubahan topologi antar jaringan, router secara otomatis akan membuat *routing* baru.

Adapun kelebihan dari *dynamic routing* yaitu:

- Cocok untuk area yang luas.

- Hanya mengenalkan alamat yang terhubung langsung dengan routernya.
- Bila terjadi penambahan suatu jaringan maka tidak perlu semua router dikonfigurasi, hanya *router* yang berkaitan saja.
- Router berbagi informasi secara otomatis.
- *Routing table* dibuat secara dinamik.
- Tidak memerlukan campur tangan administrator.

Sedangkan kelemahan dari *dynamic routing* yaitu:

- Beban kerja *router* menjadi lebih berat.
- Kecepatan pengenalan dan kelengkapan *routing table* terbilang lama karena *router* mem-*broadcast* ke semua *router* lainnya sampai ada yang cocok sehingga setelah konfigurasi harus menunggu beberapa saat agar setiap *router* mendapatkan semua alamat yang ada.

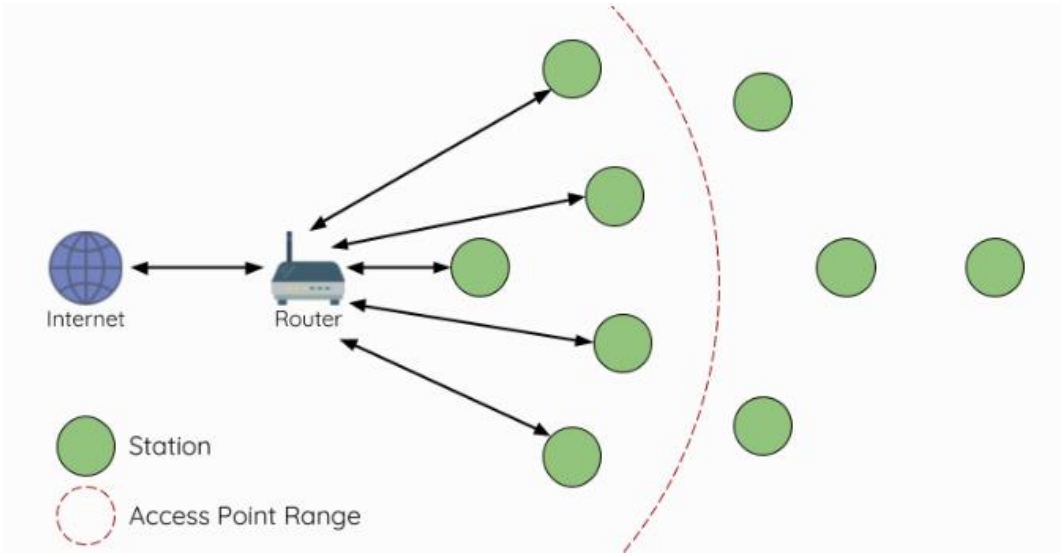
2.5. *Esp Mesh*

Topologi mesh merupakan suatu bentuk dari topologi jaringan yang menghubungkan antara satu node dengan node lainnya didalam jaringan tersebut. Hubungan antar node ini akan membentuk sebuah rangkaian yang menyerupai jaring. Perbedaan utama pada topologi mesh dan topologi star adalah pada topologi mesh memiliki kemampuan untuk *self organizing* dan *self configuration*. (Liu, 2017)

Topologi ini tidak membutuhkan *dedicated link* (perantara), setiap node akan saling berhubungan satu sama lainnya sehingga pada waktu yang sama akan terbentuk komunikasi langsung. Dikarenakan komunikasinya terhubung secara

langsung, maka apabila ada satu node yang mati, maka tidak akan berpengaruh pada node yang lainnya.

Pada arsitektur jaringan WiFi tradisional menggunakan *point to multipoint* dimana terdapat sebuah node pusat yang dikenal dengan *Access Point* (AP) yang terhubung langsung ke semua node (*station)* yang ada pada jaringan. *Access point* bertanggung jawab sebagai penengah dan meneruskan transmisi data antar *station*. Pada arsitektur tradisional ini memiliki kekurangan dimana area jangkauan yang kurang luas karena setiap *station* harus terhubung langsung dengan *access point*. Selain itu, jaringan ini rentan terjadi *overloading* dikarenakan jumlah *station* yang dapat terhubung ke *access point* terbatas.

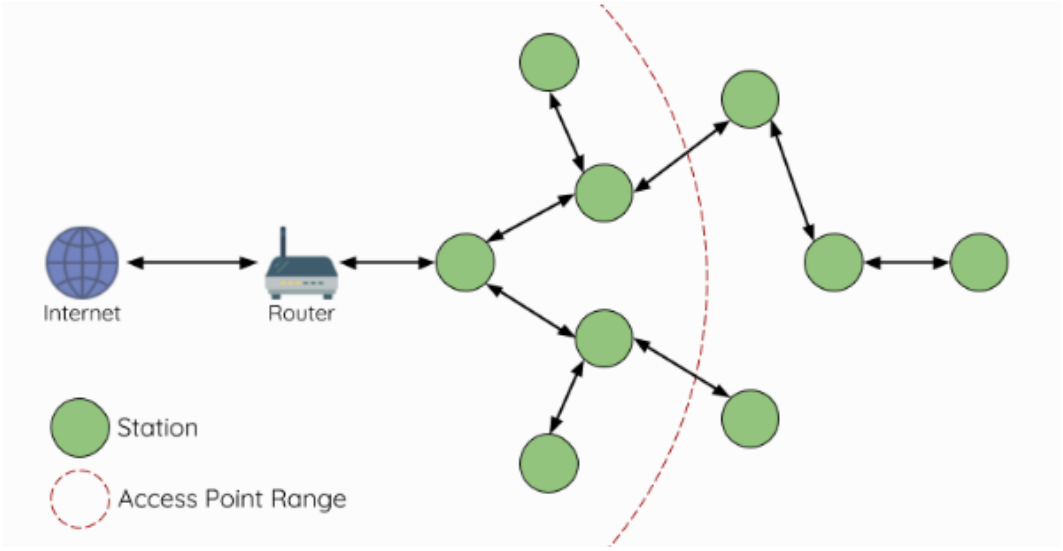


Gambar 2.5 Arsitektur Jaringan WiFi Tradisional

Esp Mesh adalah sebuah protokol jaringan pada protokol WiFi. *Esp Mesh* memungkinkan banyak perangkat atau node tersebar di area yang luas untuk saling terhubung di suatu jaringan WLAN (*Wireless Local Area Network). Esp Mesh*

dapat melakukan *self-organizing* dan *self-healing*, artinya jaringan ini dapat dibuat dan maintenance secara mandiri. (Espressif Systems, 2016)

Esp Mesh berbeda dengan jaringan WiFi tradisional dimana setiap node tidak perlu untuk terhubung langsung ke node pusat. Sebagai gantinya, node diizinkan untuk terhubung dengan node tetangganya. Setiap node saling bertanggung jawab untuk menyampaikan transmisi data satu sama lain. Hal ini memungkinkan jaringan *Esp Mesh* untuk memiliki jangkauan yang lebih luas karena node masih dapat melakukan interkonektivitas tanpa perlu berada dalam jangkauan node pusat. Jaringan ini juga tidak rentan terhadap *overloading* karena jumlah node yang diizinkan pada jaringan tidak lagi dibatasi oleh node pusat *(Access Point)*.

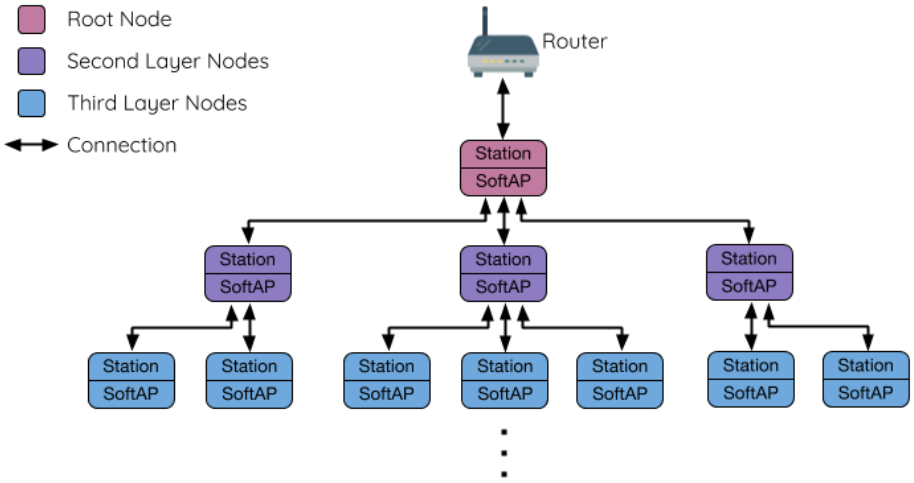


Gambar 2.6 Arsiektur Jaringan Esp Mesh

2.5.1. Topologi Esp Mesh

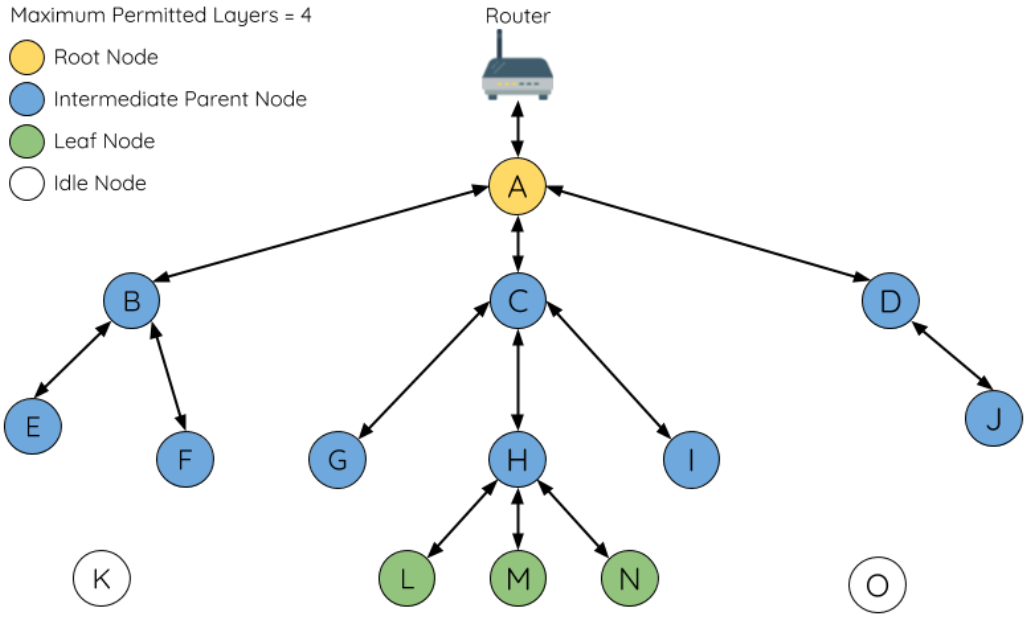
Esp Mesh dibuat diatas protokol WiFi dan bisa dianggap sebagai protokol jaringan yang menggabungkan banyak jaringan WiFi menjadi satu *Wireless Local Area Network* (WLAN). Pada jaringan WiFi, *station* hanya dapat terkoneksi ke satu *Access Point (Upstream Connection)*, sedangkan *Access Point* dapat terhubung ke

beberapa *station* secara bersamaan *(Downstream Connection)*. Akan tetapi, Esp Mesh memungkinkan untuk setiap node bertindak sebagai *station* dan *access point* secara bersamaan. Oleh karena itu setiap node dalam *Esp Mesh* dapat memiliki beberapa *downstream connection* menggunakan interface *softAP*, dan secara bersamaan memiliki sebuah *upstream connection* menggunakan interface *station*-nya. Hal ini menghasilkan topologi *tree* yang terdiri dari beberapa lapisan *(layer)*.



Gambar 2.7 Topologi Mesh

2.5.2. Tipe Node



Gambar 2.8 Tipe-Tipe Node dalam Esp Mesh

- *Root Node*

  Root Node adalah node paling atas pada jaringan dan berfungsi sebagai satu-satunya interface antara jaringan *Esp Mesh* dan eksternal IP network. *Root node* terhubung ke WiFi router kovensional dan mengirimkan paket dari node yang berada dalam jaringan *Esp Mesh* ke eksternal IP network ataupun sebaliknya. Dalam jaringan *Esp Mesh*, hanya diperbolehkan memiliki satu *root node*.

- *Leaf node*

  Leaf node node yang tidak diizinkan untuk memiliki *child node* (tidak ada *downstream connection*). Oleh karena itu *leaf node* hanya dapat mengirim paketnya sendiri dan tidak dapat menerima atau meneruskan paket dari node lainnya. Apabila sebuah node berada pada lapisan atau *layer* terakhir, maka

node tersebut akan ditugaskan sebagai *leaf node*. Hali ini untuk mencegah agar node tidak membuat *downstream connection* dan memastikan bahwa jaringan tidak membuat lapisan baru. Beberapa node yang hanya bertindak sebagai *station* dan tidak bertindak sebagai *access point* juga akan ditugaskan sebagai *leaf node*.

- *Intermediate Parent Nodes*

  Intermediate parent nodes adalah semua node selain *root node* dan *leaf node* yang terhubung ke jaringan *Esp Mesh*. Sebuah *intermediate parent node* harus memiliki satu *upstream connection*, tapi tidak harus memiliki *multiple downstream connection*. Oleh karena itu, sebuah *intermediate parent node* dapat mengirim dan menerima paket, serta dapat juga meneruskan paket dari node lainnya.

- *Idle Node*

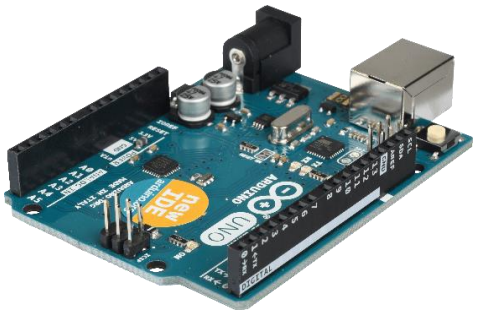
  Idle node adalah node yang belum terhubung ke jaringan *Esp Mesh*. *Idle node* akan mencoba untuk membetuk *upstream connection* dengan *intermediate parent node* atau mencoba untuk menjadi *root node*.

2.6. Mikrokontroller

Mikrokontroller adalah sebuah chip yang berfungsi sebagai pengendali/pengontrol alat elektronik dan umumnya dapat menyimpan program di dalamnya. Mikrokontroller tersusun dari CPU (*Central Processing Unit*), memori, *port input/output* tertentu dan unit pendukung tertentu seperti *Analog to Digital Converter* (ADC) yang sudah terintegrasi di dalamnya. Adapun perbedaan antara mikrokontroller dan mikroprosesor adalah pada mikroprosesor tidak memiliki

memori, *port input/output*, dan komponen pendukung lainnya. Beberapa contoh mikrokontroller adalah sebagai berikut:

2.6.1. Arduino Uno



Gambar 2.9 Arduino Uno

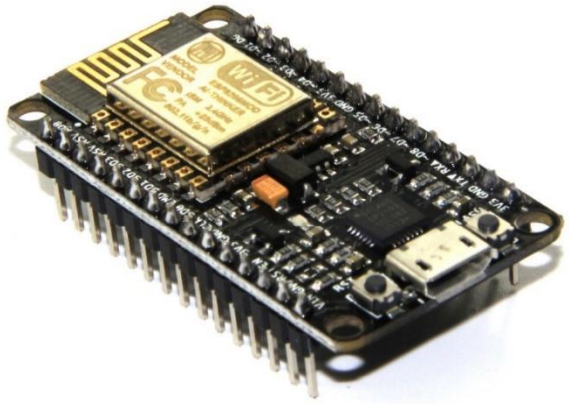
Arduino merupakan sebuah board mikrokontroller yang bersifat open source berbasis ATmega328. Arduino memiliki 14 pin input/output digital yang mana 6 pin dapat digunakan sebagai output PWM, 6 analog input, crystal osilator 16 MHz, koneksi USB, jack power, kepala ICSP, dan tombol reset. Berikut merupakan spesifikasi board Arduino Uno:

Tabel 2.1 Spesifikasi Arduino Uno

Tegangan Operasi	5 V
Tegangan Input	7-12 V
Batas Tegangan Input	6-20 V
Pin I/O Digital	14 (dimana 6 pin output PWM)
Pin Input Analog	6
Arus DC per pin I/O	40 mA
Arus DC untuk pin 3,3 V	50 mA
Flash Memory	32 KB

SRAM	2 KB
EEPROM	1 KB
Clock	16 MHz
LED_BUILTIN	Pin 13

2.6.2. NodeMCU Esp8266



Gambar 2.10 NodeMCU Esp8266

NodeMCU merupakan board pengembangan produk *internet of things* (IoT) yang berbasiskan firmware eLua dan *y* (SoC) ESP8266-12E menggunakan mikroprosesor Tensilica Xtensa 32-bit LX106. ESP8266 sendiri merupakan *chip* WiFi dengan protocol stack TCP/IP yang lengkap. Berikut merupakan spesifikasi board NodeMCU Esp8266:

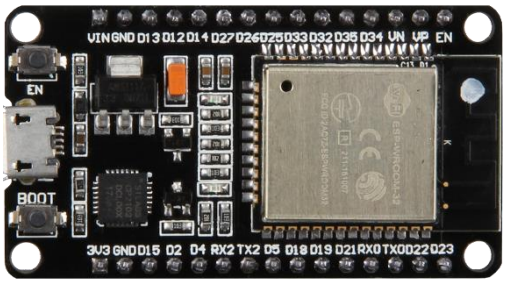
Tabel 2.2 Spesifikasi NodeMCU Esp8266

Prosesor	Tensilica Xtensa 32-bit LX106
Flash Memory	4 KB
Tegangan Operasi	3,3 V
Tegangan Input	7-12 V

Pin I/O Digital	16
Pin Input Analog	1 (10 bit)
Interface UART	1
Interface SPI	1
Interface I2C	1

Untuk tegangan kerja Esp8266 menggunakan standar tegangan JEDEC (3,3V) untuk bisa berfungsi. Tidak seperti mikrokontroller AVR dan Sebagian besar board Arduino yang memiliki tegangan TTL 5 V. Meskipun begitu, NodeMCU masih bisa terhubung dengan 5 V namun melalui *port micro USB* atau pin Vin yang disediakan pada boardnya.

2.6.3. ESP32



Gambar 2.11 ESP32

ESP32 merupakan sebuah mikrokontroller yang dibuat oleh Espressif yang merupakan penerus dari ESP8266. ESP32 mampu bekerja dengan baik dalam lingkungan industrial, dengan suhu operasi antara -40ºC sampai 125ºC. ESP32

menggunakan konsumsi daya yang dangat rendah melalui fitur hemat daya *fine resolution clock gating, multiple power modes,* dan *dynamic power scaling*. Menggunakan mikroprosesor *Tensilica Xtensa LX6 dual-core* dengan *clock rate* hingga 240 Mhz. ESP32 sudah terintegrasi dengan *RF balun, power amplifier, low-noise receive amplifier*, *filters*, dan *power management modules.* Berikut merupakan spesifikasi dari ESP32:

Tabel 2.3 Spesifikasi ESP32

Prosesor	*Xtensa LX6 dual-core*, sampai 600 MIPS
SRAM	520 KB 16 KB SRAM pada RTC
ROM	448 KB
Peripheral Interface	• 12 bit SAR ADC sampai 18 channel • 2 x 8 bit DAC • 10 touch sensors • Temperature sensors • 4 SPI • 2 I2S interfaces • 2 I2C interfaces • 3 UART • SD/SDIO/CE-ATA/MMC/eMMC host controller • SDIO/ SPI slave controller • Ethernet MAC interface dengan dedicated DMA dan IEEE 1588 Precision Time Protocol

	• CAN bus 2.0 • Infrared remote controller (RX/TX, sampai 18 channel) • Motor PWM • LED PWM (sampai 16 channel) • Hall effect sensor • Ultra low power analog pre-amplifier

Ada beberapa keunggulan Esp32 dibanding Esp8266 seperti prosesornya yang sudah menggunakan dual core, sudah mempunyai Bluetooth Low Energy, dan lain-lain. Berikut merupakan perbedaan antara Esp8266 dan Esp32:

Tabel 2.4 Perbedaan Esp8266 dan Esp32

Varian	**Esp8266**	**Esp32**
MCU	Xtensa Single-Core 32-bit L106	Xtensa Dual-Core 32-bit LX6
WiFi	802.11 b/g/n tipe HT20	802.11 b/g/n tipe HT40
Bluetooth	Tidak ada	Tipe 4.2 BLE
Typical Frequency	80 MHz	160 MHz
SRAM	Tidak Ada	Ada
Total GPIO	17	25
Total SPI-UART-I2C-I2S	2-2-1-2	4-2-2-2
Resolusi ADC	10 bit	12 bit

Suhu Operasional Kerja	-40 ^{o}C sampai 125 ^{o}C	-40 ^{o}C sampai 125 ^{o}C
Sensor didalam Modul	Tidak Ada	Touch Sensor, Temperature Sensor, Hall Efect Sensor
Ethernet MAC Interface	Tidak Ada	Ada

2.7. Sensor

Sensor merupakan penerjemah yang mengubah sifat fisik menjadi kuantitas numerik. Beberapa contoh sensor adalah sebagai berikut:

2.7.1. Sensor Ultrasonik

Gambar 2.12 Sensor Ultrasonik

Sensor ultrasonik merupakan sensor yang paling sering digunakan untuk mengukur jarak pada aplikasi *embedded system*. Biasanya pada 1 set modul terdapat satu buah *transmitter* sinyal dan satu buah *receiver* sinyal ultrasonik.

Jarak minimum yang bisa dideteksi oleh sensor ultrasonik adalah 2 cm, sementara untuk jarak maksimumnya adalah 4,5 m. Akan tetapi, ketika digunakan untuk mengukur jarak diatas 2 meter, akurasi pengukuran akan mengalami penurunan. Hal ini disebabkan karena hasil pengukuran tidak konsisten dari waktu ke waktu. Untuk mendapatkan hasil yang konsisten, maka perlu dilakukan beberapa

kali pengukuran dalam rentang waktu tertentu. Nilai dari hasil pengukuran kemudian di rata-ratakan untuk dijadikan hasil dari jarak yang didapatkan oleh sensor. (Susilawati, 2019)

Spesifikasi dari sensor ultrasonik adalah sebagai berikut :

- Tegangan : 3,3V - 5V DC
- Arus Statis : < 2mA
- Level Output : 5V – 0V
- Sudut sensor : <15 derajat
- Jarak yang bisa dideteksi : 2 cm – 450 cm

2.7.2. Sensor pH

Gambar 2.13 Sensor pH

Sensor pH adalah sebuah alat eletronik yang berfungsi untuk mengukur derajat keasaman (pH) dari suatu cairan. Prinsip kerja dari sensor ini yaitu semakin banyak elektron pada sampel maka akan semakin bernilai asam dan begitupun sebaliknya.

Probe pH mengukur pH seperti aktifitas ion-ion hydrogen yang mengelilingi bohlam kaca berdinding tipis pada ujungnya. Probe ini menghasilkan tegangan rendah (sekitar 0,06 volt per unit pH) yang diukur dan ditampilkan sebagai

pembacaan nilai pH. Untuk pengukuran yang lebih presisi, sensor pH harus dikalibrasi sebelumnya. (Nur, 2010)

Spesifikasi dari sensor pH 4502C adalah sebagai berikut:

- Tegangan Kerja : 5V DC
- Arus Kerja : 5-10 mA
- Respon Time : <= 5 s
- Output : Signal Analog

2.7.3. Sensor Salinitas

Gambar 2.14 Sensor Salinitas

Salinitas adalah tingkat kadar garam atau keasinan terlarut dalam air. Jika dilihat dari definisi, air tawar adalah air yang memiliki tingkat salinitas dibawah 0,05 ppt (Al Barqi, 2011). Salah satu cara untuk mengukur konsentrasi kadar garam pada air adalah dengan menyelupkan dua buah elektroda dengan jarak tertentu dan menghantarkan listrik pada salah satunya. Kemudian nilai tegangan dibaca pada elektroda lainnya untuk dihitung perbedaannya. (Suparta, 2018)

Spesifikasi dari sensor salinitas adalah sebagai berikut :

- Bekerja pada tegangan DC 5V
- Memiliki sensitivitas pada bahan yang bersifat konduktif

- Kedalaman cairan pada saat pengukuran sebesar 5,5 cm dari ujung sensor

2.7.4. Sensor DS18B20

Gambar 2.15 Sensor DS18B20

Modul ini merupakan termometer digital yang menyediakan 9 bit sampai 12 bit pengukuran temperatur *celcius* dan memiliki fungsi alarm dengan pemicu titi katas dan bawah yang dapat deprogram. Komunikasi dengan mikrokontroller dilakukan melalui 1 buah yang berfungsi sebagai *data line* sekaligus *power line*. Setiap modul DS18B20 memiliki 64 bit kode serial yang unik sehingga pengguna dapat memasang modul yang sama dengan menggunakan 1 kabel. (Al Barqi, 2011)

Beberapa fitur yang terdapat pada sensor DS18B20 adalah sebagai berikut:

- Interface menggunakan 1-wire sebagai komunikasi data
- Terdapat pengenal unik 64 bit pada setiap sensor
- Dapat mengukur suhu dari rentang -55°C sampai +125°C
- Keakurasian sensor yaitu sekitar 0,5°C pada suhu -10°C sampai +85°C
- Dapat mengkonversi data suhu 12 bit hanya membutuhkan waktu 750 ms
- Dapat mengkonfigurasi alarm yang dapat disetting

2.8. *Data Loss*

Data loss atau *packet loss* adalah banyaknya paket yang hilang selama proses transmisi data dari sumber ke tujuan, *data loss* dapat terjadi karena tabrakan data (*collision*), penurunan kekuatan sinyal, dan kesalahan hardware pada jaringan. Dalam suatu jaringan *data loss* akan selalu mempunyai nilai dengan satuan persen (%). (Nursan, 2013)

Data loss dapat dihitung menggunakan rumus:

$$Data\ Loss = \frac{Paket\ data\ yang\ dikirim - Paket\ data\ yang\ diterima}{Paket\ data\ yang\ dikirim} \times 100\% \qquad (2.1)$$