BAB 1

PENDAHULUAN

1.1 Latar Belakang

Wireless Sensor Network (WSN) adalah suatu jaringan nirkabel yang terdiri dari kumpulan node sensor dengan kemampuan *sensing*, komputasi, dan komunikasi yang tersebar pada suatu tempat. Setiap sensor akan mengumpulkan data dari area yang dideteksi seperti temperatur, suara, getaran, tekanan, gerakan, kelembaban udara dan deteksi lainnya tergantung kemampuan sensor tersebut. Data yang diterima ini kemudian akan diteruskan ke *base station* untuk diolah sehingga memberikan suatu informasi. WSN dapat diimplementasikan pada berbagai bidang kehidupan manusia diantaranya bidang militer untuk deteksi musuh, bidang pertanian untuk pemantauan pertumbuhan tanaman, bidang kesehatan, deteksi bahaya dan bencana alam, bidang pembangunan dan tata kota, dan bidang pendidikan.

Terdapat dua macam arsitektur WSN, yaitu hierarki dan flat. Pada arsitektur hierarki, node sensor akan disusun secara berkelompok (*cluster*) dan terdapat node sensor yang memiliki peran sebagai *cluster head*. *Cluster head* berfungsi untuk mengumpulkan data dari node sensor pada suatu *cluster* dan mengirimkan data tersebut ke *base station*. Sedangkan pada arsitektur flat hanya terdapat dua macam node sensor secara fungsional, yaitu *source node* dan *sink node*. Setiap node sensor (*source node*) akan mengirim data ke satu tujuan akhir yaitu *sink node* atau *base station*. Pada arsitektur flat data dari sebuah node sensor dapat diteruskan ke node tetangganya dan seterusnya hingga sampai ke *base station*.

Dalam praktiknya, pengiriman data merupakan suatu hal yang penting pada WSN. Data yang didapat dari sensor harus sampai ke *base station* dengan utuh dan akurat (*reliable*). Data yang *reliable* ini sangat penting karena hasil pengukuran dan tindakan selanjutnya yang akan diambil akan bergantung pada data-data tersebut. Terdapat beberapa protokol untuk memastikan transfer data *reliable* yaitu dengan protokol *Event to Sink Reliable Transport*,*Reliable Multi Segment Transport*, *Price Oriented Reliable Transport*, *Delay Sensitive Transport*, dan lain-lain.

Pada skripsi ini dibangun aplikasi untuk transfer data pada WSN. Aplikasi WSN yang dibuat juga dapat melakukan transfer data ke node sensor tetangganya hingga sampai ke node sensor yang berperan sebagai *base station*. Karena data yang akurat sangat dibutuhkan untuk menentukan tindakan selanjutnya, maka akan dibangun juga WSN yang memiliki sifat *reliable*.

1.2 Rumusan Masalah

- Bagaimana cara membangun aplikasi transfer data dari setiap node sensor pada *Wireless Sensor Network*?
- Bagaimana cara membangun aplikasi transfer data yang *reliable* pada *Wireless Sensor Network*?

1.3 Tujuan

- Membangun aplikasi transfer data yang *reliable* pada *Wireless Sensor Network*.

1.4 Batasan Masalah

Penelitian ini dibuat berdasarkan batasan-batasan sebagai berikut:

1. Sensor yang digunakan sebagai penelitian hanya sensor untuk mengukur temperatur, kelembapan, getaran, dan tekanan udara.
2. Arsitektur yang digunakan untuk membangun *Wireless Sensor Network* ini adalah flat dan hirarkikal dengan multi-hop.

1.5 Metodologi

Berikut adalah metode penelitian yang digunakan dalam penelitan ini:

1. Melakukan studi literatur mengenai *Wireless Sensor Network*.
2. Mempelajari protokol transfer data yang biasa pada *Wireless Sensor Network*.
3. Mempelajari prinsip *Reliable Data Transfer* pada *Wireless Sensor Network*.
4. Mempelajari pemrograman pada *Wireless Sensor Network* dengan Bahasa Pemrograman JAVA.
5. Melakukan perancangan perangkat lunak.
6. Mengimplementasi rancangan perangkat lunak pada *Wireless Sensor Network*.

1.6 Sistematika Pembahasan

Setiap bab dalam penelitian ini memiliki sistematika penulisan yang dijelaskan ke dalam poin-poin sebagai berikut:

1. Bab 1: Pendahuluan yaitu membahas mengenai gambaran umum penelitian ini. Berisi tentang latar belakang, rumusan masalah, tujuan, batasan masalah, metode penelitian, dan sistematika penulisan.
2. Bab 2: Dasar Teori, yaitu membahas teori-teori yang mendukung berjalannya penelitian ini. Berisi tentang *Wireless Sensor Network*, Reliable Data Transfer di WSN, dan PreonVM.
3. Bab 3: Analisis, yaitu membahas mengenai analisis masalah. Berisi tentang analisis pengiriman data dari node sensor ke base station pada WSN, analisis protokol yang reliable saat melakukan pengiriman data dari node sensor ke base station pada WSN, dan analisis aplikasi pengiriman data pada WSN.
4. Bab 4: Perancangan, yaitu membahas perancangan aplikasi WSN yang reliable untuk pengiriman data
5. Bab 5: Implementasi dan Pengujian, yaitu membahas implementasi dari hasil rancangan dan pengujian dari aplikasi WSN yang telah dibuat.
6. Bab 6: Kesimpulan, yaitu membahas kesimpulan dari hasil pengujian.

BAB 2

LANDASAN TEORI

Pada bab ini dijelaskan dasar-dasar teori mengenai *Wireless Sensor Network*, Reliable Data Transfer di WSN, dan pemrograman pada WSN.

2.1 Wireless Sensor Network

Wireless Sensor Network (WSN) merupakan jaringan nirkabel yang terdiri dari sekumpulan node sensor yang diletakan pada suatu tempat dan memiliki kemampuan untuk mengukur kondisi lingkungan sekitar(*sensing*), melakukan komputasi dan dilengkapi dengan alat komunikasi *wireless* untuk komunikasi antara node sensor. Sensor ini akan mengumpulkan data dari kondisi lingkungannya, seperti: cahaya, suara, kelembapan, getaran, gerakan, temperatur, tekanan udara, kualitas air, komposisi tanah, dan lain-lain. Data ini kemudian dapat dikirimkan langsung ke *base station* atau diteruskan melalui node sensor tetangganya hingga sampai ke *base station* sebagai pusat untuk dikelola.

2.1.1 Penerapan Wireless Sensor Network

Pada awalnya *sensor network* (jaringan sensor) digunakan dalam teknologi militer untuk mendeteksi musuh di laut dan di darat. Semakin lama node sensor ini banyak dikembangkan untuk membantu berbagai bidang kehidupan manusia. Pemanfaatan WSN pada kehidupan manusia dapat dilihat pada ilustrasi Gambar 2.1. Berikut adalah beberapa penerapan WSN:

- Bidang Militer
  Pada bidang militer WSN digunakan untuk melakukan pemantauan gerakan musuh dan melindungi wilayah. WSN juga dapat digunakan untuk mendeteksi serangan dari musuh.

- Monitoring area
  Pada *monitoring area*, node sensor akan disebar pada suatu tempat yang akan di monitoring. Saat node sensor mendeteksi kejadian(panas, tekanan, dan lain-lain) pada suatu tempat, data akan dikirimkan ke *base station* untuk ditentukan tindakan selanjutnya.

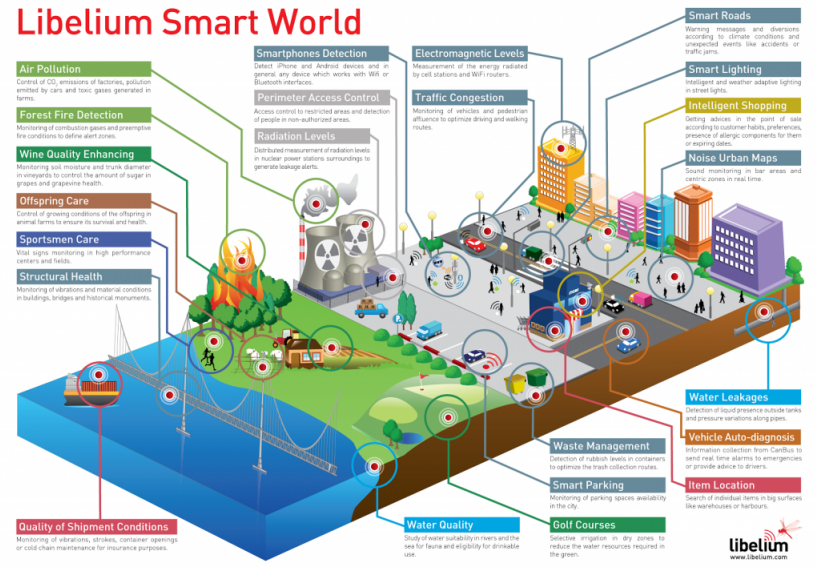
- Bidang Transportasi
  Pada bidang transportasi, WSN digunakan untuk mendeteksi arus lalu lintas secara aktual yang nantinya akan disampaikan kepada pengendara seperti kemacetan lalu lintas.

- Bidang Kesehatan
  WSN dapat digunakan pada aplikasi kesehatan seperti membantu pada disabilitas, monitoring pasien, diagnosis, pengaturan penggunaan obat, dan pelacakan dokter dan pasien di rumah sakit.

- Deteksi Lingkungan
  Deteksi lingkungan yang dapat dilakuan antara lain deteksi gunung berapi, polusi udara, kebakaran hutan, efek rumah kaca, dan deteksi longsor.

- Monitoring Struktur
  WSN dapat melakukan deteksi pergerakan bangunan dan infrastruktur seperti jembatan, *flyover*, terowongan dan fasilitas lain tanpa mengeluarkan biaya untuk melakukan deteksi manual dengan mendatangi tempatnya secara langsung.

- Bidang Pertanian
  Pada bidang pertanian dapat membantu pengelola pertanian untuk pemantauan penggunaan air dalam irigasi dan mengelola buangan pertanian mereka.

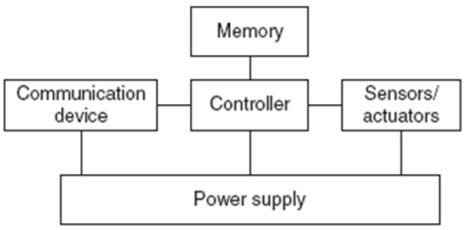


Gambar 2.1: Ilustrasi Pemanfaatan *Wireless Sensor Network*

2.1.2 Node Sensor

Struktur Node Sensor

Setiap node sensor memiliki kemampuan deteksi, komputasi dan komunikasi. Node sensor memiliki lima komponen utama yaitu *controller*, *memory*, *sensor dan actuator*, *communication device*, dan *power supply*, (Gambar 2.2). Semua komponen akan bekerja secara seimbang dalam melakukan *sensing*, komputasi, komunikasi, dan menjaga penggunaan energi seminimal mungkin.



Gambar 2.2: Struktur Node Sensor

Controller

Controller adalah inti utama pada node sensor. *Controller* mengumpulkan data dari sensor dan memproses data tersebut hingga menentukan kapan dan kemana data tersebut dikirim. *Controller* juga dapat menerima data dari node sensor lain. Pada *controller* biasanya terdapat *microcontroller* atau *microprocessor* yang mengatur dan melakukan komputasi data. *Microcontroller* ini juga dapat

mengurangi penggunaan energi dengan adanya *sleep states* yang berarti hanya bagian dari controller saja yang aktif.

Beberapa *microcontroller* yang digunakan dalam *Wireless Sensor Node*:

- Intel StrongARM (32-bit RISC, up to 206 MHz)
- Texas Instrument MSP 430 (16-bit RISC, up to 4 MHz,RAM 2-10 kB)
- Atmel Atmega 128L (8-bit)

Memory

Random Access Memory (RAM) digunakan untuk menyimpan sementara hasil yang didapat dari sensor. RAM juga menyimpan sementara paket dari node sensor lain. Jika node sensor mati atau energi habis maka data pada RAM ini akan hilang (*volatile*). Data yang hilang saat node sensor mati merupakan salah satu kekurangan dari penggunaan RAM. Untuk itu dalam menyimpan kode program digunakanlah *Read Only Memory* (ROM). ROM ini biasa disebut *Electrically Erasable Programmable Read-Only Memory* (EEPROM) atau *Flash Memory*.

Communication Device

Communication Device digunakan untuk bertukar data antar node sensor. Pada aplikasi WSN, *Radio Frequency(RF)* adalah media komunikasi yang paling relevan untuk saat ini. RF-based mendukung jangkauan yang jauh, memiliki data rate yang tinggi dan tidak perlu saling mengetahui posisi antara penerima dan pengirim.

Pada node sensor dibutuhkan *transmitter* untuk mengirim data dan *receiver* untuk menerima data. Kedua hal ini dapat digabung dan disebut dengan *transceiver*. Tugas *transceiver* adalah mengubah aliran *bit* menjadi gelombang radio. Selain itu *transceiver* juga dapat mengubah gelombang radio menjadi aliran *bit*.

Sensor dan Actuator

Sensor dan Aktuator adalah hal yang penting pada WSN. Tanpa sensor dan aktuator maka node sensor tidak berguna dan tidak dapat digunakan. Tabel 2.1 adalah jenis - jenis sensor yang dapat dimiliki node sensor. Sensor dikategorikan menjadi tiga:

1. **Passive, omnidirectional sensors** Sensor ini dapat mengukur kualitas dari lingkungan fisik tempat node sensor tersebut tanpa mengubah lingkungannya. Beberapa sensor dikategori ini *self-powered* yaitu sensor mendapatkan energi yang mereka butuhkan dari lingkungannya. *Omnidirectional* berarti tidak ada arah pada sensor ini. Sensor akan memancarkan sinyalnya ke segala arah. Contoh sensor ini adalah termometer, sensor cahaya, sensor getaran, mikrofon, sensor kelembapan, sensor tekanan udara, dan sensor deteksi asap.

2. **Passive, narrow-beam sensors** Sensor ini memiliki sifat yang sama dengan sensor *Passive, omnidirectional sensors* yaitu tidak mengubah lingkungannya. Sensor ini dapat melakukan gerakan dan memiliki arah atau daerah pengukuran. Contoh dari sensor ini adalah kamera yang bisa mengukur sesuai dengan arah yang dituju.

3. **Active Sensor** Sensor ini aktif dalam memeriksa lingkungannya. Contoh dari sensor ini adalah sonar, radar atau sensor seismik. Sensor ini menghasilkan gelombang untuk melakukan deteksi.

Aktuator adalah penerima sinyal dan yang mengubahnya menjadi aksi fisik. Aktuator jumlahnya beragam seperti sensor. Contoh aktuator adalah LED, yang mengubah listrik menjadi cahaya dan motor (motor elektrik) juga mengubah listrik menjadi gerakan.

Tabel 2.1: Jenis - jenis sensor yang dapat dimiliki node sensor

Sensor	Penggunaan
Accelerometer	Pergerakan 2D & 3D untuk objek dan manusia
Acoustic emission sensor	Elastic Waves Generation
Acoustic sensor	Acoustic pressure vibration
Capacitance sensor	Solute Concentration
ECG	Heart Rate
EEG	Brain Electric Activity
EMG	Muscle Activity
Electrical/electromagnetic sensor	Electrical Resistivity
Gyroscope	Angular Velocity
Humidity Sensor	Mendeteksi Humidity
Infrasonic sensor	Gelombang untuk deteksi gempa dan volkanik
Magnetic sensor	Mendeteksi magnetik
Oximeter	Tekanan Oxigenation pada darah
pH sensor	Tingkat Keasaman
Photo acoustic spectroscopy	Gas Sensing
Piezoelectric cylinder	Gas Velocity
Soil moisture sensor	Mengukur tanah
Temperature sensor	Temperatur
Barometer sensor	tekanan air
Passive infrared sensor	Pergerakan infrared
Seismic sensor	Pergerakan Seismik (Gempa)
Oxygen sensor	Oksigen pada darah
Blood flow sensor	Gelombang ultrasonik pada darah

Power Supply

Power supply atau sumber energi pada WSN dapat berasal dari dua cara yaitu ***storing energy*** dan ***energy scavenging***. *Storing energy* adalah penggunaan baterai sebagai sumber energinya. Baterai yang digunakan dapat diisi ulang maupun tidak dapat diisi ulang. *Energy scavenging* digunakan saat membuat WSN yang akan digunakan dalam waktu yang lama. Dibutuhkan energi yang bisa dikatakan tidak terbatas. Salah satu cara *energy scavenging* adalah *photovoltaics*. *Photovoltaics* dapat disebut juga *solar cell* yang memanfaatkan cahaya matahari dan mengubahnya menjadi energi sebagai pembangkit daya. Cara lain yang dapat digunakan adalah pemanfaatan angin dan air untuk mengerakan kincir atau turbin yang akan menghasilkan listrik dan digunakan sebagai sumber energi pada node sensor.

2.1.3 Arsitektur dan Topologi Wireless Sensor Network

Pada WSN biasanya akan terdapat banyak node sensor yang disebar pada suatu tempat. Terdapat satu atau lebih *sink node* atau *base station* dalam area sensing tersebut (Gambar 2.10). *Sink node* atau *base station* adalah node sensor yang bertugas untuk mendapatkan data dari node sensor lain. Dalam membuat WSN perlu diperhatikan arsitektur dan topologi yang akan digunakan. Tidak semua topologi jaringan komputer dapat digunakan untuk *Wireless Sensor Network*.

Ada banyak topologi pada jaringan sensor (*sensor network*). Pada jaringan sensor dengan menggunakan kabel, topologi yang biasa digunakan adalah topologi *star*, *line*, atau *bus*. Sedangkan pada jaringan sensor tanpa kabel (WSN), topologi yang biasa digunakan adalah *star*, *tree*, atau *mesh*.

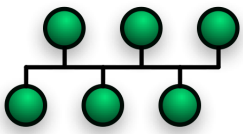
Topologi Point-to-Point

Topologi *Point-to-Point* adalah topologi yang menghubungkan dua titik (Gambar 2.3). Topologi *Point-to-Point* dibagi menjadi dua yaitu *permanent point-to-point* dan *switched point-to-point*. *Permanent point-to-point* adalah koneksi perangkat keras antara dua titik dan tidak dapat diubah. *Switched point-to-point* adalah koneksi *point-to-point* yang dapat berpindah antara node yang berbeda.

Gambar 2.3: Topologi Point-to-Point

Topologi Bus

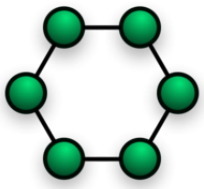
Topologi Bus seperti pada Gambar 2.4 akan terdiri dari node-node dan sebuah jalur. Setiap node akan terhubung dengan satu jalur yang sama. Untuk mengirim data atau komunikasi akan dilakukan bergantian antar node. Kekurangan dari topologi bus ini adalah jika suatu saat jalur atau bus ini mengalami kerusakan maka setiap node tidak dapat saling berkomunikasi lagi.



Gambar 2.4: Topologi Bus

Topologi Ring

Pada Topologi *Ring* node akan disusun dengan bentuk melingkar (Gambar 2.5). Setiap node akan terkoneksi dengan dua node lain. Transfer data terjadi dengan cara data akan berjalan dari satu node ke node lain mengikuti jalur melingkar tersebut hingga menemukan node tujuan yang tepat. Topologi ini mudah untuk diimplementasikan tapi kekurangan dari topologi ring adalah saat ada node yang rusak maka perlu biaya lebih untuk memperbaikinya. Biasanya untuk menangani kegagalan komunikasi akibat node yang rusak, akan diatur komunikasi node tidak hanya satu arah tetapi dapat ke arah sebaliknya.



Gambar 2.5: Topologi Ring

Topologi Star

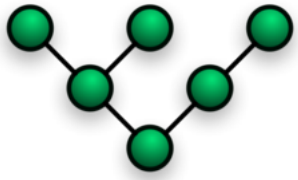
Topologi *Star* terdiri dari satu node yang berada di tengah biasanya berupa *hub* atau *switch* seperti pada Gambar 2.6. Setiap node akan terkoneksi dengan node yang berada di tengah ini. Saat node

akan berkomunikasi dengan node lain, node tersebut harus mengirimkan data tersebut ke node yang ada ditengah. Setelah itu node yang berada ditengah ini akan meneruskan data tersebut ke node tujuan. Hal yang paling penting pada topologi ini adalah node yang berada di tengah, karena semua komunikasi harus melalui node tersebut. Jika node tengah mengalami kerusakan maka tidak akan terjadi komunikasi antar node pada jaringan tersebut.

Gambar 2.6: Topologi Star

Topologi Tree

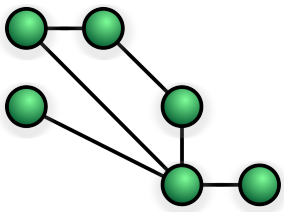
Pada Topologi *Tree* node-node akan disusun secara hierarki dengan satu node yang berada pada level paling atas sebagai *root node* (Gambar 2.7). *Root node* akan terhubung dengan satu atau lebih node level dibawahnya. Dengan Topologi *Tree* lebih mudah untuk melakukan identifikasi dan meminimalisir kesalahan, namun jika *tree* sudah sangat besar atau *level tree* sudah sangat banyak maka akan sulit untuk melakukan konfigurasi.



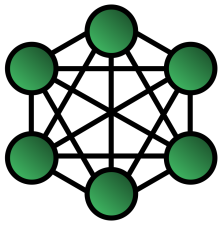
Gambar 2.7: Topologi Tree

Topologi Mesh

Topologi *Mesh* dibagi menjadi dua yaitu *partially connected mesh* dan *fully connected mesh*. Pada *partially connected mesh* (Gambar 2.8), node akan terhubung dengan lebih dari satu node. Pada *fully connected mesh* (Gambar 2.9), setiap node akan terhubung dengan semua node lain pada jaringan tersebut.

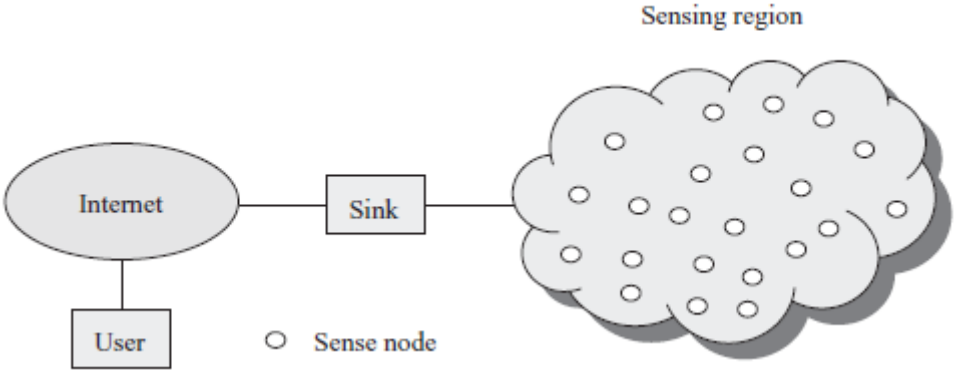


Gambar 2.8: Topologi Partially Connected Mesh



Gambar 2.9: Topologi Fully Connected Mesh

Arsitektur yang biasanya dipakai pada WSN adalah **arsitektur flat atau peer-to-peer** dan **arsitektur hierarki**. Selain itu dalam membangun WSN perlu juga diperhatikan jalur komunikasi yang digunakan untuk menghubungkan antar node sensor saat transfer data. Untuk area *sensing* yang tidak terlalu luas dan hanya menggunakan sedikit node sensor dapat menggunakan cara komunikasi ***single hop***. Sedangkan untuk daerah yang luas dan memerlukan banyak node sensor dapat menggunakan cara komunikasi ***multi hop***.



Gambar 2.10: Arsitektur Wireless Sensor Network

Single-Hop dan Multi-Hop

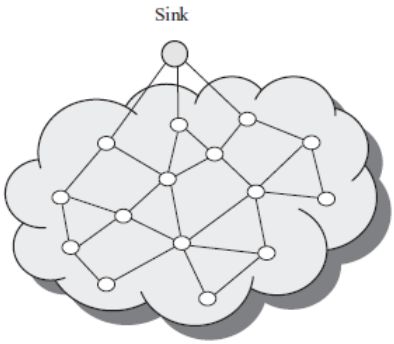
Untuk mengirim data ke *sink node* setiap node sensor dapat menggunakan ***single-hop long-distance transmission***. *Single-hop long-distance* ini berarti setiap node sensor akan mengirimkan data ke sink node hanya satu kali lompatan walaupun jarak antara sink node dengan node sensor itu sangat jauh. Dalam jaringan sensor, penggunaan energi paling besar adalah saat melakukan komunikasi dibandingan saat sensing. Penggunaan energi akan semakin bertambah jika jarak sink dan node sensor semakin jauh. Untuk menangani masalah tersebut muncul protokol *multi-hop*.

Pada protokol ***multi-hop***, node sensor akan disusun saling berdekatan dan terhubung dengan yang lain. Jadi saat akan mengirimkan data ke *sink node*, node sensor harus mengirimkan data tersebut ke node sensor tetangganya dan diteruskan hingga sampai ke *sink node*. Karena jarak yang saling berdekatan maka penggunaan energi dapat efektif. *Single-hop* dan *multi-hop* ini dapat digunakan pada topologi flat maupun hierarki sesuai dengan kebutuhan.

Arsitektur Flat / Peer-to-Peer

Pada arsitektur flat, setiap node sensor memiliki peran atau *role* yang sama dalam melakukan *sensing*. Secara fungsional hanya terdapat dua macam node sensor pada arsitektur flat, yaitu *source node* dan *sink node*. Untuk mendapatkan data dilakukan dengan cara *sink node* melakukan pengiriman data ke semua node sensor pada area *sensing* dengan cara *flooding* dan hanya node sensor yang sesuai yang akan merespon *sink node*. Gambar 2.11 adalah ilustrasi dari arsitektur flat.

Setiap node sensor mengirimkan data ke *sink node* dengan *multi-hop* dan melalui node tetangganya yang terhubung dengannya untuk meneruskan data.

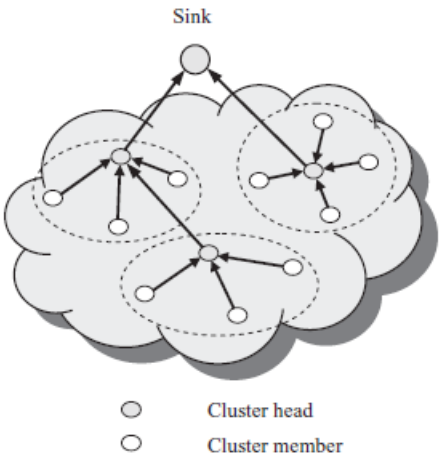


Gambar 2.11: Arsitektur flat pada *Wireless Sensor Network*

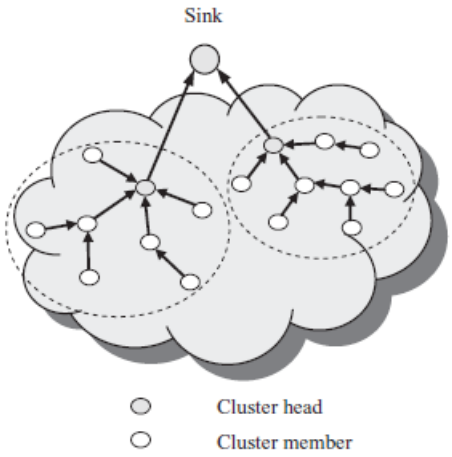
Arsitektur Hierarki

Pada arsitektur hierarki, semua node sensor dikelompokan ke dalam cluster-cluster. Terdapat *cluster head* pada setiap cluster. *Cluster head* ini yang mengumpulkan data dari setiap node sensor yang berada pada cluster tersebut dan meneruskan data yang telah diterima ke *base station* atau *sink node*. Hal yang perlu diperhatikan pada arsitektur hierarki adalah pemilihan node sensor sebagai *cluster head* dan node sensor yang melakukan sensing. Penggunaan energi yang paling besar dalam WSN ini adalah saat melakukan komunikasi yaitu saat mengirimkan data ke node sensor lain. Maka untuk node sensor yang memiliki energi kecil dapat digunakan untuk *sensing*, karena node sensor ini hanya melakukan komunikasi ke *cluster head*. *Cluster head* harus memiliki energi atau daya yang lebih banyak, karena *cluster head* akan bertugas menerima hasil sensing node sensor di cluster tersebut dan meneruskan data ke *sink node*.

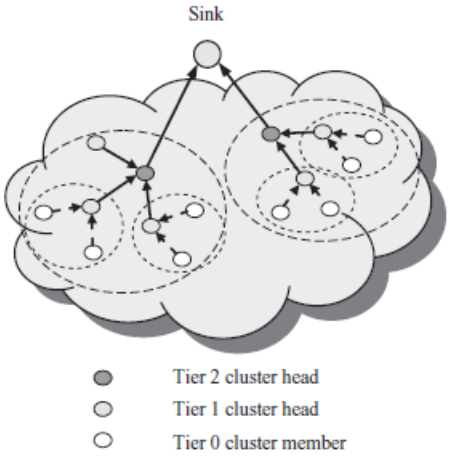
Masalah yang utama pada clustering ini adalah pemilihan *cluster head* dan bagaimana cara mengatur setiap cluster. Terdapat beberapa cara untuk membuat clustering ini. Bedasarkan jarak antara *cluster head* dengan cluster member, dapat dibuat clustering dengan *single-hop* atau *multi-hop* seperti pada Gambar 2.12 dan Gambar 2.13. Sedangkan jika berdasarkan jumlah *tier* atau tingkat dapat dibangun *clustering single tier* atau *multi tier* (Gambar 2.14).



Gambar 2.12: Arsitektur hierarki pada *Wireless Sensor Network* dengan *single hop* terhadap *Cluster Head*



Gambar 2.13: Arsitektur hierarki pada *Wireless Sensor Network* dengan *multi hop*



Gambar 2.14: Clustering dengan multi tier

2.1.4 Sistem Operasi

Setiap node sensor memerlukan sistem operasi (OS) untuk mengontrol perangkat keras dan perangkat lunak. Sistem operasi tradisional tidak dapat digunakan pada WSN. Pada sistem operasi tradisional digunakan untuk mengatur proses, memori, CPU, dan sistem berkas. Terdapat beberapa hal yang harus ditangani oleh sistem operasi dalam WSN yaitu:

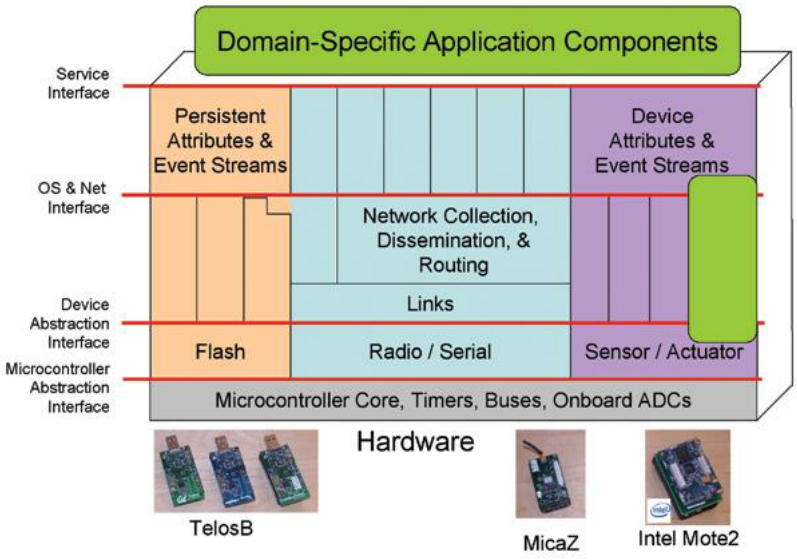
1. WSN memerlukan *real-time scheduler*. Data yang didapat harus segera dikirim atau diproses.
2. Pengaturan memori karena memori pada WSN sangat kecil.
3. Pengaturan data yang efisien terkait dengan *microprocessor* dan memori yang terbatas
4. Mendukung kode pemrograman yang efisien dan *reliable* karena dapat terjadi perubahan kode saat implementasi.
5. Mendukung pengaturan sumber energi untuk menambah waktu hidup dari node sensor dan meningkatkan performa dengan *sleep time* saat tidak ada kegiatan atau *wake up time* saat terdapat interupsi dari lingkungan.
6. Mendukung antarmuka untuk pemrograman dan antarmuka perangkat lunak.

Beberapa sistem operasi yang umum digunakan pada WSN antara lain :

1. TinyOS
2. Contiki
3. LiteOS
4. PreonVM

TinyOS

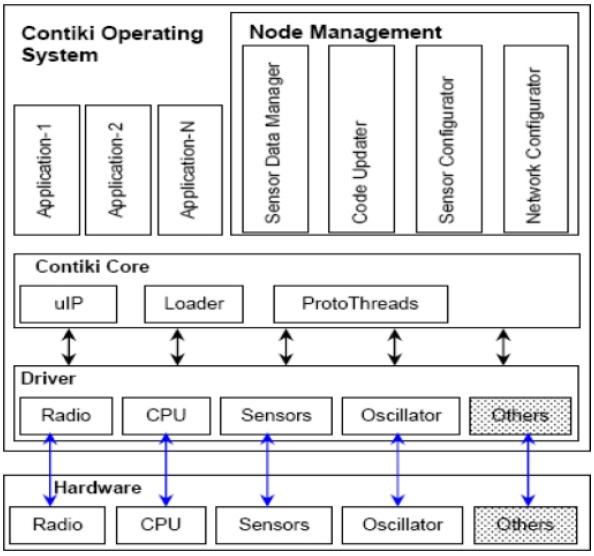
TinyOS adalah sistem operasi *open-source* yang digunakan pada WSN. TinyOS dapat menjalankan program dengan memori yang sangat kecil. Ukurannya hanya 400 Byte. Komponen *library* TinyOS terdiri dari protokol jaringan, layanan distribusi sensor, *driver sensor*, dan perangkat lunak pengamatan data sensor yang dapat digunakan untuk melakukan monitoring jaringan sensor. Gambar 2.15 adalah arsitektur pada TinyOS.



Gambar 2.15: Arsitektur TinyOS

Contiki

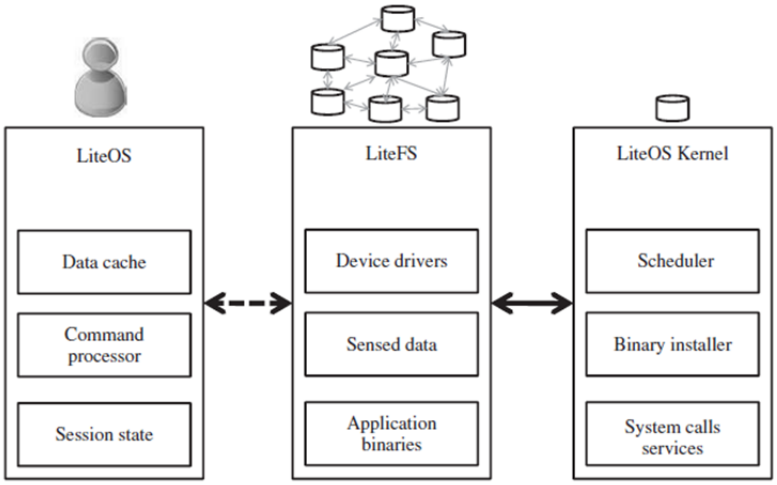
Contiki adalah sistem operasi *open-source* dengan Bahasa Pemrograman C yang digunakan pada WSN. Pengaturan Contiki hanya memerlukan 2KB dari RAM dan 40KB dari ROM. Fitur yang dimiliki oleh Contiki antara lain: *multitasking*, *multithreading*, jaringan TCP/IP, IPv6, GUI, *Web Browser*, *Web Server*, telnet, dan komputasi jaringan virtual. Gambar 2.16 adalah arsitektur pada Contiki



Gambar 2.16: Arsitektur Contiki

LiteOS

LiteOS adalah sistem operasi mirip UNIX yang didisain untuk WSN. Tujuan dibuat LiteOS adalah membuat sistem operasi yang mirip dengan UNIX agar lebih lebih familiar dengan paradigma pemrograman UNIX. Pada LiteOS terdapat sistem berkas yang hiearki, dan mendukung Bahasa Pemrograman LiteC++ dan UNIX Shell. LiteOS dapat digunakan untuk MicaZ yang memiliki 8 Mhz CPU, 128 byte flash, dan 4KB RAM. LiteOS memiliki tiga komponen utama yaitu: LiteShell, LiteFS, dan Kernel. Gambar 2.16 adalah arsitektur pada LiteOS



Gambar 2.17: Arsitektur LiteOS

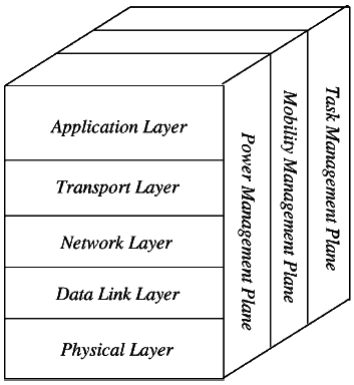
Perbandingan dari TinyOS, Contiki, dan LiteOS dapat dilihat pada Tabel 2.2 berikut ini:

Tabel 2.2: Tabel Perbandingan Sistem Operasi

OS	Programming Paradigm	Scheduling	Memory Allocation	System Call
TinyOS	Event-based	FIFO	Static	Not Available
Contiki	Predominant event-based	FIFO	Dynamic	Runtime libraries
LiteOS	Thread-based	Priority-based scheduling with optional round-robin support	Dynamic	A host of system calls available to the user (file, process, environment, debugging, and device command)

2.1.5 Protokol Stack pada Wireless Sensor Network

WSN memiliki lima layer protokol: *physical layer*, *data link layer*, *network layer*, *transport layer*, dan *application layer*, seperti pada Gambar 2.18.



Gambar 2.18: Layer pada *Wireless Sensor Network*

Selain itu protokol stack pada WSN dibagi kedalam 3 grup manajemen yaitu *power management plane*, *connection management plane*, dan *task management plane*. ***Power Management Plane*** bertanggung jawab untuk melakukan manajemen sumber daya atau energi pada setiap node sensor saat melakukan *sensing*, komputasi, mengirim dan menerima data. Contohnya pada layer MAC, node sensor dapat mematikan transceiver saat tidak ada data untuk dikirim dan diterima. Pada *network layer*, node sensor dapat memilih node sensor tetangga saat akan melakukan transfer data agar penggunaan energi yang minimal. ***Connection Management Plane*** bertanggung jawab untuk melakukan pengaturan terhadap node sensor terkait dengan koneksi antar node sensor. ***Task Management Plane*** bertanggung jawab untuk mengatur distribusi tugas atau *task* pada node sensor dalam melakukan *sensing* agar tercapai penggunaan energi yang efisien.

Physical Layer

Physical layer bertanggung jawab untuk mengubah bit stream dari *data link layer* menjadi sinyal agar bisa dilakukan transmisi melalui media komunikasi yang terdapat (transceiver). Pemilihan media dan frekuensi adalah hal yang penting dalam komunikasi antar node sensor. Salah satu cara yang bisa digunakan adalah dengan menggunakan *Radio frequency* (RF). Jaringan sensor memerlukan biaya yang kecil, ukuran yang kecil, dan penggunaan daya yang kecil untuk tranceivernya. Karena itu banyak yang menggunakan Radio Frequency (RF) untuk desain perangkat keras node sensornya.

Data Link Layer

Data link layer bertanggung jawab untuk melakukan *multiplexing* pada aliran data, membentuk *data frame*, mendeteksi *data frame*, *medium access*, dan mengatur kesalahan saat melakukan transfer data. Fungsi paling penting *data link layer* adalah *Medium Access Control* (MAC). Protokol MAC menentukan kapan node sensor mengakses media untuk mengirim data, melakukan kontrol dan mengatur paket ke node sensor lain. Hal itu dilakukan agar tidak terjadi paket yang bertabrakan.

Network Layer

Network layer bertanggung jawab untuk *routing* dari node sensor ke *sink node*. Pada WSN node sensor tersebar pada suatu tempat untuk melakukan *sensing*. Data hasil *sensing* tersebut harus dikirimkan ke *sink node* untuk diolah. Dalam mengirimkan data tersebut dapat menggunakan *single-hop* atau *multi hop*. Dalam mengirim data diperlukan protokol routing yang tepat agar hemat energi.

Transport Layer

Secara umum *transport layer* bertanggung jawab untuk pengiriman data yang *reliable* antara node sensor dan *sink node*. Protokol transpor yang biasa pada jaringan komputer tidak bisa diterapkan pada WSN tanpa modifikasi. Setiap jaringan sensor memiliki fungsi khusus. Untuk aplikasi yang berbeda memerlukan kebutuhkan reliabilitas yang berbeda. Pengiriman data pada WSN dibagi menjadi dua yaitu: ***downstream*** dan ***upstream***. ***Upstream*** berarti node sensor mengirimkan hasil *sensing* ke *sink node*. ***Downstream*** berarti data berasal dari *sink node* contohnya, kueri, dan perintah-perintah yang dikirimkan ke setiap node sensor. Aliran data yang *reliable* untuk kedua jenis pengiriman ini berbeda. Pada *upstream*, *reliable* data dapat ditoleransi karena sensor akan melakukan sensing terus menerus dan dapat terjadi pengulangan data sehingga data yang hilang tadi dapat dikoreksi. Sedangkan pada *downstream* memerlukan 100% pengiriman data yang *reliable* karena aplikasi tidak dapat berjalan jika kode program tidak lengkap.

Application Layer

Application layer meliputi berbagai macam protokol yang ada pada layer ini untuk menjalankan berbagai macam aplikasi seperti *query dissemination*, *node localization*, *time synchronization*, dan *network security*. Protokol yang ada pada layer ini antara lain:

1. Sensor management protocol (SMP) adalah protokol untuk melakukan pertukaran lokasi data, sinkornisasi node sensor, mengatur ulang node sensor, dan menyimpan status dari node sensor.
2. Sensor query and data dissemination protocol (SQDDP) adalah protokol yang mendukung antarmuka aplikasi untuk memasukan kueri, merespon kueri, dan mengumpulkan respon.
3. Sensor query and tasking language (SQTL) adalah protokol untuk mendukung bahasa pemrograman pada WSN.

2.2 Reliable Data Transfer di WSN

WSN terdiri dari *sink node* dan banyak node sensor. Saat melakukan *sensing* dan mengirim data ke *sink node*, semua node sensor akan mengirimkan data melalui media yang sama. Hal ini dapat membanjiri jaringan dengan data tersebut dan menyebabkan *congestion* atau kemacetan pada jaringan yang berakibat *data loss* atau hilang. Untuk menangani *data loss*, maka salah satu yang diperlukan dalam membangun WSN adalah protokol yang menangani *reliability*. *Reliability* berarti memastikan data yang dikirim dari setiap node sensor diterima oleh *base station* secara lengkap dan sesuai dengan urutan pengiriman.

2.2.1 Jenis Reliability

Berdasarkan tingkat, reliability ada dua macam yaitu:

- Packet Reliability
- Event Reliability

Packet Reliability berarti paket yang dikirim harus sampai kepada tujuan (*base station*) secara utuh. *Packet reliability* ini membutuhkan *acknowledge* dari node sensor. Tantangan yang harus dihadapi dari *packet reliability* adalah dalam mengirim ulang paket yang hilang akan menghabiskan energi atau daya.

Event Reliability berarti hanya data hasil *sensing* yang akan dikirim ke *base station*. Pada *Event reliability* tidak membutuhkan *acknowledge*. Karena data yang hilang itu tidak banyak berpengaruh sehingga tidak dibutuhkan pengiriman ulang (*retransmission*) data.

Berdasarkan arah, *reliability* ada dua macam yaitu:

- Upstream Reliability
- Downstream Reliability

Upstream Reliability adalah komunikasi dari node sensor ke *sink node*. Banyak protokol yang mendukung *upstream reliability*. Pengiriman data yang dilakukan adalah *unicast* yang berarti hanya dari satu titik ke titik lain.

Downstream Reliability adalah adalah komunikasi dari *sink node* ke node sensor. Pengiriman data biasanya dilakukan dengan cara *broadcast* ke semua node sensor. Data yang dikirim ini biasanya adalah kode program untuk melakukan *sensing*.

Pada protokol transport tradisional dikenal istilah TCP dan UDP, namun keduanya tidak bisa diimplementasikan pada WSN. Protokol transport tersebut dapat dibuat dengan beberapa pertimbangan seperti:

- WSN membutuhkan mekanisme untuk mengembalikan paket yang hilang seperti *acknowledge*.
- Proses awal dalam membangun koneksi seperti *handshake* harus disederhanakan karena akan membuang banyak daya atau energi.
- Protokol harus dapat menangani *congestion*.
- Protokol harus dapat adil terhadap setiap node sensor seperti pembagian penggunaan jaringan. (*bandwidth*)

Retransmission dapat dilakukan dengan *End-to-End Retransmission* dan *Hop-by-Hop Retransmission (Link Level Retransmission)*. Pada ***End-to-End Transmission*** dan terjadi *data loss*, maka pengirim harus mengirim ulang semua paket dan akan menghabiskan lebih banyak daya. *End-to-End Retransmission* adalah salah satu metode yang digunakan di Internet. Cara ini dapat memastikan *reliable* data tanpa harus mengetahui apa yang terjadi di tengah jaringan. Pada *End-to-End Retransmission* diperlukan *handshake* seperti pada komunikasi jaringan komputer. Pada awalnya data dikirim sebagai permintaan transfer. Jika penerima (*receiver*) memiliki cukup RAM, dan layer aplikasi dapat menerima data tersebut, maka *receiver* mengirimkan *acknowledge* untuk permintaan data tersebut. Saat koneksi sudah terbentuk, data yang sebenarnya dapat dikirim. Data tersusun dari beberapa *round*. Setiap *round*, pengirim (*sender*) mengirim paket yang hilang pada round sebelumnya. Diakhir setiap round, *receiver* mengirimkan *acknowledge* kepada *sender* yang berisi informasi paket yang hilang. *Sender* menerima *acknowledge* tersebut dan mengirim

paket yang hilang tersebut. Untuk round pertama terdapat pengecualian karena semua pake pada round sebelumnya tidak ada (belum dikirim).

Sedangkan pada ***Hop-by-Hop Transmission*** dilakukan antara node sensor dengan node sensor tetangganya. Jadi saat terjadi *data loss* maka pengiriman ulang lebih sedikit dalam menghabiskan daya. *Hop-by-hop* ini membutuhkan *buffer* atau penyimpanan sementara pada setiap node sensor dan lebih efektif dilakukan pada topologi WSN *multi-hop*. *Buffer* ini digunakan untuk menyimpan data sementara hingga mendapatkan *acknowledge* dari hop berikutnya.

2.2.2 Jenis - jenis Acknowledge

Dalam mencapai reliability dapat digunakan acknowledge saat melakukan pengiriman data. Terdapat empat jenis acknowledge yang dapat digunakan, yaitu:

1. Explicit Acknowledge (eACK) : Penerima memberitahu pengirim bahwa paket telah diterima dengan tepat sekaligus memberitahu pengirim paket mana yang belum diterima untuk dilakukan *retransmission*.
2. Negative Acknowledge (nACK) : Penerima memberitahu pengirim bahwa paket yang diterima tidak benar dan diperlukan *retransmission*.
3. Implicit Acknowledge (iACK) : Setelah pengirim mengirim pesan, pengirim akan memastikan paket data dikirim ke tetangganya memberikan acknowledge.
4. Selective Acknowledge (sACK) : Hanya paket yang hilang dari sebuah pesan yang akan dikirim ulang

2.2.3 Protokol Transport yang Reliable

Ada banyak protokol untuk memastikan *reliability* pada WSN. Beberapa protokol mendukung *upstream* dan beberapa protokol mendukung *downstream*. Hanya ada sedikit protokol yang dapat mendukung keduanya. Selain itu ada protokol yang memiliki fokus utama untuk menangani *reliability* saja dan ada yang menangani *congestion* sekaligus *reliability*.

Beberapa protokol transport yang sering digunakan antara lain:

- GARUDA
- Event-to-Sink Reliable Transport (ESRT)
- Reliable Multi Segment Transport (RMST)
- Pump Slowly Fetch Quickly (PSFQ)
- Asymmetric Reliable Transport (ART)
- Price Oriented Reliable Transport (PORT)
- Delay Sensitive Transport (DST)

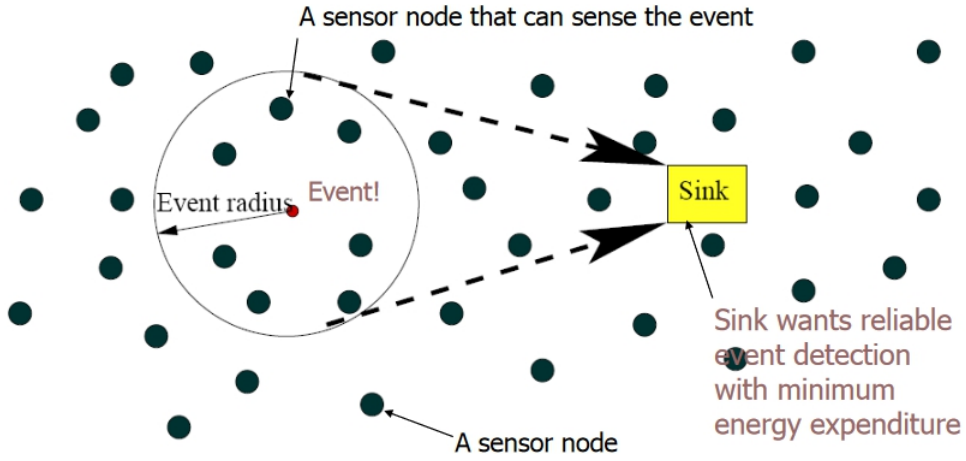
GARUDA

GARUDA adalah salah satu protokol *downstream* yang menggunakan *hop-by-hop retransmission* dalam memastikan *reliability*. Protokol ini dapat berjalan pada arsitektur dengan dua *tier*. Node sensor dengan 32hop dari sink node akan dipilih menjadi node sensor inti (*core*). Node sensor lain (*noncore*) akan disebut dengan *second-tier nodes*. Setiap node noncore menggunakan node *core* untuk memulihkan paket yang hilang. GARUDA menggunakan mekanisme NACK untuk mendeteksi paket yang hilang. Pemulihan paket yang hilang dilakukan dengan dua fase. Pemulihan paket dalam node *core*, dan pemulihan paket antara node noncore dengan node core.

GARUDA menggunakan mekanisme WFP *(Wait for First Packet) pulse transmission* untuk memastikan keberhasilan dalam mengirim satu atau paket pertama. Pulse transmission ini juga digunakan untuk menghitung jumlah hop dan memilih core node. Kekurangan dari GARUDA adalah tidak dapat menangani *upstream reliability* dan tidak menangani]*congestion control*.

Event-to-Sink Reliable Transport

Pada WSN terdapat *event detection*, yaitu saat node sensor mendeteksi sebuah *event* pada sebuah radius tertentu. Hasil deteksi tersebut yang kemudian dikirimkan ke *sink node*. Gambar 2.19 menunjukan *event detection* pada WSN. Event-to-Sink Reliable Transport (ESRT) adalah protokol *upstream* yang lebih banyak digunakan untuk mengirim event dibandingkan paket data. ESRT menggunakan pendekatan end-to-end untuk menjamin *reliability*. Pada ESRT tingkat pengiriman pada setiap node sensor bergantung pada tingkat *reliable* di *sink node* dan dengan status jaringan tersebut (terdapat kemacetan atau tidak). ESRT adalah protokol pertama yang menangani *congestion control* dan *reliability* sekaligus.



Gambar 2.19: Event Detection pada WSN

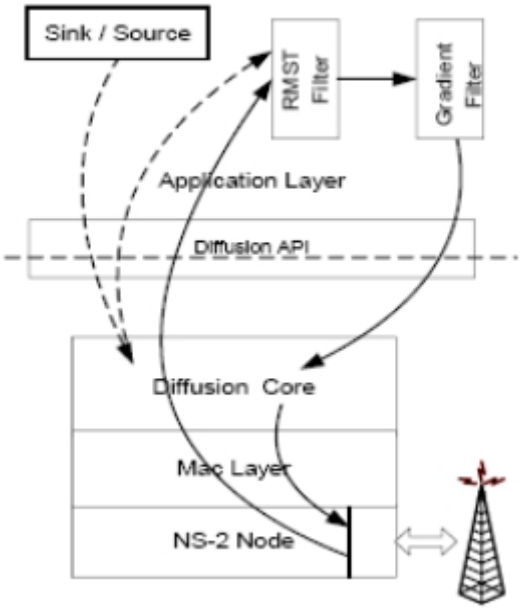
Pada ESRT setiap node sensor mendeteksi *congestion* berdasarkan peningkatan *buffer* lalu menambahkan N bit pada *header* sebuah paket dan meneruskannya ke sink node. Berdasarkan paket yang diterima oleh *sink node*, dapat diketahui keadaan dari jaringan dan juga *reliability* suatu paket untuk menentukan jumlah paket yang telah berhasil diterima pada periode tertentu. ESRT menggunakan ACK untuk memastikan pengiriman data yang reliable. ESRT dapat berjalan saat *sink node* melihat bahwa jumlah data yang diterima tidak sesuai dengan jumlah yang dikirimkan oleh node sensor.

Reliable Multi Segment Transport

Reliable Multi Segment Transport (RMST) adalah protokol *upstream* yang menjamin transfer paket yang reliable menggunakan *selective NACK* untuk memastikan paket yang *loss* dan mengirim ulang paket tersebut. Setiap node sensor pada RMST ini menyimpan data sementara (*cache*). Data *loss* dapat terjadi pada setiap node sensor. Dalam memulihkan data yang *loss* RMST menggunakan pemulihan hop-by-hop. Saat terjadi loss, node sensor akan melakukan request ke node sensor tetangganya. Jika data tidak ditemukan pada node tetangganya, maka NACK akan diteruskan ke node sensor sebelumnya hingga ke *source node*. RMST ini menggunakan mekanisme *timer-driven* untuk mendeteksi paket yang *loss*.

RMST dibuat untuk dapat berjalan diatas directed diffusion (Gambar 2.20). *Directed diffusion* berarti protokol routing untuk memastikan reliability untuk diaplikasikan. Pada RMST terdapat dua mekanisme yang dapat dilakukan yaitu mode *caching* dan mode *non caching*. Mode *Caching*

berarti node sensor yang berada di tengah dapat mendeteksi celah atau loss dan membuat request (NACK) ke node sebelumnya untuk memulihkan bagian yang hilang. Sedangkan pada mode *non caching* berarti *sink node* yang melakukan deteksi terhadap data yang hilang. Jadi sebenarnya RMST ini dapat dilakukan dengan mekanisme hop-by-hop dan end-to-end. Masalah dari RMST ini adalah tidak menangani *congestion control*, penggunaan energi yang efisien, dan reliable pada aplication layer.



Gambar 2.20: Hubungan antara RMST dengan Directed Diffusion

Pump Slowly Fetch Quickly

Pump Slowly Fetch Quickly (PSFQ) adalah protokol yang mengadaptasi perbaikan atau pemulihan dengan hop-by-hop saat terjadi data loss. PSFQ adalah protokol *downstream*. Tujuan dari PSFQ adalah mencapai pengiriman data secara bolak-balik *reliable* dari *sink node* ke node sensor dengan kecepatan yang relatif lambat, tapi memperbolehkan node sensor untuk melakukan pemulihan data yang hilang dari node tetangganya secara cepat. Protokol ini menggunakan mekanisme hop-by-hop dalam melakukan pengiriman ulang data yang hilang. Pada PSFQ terdapat tiga *operation* yaitu *pump*, *fetch*, dan *report*.

Berikut adalah cara kerja PSFQ. Pertama, secara perodik dan perlahan *sink node* melakukan *broadcast* paket yang terdiri dari ID, panjang file, *sequence number*, TTL, dan *report bit* ke node sensor tetangganya sampai semua *fragment* data dikirim semua. Kedua, node sensor akan memasuki mode *fetch* saat *gap* atau jarak pada *sequence number* terdeteksi. Kemudian NACK akan dikirimkan kepada pengirim untuk memperbaiki *fragment* yang hilang tersebut. Ketiga, dengan mekanisme hop-by-hop *fragment* yang hilang dapat dipulihkan. Protokol ini tidak menangani *congestion* dan tidak menangani untuk satu paket yang hilang, karena saat melakukan *pump* tidak hanya satu paket yang dikirimkan melainkan banyak paket sekaligus dan pemulihannya juga dilakukan semua pake yg dikirimkan pada saat tersebut.

Asymmetric Reliable Transport

Asymmetric Reliable Transport (ART) adalah protokol yang berbasis pada *event* dan *query reliability*. ART merupakan protokol *bidirectional* pertama yang sudah mendukung *congestion control*. Terdiri dari kumpulan node yang disebut *essential node* yang tersebar pada suatu area untuk melakukan

sensing dan beberapa node yang disebut *non-essential node* yang terlibat pada pengiriman data dan *congestion control*.

Price Oriented Reliable Transport

Price Oriented Reliable Transport Protocol (PORT) adalah protokol *upstream* yang berbasis pada *event reliability* dengan penggunaan energi seminimal mungkin. PORT mendukung mekanisme dengan energi yang efisien disertai *congestion control*. Selain itu PORT juga adalah protokol yang mendukung komunikasi *End-to-End*. PORT membutuhkan *sink node* untuk mengatur aliran data. Kekurangannya adalah tidak ada pemulihan paket yang hilang.

Delay Sensitive Transport

Delay Sensitive Transport (DST) adalah tambahan dari ESRT. Tujuan dari DST adalah dicapai *reliability* pada *sink node*. Pada DST terdapat aturan *Time Critical Event First* (TCEF).TCEF berarti data paket dengan diberikan prioritas untuk melakukan *retransmission*. DST dapat bekerja dengan baik pada satu event, namun pada banyak event akan menjadi lebih kompleks.

Lebih singkat setiap protokol tersebut dapat dilihat pada Tabel 2.3

Tabel 2.3: Tabel Perbandingan Protokol

Protokol	Arah	Tingkat	Mekanisme	Tipe ACK	Menangani Congestion	Energi Efisien
GARUDA	Downstream	Packet	Hop-By-Hop	NACK	Tidak	Tidak
ESRT	Upstream	Event	End-to-End	-	Ya	Ya
RMST	Upstream	Packet	Hop-By-Hop	NACK	Tidak	Ya
PSFQ	Downstream	Packet	Hop-By-Hop	NACK	Tidak	Tidak
ART	Both	Event	End-to-End	-	Ya	Ya
PORT	Upstream	Event	Hop-By-Hop	-	Ya	Ya
DST	Upstream	Event	End-to-End	-	Tidak	Ya

2.3 PreonVM

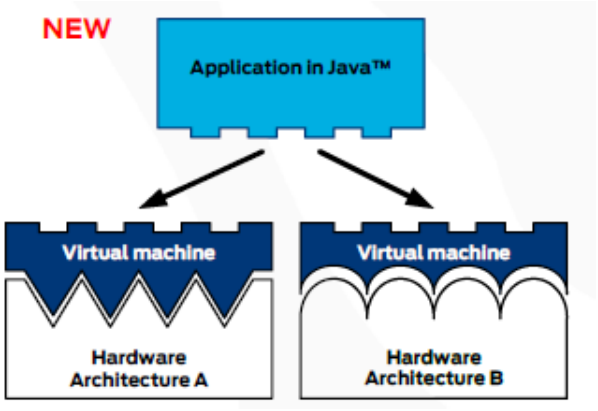
PreonVM adalah virtual machine (VM) yang dibuat oleh VIRTENIO untuk sistem komputer yang dirancang khusus (*embedded system*) dengan sumber daya yang terbatas. PreonVM dapat digunakan pada node sensor jenis Preon32. PreonVM dibuat sangat optimal dengan tidak dibutuhkannya sistem operasi tambahan dan berjalan langsung pada *microprocessor*. Dengan PreonVM ini *developer* dapat membuat aplikasi dengan mudah pada Bahasa Java yang mengumpulkan data dari sensor dan mengontrol aktuator. API pada PreonVM mendukung antarmuka radio sesuai dengan IEEE 802.15.4 dan AES encryption pada perangkat kerasnya.

PreonVM memiliki fitur sebagai berikut:

- Aplikasi dibangun dengan Bahasa Pemrograman Java
- Mendukung semua tipe data pada Java seperti char, byte, int, long, float atau double
- *Garbage collection* dengan *memory defragmentation*
- Mendukung *exception handling*, stack dan array multidimensi
- Terdapat *system properties* untuk mengatur aplikasi
- Tidak membutuhkan sistem operasi tambahan

- Mendukung threads termasuk synchronized, Object.wait, Object.notify, Object.notifyAll, Thread.sleep, Thread.interrupt

Kelebihan yang dimiliki oleh VIRTENIO ini adalah *object-oriented programming* dengan Bahasa Pemrograman Java. VIRTENIO menyediakan virtual machine sebagai sistem operasi yang inovatif untuk modul Preon32. Dengan menggunakan virtual machine, aplikasi dapat berdiri sendiri pada arsitektur yang dibuat. Dengan demikian aplikasi Java yang dibuat dapat dijalankan pada aristektur yang berbeda tanpa harus modifikasi. Gambar 2.21 menunjukan ilustrasi penggunaan virtual machine pada berbagai perangkat.

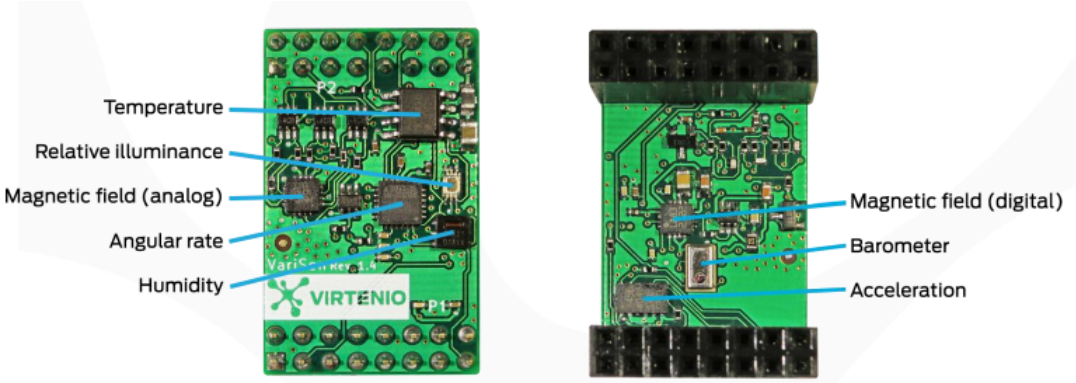


Gambar 2.21: Penggunaan virtual machine pada perangkat yang berbeda

Virtual machine juga membatu memisahkan proses pengembangan perangkat keras dengan aplikasi perangkat lunak. Program aplikasi dikembangan dengan antarmuka yang abstrak dan tidak berubah. Sebagai tambahan, virtual mesin berjalan optimal untuk aplikasi kecil pada 8-bit sampai 32-bit microprocessor dengan 8 Kbyte RAM dan 128 Kbyte Flash.

2.3.1 Preon32

Preon32 adalah salah satu jenis node sensor. Preon32 menggunakan PreonVM sebagai sistem operasi. Preon32 versi umum memiliki 5 jenis sensor pada sebuah board. Sensor yang ada pada Preon32 ini antara lain sensor suhu (temperature), sensor cahaya, sensor kelembapan udara (humidity), sensor untuk mengukur tekanan udara (barometer), dan sensor getaran. Pada versi tambahan Preon32 dapat dilengkapi dengan sensor untuk mendeteksi medan magnet (magnetometer), dan gyroscope. Semua jenis sensor tersebut dapat diatur melalui PreonVM dan pemrograman dapat dilakukan dengan Bahasa Pemrograman Java. Gambar 2.22 adalah board dan sensor-sensor pada Preon32.



Gambar 2.22: Preon32 Board

Preon32 ini dapat diaplikasikan pada berbagai hal seperti:

1. Home Automation yang digunakan untuk mengintegrasikan berbagai sistem pada rumah, monitoring area, dan navigasi.
2. Platform untuk membantu penelitian
3. Implementasi aplikais baru pada jaringan nirkabel
4. Algoritma untuk sensor data fusion

2.3.2 Class Library JVM Preon32

Preon32 mengunakan PreonVM sebagai Virtual Machine. Class library pada Virtenio Virtual Machine ini dibagi menjadi beberapa *package* utama diantaranya ***Radio Packages***, ***Route Packages***, ***Other Virtenio Packages***, dan ***Java Related Packages***. Tabel 2.3.2, sampai Tabel 2.3.2 berisi package yang terdapat pada setiap package utama.

Package	Deksripsi
com.virtenio.radio	Paket yang berisi kelas terkait radio
ESRTcom.virtenio.radio.ieee_802_15_4	Paket yang berisi kelas terkait IEEE 802.15.4

Tabel 2.4: Tabel Radio Packages

Package	Deksripsi
com.virtenio.route.aodv	Paket yang berisi kelas terkait AODV (Ad Hoc On Demand Vector) Routing

Tabel 2.5: Tabel Route Packages

Package	Deksripsi
com.virtenio.driver	Paket yang berisi drivers untuk berbagai perangkat
com.virtenio.driver.adc	Paket yang berisi kelas ADC driver
com.virtenio.driver.atmodem	
com.virtenio.driver.button	Paket yang berisi kelas button driver
com.virtenio.driver.can	Paket yang berisi kelas CAN driver
com.virtenio.driver.cpu	Paket yang berisi kelas CPU driver
com.virtenio.driver.device	Paket yang berisi kelas divice driver
com.virtenio.driver.device.at86rf212	Paket yang berisi driver untuk perangkat ATR86RF212
com.virtenio.driver.device.at86rf231	paket yang berisi driver untuk perangkat AT86RF231
com.virtenio.driver.flash	Paket yang berisi kelas Flash driver
com.virtenio.driver.gpio	Paket yang berisi kelas GPIO device driver
com.virtenio.driver.i2c	Paket yang berisi kelas I2C device driver
com.virtenio.driver.irq	Paket yang berisi kelas IRQ device driver
com.virtenio.driver.led	Paket yang berisi kelas LED device driver
com.virtenio.driver.lin	Paket yang berisi kelas LIN device driver
com.virtenio.driver.onewire	Paket yang berisi kelas OneWire device driver
com.virtenio.driver.pwm	Paket yang berisi kelas PWM (pulse-width modulation) device driver
com.virtenio.driver.ram	Paket yang berisi kelas FRAM device driver
com.virtenio.driver.rtc	Paket yang berisi kelas untuk pengaturan jam secara real-time, dan real-counter device driver
com.virtenio.driver.spi	Paket yang berisi kelas SPI (Serial Peripheral Interface) device driver
com.virtenio.driver.sw	Paket yang berisi kelas switch device driver
com.virtenio.driver.timer	Paket yang berisi kelas hardware timer device driver
com.virtenio.driver.usart	Paket yang berisi USART device driver
com.virtenio.driver.watchdog	Paket yang berisi
com.virtenio.io	Paket Virtenio VM yang berisi IO
com.virtenio.lib	Paket Virtenio VM yang berisi pengaturan classlib
com.virtenio.misc	Paket tambahan Virtenio VM
com.virtenio.net	
com.virtenio.vm	
com.virtenio.vm.event	Sistem event pada Virtenio VM untuk menangani event asynchronous dan synchronous

Tabel 2.6: Tabel Other Virtenio Packages

Package	Deksripsi
java.io	Paket Java IO
java.lang	Paket Java lang
java.lang.annotation	Paket Annotation pada Java
java.lang.ref	
java.nio	
java.nio.channels	
java.text	Paket Java Text
java.util	Paket Java Utility yang berisi collection
java.util.regex	Paket Java Regular Expression

Tabel 2.7: Tabel Java Related Packages

2.3.3 Pemrograman Pada Preon32

Pada subab ini akan dijelaskan bagaimana pemrograman pada Preon32 termasuk struktur file dan perintah-perintah yang digunakan pada ANT scripts.

Dalam melakukan pemrograman untuk Preon32 diperlukan lingkungan yang mendukung Bahasa Pemrograman Java seperti OpenJDK atau Java Development Kit. Untuk melakukan *compile* dan *transfer* aplikasi ke Preon32 harus menggunakan ANT scripts.

Struktur File Sandbox Preon32

Dalam melakukan pembangunan aplikasi pada Preon32, digunakan Sandbox yang sudah disediakan oleh Virtenio. Pembangunan aplikasi dilakukan pada Eclipse IDE. Sandbox ini terdiri dari *file-file* dan *folder-folder*. *Folder-folder* tersebut memiliki tujuannya masing-masing. Struktur *folder-folder* tersebut ditampilkan pada Gambar 2.23.

Pengaturan Build System Pada Sandbox

ANT script "build.xml"

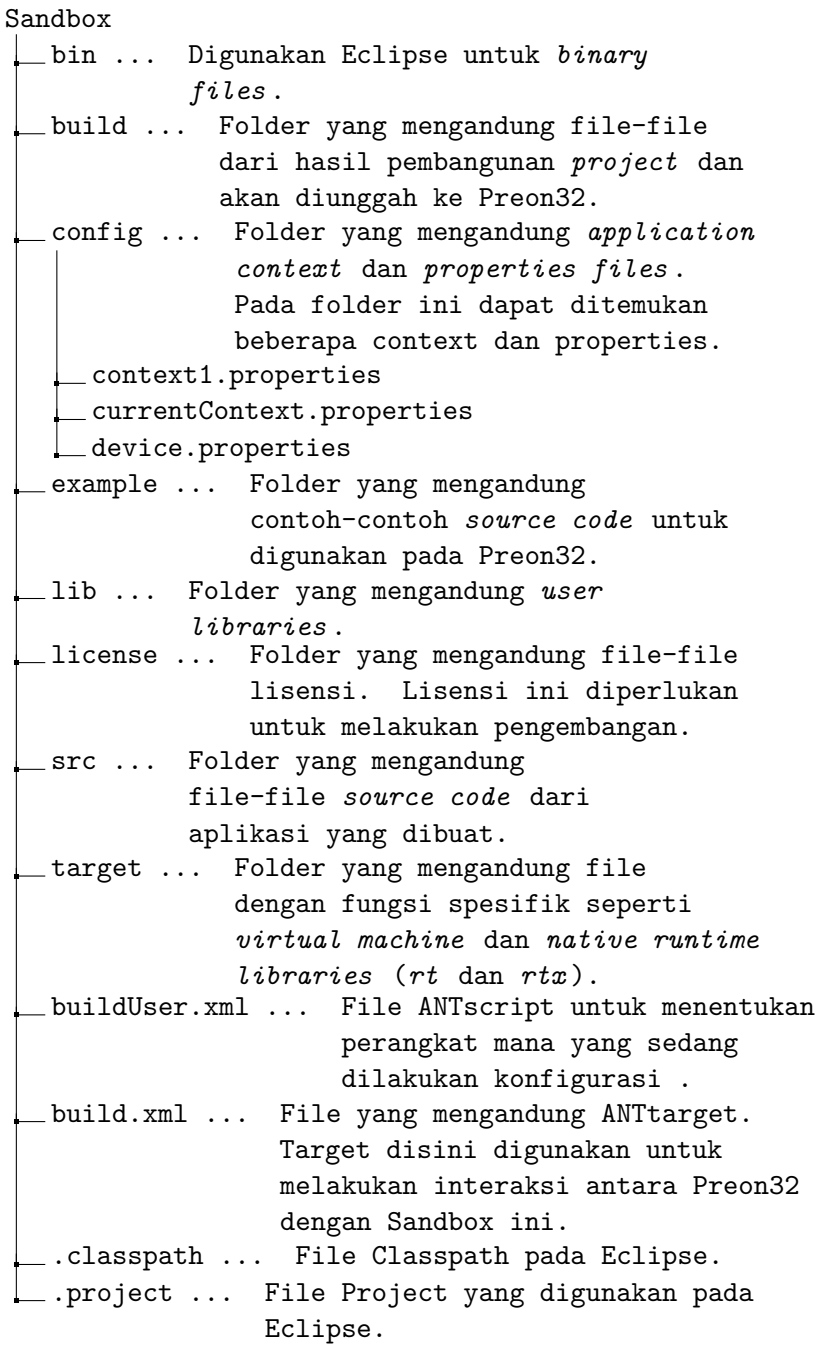
ANT script "buildUser.xml"

Pengaturan file pada direktori "config" Didalam direktori "config" terdapat 3 file awal:

1. **context1.properties:** Mendeskripsikan context pertama (1). File "buildUser.xml" secara *default* akan terhubung ke file ini.
2. **currentContext.properties:** File ini menunjuk pada context saat ini. Isi dari file ini dapat berubah secara otomatis saat mengganti context pada file "buildUser.xml"
3. **device.properties:** Pada file ini terdapat user properties.

File paling penting pada direktori ini adalah "contet1.properties". Properties penting dalam file ini antara lain:

1. mainClass.name = Prog:
2. module.name = prog:
3. target.dir = targets/Preon32:
4. device.properties.file = config/device.properties
5. comport = COM10:



Gambar 2.23: Tampilan struktur folder pada Sandbox Preon32

BAB 3

ANALISIS

Pada bab ini dijelaskan mengenai analisis pengiriman data dari node sensor ke base station pada WSN, analisis terhadap protokol transfer yang *reliable* pada *Wireless Sensor Network* untuk digunakan dalam membangun aplikasi transfer data, dan analisis aplikasi pengiriman data pada WSN.

3.1 Analisis Proses Pengiriman Data Dari Node Sensor Ke Base Station Pada WSN

Pengiriman data pada WSN dibagi menjadi dua yaitu pengiriman data dari base station ke setiap node sensor (downstream) dan pengiriman data dari setiap node sensor ke base station (upstream). Pengiriman downstream biasanya adalah perintah yang diberikan oleh base station kepada node sensor, sedangkan pengiriman upstream adalah pengiriman data hasil sensing setiap node sensor kepada base station. Dalam melakukan pengiriman ini dapat dilakukan dengan single hop atau multi hop tergantung pada komunikasi yang digunakan.

Pengiriman data hasil sensing dari node sensor ke base station sendiri dapat dilakukan dengan dua cara yaitu setiap data hasil sensing langsung dikirimkan ke base station tanpa melalui proses apapun pada node sensor dan data hasil sensing diproses dahulu pada node sensor lalu dikirimkan. Kedua cara ini sebenarnya dapat digunakan sesuai dengan kebutuhkan dan kapasitas penyimpanan pada node sensor. Node sensor memiliki tempat penyimpanan yang kecil, sehingga kode program yang dibuat juga tidak dapat terlalu besar.

Pada skripsi ini digunakan cara dengan mengirim langsung data mentah yang didapat dari node sensor tersebut. Penulis memilih cara ini karena penyimpanan node sensor yang kecil. Selain itu untuk diimplementasi cara ini lebih mudah dibandingkan dengan melakukan memprosesan terlebih dahulu. Karena tujuan utama pada skripsi ini adalah membangun aplikasi untuk transfer data maka pada base station hanya memerlukan data mentah.

3.2 Analisis Terhadap Protokol Transfer Yang Reliable Pada WSN Untuk Digunakan Dalam Membangun Aplikasi Transfer Data

Tidak semua protokol pengiriman data pada WSN mendukung pengiriman data yang *reliable*. Pada WSN terdapat beberapa protokol yang mendukung pengiriman data reliable. Untuk memastikan data yang reliable ini dilakukanlah pengiriman ulang (retransmission) data yang hilang atau *loss*. Data yang dikirim ulang ini dapat berasal dari hasil *sensing* node sensor atau data dari base station seperti perintah atau *method* untuk melakukan sesuatu. Pada skripsi ini data yang dikirim ulang adalah data hasil *sensing* sehingga protokol yang digunakan adalah protokol dengan arah pengiriman *upstream*. Setiap protokol memiliki spesifikasi yang berbeda-beda pada arah *retransmission*, data yang dikirim, mekanisme dan *acknowledge* yang digunakan.

Pada skripsi ini dibangun aplikasi transfer data pada WSN. Aplikasi yang dibangun ini juga menangani pengiriman data yang *reliable*. Pengiriman data yang terjadi adalah pengiriman data

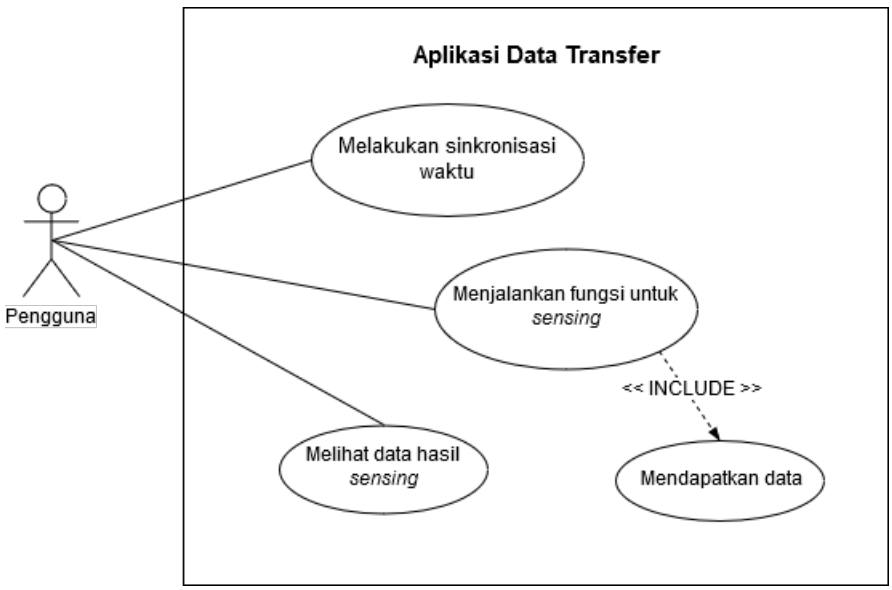
dari setiap node sensor hingga sampai ke base station (*upstream*). Data yang didapat oleh node sensor ini dapat diteruskan ke node tetangganya dahulu sebelu sampai ke tujuan akhir yaitu *base station*. Berdasarkan Tabel 2.3 terdapat beberapa protokol yang diadaptasi untuk membuat aplikasi transfer data ini.

Aplikasi transfer data yang dibuat ini akan menangani data yang loss menggunakan mekanisme hop-by-hop dengan bantuan ACK dalam memastikan data sampai ke node tetangganya. Dengan mekanisme hop-by-hop akan lebih cepat untuk mengirimkan ulang data yang loss. Jika dibandingkan dengan mekanisme end-to-end retransmission maka reliabilitas data akan diperiksa oleh base station. Jika base station menemukan ada data yang loss, maka base station akan meminta node sensor untuk mengirimkan ulang data yang loss tersebut. Namum permasalahan pada mekanisme end-to-end adalah jika arsitektur yang dibangun adalah multi hop maka base station akan meminta node yang terhubung dengan base station meneruskan pesan ke node sensor tetangganya hingga ke node sensor yang melakukan *sensing* untuk mengirim ulang data.

Dengan mekanisme hop-by-hop, saat salah satu node mengirimkan data ke node tetangganya, node pengirim akan menunggu ACK dari node penerima. Saat menunggu ACK dari node tetangganya ini digunakan juga mekanisme *timer-driven*. Dengan menggunakan mekanisme *timer-driven* ini, waktu menunggu ACK dapat dibatasi. Jika sudah melewati batas waktu menunggu ACK tersebut, maka data akan dianggap tidak diterima oleh penerima dan pengirim akan mengirimkan ulang data. Saat melakukan pengiriman ulang data ini bisa saja terjadi pengulangan data yang diterima. Jika ini terjadi maka digunakan *sequence number* untuk memastikan urutan data.

3.3 Analisis Aplikasi Pengiriman Data Pada WSN

Pada skripsi ini dibuat aplikasi yang dapat melakukan transfer data dari node sensor ke base station. Aplikasi ini digunakan untuk mengetahui keadaan suatu tempat atau daerah dengan bantuan node sensor untuk melakukan mendapatkan data (sensing). Aplikasi ini memiliki beberapa fungsi. Fungsi utama pada aplikasi ini adalah untuk melakukan transfer data dari setiap node sensor. Fungsi lain pada aplikasi ini dijelaskan menggunakan diagram *use case* pada Gambar 3.1 dan skenario pada Tabel 3.1 sampai Tabel 3.3.



Gambar 3.1: Diagram *use case* aplikasi data transfer pada WSN

Nama	Melakukan sinkronisasi waktu
Deskripsi	Melakukan sinkronisasi waktu untuk mengetahui kapan data dikirim dari node sensor dan diterima oleh base station.
Aktor	Pengguna
Pre-kondisi	Aplikasi dimulai
Alur Skenario Utama	1. Sistem memuat aplikasi. 2. Sistem menampilkan menu untuk dipilih pengguna. 3. Pengguna memilih menu sinkronisasi waktu. 4. Sistem melakukan sinkronisasi waktu pada setiap node sensor. 5. Sistem menampilkan menu.

Tabel 3.1: Tabel skenario melakukan sinkronisasi waktu.

Nama	Menjalankan fungsi untuk *sensing*
Deskripsi	Pengguna menjalankan fungsi ini untuk membuat node sensor melakukan sensing dan mengirimkan data.
Aktor	Pengguna
Pre-kondisi	Sudah dilakukan sinkronisasi waktu pada setiap node sensor
Alur Skenario Utama	1. Sistem memuat aplikasi. 2. Sistem menampilkan menu untuk dipilih pengguna. 3. Pengguna memilih menu untuk *sensing*. 4. Sistem memerintahkan node sensor untuk melakukan *sensing*. 5. Sistem mendapatkan data dari setiap node sensor.

Tabel 3.2: Tabel skenario Menjalankan fungsi untuk *sensing*.

Nama	Mendapatkan Data
Deskripsi	Pengguna mendapatkan data hasil *sensing* dari setiap node sensor
Aktor	Pengguna
Pre-kondisi	Sistem sudah melakukan *sensing* dan data hasil *sensing* sudah didapatkan
Alur Skenario Utama	1. Sistem memuat aplikasi. 2. Sistem menampilkan menu untuk dipilih pengguna. 3. Pengguna memilih menu untuk mendapatkan data. 4. Sistem mengirimkan data hasil *sensing* kepada pengguna.

Tabel 3.3: Tabel skenario mendapatkan data.

3.3.1 Diagram Sequence

Untuk aplikasi transfer data ini, aliran data dapat dilihat dengan diagram sequence. Diagram sequence dibuat menjadi dua macam sesuai dengan arsitektur yang digunakan yaitu arsitektur flat dengan multi-hop dan arsitektur hierarkikal.

——- Gambar Diagram Seq 1 ——- Keterangan...

——- Gambar Diagram Seq 2 ——- Keterangan...

3.3.2 Fitur dan Kebutuhkan Sistem

Pada skripsi ini aplikasi yang dibangun memiliki fitur utama untuk mengirim data hasil sensing. Sesuai dengan diagram sequence yang dibuat, untuk arsitektur flat dan hierarki terdapat sedikit perbedaan untuk aplikasi yang dibuat. Pada arsitektur flat tidak terdapat cluster head jadi node sensor hanya mengirimkan data ke node tetangganya hingga sampai ke base station, sedangkan pada arsitektur hierarkikal data dikirimkan dahulu ke cluster head sebelum diteruskan ke base station. .....

LAMPIRAN A

KODE PROGRAM

Listing A.1: MyCode.c

```c

// This does not make algorithmic sense,
// but it shows off significant programming characters.

#include<stdio.h>

void myFunction( int input, float* output ) {
    switch ( array[i] ) {
        case 1: // This is silly code
            if ( a >= 0 || b <= 3 && c != x )
                *output += 0.005 + 20050;
            char = 'g';
            b = 2^n + ~right_size - leftSize * MAX_SIZE;
            c = (--aaa + &daa) / (bbb++ - ccc % 2 );
            strcpy(a,"hello $@?");
    }
    count = ~mask | 0x00FF00AA;
}

// Fonts for Displaying Program Code in LATEX
// Adrian P. Robson, nepsweb.co.uk
// 8 October 2012
// http://nepsweb.co.uk/docs/progfonts.pdf
```

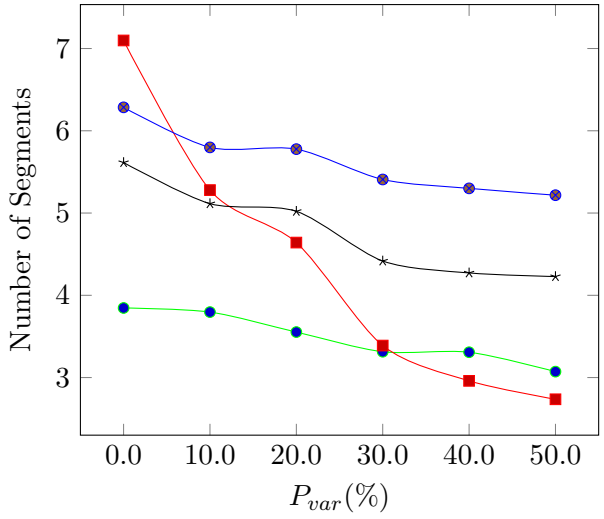
Listing A.2: MyCode.java

```java
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashSet;

//class for set of vertices close to furthest edge
public class MyFurSet {
    protected int id;                                   //id of the set
    protected MyEdge FurthestEdge;                      //the furthest edge
    protected HashSet<MyVertex> set;                    //set of vertices close to furthest edge
    protected ArrayList<ArrayList<Integer>> ordered;    //list of all vertices in the set for each trajectory
    protected ArrayList<Integer> closeID;               //store the ID of all vertices
    protected ArrayList<Double> closeDist;              //store the distance of all vertices
    protected int totaltrj;                             //total trajectories in the set

    /*
     * Constructor
     * @param id : id of the set
     * @param totaltrj : total number of trajectories in the set
     * @param FurthestEdge : the furthest edge
     */
    public MyFurSet(int id,int totaltrj,MyEdge FurthestEdge) {
        this.id = id;
        this.totaltrj = totaltrj;
        this.FurthestEdge = FurthestEdge;
        set = new HashSet<MyVertex>();
        ordered = new ArrayList<ArrayList<Integer>>();
        for (int i=0;i<totaltrj;i++) ordered.add(new ArrayList<Integer>());
        closeID = new ArrayList<Integer>(totaltrj);
        closeDist = new ArrayList<Double>(totaltrj);
        for (int i = 0;i <totaltrj;i++) {
            closeID.add(-1);
            closeDist.add(Double.MAX_VALUE);
        }
    }

}
```

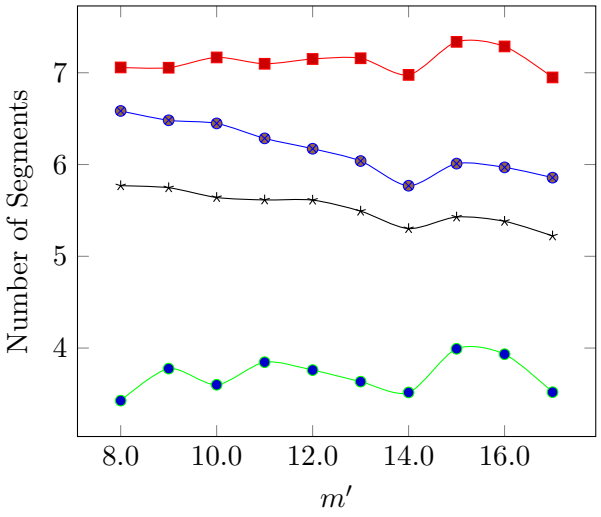
LAMPIRAN B

HASIL EKSPERIMEN

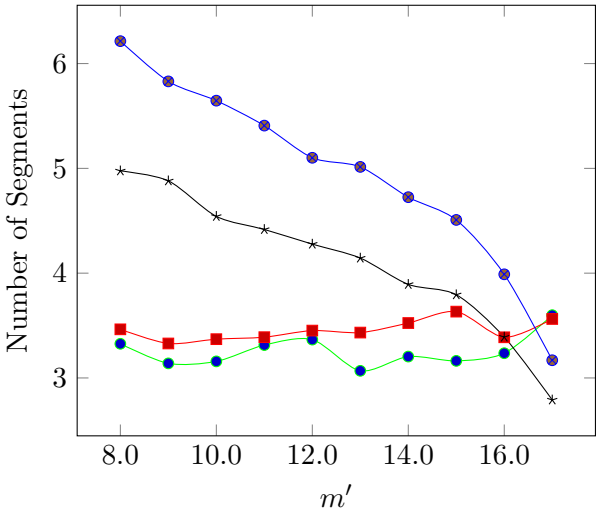
Hasil eksperimen berikut dibuat dengan menggunakan TIKZPICTURE (bukan hasil excel yg diubah ke file bitmap). Sangat berguna jika ingin menampilkan tabel (yang kuantitasnya sangat banyak) yang datanya dihasilkan dari program komputer.



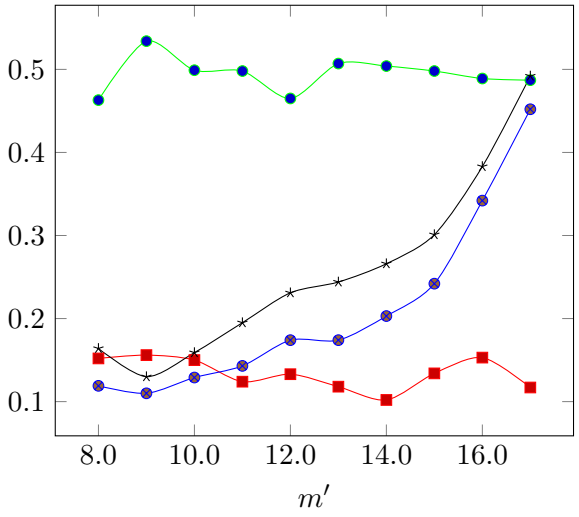
Gambar B.1: Hasil 1



Gambar B.2: Hasil 2



Gambar B.3: Hasil 3



Gambar B.4: Hasil 4