

SKRIPSI

PENGEMBANGAN APLIKASI TRANSFER DATA DI WSN



Jonathan Alva

NPM: 2015730047

**PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI DAN SAINS
UNIVERSITAS KATOLIK PARAHYANGAN
2019**

UNDERGRADUATE THESIS

**DEVELOPMENT OF DATA TRANSFER APPLICATION IN
WSN**



Jonathan Alva

NPM: 2015730047

**DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY AND SCIENCES
PARAHYANGAN CATHOLIC UNIVERSITY
2019**

LEMBAR PENGESAHAN

PENGEMBANGAN APLIKASI TRANSFER DATA DI WSN

Jonathan Alva

NPM: 2015730047

Bandung, 22 Mei 2019

Menyetujui,

Pembimbing

Elisati Hulu, M.T.

Ketua Tim Penguji

Anggota Tim Penguji

Chandra Wijaya, M.T.

Pascal Alfadian, M.Comp.

Mengetahui,

Ketua Program Studi

Mariskha Tri Adithia, P.D.Eng

PERNYATAAN

Dengan ini saya yang bertandatangan di bawah ini menyatakan bahwa skripsi dengan judul:

PENGEMBANGAN APLIKASI TRANSFER DATA DI WSN

adalah benar-benar karya saya sendiri, dan saya tidak melakukan penjiplakan atau pengutipan dengan cara-cara yang tidak sesuai dengan etika keilmuan yang berlaku dalam masyarakat keilmuan.

Atas pernyataan ini, saya siap menanggung segala risiko dan sanksi yang dijatuhkan kepada saya, apabila di kemudian hari ditemukan adanya pelanggaran terhadap etika keilmuan dalam karya saya, atau jika ada tuntutan formal atau non-formal dari pihak lain berkaitan dengan keaslian karya saya ini.

Dinyatakan di Bandung,
Tanggal 22 Mei 2019

Meterai Rp. 6000

Jonathan Alva
NPM: 2015730047

ABSTRAK

Wireless Sensor Network (WSN) adalah jaringan nirkabel yang terdiri dari sekumpulan node sensor dengan kemampuan *sensing*, komputasi, dan komunikasi. Setiap node sensor akan mengumpulkan data seperti suhu, tekanan, dan kelembaban. WSN dapat dibangun menggunakan arsitektur flat dan hierarki. Setiap arsitektur dapat menggunakan jalur komunikasi *single-hop* atau *multi-hop*. Pada komunikasi *single-hop*, data dari node sensor langsung dikirim ke *base station* dalam 1 kali lompatan (hop). Sedangkan pada komunikasi *multi-hop*, data dari node sensor akan diteruskan kepada node sensor lain hingga sampai ke *base station*. Dalam melakukan pengiriman data atau pesan dapat terjadi *loss*. Untuk melakukan monitoring suatu area, data harus dapat diterima secara utuh. Dalam monitoring area, data yang *loss* dapat berpengaruh pada tingkat akurasi data. Semakin banyak data *loss*, tingkat akurasi akan semakin rendah.

Ada beberapa protokol yang sudah ada untuk menangani *loss* pada transfer data. Salah satu protokol transport yang ada adalah RMST (Reliable Multi Segment Transport). RMST menggunakan beberapa mekanisme antara lain *retransmission*, *end-to-end*, *ACK*, *sequence number* dan *timer* untuk melakukan transfer data yang *reliable*. Aplikasi yang dikembangkan akan menggunakan kelima mekanisme ini untuk menangani transfer data yang *reliable* pada WSN.

Pengembangan aplikasi ini berhasil dibangun dan transfer data dapat dilakukan secara *reliable* dimana tidak ada data yang *loss*. Tetapi transfer data yang *reliable* ini membutuhkan waktu yang lebih lama, energi yang lebih besar dan memerlukan tempat penyimpanan yang besar pada setiap node sensor.

Kata-kata kunci: Wireless Sensor Network, Arsitektur Flat, Single-hop, Reliable, RMST

ABSTRACT

Wireless Sensor Network (WSN) is a wireless network consists of a set sensor nodes with sensing, computing, and communication. Each sensor node will collect data such as temperature, pressure, and humidity. WSN can be built using flat and hierarchy architecture. Each architecture can use single-hop or multi-hop communication. In single-hop communication, data from sensor node is sent directly to the base station with 1 hop. Whereas in multi-hop communication, data from the sensor node will be forwarded to the sensor node until it reach the base station. In monitoring area, data must be received in its entirety. Data loss can affect to the level of data accuracy. The more data loss, the lower accuracy will be.

There are several protocols that can handle loss data transfer. One of the existing transport protocols is RMST (Reliable Multi Segment Transport). RMST uses mechanisms such as retransmission, end-to-end, ACK, sequence number, and timer for reliable data transfer. The application developed will use these five machanisms to handle the reliable data transfer in WSN.

This application development was successfully built and data transfer can be done reliable with no data loss. But the reliable data transfer need more time, more energy, and large storage on each sensor node

Keywords: Wireless Sensor Network, Arsitektur Flat, Single-hop, Reliable, RMST

Teknik Informatika UNPAR dan diri sendiri

KATA PENGANTAR

Puji syukur kepada Tuhan Yang Maha Esa atas berkat yang diberikan kepada penulis sehingga dapat menyelesaikan tugas akhir dengan judul **Pengembangan Aplikasi Transfer Data Di WSN** dengan baik dan tepat waktu. Penulis juga berterima kasih kepada pihak-pihak yang telah memberikan dukungan kepada penulis dalam menyelesaikan tugas akhir ini, yaitu

1. Keluarga yang selalu memberikan dukungan dan semangat kepada penulis.
2. Bapak Elisati Hulu sebagai dosen pembimbing yang telah membimbing penulis hingga dapat menyelesaikan tugas akhir ini dan meminjamkan node sensor selama mengerjakan tugas akhir ini.
3. Bapak Chandra Wijaya dan Bapak Pascal Alfadian sebagai dosen penguji yang telah membantu dalam menguji dan memperbaiki tugas akhir ini.
4. Teman-teman *admin* lab yang selalu menyediakan lab skripsi agar dapat mengerjakan skripsi dengan nyaman.
5. Felicia Christiany, Joshua Riyadi, Dandy Unggana, Victor Christian, Sutyoso, Thoby, Emmanuel Yudhistira, Irvan Wijaya Ludianto, Henry, Raymond Nagawijaya, Nico Samuel, Kezia, Edrick, Himawan Saputra sebagai teman seperjuangan dan bertukar pikiran sekaligus telah memberikan semangat kepada penulis
6. Teman-teman Teknik Informatika UNPAR angkatan 2015 yang telah berbagi ilmu kepada penulis.
7. Pihak-pihak lain yang belum disebutkan, yang berperan dalam penyelesaian tugas akhir ini.

Bandung, Mei 2019

Penulis

DAFTAR ISI

KATA PENGANTAR	xv
DAFTAR ISI	xvii
DAFTAR GAMBAR	xix
DAFTAR TABEL	xxi
1 PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah	1
1.3 Tujuan	2
1.4 Batasan Masalah	2
1.5 Metodologi	2
1.6 Sistematika Pembahasan	2
2 LANDASAN TEORI	5
2.1 Wireless Sensor Network	5
2.1.1 Penerapan Wireless Sensor Network [1]	5
2.1.2 Node Sensor	6
2.1.3 Arsitektur dan Topologi Wireless Sensor Network [2]	8
2.1.4 Sistem Operasi	13
2.1.5 Protokol Stack pada Wireless Sensor Network [3]	16
2.2 Reliable Data Transfer di WSN [4]	17
2.2.1 Jenis Reliability	18
2.2.2 Jenis - jenis Acknowledge [5]	19
2.2.3 Protokol Transport yang Reliable [5]	19
2.3 PreonVM	22
2.3.1 Preon32	23
2.3.2 Class Library JVM Preon32	25
2.3.3 Pemrograman Pada Preon32	27
2.4 Format Struktur Frame	30
3 ANALISIS	33
3.1 Analisis Aplikasi Pengiriman Data Pada WSN	33
3.1.1 Fitur dan Kebutuhan Sistem	40
3.1.2 Ukuran Reliability	42
3.2 Analisis Proses Pengiriman Data Dari Node Sensor Ke Base Station Pada WSN	42
3.2.1 Format Pesan Yang Dikirim Saat Melakukan Sensing	43
3.2.2 Format Pesan ACK Yang Dikirim	43
3.2.3 Format Pesan Untuk Mengetahui Node Yang Menyala	43
3.2.4 Format Pesan Untuk Mengirimkan Waktu Node Sensor	44

3.3	Analisis Protokol Transfer Yang Reliable Pada WSN Untuk Digunakan Dalam Membangun Aplikasi Transfer Data	44
4	PERANCANGAN	45
4.1	Perancangan Interaksi Antar Node Untuk Transfer Data Di WSN	45
4.1.1	Diagram Sequence "Check Online"	45
4.1.2	Diagram Sequence "Synchronize Time"	46
4.1.3	Diagram Sequence "Get Time"	47
4.1.4	Diagram Sequence "Start Sensing"	48
4.1.5	Diagram Sequence "Exit"	49
4.2	Perancangan Kelas Aplikasi Transfer Data Di WSN	49
4.3	Perancangan Masukan dan Keluaran	58
4.4	Perancangan Pseudocode Aplikasi Transfer Data Yang Reliable	58
4.4.1	Base Station	58
4.4.2	Node Sensor	58
4.5	Perancangan <i>Routing</i> Pada Aplikasi Transfer Data	63
4.6	Perancangan Format Pesan	63
5	IMPLEMENTASI DAN PENGUJIAN	67
5.1	Implementasi	67
5.1.1	Lingkungan Implementasi	67
5.1.2	Hasil Implementasi	68
5.2	Pengujian	74
5.2.1	Pengujian Fungsional	74
5.2.2	Pengujian Eksperimental	79
5.2.3	Kesimpulan Hasil Eksperimen	90
5.3	Masalah yang Dihadapi pada Saat Implementasi	91
6	KESIMPULAN DAN SARAN	93
6.1	Kesimpulan	93
6.2	Saran	93
	DAFTAR REFERENSI	95
	A KODE PROGRAM	97
	B HASIL EKSPERIMEN SINGLE HOP	113
	C HASIL EKSPERIMEN MULTI HOP TIPE 1	115
	D HASIL EKSPERIMEN MULTI HOP TIPE 2	117
	E HASIL EKSPERIMEN MULTI HOP TIPE 3	119
	F HASIL EKSPERIMEN MULTI HOP TIPE 4	121
	G HASIL EKSPERIMEN MULTI HOP TIPE 5	123
	H CONTOH HASIL EKSPERIMEN YANG DISIMPAN PADA FILE TEXT	125

DAFTAR GAMBAR

2.1	Ilustrasi Pemanfaatan <i>Wireless Sensor Network</i>	6
2.2	Struktur Node Sensor	6
2.3	Topologi Point-to-Point	9
2.4	Topologi Bus	9
2.5	Topologi Ring	9
2.6	Topologi Star	10
2.7	Topologi Tree	10
2.8	Topologi Partially Connected Mesh	10
2.9	Topologi Fully Connected Mesh	11
2.10	Arsitektur Wireless Sensor Network	11
2.11	Arsitektur flat pada <i>Wireless Sensor Network</i>	12
2.12	Arsitektur hierarki pada <i>Wireless Sensor Network</i> dengan <i>single hop</i> terhadap <i>Cluster Head</i>	12
2.13	Arsitektur hierarki pada <i>Wireless Sensor Network</i> dengan <i>multi hop</i>	13
2.14	Clustering dengan multi tier	13
2.15	Arsitektur TinyOS	14
2.16	Arsitektur Contiki	15
2.17	Arsitektur LiteOS	15
2.18	Layer pada <i>Wireless Sensor Network</i>	16
2.19	Event Detection pada WSN	20
2.20	Hubungan antara RMST dengan Directed Diffusion	21
2.21	Penggunaan virtual machine pada perangkat yang berbeda	23
2.22	Preon32 Board	24
2.23	Tampilan struktur folder pada Sandbox Preon32	28
2.24	Struktur Frame Pada Physical Layer (PHY)	30
2.25	Struktur Frame Pada Medium Access Control (MAC) Layer	30
3.1	Diagram <i>use case</i> aplikasi data transfer pada WSN	34
3.2	Diagram Kelas Sederhana Dari Aplikasi Pada Node Sensor Biasa	38
3.3	Diagram Kelas Sederhana Dari Aplikasi Pada Handler	38
3.4	Diagram Kelas Sederhana Dari Aplikasi Pada Base Station	39
3.5	Contoh arsitektur flat	40
3.6	Flowchart transfer data dengan ACK dan <i>timer</i>	41
4.1	Diagram Sequence "Check Online"	45
4.2	Diagram Sequence "Synchronize Time"	46
4.3	Diagram Sequence "Get Time"	47
4.4	Diagram Sequence "Start Sensing"	48
4.5	Diagram Sequence "Exit"	49
4.6	Diagram Kelas Sensor-Sensor	50
4.7	Diagram Kelas Sensing	51
4.8	Diagram Kelas BS	52
4.9	Diagram Kelas NS	53

4.10	Diagram Kelas BS_Testing	55
4.11	Diagram Kelas NS_Testing	56
4.12	Diagram Kelas Handler	57
5.1	Tampilan Utama Aplikasi	75
5.2	Tampilan Check Online	75
5.3	Tampilan Synchronize Time	76
5.4	Tampilan Get Time	77
5.5	Tampilan Exit	77
5.6	Kesalahan Input 1	78
5.7	Kesalahan Input 2	78
5.8	Arsitektur flat <i>single-hop</i>	79
5.9	Grafik node DAAA pada single-hop	79
5.10	Grafik node DAAB pada single-hop	80
5.11	Grafik node DAAC pada single-hop	80
5.12	Grafik node DAAD pada single-hop	80
5.13	Arsitektur flat <i>multi-hop</i> tipe 1	81
5.14	Grafik node DAAA pada multi-hop tipe 1	81
5.15	Grafik node DAAB pada multi-hop tipe 1	81
5.16	Grafik node DAAC pada multi-hop tipe 1	82
5.17	Grafik node DAAD pada multi-hop tipe 1	82
5.18	Arsitektur flat <i>multi-hop</i> tipe 2	83
5.19	Grafik node DAAA pada multi-hop tipe 2	83
5.20	Grafik node DAAB pada multi-hop tipe 2	83
5.21	Grafik node DAAC pada multi-hop tipe 2	84
5.22	Grafik node DAAD pada multi-hop tipe 2	84
5.23	Arsitektur flat <i>multi-hop</i> tipe 3	84
5.24	Grafik node DAAA pada multi-hop tipe 3	85
5.25	Grafik node DAAB pada multi-hop tipe 3	85
5.26	Grafik node DAAC pada multi-hop tipe 3	85
5.27	Grafik node DAAD pada multi-hop tipe 3	86
5.28	Arsitektur flat <i>multi-hop</i> tipe 4	86
5.29	Grafik node DAAA pada multi-hop tipe 4	87
5.30	Grafik node DAAB pada multi-hop tipe 4	87
5.31	Grafik node DAAC pada multi-hop tipe 4	87
5.32	Grafik node DAAD pada multi-hop tipe 4	88
5.33	Arsitektur flat <i>multi-hop</i> tipe 5	88
5.34	Grafik node DAAA pada multi-hop tipe 5	89
5.35	Grafik node DAAB pada multi-hop tipe 5	89
5.36	Grafik node DAAC pada multi-hop tipe 5	89
5.37	Grafik node DAAD pada multi-hop tipe 5	90

DAFTAR TABEL

2.1	Jenis - jenis sensor yang dapat dimiliki node sensor	8
2.2	Tabel Perbandingan Sistem Operasi	16
2.3	Tabel Perbandingan Protokol	22
2.4	Tabel Radio Packages	25
2.5	Tabel Route Packages	26
2.6	Tabel Other Virtenio Packages	26
2.7	Tabel Java Related Packages	27
2.8	Tabel Fungsi-Fungsi Yang Dapat Digunakan Pada File "puild.xml"	29
3.1	Tabel skenario Mengetahui Node Yang Menyala.	34
3.2	Tabel skenario melakukan sinkronisasi waktu.	35
3.3	Tabel skenario mendapatkan waktu setiap node.	35
3.4	Tabel skenario melakukan perintah <i>sensing</i>	36
3.5	Tabel skenario mendapatkan data.	36
3.6	Tabel skenario keluar dari program (<i>exit</i>).	37
4.1	Tabel <i>routing</i> pada komunikasi <i>single-hop</i>	63
4.2	Tabel <i>routing</i> pada komunikasi <i>multi-hop</i>	64
5.1	Perbandingan waktu yang diperlukan node sensor mengumpulkan data (Single-Hop dan Multi-Hop tipe 1)	90
5.2	Perbandingan waktu yang diperlukan node sensor mengumpulkan data (Multi-Hop tipe 2 dan Multi-Hop tipe 3)	91
5.3	Perbandingan waktu yang diperlukan node sensor mengumpulkan data (Multi-Hop tipe 4 dan Multi-Hop tipe 5)	91
B.1	Jumlah Data Yang Diterima Node DAAA Single Hop	113
B.2	Jumlah Data Yang Diterima Pada Node DAAB Single Hop	113
B.3	Jumlah Data Yand Diterima Pada Node DAAC Single Hop	114
B.4	Jumlah Data Yang Diterima Pada Node DAAD Single Hop	114
C.1	Jumlah Data Yang Diterima Pada Node DAAA Multi Hop Tipe 1	115
C.2	Jumlah Data Yang Diterima Pada Node DAAB Multi Hop Tipe 1	115
C.3	Jumlah Data Yang Diterima Pada Node DAAC Multi Hop Tipe 1	116
C.4	Jumlah Data Yang Diterima Pada Node DAAD Multi Hop Tipe 1	116
D.1	Jumlah Data Yang Diterima Pada Node DAAA Multi Hop Tipe 2	117
D.2	Jumlah Data Yang Diterima Pada Node DAAB Multi Hop Tipe 2	117
D.3	Jumlah Data Yang Diterima Pada Node DAAC Multi Hop Tipe 2	118
D.4	Jumlah Data Yang Diterima Pada Node DAAD Multi Hop Tipe 2	118
E.1	Jumlah Data Yang Diterima Pada Node DAAA Multi Hop Tipe 3	119
E.2	Jumlah Data Yang Diterima Pada Node DAAB Multi Hop Tipe 3	119
E.3	Jumlah Data Yang Diterima Pada Node DAAC Multi Hop Tipe 3	120

E.4	Jumlah Data Yang Diterima Pada Node DAAD Multi Hop Tipe 3	120
F.1	Jumlah Data Yang Diterima Pada Node DAAA Multi Hop Tipe 4	121
F.2	Jumlah Data Yang Diterima Pada Node DAAB Multi Hop Tipe 4	121
F.3	Jumlah Data Yang Diterima Pada Node DAAC Multi Hop Tipe 4	122
F.4	Jumlah Data Yang Diterima Pada Node DAAD Multi Hop Tipe 4	122
G.1	Jumlah Data Yang Diterima Pada Node DAAA Multi Hop Tipe 5	123
G.2	Jumlah Data Yang Diterima Pada Node DAAB Multi Hop Tipe 5	123
G.3	Jumlah Data Yang Diterima Pada Node DAAC Multi Hop Tipe 5	124
G.4	Jumlah Data Yang Diterima Pada Node DAAD Multi Hop Tipe 5	124

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 suhu, 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* (*multi-hop*) dan langsung dari node sensor ke *base station* (*single-hop*).

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 dibangun dengan menggunakan node sensor jenis Preon32. Preon32 adalah node sensor yang dibuat oleh perusahaan Virtenio dan menggunakan bahasa pemrograman JAVA. Aplikasi WSN yang dibangun dapat melakukan transfer data ke node sensor tetangganya hingga sampai ke node sensor yang berperan sebagai *base station* dengan *single-hop* dan *multi-hop*. Karena data yang akurat sangat dibutuhkan untuk menentukan tindakan selanjutnya. Data yang lengkap juga membantu dalam melihat riwayat kejadian pada suatu tempat. Oleh karena itu dibangun juga aplikasi WSN yang memiliki sifat *reliable*.

1.2 Rumusan Masalah

- 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 suhu, kelembaban udara, dan getaran.
2. Arsitektur yang digunakan untuk membangun *Wireless Sensor Network* ini adalah arsitektur flat dengan *single-hop* dan *multi-hop*.
3. Fokus utama penelitian ini adalah membangun aplikasi transfer data yang *reliable* oleh karena itu tidak memperhitungkan konsumsi energi dan *delay* dalam melakukan pengiriman waktu.

1.5 Metodologi

Berikut adalah metode penelitian yang digunakan dalam penelitian 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* yang dapat digunakan 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*.
7. Melakukan pengujian aplikasi terkait *reliability* dalam pengiriman data.

1.6 Sistematika Pembahasan

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

Bab 1 Pendahuluan, yaitu membahas mengenai gambaran umum penelitian ini. Berisi tentang latar belakang, rumusan masalah, tujuan, batasan masalah, metode penelitian, dan sistematika penulisan.

Bab 2 Dasar Teori, yaitu membahas teori-teori yang mendukung berjalannya penelitian ini. Berisi tentang *Wireless Sensor Network*, *Reliable Data Transfer* di WSN, *PreonVM*, dan *Format Struktur Frame*.

Bab 3 Analisis, yaitu membahas mengenai analisis masalah. Berisi tentang analisis aplikasi pengiriman data pada WSN, analisis proses pengiriman data dari node sensor ke *base station* pada WSN, dan analisis protokol transfer yang *reliable* pada *Wireless Sensor Network*.

Bab 4 Perancangan, yaitu membahas perancangan aplikasi WSN yang *reliable* untuk pengiriman data. Berisi tentang perancangan interaksi antar node untuk transfer data di WSN, perancangan kelas aplikasi transfer data di WSN, perancangan masukan dan keluaran, perancangan pseudocode aplikasi transfer data yang *reliable*, perancangan *routing* pada aplikasi transfer data, dan perancangan format pesan.

Bab 5 Implementasi dan Pengujian, yaitu membahas implementasi dari hasil rancangan dan pengujian dari aplikasi WSN yang telah dibuat dan pengujian aplikasi yang telah dibuat.

Bab 6 Kesimpulan, yaitu membahas kesimpulan dari hasil pengujian dan saran untuk pengembangan selanjutnya.

BAB 2

LANDASAN TEORI

Pada bab ini dijelaskan dasar-dasar teori mengenai *Wireless Sensor Network*, Reliable Data Transfer di WSN, PreonVM, dan Format Struktur Frame.

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 [6]. Sensor ini akan mengumpulkan data dari kondisi lingkungannya, seperti: cahaya, suara, kelembaban, 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.

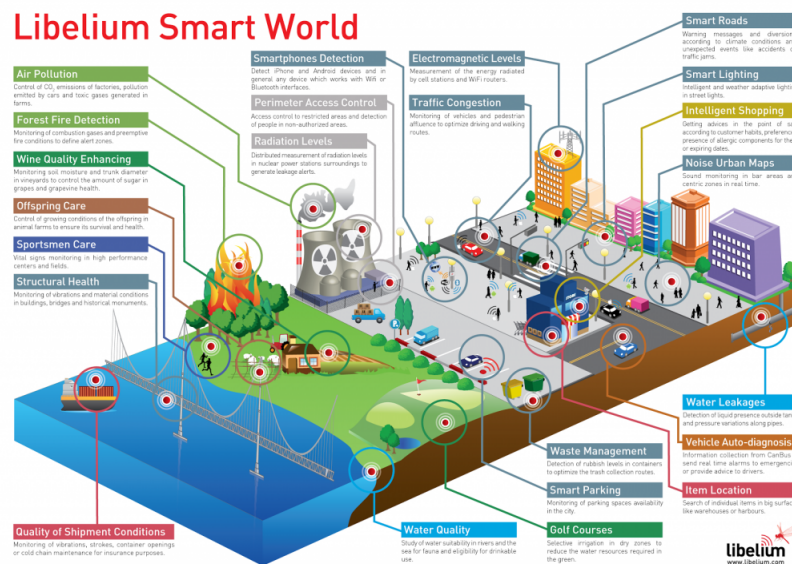
2.1.1 Penerapan Wireless Sensor Network [1]

Pada awalnya 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 2.1¹. 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 dilakukan antara lain deteksi gunung berapi, polusi udara, kebakaran hutan, efek rumah kaca, dan deteksi longsor.

¹http://www.libelium.com/resources/top_50_iot_sensor_applications_ranking/#show_infographic

- **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.

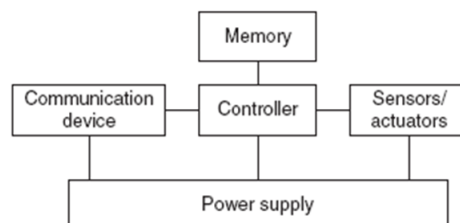


Gambar 2.1: Ilustrasi Pemanfaatan *Wireless Sensor Network*

2.1.2 Node Sensor

Struktur Node Sensor

Setiap node sensor memiliki kemampuan *sensing*, komputasi dan komunikasi. Node sensor memiliki lima komponen utama yaitu *controller*, *memory*, *sensor* dan *actuator*, *communication device*, dan *power supply*, (Gambar 2.2) [7]. 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 kelembaban, 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 vulkanik
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 menggerakkan kincir atau turbin yang akan menghasilkan listrik dan digunakan sebagai sumber energi pada node sensor.

2.1.3 Arsitektur dan Topologi Wireless Sensor Network [2]

Pada WSN biasanya akan terdapat banyak node sensor yang tersebar 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 biasa 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

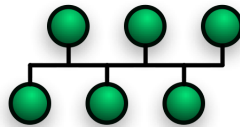
Topologi *Point-to-Point* adalah topologi yang menghubungkan dua titik (Gambar 2.3). Topologi *Point-to-Point* dibagi menjadi dua yaitu *permanent point-to-point* dan *switched point-to-point*. *Permanent point-to-point* adalah koneksi perangkat keras antara dua titik dan tidak dapat diubah. *Switched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



Gambar 2.3: Topologi Point-to-Point

Topologi Bus

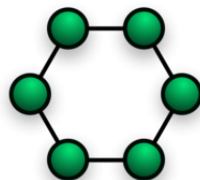
Topologi Bus seperti pada Gambar 2.4 akan terdiri dari node-node dan sebuah jalur. Setiap node akan terhubung dengan satu jalur yang sama. Untuk mengirim data atau komunikasi akan dilakukan bergantian antar node. Kekurangan dari topologi bus ini adalah jika suatu saat jalur atau bus ini mengalami kerusakan 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 terhubung 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 terhubung dengan node yang berada di tengah ini. Saat node

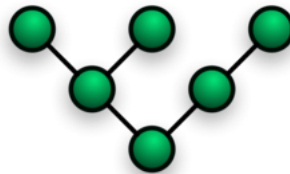
akan berkomunikasi dengan node lain, node tersebut harus mengirimkan data tersebut ke node yang ada di tengah. Setelah itu node yang berada di tengah ini akan meneruskan data tersebut ke node tujuan. Hal yang paling penting pada topologi ini adalah node yang berada di tengah, karena semua komunikasi 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

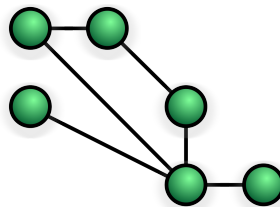
Pada Topologi *Tree* node-node akan disusun secara hierarki dengan satu node yang berada pada level paling atas sebagai *root node* (Gambar 2.7). *Root node* akan terhubung dengan satu atau lebih node level di bawahnya. Dengan Topologi *Tree* lebih mudah untuk melakukan identifikasi dan meminimalisir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



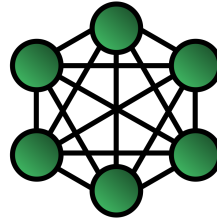
Gambar 2.7: Topologi Tree

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.

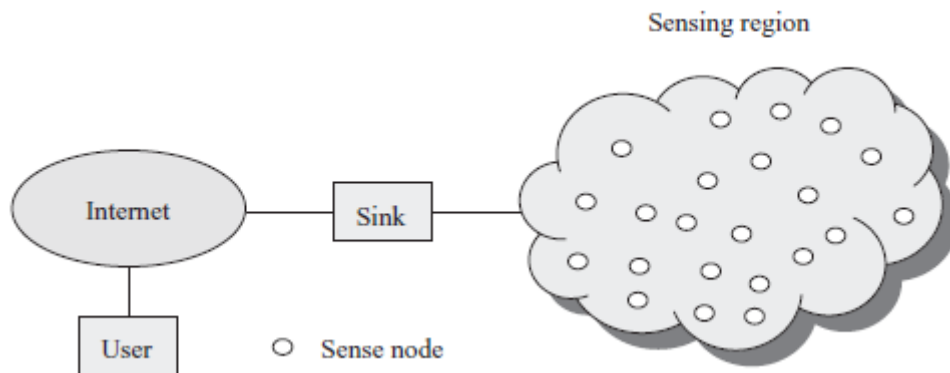


Gambar 2.8: Topologi Partially Connected Mesh



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 [7]

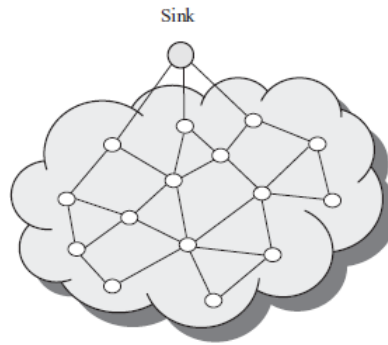
Untuk mengirim data ke *sink node* setiap node sensor dapat menggunakan *single-hop long-distance transmission*. *Single-hop long-distance* ini berarti setiap node sensor akan mengirimkan data ke sink node hanya satu kali lompatan walaupun jarak antara *sink node* dengan node sensor itu sangat jauh. Dalam jaringan sensor, penggunaan energi paling besar adalah saat melakukan komunikasi dibandingkan saat *sensing*. Penggunaan energi akan semakin bertambah jika jarak *sink node* dengan 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 [3]

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.

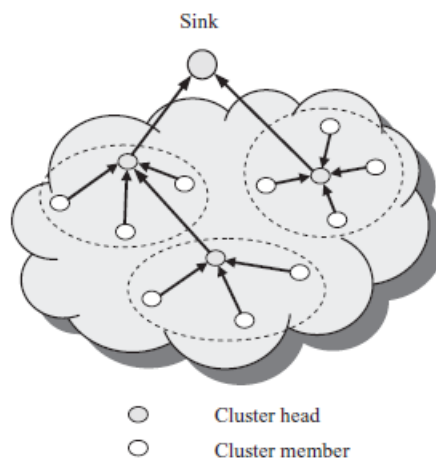


Gambar 2.11: Arsitektur flat pada *Wireless Sensor Network*

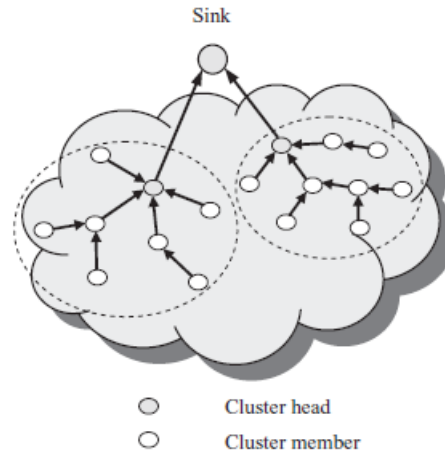
Arsitektur Hierarki [3]

Pada arsitektur hierarki, semua node sensor dikelompokkan ke dalam *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*.

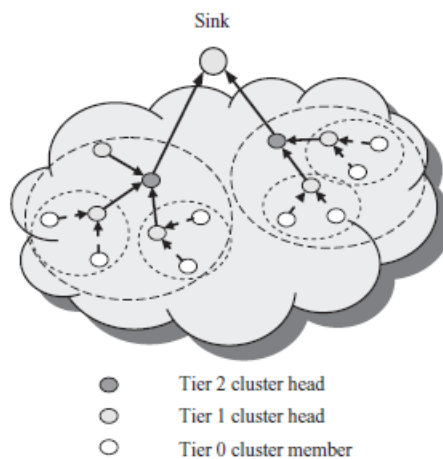
Masalah yang utama pada *clustering* ini adalah pemilihan *cluster head* dan bagaimana cara mengatur setiap *cluster*. Terdapat beberapa cara untuk membuat *clustering* ini. Berdasarkan jarak antara *cluster head* dengan *cluster member*, dapat dibuat *clustering* dengan *single-hop* atau *multi-hop* 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.[6] Terdapat beberapa hal yang harus ditangani oleh sistem operasi dalam WSN yaitu:

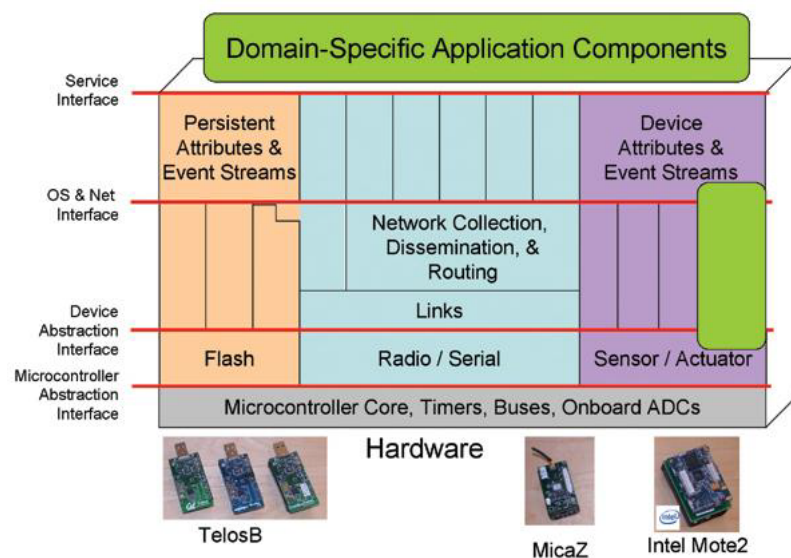
1. WSN memerlukan *real-time scheduler*. Data yang didapat harus segera dikirim atau diproses.
2. Pengaturan memori karena memori pada WSN sangat kecil.
3. Pengaturan data yang efisien terkait dengan *microprocessor* dan memori yang terbatas
4. Mendukung kode pemrograman yang efisien dan *reliable* karena 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 [8]

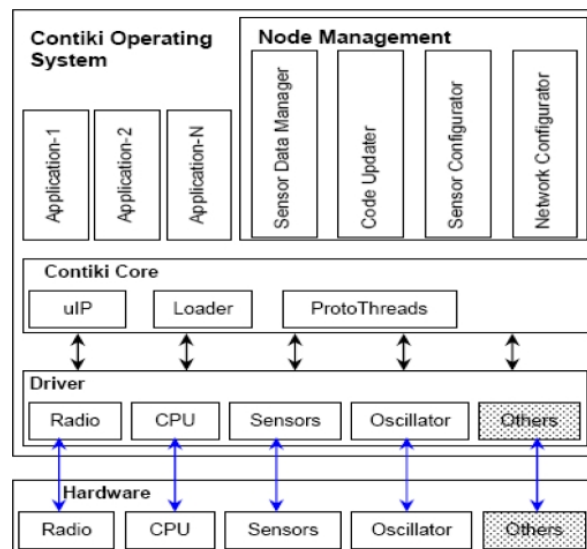
TinyOS adalah sistem operasi *open-source* yang digunakan pada WSN. TinyOS dapat menjalankan program dengan memori yang sangat kecil. Ukurannya hanya 400 Byte. Komponen *library* TinyOS terdiri dari protokol jaringan, layanan distribusi sensor, *driver sensor*, dan perangkat lunak pengamatan data sensor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



Gambar 2.15: Arsitektur TinyOS

Contiki [8]

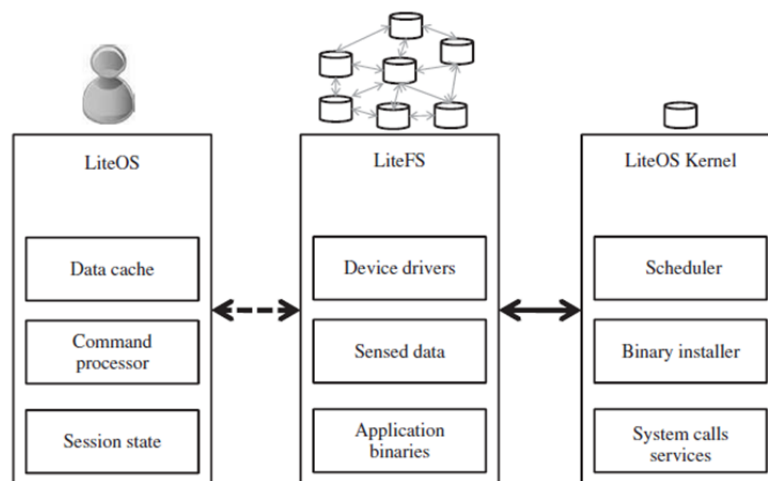
Contiki adalah sistem operasi *open-source* dengan Bahasa Pemrograman C yang digunakan pada WSN. Pengaturan Contiki hanya memerlukan 2KB dari RAM dan 40KB dari ROM. Fitur yang dimiliki oleh Contiki antara lain: *multitasking*, *multithreading*, jaringan TCP/IP, IPv6, GUI, *Web Browser*, *Web Server*,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



Gambar 2.16: Arsitektur Contiki

LiteOS [8]

LiteOS adalah sistem operasi mirip UNIX yang didisain untuk WSN. Tujuan dibuat LiteOS adalah membuat sistem operasi yang mirip dengan UNIX agar lebih familiar dengan paradigma pemrograman UNIX. Pada LiteOS terdapat sistem berkas yang hierarki, 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

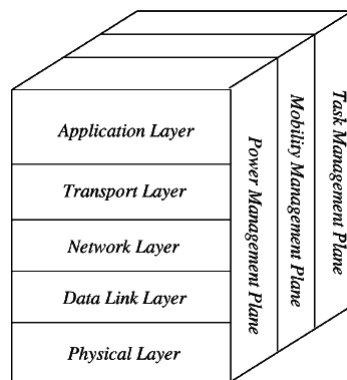
Perbandingan dari TinyOS, Contiki, dan LiteOS dapat dilihat pada Tabel 2.2 berikut ini:

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 [3]

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.[9] 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 trancievernya. 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 kebutuhan 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 [4]

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

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

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

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

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 dapat diimplementasikan pada WSN. Protokol transport tersebut dapat dibuat, namun 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*) [10]. 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 [5]

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 [5]

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)

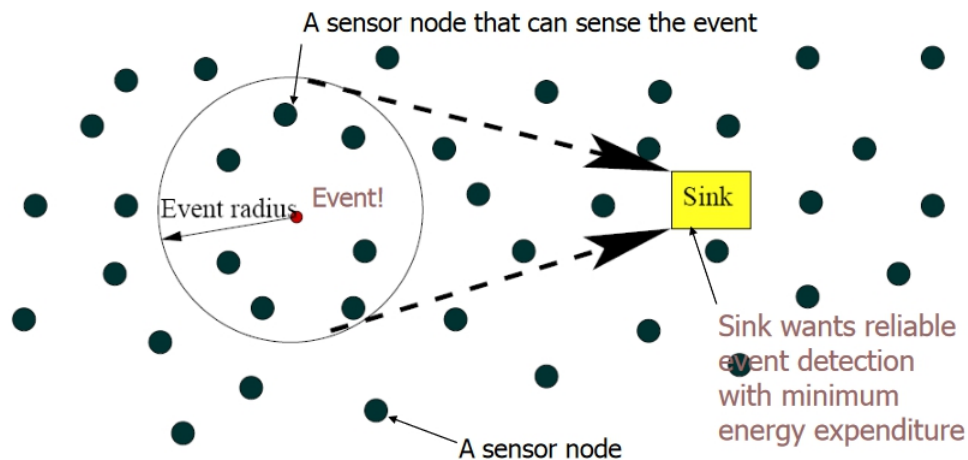
GARUDA

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 [11]

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 menunjukkan *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

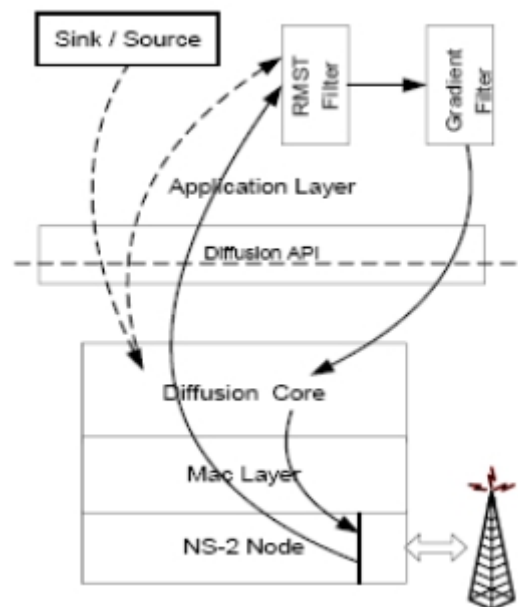
Pada ESRT setiap node sensor mendeteksi *congestion* berdasarkan peningkatan *buffer* lalu menambahkan N bit pada *header* sebuah paket dan meneruskannya ke sink node. Berdasarkan paket yang diterima oleh *sink node*, dapat diketahui keadaan dari jaringan dan juga *reliability* suatu paket untuk menentukan 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 [12]

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* atau *end-to-end*. Saat terjadi data *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 *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* dan penggunaan energi yang efisien.



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. Pada PSFQ terdapat tiga *operation* yaitu *pump*, *fetch*, dan *report*.

Berikut adalah cara kerja PSFQ. Pertama, secara periodik 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. 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 dengan semua paket yang dikirimkan pada saat tersebut.

Asymmetric Reliable Transport [13]

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 [14]

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 [15]

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	End-to-End	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 terbatas². 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 Pemrograman Java yang mengumpulkan data dari sensor dan mengontrol aktuator. API pada PreonVM mendukung antarmuka radio sesuai dengan IEEE 802.15.4.

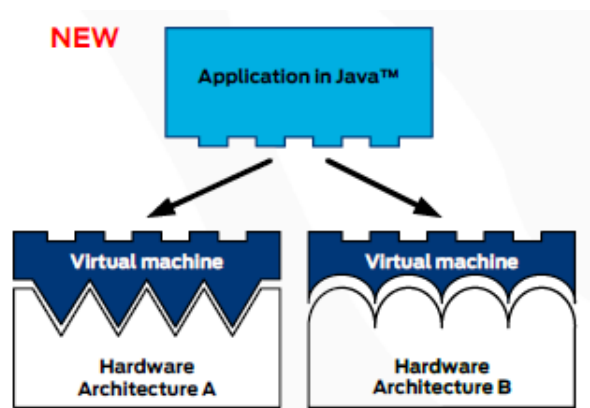
PreonVM memiliki fitur sebagai berikut:

- Aplikasi dibangun dengan Bahasa Pemrograman Java

²<https://www.virtenio.com/en/preonvm-virtual-machine.html>

- 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 *thread* termasuk *synchronized*, *Object.wait*, *Object.notify*, *Object.notifyAll*, *Thread.sleep*, atau *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 arsitektur yang berbeda tanpa harus modifikasi. Gambar 2.21 menunjukkan ilustrasi penggunaan virtual machine pada berbagai perangkat.



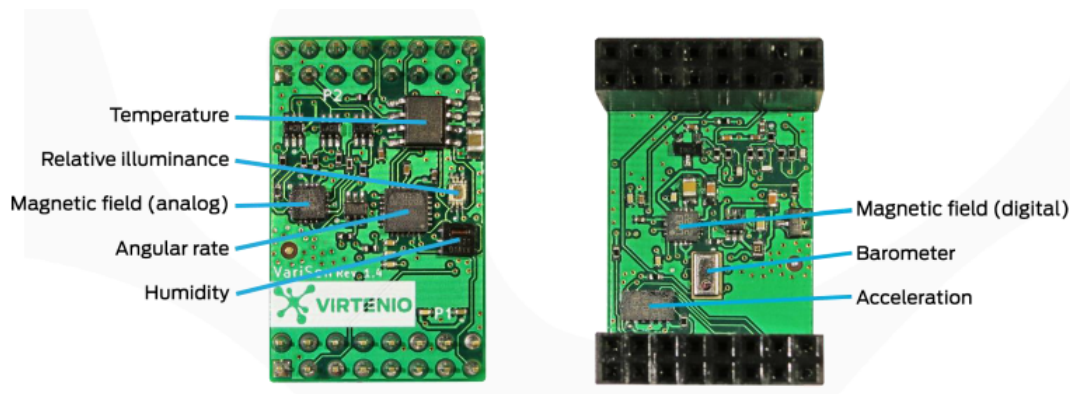
Gambar 2.21: Penggunaan virtual machine pada perangkat yang berbeda

Virtual machine juga membantu memisahkan proses pengembangan perangkat keras dengan aplikasi perangkat lunak. Program aplikasi dikembangkan 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 sensor³. Preon32 menggunakan PreonVM sebagai *operating software*. Preon32 versi umum memiliki 5 jenis sensor pada sebuah *board*. Sensor yang ada pada Preon32 ini antara lain sensor suhu (*temperature sensor*), sensor cahaya (*light intensity sensor*), sensor kelembaban udara (*relative humidity sensor*), sensor tekanan udara (*air pressure sensor*), dan sensor getaran (*acceleration sensor*). Pada versi tambahan Preon32 dapat dilengkapi dengan sensor untuk mendeteksi medan magnet, dan *gyroscope*. Semua jenis sensor tersebut dapat diatur melalui PreonVM dan pemrograman dapat dilakukan dengan Bahasa Pemrograman Java. Gambar 2.22 adalah *board* Preon32 beserta letak sensor-sensor tersebut.

³<https://www.virtenio.com/en/products/radio-module.html>



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. Aplikasi pada jaringan nirkabel
4. Algoritma *sensor data fusion*

Spesifikasi Sensor Pada Preon32

Berikut adalah spesifikasi setiap sensor yang dimiliki oleh Preon32 ⁴:

Sensor Suhu *Temperature Sensor*

- Manufacture : Analog Devices
- Model : ADT7410
- Interface : digital, I2C
- Resolution : 16-Bit
- Range : -40°C sampai +105°C
- Accuracy : $\pm 0.5^\circ\text{C}$

Sensor Intensitas Cahaya *Light Intensity Sensor*

- Manufacture : Rohm
- Model : BH1715FVC
- Interface : digital, I2C
- Resolution : 16-Bit
- Range : 1 lx to 65355 lx

⁴<https://www.virtenio.com/en/products/sensor-module.html>

Sensor kelembaban Udara Relatif *Relative Humidity Sensor*

- Manufacture : Sensirion
- Model : SHT21
- Interface : digital, I2C
- Resolution : 12-Bit
- Range : 0 %RH sampai 100 %RH
- Accuracy : $\pm 2,0$ %RH (typ.)

Sensor Tekanan Udara *Air Pressure Sensor*

- Manufacture : Freescale
- Model : MPL115A2
- Interface : digital, I2C
- Resolution : 0,15 kPa
- Range : 50 kPa sampai 115 kPa
- Accuracy : ± 1.0 kPa

Sensor Getaran *Acceleration Sensor*

- Manufacture : Analog Devices
- Model : ADXL345
- Interface : digital, SPI
- Resolution : 13 Bit per axis
- Range : ± 16 g, 3 axis
- Accuracy : 3,9 mg/LSB

2.3.2 Class Library JVM Preon32

Preon32 menggunakan 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***. *Class Library JVM Preon32* dapat dilihat pada website resmi milih Virtenio ⁵. Tabel 2.4, sampai Tabel 2.7 berisi package yang terdapat pada setiap package utama.

Tabel 2.4: Tabel Radio Packages

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

⁵<https://www.virtenio.com/assets/vm/javadoc/overview-summary.html>

Tabel 2.5: Tabel Route Packages

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

Tabel 2.6: Tabel Other Virtenio 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 device 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 2.7: Tabel Java Related 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

2.3.3 Pemrograman Pada Preon32

Pada subbab 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 membangun 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

Pada bagian ini dijelaskan pengaturan dan adaptasi dari Sandbox.

ANT script "build.xml"

Pada file "build.xml" ini terdapat beberapa fungsi-fungsi yang dapat digunakan saat membangun aplikasi Preon32. Fungsi-fungsi tersebut ditunjukkan pada Tabel 2.8 :

Sandbox

- bin ... Digunakan Eclipse untuk *binary files*.
- build ... Folder yang mengandung file-file dari hasil pembangunan *project* dan akan diunggah ke Preon32.
- config ... Folder yang mengandung *application context* dan *properties files*. Pada folder ini dapat ditemukan beberapa context dan properties.
 - context1.properties
 - currentContext.properties
 - device.properties
- example ... Folder yang mengandung contoh-contoh *source code* untuk digunakan pada Preon32.
- lib ... Folder yang mengandung *user libraries*.
- license ... Folder yang mengandung file-file lisensi. Lisensi ini diperlukan untuk melakukan pengembangan.
- src ... Folder yang mengandung file-file *source code* dari aplikasi yang dibuat.
- target ... Folder yang mengandung file dengan fungsi spesifik seperti *virtual machine* dan *native runtime libraries* (*rt* dan *rtx*).
- buildUser.xml ... File ANTscript untuk menentukan perangkat mana yang sedang dilakukan konfigurasi .
- build.xml ... File yang mengandung ANTtarget. Target disini digunakan untuk melakukan interaksi antara Preon32 dengan Sandbox ini.
- .classpath ... File Classpath pada Eclipse.
- .project ... File Project yang digunakan pada Eclipse.

Gambar 2.23: Tampilan struktur folder pada Sandbox Preon32

Tabel 2.8: Tabel Fungsi-Fungsi Yang Dapat Digunakan Pada File "puild.xml"

Nama Fungsi	Nama Fungsi
.all	cmd.license.upload
.run	cmd.module.run
boot.erase	cmd.module.upload
boot.firmware.upload	cmd.module.upload.run
boot.info	cmd.modules.list
boot.options.cmd.disable	cmd.properties.all
boot.options.cmd.enable	cmd.properties.clear
boot.options.modify.disable	cmd.properties.list
boot.options.modify.enable	cmd.properties.upload
boot.options.print	cmd.terminal.open
boot.options.read.protect	cmd.time.synchronize
boot.options.read.unprotect	devel.all
boot.options.reset	devel.clean
boot.options.wdg.disable	devel.compile
boot.options.wdg.enable	devel.jar
boot.options.write.protect	devel.proguard
boot.options.write.unprotect	devel.properties
boot.reset	devel.vmm
cmd.info	init

ANT script "buildUser.xml"

ANT script "buildUser.xml" digunakan untuk memilih *context* yang akan digunakan. *Context* yang telah dipilih dapat menjalankan fungsi-fungsi yang terdapat pada Tabel 2.8. Pada file "buildUser.xml" ini dapat dibuat *context* baru sesuai yang diperlukan. Untuk membuat *context* baru, perlu menambahkan Listing 2.1 dibawah *context* yang sudah ada.

Listing 2.1: Contoh membuat context baru

```

1 | <target name="context.set.1">
2 |   <switchContext to="config/context1.properties" />
3 | </target>

```

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.

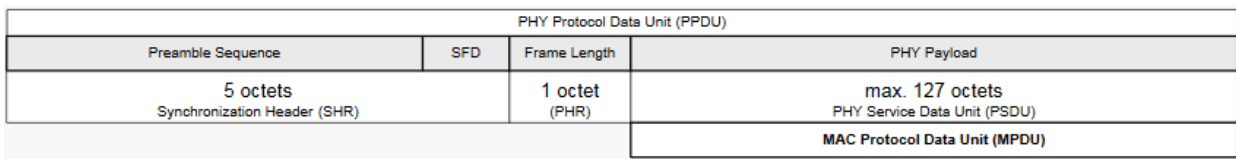
File paling penting pada direktori "config" adalah "context1.properties". Pada file ini terdapat pengaturan atau properties dari program yang dibuat. Properties penting dalam file ini antara lain:

1. **mainClass.name = Prog.** Merupakan nama kelas utama yang digunakan dan akan di eksekusi.
2. **module.name = prog.** Merupakan nama dari sebuah modul yang telah dibuat. Modul yang dibuat dapat berjalan secara otomatis (*autostart*) dengan menggunakan nama "autostart" pada module.name ini.

3. **target.dir = targets/Preon32.** Merupakan direktori yang berisi firmware sesuai dengan perangkat yang digunakan (Preon32).
4. **device.properties.file = config/device.properties.** Properties yang didefinisikan pada file ini dapat dibaca dalam aplikasi pengguna.
5. **comport = COM10.** Merupakan *port* mana yang terhubung dengan Preon32.

2.4 Format Struktur Frame

Berdasarkan Standard IEEE 802.15.4, Gambar 2.24 adalah format dari struktur frame pada Physical Layer (PHY). Gambar 2.25 adalah format dari struktur frame pada Medium Access Control (MAC) layer.



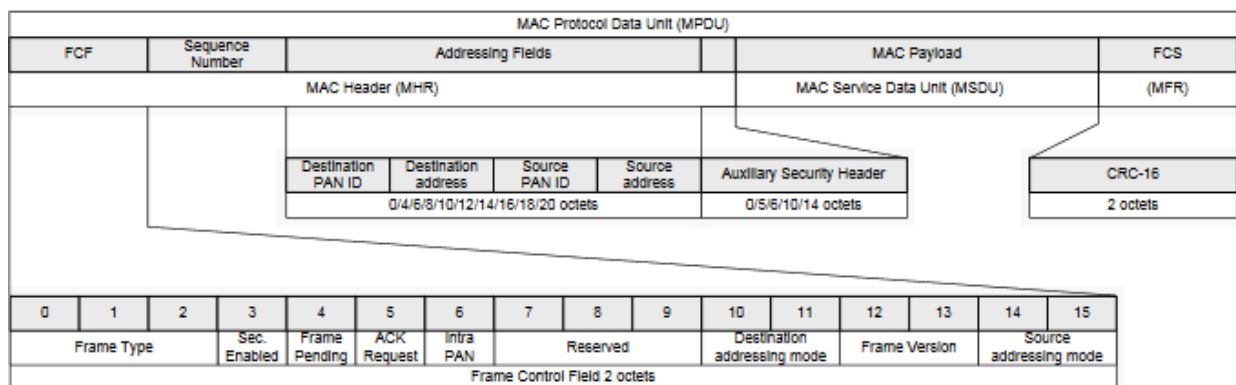
Gambar 2.24: Struktur Frame Pada Physical Layer (PHY)

PHY Protocol Layer Data Unit (PPDU)

PHY Protocol Layer Data Unit terdiri dari beberapa bagian yaitu:

1. **Synchronization Header (SHR)**, Saat transfer data, SHR ini akan secara otomatis dihasilkan oleh *microcontroller*. Sehingga pada *frame buffer* akan terdiri dari PHR dan PSDU.
2. **PHY Header (PHR)**, PHY Header terdiri dari 1 byte untuk menunjukan panjang dari *frame* tersebut.
3. **PHY Payload (PHY Service Data Unit, PSDU)**, PSDU memiliki panjang antara 0 sampai maksimal 127 byte dengan 2 byte akhir digunakan sebagai *Frame Check Sequence* (FCS). Panjang dari PSDU ini ditentukan oleh PHR. Pada PSDU terdapat MAC Protocol Layer Data Unit (MPDU).

MAC Protocol Layer Data Unit (MPDU)



Gambar 2.25: Struktur Frame Pada Medium Access Control (MAC) Layer

MAC Protocol Layer Data Unit terdiri dari beberapa bagian yaitu:

-
1. **Frame Control Field (FCF)**, terdiri dari 16 bit dan setiap bit memiliki fungsi yang berbeda-beda.
 2. **Sequence Number**, terdiri dari 1 byte untuk mendeteksi duplikasi *frame*.
 3. **Addressing Fields**, terdiri dari alamat - alamat yang diperlukan untuk melakukan transfer data. Alamat - alamat ini antara lain *Destination PAN ID*, *Destination Address*, *Source PAN ID*, dan *Source Address*.
 4. **MAC Payload**, merupakan data yang akan dikirimkan dari node sensor ke node sensor lain.
 5. **Frame Check Sequence (FCS)**, merupakan *checksum* untuk memastikan *bit error* atau redundansi data.

BAB 3

ANALISIS

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

3.1 Analisis Aplikasi Pengiriman Data Pada WSN

Aplikasi transfer data pada WSN dapat menggunakan arsitektur flat maupun hierarki. Komunikasi yang dilakukan pada setiap arsitektur dapat menggunakan *single-hop* maupun *multi-hop*. Dalam melakukan transfer data dapat terjadi *loss*. Untuk menangani *loss* harus dilakukan pengiriman ulang data atau *retransmission*.

Pada skripsi ini dibangun aplikasi yang dapat melakukan transfer data dari node sensor ke *base station*. Aplikasi WSN yang dibangun menggunakan arsitektur flat dengan *single-hop* dan *multi-hop*. Aplikasi ini digunakan untuk mengetahui keadaan suatu tempat atau daerah dengan bantuan node sensor untuk mendapatkan data (*sensing*).

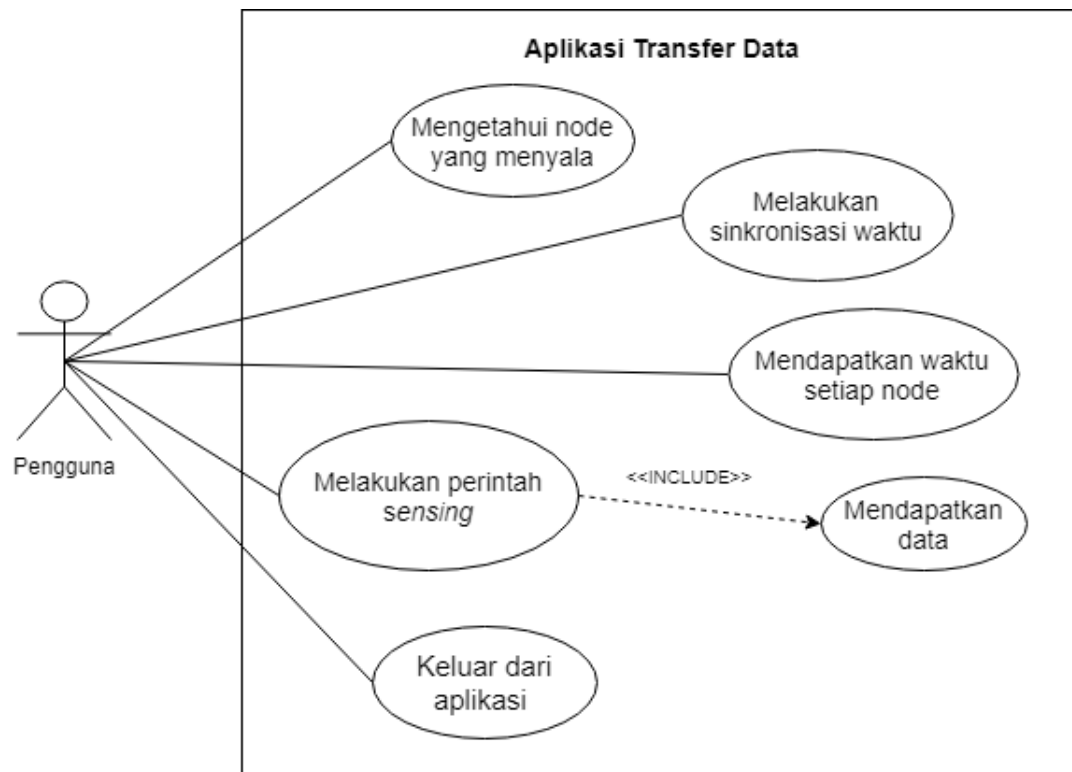
Aplikasi ini memiliki beberapa fungsi. Fungsi utama pada aplikasi ini adalah untuk melakukan transfer data dari setiap node sensor. Fungsi lain pada aplikasi ini meliputi:

1. Mengetahui node sensor yang menyala.
2. Melakukan sinkronisasi waktu.
3. Mendapatkan waktu dari setiap node sensor.

Fungsi mengetahui node sensor yang menyala digunakan agar pengguna tahu node sensor mana saja yang sedang menyala. Sehingga jika diperlukan lebih mudah melakukan perbaikan terhadap node sensor yang tidak menyala tersebut.

Fungsi melakukan sinkronisasi waktu digunakan untuk membuat semua node sensor memiliki waktu yang sama dengan waktu pada *base station*. *Base station* akan mendapatkan waktu dari komputer pengguna. Waktu tersebut akan disebarkan kepada node sensor lain pada jaringan tersebut. Dalam melakukan sinkronisasi waktu ini, penulis tidak memperhitungkan *delay* waktu saat waktu diterima oleh node sensor. Sinkronisasi waktu ini juga diperlukan untuk mengetahui kapan data didapatkan. Selain itu waktu awal saat node sensor dinyalakan adalah 1 Januari 1970, maka perlu disesuaikan dengan waktu saat ini. Cara sinkronisasi waktu yang digunakan adalah mengirim langsung waktu dari komputer pengguna ke setiap node sensor. Sinkronisasi ini tidak menggunakan algoritma. *Base station* yang sudah mendapatkan waktu akan meneruskan waktu kepada node sensor lain hingga semua node sensor memiliki waktu yang sinkron.

Fungsi mendapatkan waktu setiap node sensor digunakan untuk mengetahui waktu dari setiap node sensor. Fungsi ini juga digunakan untuk mengetahui apakah node sensor sudah memiliki waktu yang sinkron atau belum. Fungsi lain ini dijelaskan menggunakan diagram *use case* pada Gambar 3.1 dan skenario pada Tabel 3.1 sampai Tabel 3.6.

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

Tabel 3.1: Tabel skenario Mengetahui Node Yang Menyala.

Nama	Mengetahui Node Yang Menyala
Deskripsi	Mengetahui node-node mana saja yang sedang menyala (<i>online</i>).
Aktor	Pengguna
Pre-kondisi	Aplikasi sudah dijalankan dan pengguna memilih pilihan "Check Online".
Alur Skenario Utama	<ol style="list-style-type: none"> 1. Sistem memuat aplikasi. 2. Sistem menampilkan pilihan untuk dipilih pengguna. 3. Pengguna memilih "Check Online". 4. Sistem menampilkan semua node yang sedang menyala (<i>online</i>). 5. Sistem kembali pada tampilan utama.

Tabel 3.2: Tabel skenario melakukan sinkronisasi waktu.

Nama	Melakukan sinkronisasi waktu
Deskripsi	Melakukan sinkronisasi waktu untuk mengetahui kapan data dikirim dari node sensor dan diterima oleh <i>base station</i> .
Aktor	Pengguna
Pre-kondisi	Aplikasi sudah dijalankan dan pengguna memilih pilihan "Synchronize Time".
Alur Skenario Utama	<ol style="list-style-type: none"> 1. Sistem memuat aplikasi. 2. Sistem menampilkan pilihan untuk dipilih pengguna. 3. Pengguna memilih melakukan sinkronisasi waktu ("Synchronize Time"). 4. Sistem melakukan sinkronisasi waktu pada setiap node sensor. 5. Sistem menampilkan "Done synchronize". 6. Sistem kembali pada tampilan utama.

Tabel 3.3: Tabel skenario mendapatkan waktu setiap node.

Nama	Mendapatkan waktu setiap node sensor
Deskripsi	Pengguna akan meminta waktu setiap node sensor untuk ditampilkan pada aplikasi.
Aktor	Pengguna
Pre-kondisi	Aplikasi sudah dijalankan dan pengguna memilih pilihan "Get Time".
Alur Skenario Utama	<ol style="list-style-type: none"> 1. Sistem memuat aplikasi. 2. Sistem menampilkan pilihan untuk dipilih pengguna. 3. Pengguna memilih mendapatkan waktu setiap node sensor ("Get Time"). 4. Sistem menampilkan waktu dari setiap sensor. 5. Sistem kembali pada tampilan utama.

Tabel 3.4: Tabel skenario melakukan perintah *sensing*.

Nama	Menjalankan fungsi untuk <i>sensing</i>
Deskripsi	Pengguna menjalankan fungsi ini untuk membuat node sensor melakukan <i>sensing</i> dan mengirimkan data.
Aktor	Pengguna
Pre-kondisi	Sudah dilakukan sinkronisasi waktu pada setiap node sensor
Alur Skenario Utama	<ol style="list-style-type: none"> 1. Sistem memuat aplikasi. 2. Sistem menampilkan pilihan untuk dipilih pengguna. 3. Pengguna memilih untuk memulai <i>sensing</i>. 4. Sistem memerintahkan node sensor untuk melakukan <i>sensing</i>. 5. Sistem mendapatkan data dari setiap node sensor.

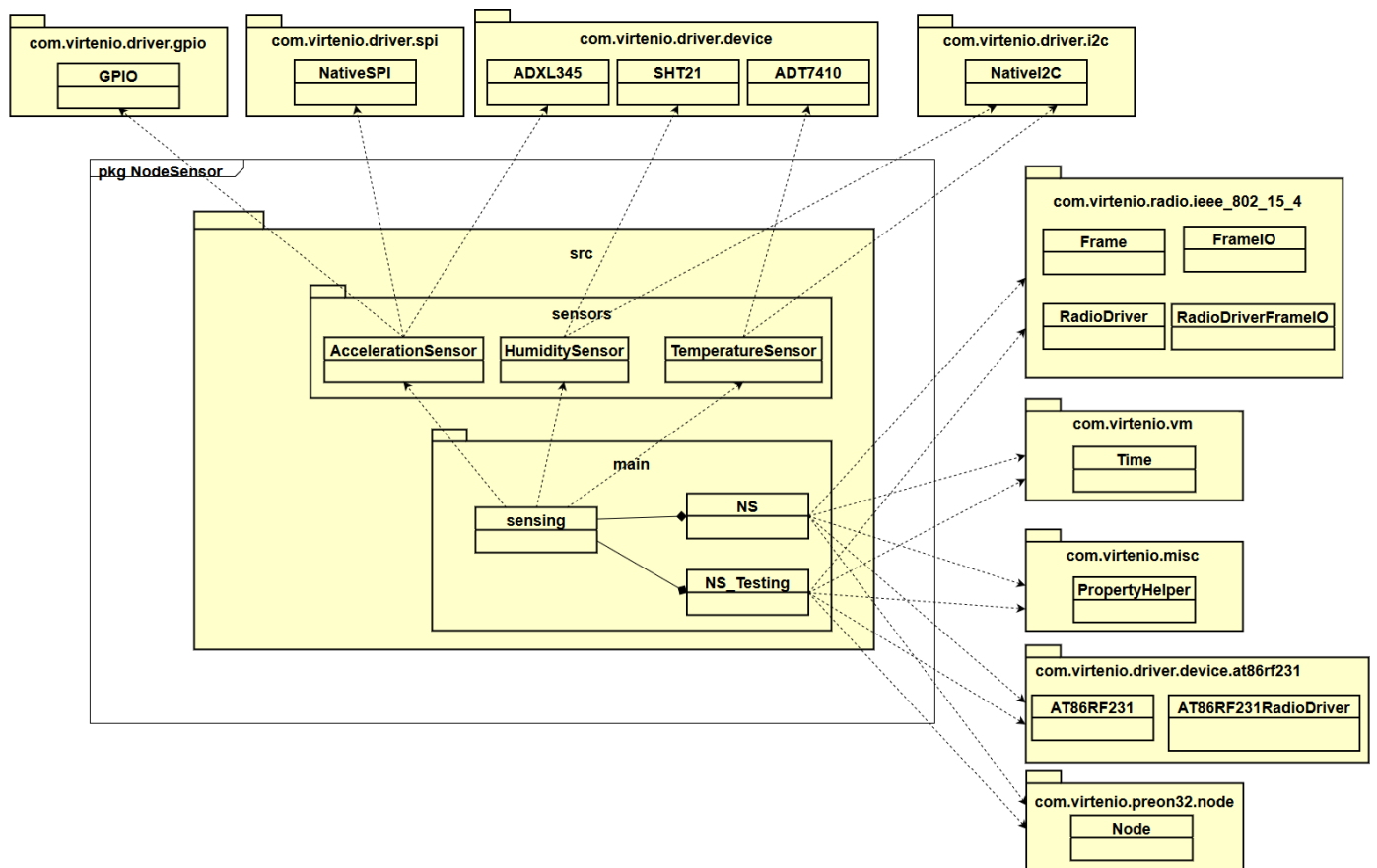
Tabel 3.5: Tabel skenario mendapatkan data.

Nama	Mendapatkan Data
Deskripsi	Pengguna mendapatkan data hasil <i>sensing</i> dari setiap node sensor berupa text file
Aktor	Pengguna
Pre-kondisi	Sistem sedang melakukan <i>sensing</i> .
Alur Skenario Utama	<ol style="list-style-type: none"> 1. Sistem melakukan <i>sensing</i>. 2. Sistem menyimpan data <i>sensing</i> ke dalam text file.

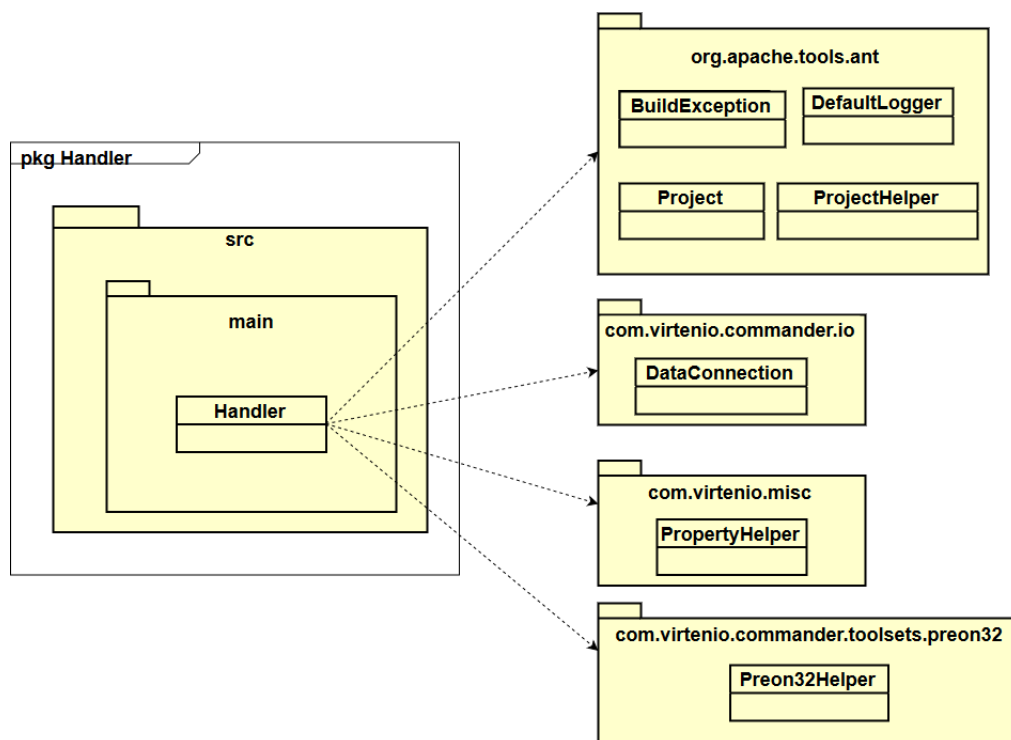
Tabel 3.6: Tabel skenario keluar dari program (*exit*).

Nama	Keluar dari program
Deskripsi	Pengguna keluar dari program dan aplikasi berhenti otomatis.
Aktor	Pengguna
Pre-kondisi	Sistem sedang berjalan (pada halaman utama atau sedang melakukan <i>sensing</i>).
Alur Skenario Utama	<ol style="list-style-type: none"> 1. Sistem memuat aplikasi. 2. Sistem menampilkan pilihan untuk dipilih pengguna. 3. Pengguna menjalankan fitur yang ada. 4. Pengguna memilih "Exit". 5. Sistem keluar dari aplikasi dan aplikasi berhenti berjalan.

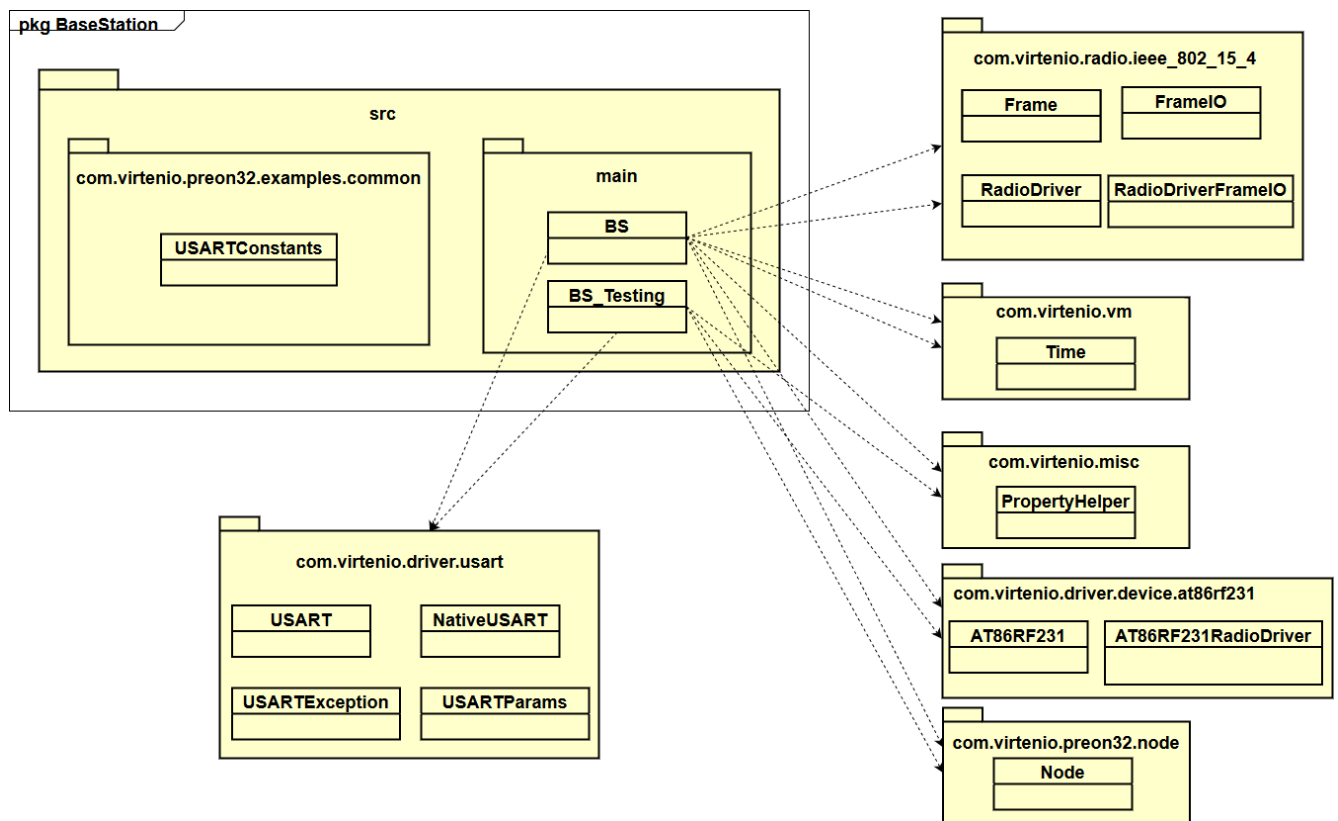
Pada subbab ini dibuat suatu diagram kelas sederhana untuk menjelaskan kelas-kelas yang dibutuhkan dalam membangun aplikasi transfer data di WSN. Aplikasi dibangun dengan Eclipse IDE dan menggunakan menggunakan Sandbox yang sudah disediakan oleh perusahaan Virmenio. Aplikasi yang dibuat ini terdiri dari program pada *base station* dan program pada node sensor biasa (Gambar 3.2). Aplikasi ini juga memerlukan program tambahan untuk menghubungkan *base station* dengan komputer pengguna (Handler).



Gambar 3.2: Diagram Kelas Sederhana Dari Aplikasi Pada Node Sensor Biasa



Gambar 3.3: Diagram Kelas Sederhana Dari Aplikasi Pada Handler



Gambar 3.4: Diagram Kelas Sederhana Dari Aplikasi Pada Base Station

Berikut adalah penjelasan dari kelas-kelas pada diagram di Gambar 3.2 hingga Gambar 3.4.

- Package Sensor

- Kelas AccelerationSensor
Kelas ini digunakan untuk melakukan pengukuran getaran pada satu waktu.
- Kelas HumiditySensor
Kelas ini digunakan untuk melakukan pengukuran kelembaban pada satu waktu.
- Kelas TemperatureSensor
Kelas ini digunakan untuk melakukan pengukuran suhu pada satu waktu.

- Package main

- Kelas Sensing
Kelas ini menyatukan ketiga buah kelas sensor dan digunakan pada kelas NodeSensor.
- Kelas BS
Kelas ini menangani fungsi-fungsi yang ada pada *base station* seperti mengirimkan perintah-perintah kepada node sensor yang terhubung dengan *base station*.
- Kelas NS
Kelas ini melakukan *sensing* dan mengirimkan data ke *base station* atau ke node sensor lain.
- Kelas BS_Testing
Kelas ini menangani fungsi-fungsi yang ada pada *base station*. Kelas ini digunakan untuk melakukan pengujian aplikasi yang tidak *reliable*.
- Kelas NS_Testing
Kelas ini melakukan *sensing* dan mengirimkan data ke *base station* atau ke node sensor lain. Kelas ini digunakan untuk melakukan pengujian aplikasi yang tidak *reliable*.

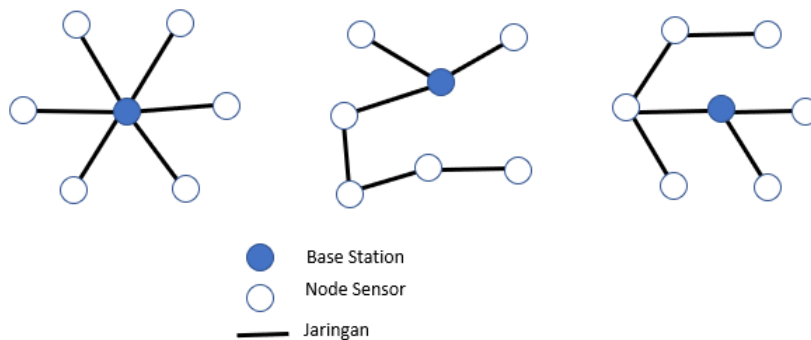
- Kelas handler
Kelas yang digunakan untuk menangani semua interaksi antara pengguna dengan Base Station.
- Package `com.virtenio.preon32.example.common`
 - Kelas `USARTConstants`
Kelas yang digunakan untuk membuat koneksi antara *base station* dengan program pada komputer pengguna.

3.1.1 Fitur dan Kebutuhan Sistem

Pada subbab ini dijelaskan fitur-fitur dan kebutuhan sistem dalam membangun aplikasi transfer data yang *reliable* di WSN.

Arsitektur Flat

Pada arsitektur flat tidak terdapat hierarki dan *cluster head*. Setiap node sensor yang telah melakukan *sensing* akan mengirimkan data dengan cara meneruskan data ke node sensor tetangganya. Node sensor tetangganya tersebut yang akan meneruskan data ke node sensor lain hingga sampai ke *base station*. Beberapa contoh arsitektur flat dapat dilihat pada Gambar 3.5.

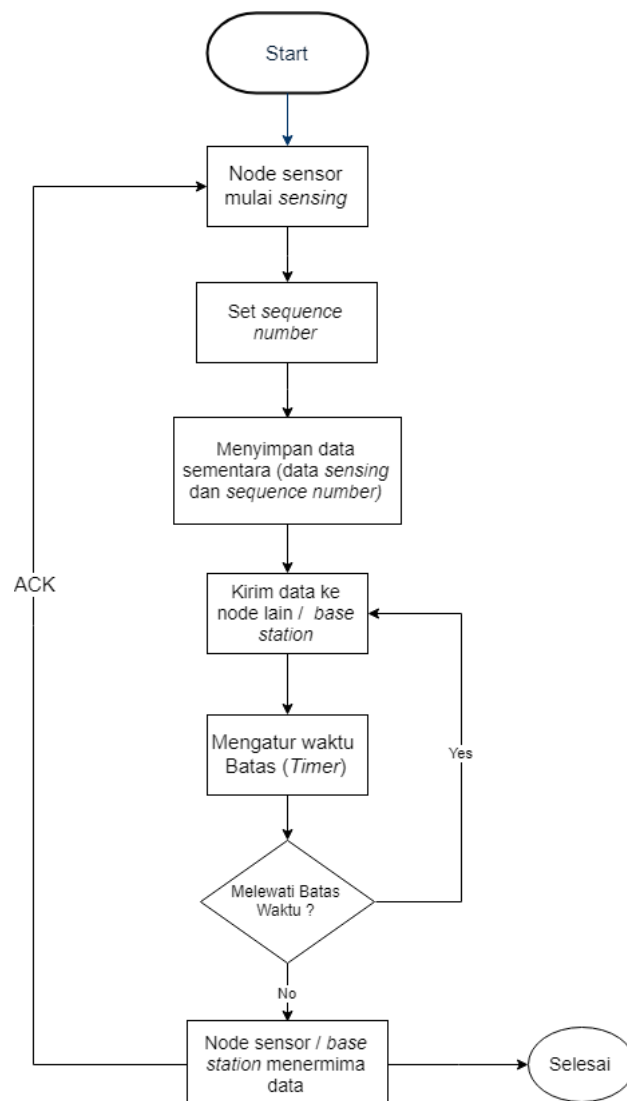


Gambar 3.5: Contoh arsitektur flat

Gambar 3.6 adalah *flowchart* dalam melakukan transfer data pada arsitektur flat dan cara menangani data yang hilang.

Berikut adalah penjelasan dari Gambar 3.6.

1. Node sensor melakukan *sensing*.
2. Disaat yang bersamaan node sensor menetapkan waktu *timer* untuk data yang akan dikirim.
3. Node sensor mendapatkan data-data hasil *sensing*.
4. Data yang didapat ini berisi *sequence number*, *timestamp* dan data utama hasil *sensing*.
5. Data disimpan sementara dan diteruskan ke node sensor tetangganya.
6. Node sensor(*receiver*) yang menerima data dari node sensor sebelumnya (*source*) akan meneruskan data tersebut ke node lain hingga sampai ke *base station*.
7. Jika data sudah sampai ke *base station*, *base station* akan mengirimkan ACK ke node yang terhubung dengan *base station*.
8. ACK akan diteruskan ke node lain jika tidak sesuai dengan pengirim data *sensing* tersebut.

Gambar 3.6: Flowchart transfer data dengan ACK dan *timer*

9. Saat node sensor pengirim data menerima ACK, node sensor tersebut akan melakukan *sensing* lagi.
10. Jika sudah melewati batas waktu *timer*, node sensor akan mengirimkan ulang data yang sudah tersimpan.

Routing Pada Aplikasi Transfer Data

Untuk melakukan monitoring area, node sensor mengirimkan data langsung kepada *base station* atau melalui node sensor lain hingga sampai *base station*. Dalam mengirimkan data diperlukan alamat tujuan data akan dikirimkan. Pada *single-hop* pertukaran data hanya antara node sensor dengan *base station*, sedangkan pada *multi-hop* data akan diteruskan kepada node sensor beberapa kali hingga sampai kepada alamat yang dituju.

Aplikasi yang dibangun akan menyimpan alamat *base station* itu sendiri dan alamat node lain. Pada *base station* menyimpan alamat dirinya dan juga alamat node-node yang terhubung langsung dengan *base station*. Sedangkan pada node sensor menyimpan alamat node sensor tersebut, alamat tujuan untuk mengirimkan data hasil *sensing* dan alamat node sensor lain untuk mengirimkan perintah-perintah. Sehingga untuk mengirimkan data hasil *sensing*, node sensor akan mengirimkan kepada 1 tujuan (*unicast*). Sedangkan untuk mengirimkan perintah, node sensor dapat mengirimkan kepada banyak node sensor lain secara *multicast*.

3.1.2 Ukuran Reliability

Pada aplikasi yang dibangun, setiap node sensor akan melakukan *sensing* secara terus menerus hingga pengguna menghentikan program. Node sensor akan berhenti jika menerima perintah berhenti dari *base station*. Data yang didapat ini yang akan dikirimkan ke *base station* dan digunakan untuk monitoring.

Setiap data yang dikirim memiliki *sequence number*. *Sequence number* ini yang akan menjadi perhitungan untuk memastikan *reliability*. Pengguna melihat keterurutan dari *sequence number* setiap data yang telah diterima oleh *base station*. Jika didapat *sequence number* dengan urut, maka *reliability* pada aplikasi ini berhasil.

3.2 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*.

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 kebutuhan dan kapasitas penyimpanan pada node sensor. Pada umumnya node sensor memiliki tempat penyimpanan yang kecil, sehingga kode program yang dibuat tidak dapat terlalu besar dan data hasil *sensing* harus dikelola saat akan melakukan pengiriman.

Pada skripsi ini, aplikasi yang dibangun adalah aplikasi transfer data yang *reliable*. Sehingga data yang dikirim harus dikelola dahulu. Sedangkan untuk membuktikan aplikasi *reliable* ini berhasil, dibuat juga aplikasi yang tidak *reliable* sebagai perbandingan data yang diterima. Untuk aplikasi yang tidak *reliable* ini data yang dikirimkan adalah data mentah hasil *sensing*.

3.2.1 Format Pesan Yang Dikirim Saat Melakukan Sensing

Setelah sensor mendapatkan data *sensing* dari lingkungannya, perlu ditentukan bagaimana format data yang akan dikirimkan ke node sensor lain. Data hasil *sensing* ini akan dikirim melalui *frame*. Setiap *frame* memiliki batasan panjang data yang bisa dikirimkan (PHY Payload). Untuk setiap data ada beberapa hal penting yang perlu ada untuk mendukung transfer data yang *reliable*, yaitu:

1. *Sequence number*.
2. *Timestamp*.
3. Data utama.

Sequence number digunakan untuk menentukan urutan pengiriman data dari node sensor ke node sensor lain. Data yang *loss* dapat dilihat dari *sequence number* ini. ***Timestamp*** digunakan untuk menentukan kapan waktu data diambil. **Data utama** adalah data hasil *sensing* yang diperoleh dari setiap sensor.

3.2.2 Format Pesan ACK Yang Dikirim

Untuk memastikan data yang diterima telah sampai ke *base station*, maka *base station* harus melakukan pengiriman ACK kepada node sensor yang melakukan *sensing*. ACK ini juga dimaksudkan untuk mengatur ulang waktu *timer* pada node sensor. Untuk setiap pesan ACK yang dikirimkan ada beberapa hal penting yang perlu ada untuk mendukung transfer data yang *reliable*, yaitu:

1. Kata "ACK".
2. Alamat node sensor sumber data.
3. *Sequence number* dari data yang diterima.

Kata "ACK" digunakan untuk menandakan bahwa pesan yang dikirim dari *base station* kepada node sensor adalah ACK. Kata "ACK" ini akan diikuti dengan ***alamat node sensor sumber data***. Dari *base station* pesan ini akan dikirimkan kepada node sensor yang terhubung kepada *base station*, saat node sensor menerima pesan ini dan alamat node sensor pada pesan tersebut bukan alamat node sensor penerima, maka pesan tersebut akan diteruskan kepada node sensor lain yang terhubung. Setelah alamat node sensor, pesan akan diikuti dengan ***sequence number***. *Sequence number* digunakan node sensor untuk mengetahui apakah pesan yang diterima *base station* sama dengan pesan yang node sensor kirim.

3.2.3 Format Pesan Untuk Mengetahui Node Yang Menyala

Untuk mengetahui node yang menyala, node sensor harus mengirimkan pesan kepada *base station*. Pesan yang dikirim terdiri dari:

1. Kata "Node".
2. Alamat node sensor.
3. Kata "Online"

Kata "Node" digunakan untuk menandakan bahwa pesan yang dikirim dari node sensor ini adalah status node sensor yang menyala. Kata "Node" ini diikuti dengan **Alamat node sensor** tersebut. Lalu pesan ini diikuti dengan kata "Online" yang menandakan bahwa node sensor sedang menyala.

3.2.4 Format Pesan Untuk Mengirimkan Waktu Node Sensor

Untuk mengetahui waktu dari setiap node sensor, setiap node sensor perlu mengirimkan pesan yang terdiri dari waktu kepada *base station*. Pesan yang dikirim terdiri dari:

1. Kata "Time".
2. Alamat node sensor.
3. Waktu node sensor.

Pesan yang dikirimkan diawali dengan **Kata "Time"** untuk menandakan pesan tersebut adalah waktu node sensor. Pesan ini akan diikuti oleh **alamat node sensor** untuk mengetahui waktu tersebut adalah waktu dari node sensor yang mana. Pesan ini diikuti juga oleh waktu node sensor.

3.3 Analisis 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, dan mekanisme *acknowledge* yang digunakan. Data yang diperoleh node sensor ini dapat diteruskan ke node sensor tetangganya dahulu sebelum sampai ke tujuan akhir yaitu *base station* (*multihop*).

Aplikasi transfer data yang dibangun ini mengadaptasi beberapa cara yang terdapat pada protokol RMST untuk menangani *reliability* transfer data dari node sensor ke *base station*. RMST dapat melakukan *retransmission* dengan mekanisme *hop-by-hop* dan *end-to-end* [12]. Aplikasi dibangun menggunakan mekanisme *end-to-end* dengan bantuan ACK dalam memastikan data sampai *base station*. Dengan mekanisme *end-to-end*, node sensor akan mengirimkan data ke node tetangga dan diteruskan ke node sensor lain hingga sampai di *base station*. Setelah *base station* mendapatkan data hasil *sensing*, *base station* akan mengirimkan ACK kepada node sensor di bawahnya dan diteruskan hingga sampai kepada node sensor pengirim data tersebut. Node sensor yang sedang mengirim data akan menunggu ACK yang didapat dari node di atasnya.

Protokol RMST ini juga menggunakan *timer-driven*. Mekanisme *timer-driven* ini adalah batas waktu menunggu ACK atau NACK untuk dilakukan pengiriman ulang. Pada aplikasi yang dibangun juga menggunakan mekanisme *timer-driver*. Saat node sensor telah mengirimkan data, node sensor tersebut akan mengatur (*timer*) atau batas waktu menunggu. Jika sudah melewati batas waktu, maka node sensor akan mengirimkan ulang data kepada node sensor tujuan. Saat melakukan pengiriman ulang data, dapat terjadi duplikasi data yang pada *base station*. Jika ini terjadi maka digunakan *sequence number* untuk memastikan urutan data. Saat data yang diterima memiliki *sequence number* yang sama dengan dengan sebelumnya, maka data tidak akan disimpan.

BAB 4

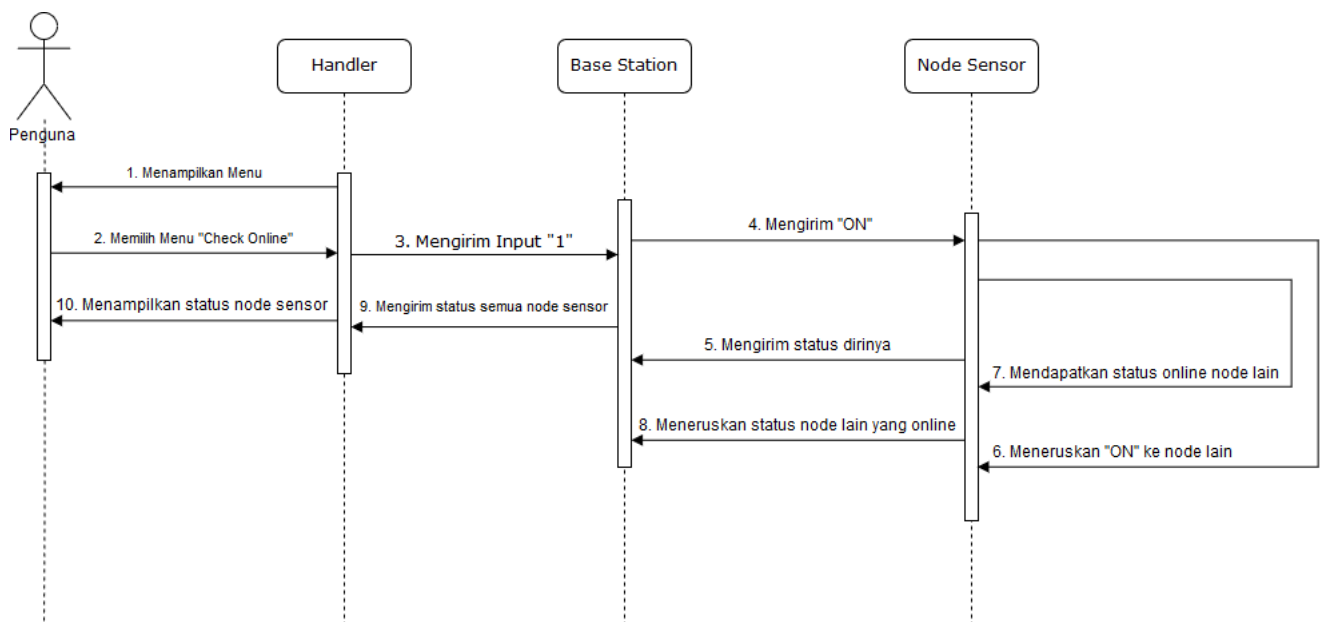
PERANCANGAN

Pada bab ini dijelaskan mengenai perancangan aplikasi yang dibangun meliputi perancangan interaksi antar node untuk transfer data di WSN, perancangan kelas aplikasi transfer data di WSN, perancangan *routing* pada aplikasi transfer data di WSN, perancangan format pesan, dan perancangan masukan dan keluaran.

4.1 Perancangan Interaksi Antar Node Untuk Transfer Data Di WSN

Pada subbab 3.1.1 telah dijelaskan mekanisme aplikasi transfer data pada arsitektur flat secara prosedural. Untuk lebih jelas mengenai aplikasi transfer data, dibuatlah diagram sequence untuk setiap fitur pada aplikasi transfer data ini.

4.1.1 Diagram Sequence "Check Online"

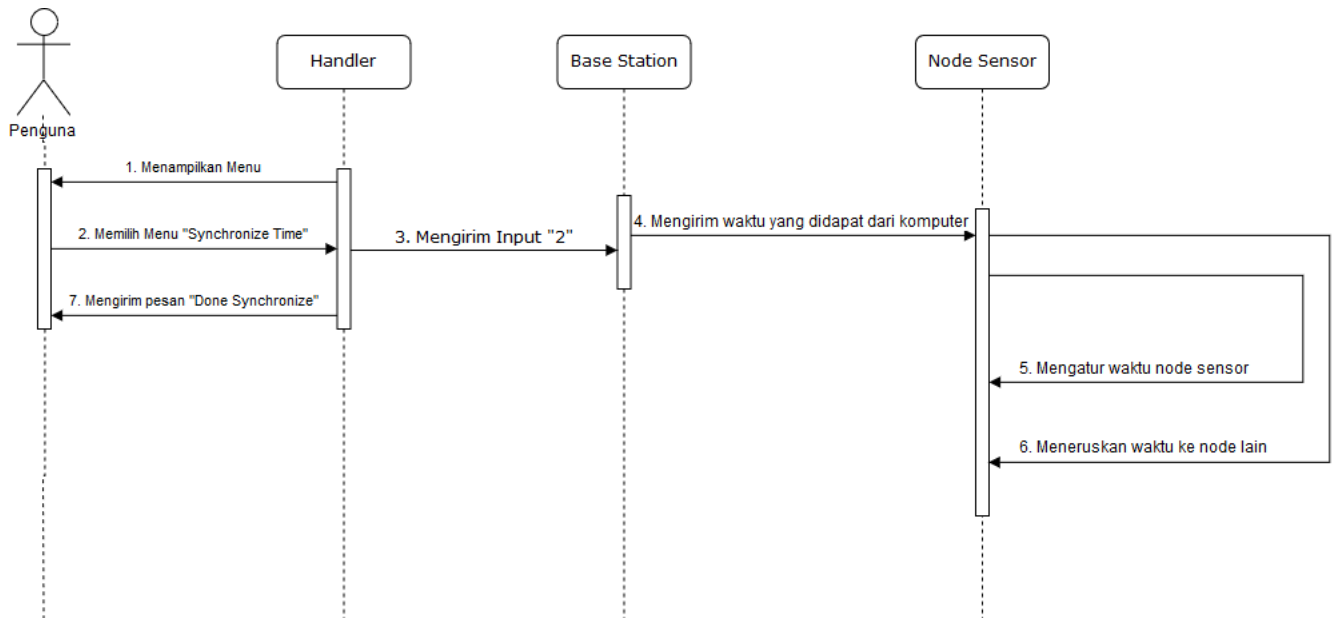


Gambar 4.1: Diagram Sequence "Check Online"

Fitur ini diawali dengan Handler sebagai program komputer pengguna menampilkan pilihan yang dapat dipilih oleh pengguna. Kemudian pengguna memilih "Check Online". "Check Online" ini digunakan untuk mengetahui node sensor yang sedang menyala (*online*). Handler mengirimkan input "1" kepada *base station*. Lalu *base station* mengirimkan "ON" kepada node sensor. Saat node sensor mendapatkan pesan "ON", node sensor akan mengirimkan status dirinya dan meneruskan

pesan "ON" kepada node sensor lain. Node sensor juga mendapatkan status dari node sensor lain dan diteruskan hingga sampai ke *base station*. Kemudian *base station* mendapatkan status dari semua node sensor dan meneruskan Handler. Handler yang akan menampilkan status node sensor kepada pengguna.

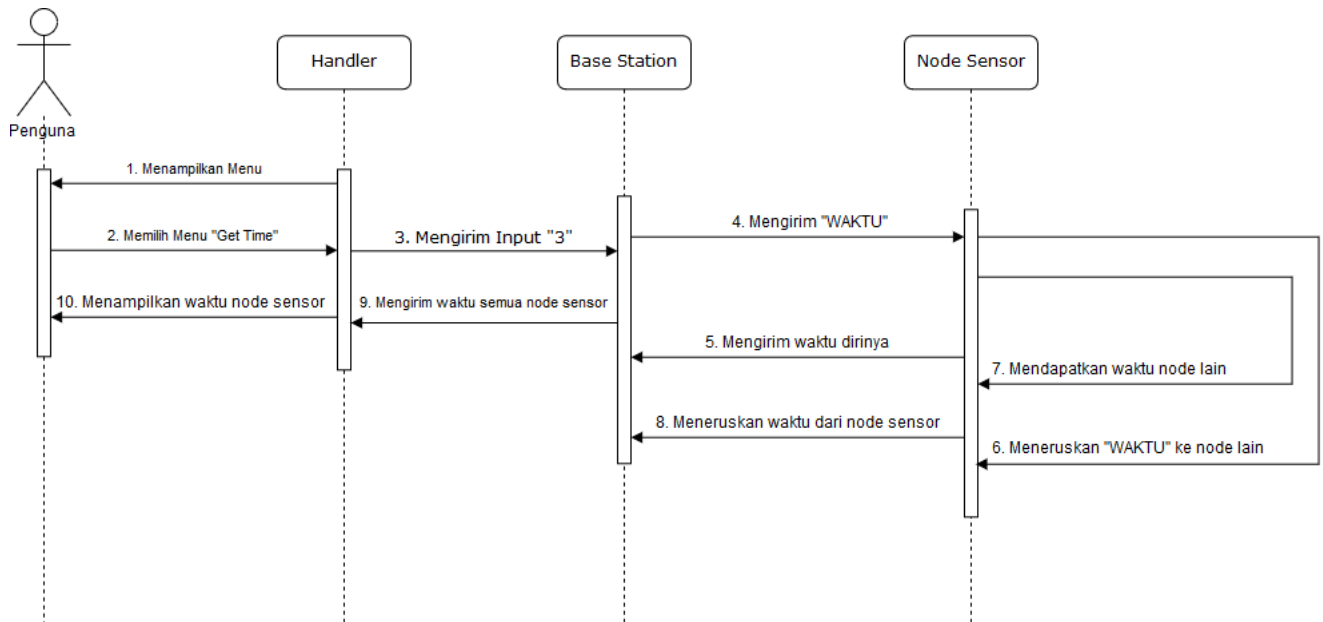
4.1.2 Diagram Sequence "Synchronize Time"



Gambar 4.2: Diagram Sequence "Synchronize Time"

Fitur ini diawali dengan Handler menampilkan pilihan yang dapat dipilih oleh pengguna. Kemudian pengguna memilih "Synchronize Time". "Synchronize Time" ini digunakan untuk melakukan sinkronisasi waktu semua node sensor sesuai dengan waktu lokal pada komputer pengguna. Saat pertama menjalankan program, *base station* mengatur waktu sesuai waktu komputer pengguna. Handler mengirimkan input "2" kepada *base station*. Kemudian *base station* meneruskan waktu kepada node sensor dan node sensor ini meneruskan waktu kepada node sensor lain. Setelah pengguna memilih "Synchronize Time", Handler akan mengirimkan pesan "Done Synchronize".

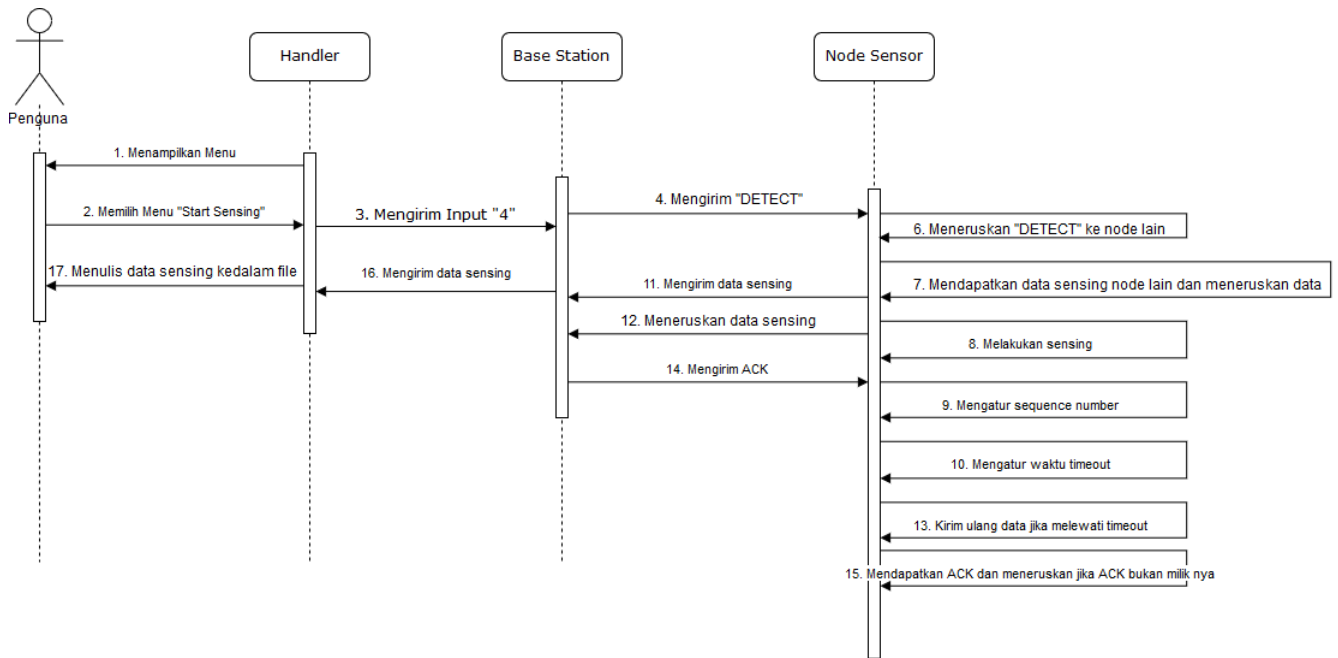
4.1.3 Diagram Sequence "Get Time"



Gambar 4.3: Diagram Sequence "Get Time"

Fitur ini diawali dengan Handler menampilkan pilihan yang dapat dipilih oleh pengguna. Kemudian pengguna memilih "Get Time". "Get Time" ini digunakan untuk mendapatkan waktu dari setiap node sensor. Handler mengirimkan input "3" kepada *base station*. Kemudian *base station* mengirimkan pesan "WAKTU" kepada node sensor dan node sensor meneruskan pesan "WAKTU" kepada node sensor tetangganya. Setelah itu node sensor akan mengirimkan waktu kepada *base station* atau node sensor lain. *Base station* mengirimkan waktu node sensor kepada Handler dan Handler yang akan menampilkan kepada pengguna.

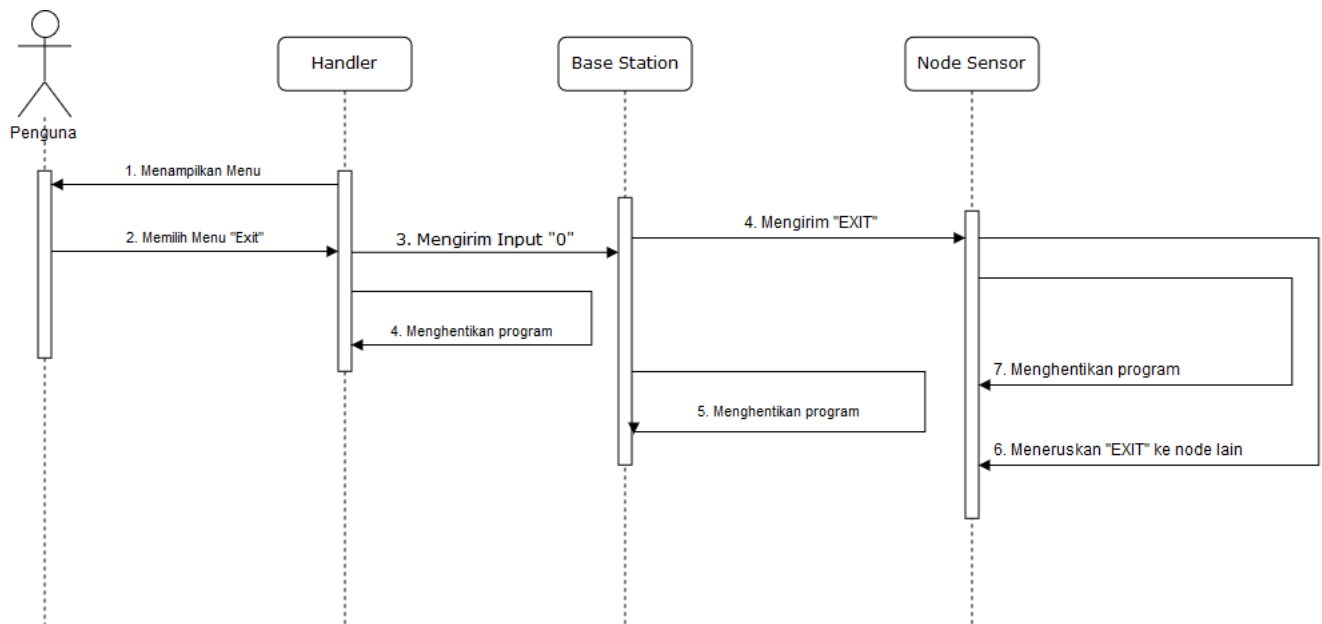
4.1.4 Diagram Sequence "Start Sensing"



Gambar 4.4: Diagram Sequence "Start Sensing"

Fitur ini digunakan untuk mendapatkan data sensing secara *reliable*. Setelah Handler menampilkan pilihan untuk dipilih pengguna dan pengguna memilih untuk memulai *sensing*, Handler meneruskan pilihan tersebut kepada *base station*. *Base station* mengirimkan pesan "DETECT" kepada node sensor. Node sensor melakukan *sensing*, mengatur *sequence number*, dan mengatur *timeout*. Kemudian node sensor meneruskan pesan "DETECT" kepada node sensor lain. Setelah melakukan *sensing*, node sensor akan mengirimkan data kepada node sensor lain atau langsung kepada *base station*. Jika data hasil *sensing* dikirim kepada node sensor lain, maka data tersebut akan diteruskan hingga sampai ke *base station*. Data yang sudah sampai *base station* akan diteruskan kepada Handler untuk ditulis ke dalam *file* agar dapat dibaca oleh pengguna.

4.1.5 Diagram Sequence "Exit"



Gambar 4.5: Diagram Sequence "Exit"

Fitur ini digunakan untuk menghentikan semua proses yang sedang berlangsung pada Handler, *Base Station*, dan Node Sensor. Fitur ini diawali dengan Handler menampilkan pilihan yang dapat dipilih oleh pengguna. Kemudian pengguna memilih "Exit". Handler mengirimkan input "0" yang berarti *exit* kepada *base station* dan menghentikan program yang sedang berjalan. *Base station* mengirimkan pesan "EXIT" kepada node sensor lain dan menghentikan program pada *base station*. Node sensor yang menerima pesan "EXIT" juga menghentikan program yang sedang berjalan.

4.2 Perancangan Kelas Aplikasi Transfer Data Di WSN

Pada subbab 3.1.1 telah dijelaskan kelas diagram aplikasi transfer data secara sederhana. Pada subbab ini dijelaskan kelas diagram secara detail dengan Gambar 4.6 sampai Gambar 4.12. Deskripsi setiap kelas dijelaskan sebagai berikut:

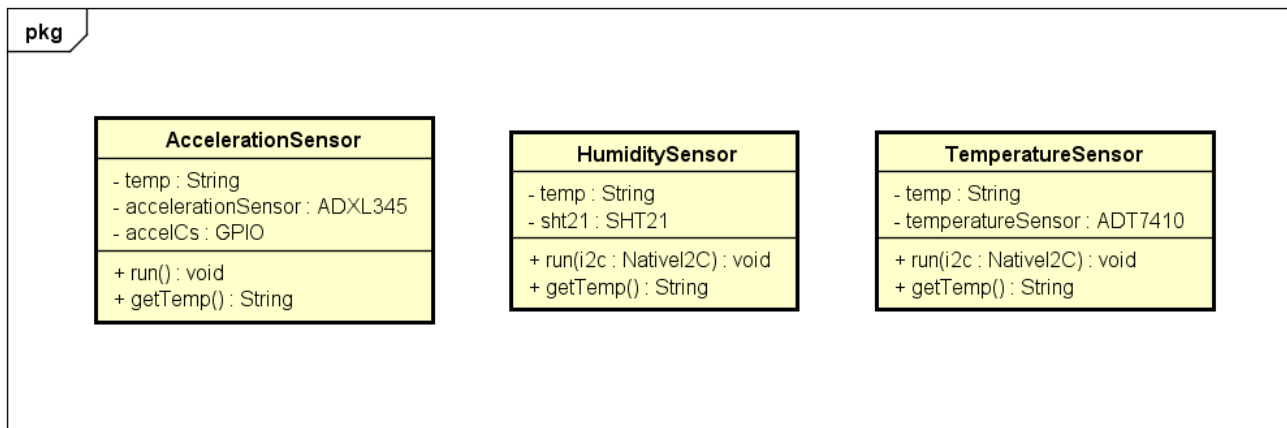
- **Package Sensor**

- **Kelas AccelerationSensor**

Kelas ini digunakan untuk melakukan pengukuran tekanan udara pada satu waktu. Atribut-atribut pada kelas ini adalah sebagai berikut:

- * private ADXL345 accelerationSensor
Atribut ini digunakan sebagai *import* dari *driver sensor* untuk mengukur getaran yang ada pada node sensor Preon32.
- * private String temp
Atribut ini digunakan untuk menyimpan sementara data hasil pengukuran tekanan udara pada satu waktu.
- * private GPIO accelSc
Atribut ini digunakan sebagai *import* dari *driver GPIO* (general purpose IO).

Metode-metode pada kelas ini adalah sebagai berikut:



Gambar 4.6: Diagram Kelas Sensor-Sensor

- * `public void run()`
Metode ini digunakan untuk melakukan *sensing* getaran dan datanya akan disimpan pada atribut `temp`.
- * `public String getTemp()`
Metode ini digunakan untuk mengembalikan atribut `temp` yang nanti digunakan pada kelas-kelas lain.

– Kelas **HumiditySensor**

Kelas ini digunakan untuk melakukan pengukuran kelembaban pada satu waktu. Atribut-atribut pada kelas ini adalah sebagai berikut:

- * `private SHT21 sht21`
Atribut ini digunakan sebagai *import* dari *driver sensor* untuk mengukur kelembaban yang ada pada node sensor Preon32.
- * `private String temp`
Atribut ini digunakan untuk menyimpan sementara data hasil pengukuran kelembaban pada satu waktu.

Metode-metode pada kelas ini adalah sebagai berikut:

- * `public void run(NativeI2C i2c)`
Metode ini digunakan untuk melakukan *sensing* kelembaban dan datanya akan disimpan pada atribut `temp`.
- * `public String getTemp()`
Metode ini digunakan untuk mengembalikan atribut `temp` yang nanti digunakan pada kelas-kelas lain.

– Kelas **TemperatureSensor**

Kelas ini digunakan untuk melakukan pengukuran suhu pada satu waktu. Atribut-atribut pada kelas ini adalah sebagai berikut:

- * `private ADT7410 temperatureSensor`
Atribut ini digunakan sebagai *import* dari *driver sensor* untuk mengukur suhu yang ada pada node sensor Preon32.
- * `private String temp`
Atribut ini digunakan untuk menyimpan sementara data hasil pengukuran suhu pada satu waktu.

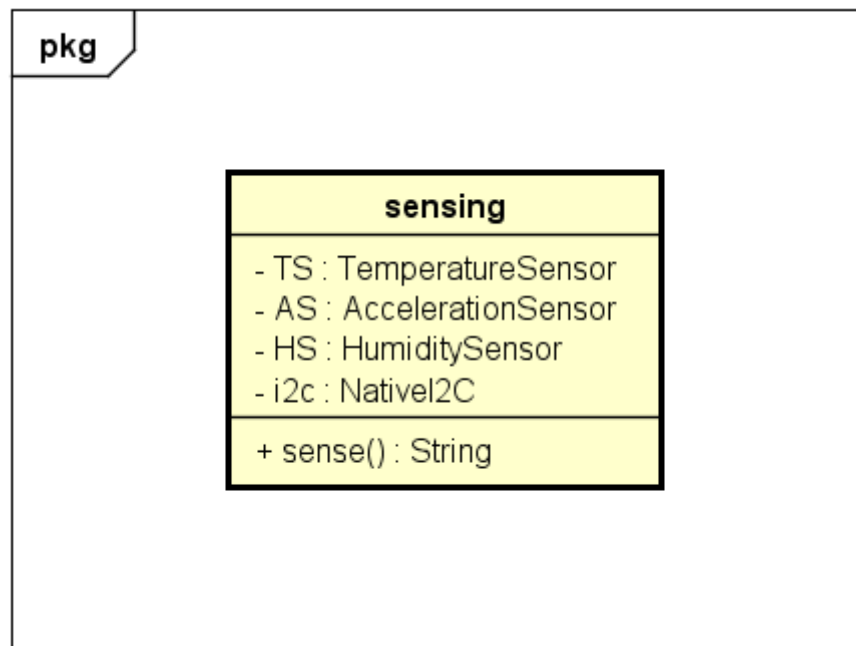
Metode-metode pada kelas ini adalah sebagai berikut:

- * `public void run(NativeI2C i2c)`
Metode ini digunakan untuk melakukan *sensing* suhu dan datanya akan disimpan pada atribut `temp`.
- * `public String getTemp()`
Metode ini digunakan untuk mengembalikan atribut `temp` yang nanti digunakan pada kelas-kelas lain.

- **Package main**

- **Kelas Sensing**

Kelas ini menyatukan tiga buah kelas sensor dan digunakan pada kelas NS. Atribut-



Gambar 4.7: Diagram Kelas Sensing

atribut pada kelas ini adalah sebagai berikut:

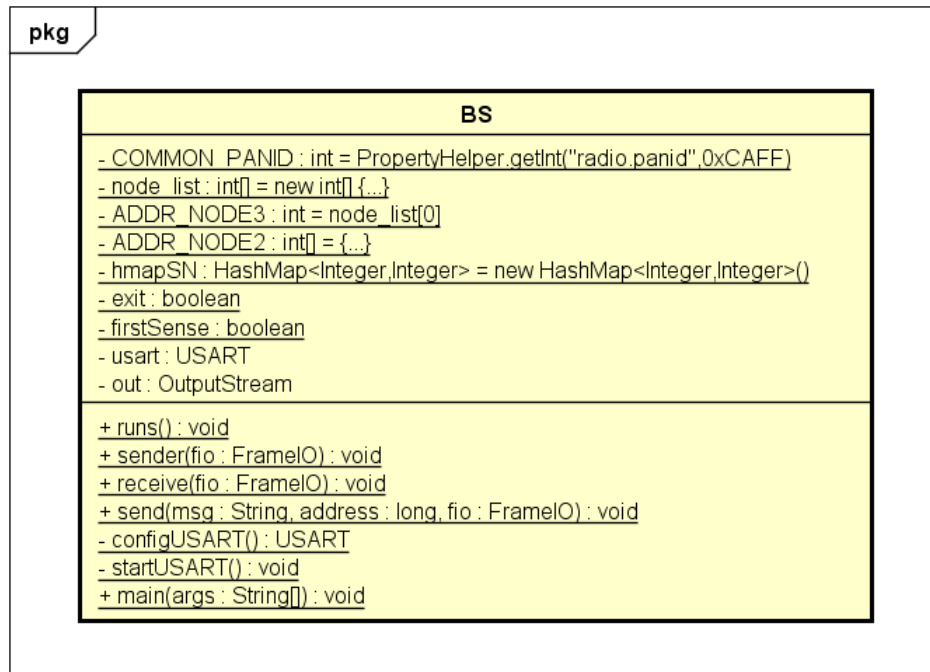
- * `private TemperatureSensor TS = new TemperatureSensor()`
Atribut untuk membuat objek baru dari kelas `TemperatureSensor`.
- * `private AccelerationSensor AS = new AccelerationSensor()`
Atribut untuk membuat objek baru dari kelas `AccelerationSensor`.
- * `private HumiditySensor HS = new HumiditySensor()`
Atribut untuk membuat objek baru dari kelas `HumiditySensor`.
- * `private NativeI2C i2c = NativeI2C.getInstance(1)`
Atribut untuk melakukan inisialisasi *driver NativeI2C* untuk digunakan oleh sensor-sensor.

Metode-metode pada kelas ini adalah sebagai berikut:

- * `public String sense()`
Metode ini digunakan untuk melakukan *sensing*. Metode ini mengembalikan *string* yang berisi nilai *sensing* dari setiap sensor yang digunakan.

- **Kelas BS**

Kelas ini menangani fungsi-fungsi yang ada pada *base station* seperti mengirimkan perintah-perintah kepada node sensor dibawahnya dan menerima data dari node sensor.



Gambar 4.8: Diagram Kelas BS

Kelas ini yang akan diunggah kedalam node sensor *base station*. Atribut-atribut pada kelas ini adalah sebagai berikut:

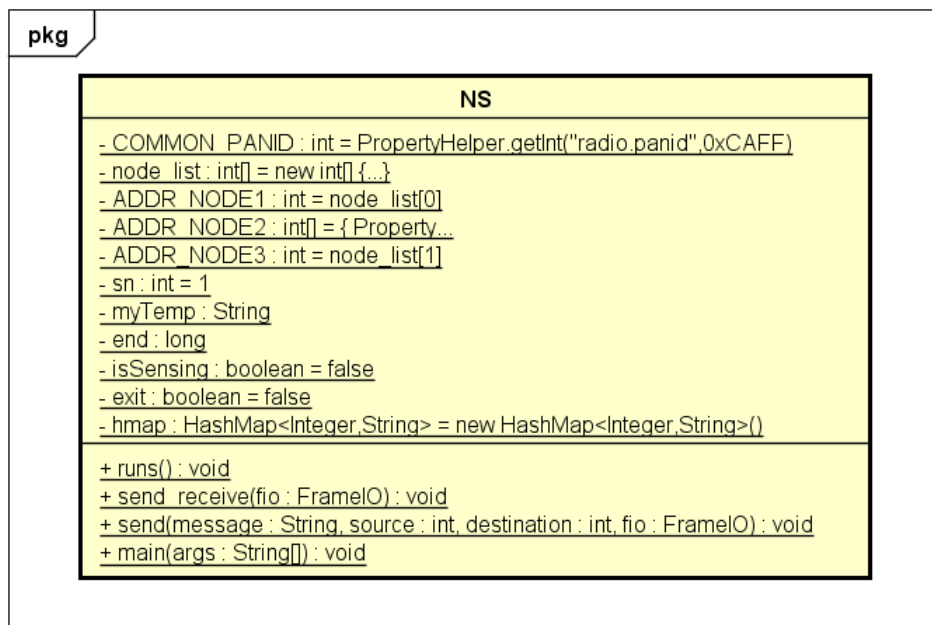
- * `private static int COMMON_PANID`
Atribut ini digunakan untuk menyimpan PAN ID dari satu jaringan node sensor.
- * `private static int [] node_list`
Atribut ini digunakan untuk menyimpan daftar alamat node sensor pada satu jaringan.
- * `private static int ADDR_NODE3`
Atribut ini digunakan untuk menyimpan alamat dari *base station*.
- * `private static int[] ADDR_NODE2`
Atribut ini menyimpan alamat node sensor yang terhubung langsung *base station*.
- * `private static HashMap<Long, Integer> hmap`
Atribut ini digunakan untuk menyimpan sementara data dari setiap node sensor yang ada pada `node_list` untuk memastikan *reliability*.
- * `private static boolean exit`
Atribut ini digunakan menyimpan kondisi exit saat mendapatkan input "Exit" dari user.
- * `private static boolean firstSense`
Atribut ini digunakan untuk menyimpan kondisi sudah melakukan *sensing* atau belum.
- * `private static USART usart`
Atribut ini digunakan untuk inialisasi USART yang menghubungkan *base station* dengan komputer yang terhubung langsung.
- * `private static OutputStream out`
Atribut ini digunakan untuk inialisasi OutputStream yang menulis data dari *base station* ke program komputer agar bisa dilihat oleh pengguna.

Metode-metode pada kelas ini adalah sebagai berikut:

- * `public void runs()`
Metode ini digunakan untuk memanggil metode `sender()`.
- * `public static void sender(Final FrameIO fio)`
Metode ini digunakan untuk melakukan mengirim perintah-perintah ke node sensor.
- * `public static void receiver(Final FrameIO fio)`
Metode ini digunakan untuk menerima data dari node sensor.
- * `public void send(String msg, long address, FrameIO fio)`
Metode ini digunakan untuk mengirimkan pesan (*msg*) dari *base station* ke alamat tujuan (*address*) dengan menggunakan `FrameIO` (*fio*).
- * `public static void main(String[] args)`
Metode ini digunakan sebagai metode utama dari kelas ini dan memanggil metode `runs()`.

– Kelas NS

Kelas ini diunggah ke dalam node sensor untuk melakukan *sensing* dan mengirimkan



Gambar 4.9: Diagram Kelas NS

data ke *base station* atau ke node sensor lain. Atribut-atribut pada kelas ini adalah sebagai berikut:

- * `private static int COMMON_PANID`
Atribut ini digunakan untuk menyimpan PAN ID dari satu jaringan node sensor.
- * `private static int [] node_list`
Atribut ini digunakan untuk menyimpan daftar alamat node sensor pada satu jaringan.
- * `private int ADDR_NODE1`
Atribut ini digunakan untuk menyimpan alamat dari node sensor diatas node sensor ini.
- * `private static int[] ADDR_NODE2`
Atribut ini menyimpan array alamat yang terhubung langsung node sensor ini kecuali node *base station*.
- * `private static int ADDR_NODE3`
Atribut ini digunakan untuk menyimpan alamat dari node sensor ini.

- * `private sensing s`
Atribut ini digunakan untuk membuat objek dari kelas `Sensing`.
- * `private int sn = 1`
Atribut ini digunakan untuk menyimpan *counter* dari *sequence number* pada satu kali pengiriman *frame*.
- * `private static String myTemp`
Atribut ini digunakan untuk menyimpan sementara data hasil sensing dari node sensor ini.
- * `private static long end`
Atribut ini digunakan untuk menyimpan batas waktu node sensor menunggu ACK / NACK (*timeout*).
- * `private static boolean isSensing`
Atribut ini digunakan sebagai penanda bahwa sudah pernah melakukan sensing agar tidak melakukan sensing lagi sebelum menerima ACK.
- * `private static boolean exit`
Atribut ini digunakan sebagai penanda program telah dihentikan dari *base station* dan harus dihentikan juga pada node sensor ini.
- * `private static HashMap<Integer, String> hmap`
Atribut ini digunakan untuk menyimpan sementara data sensing dari node tetangga pada atribut `ADDR_NODE2`. untuk memastikan *reliability* sampai ke *base station*.

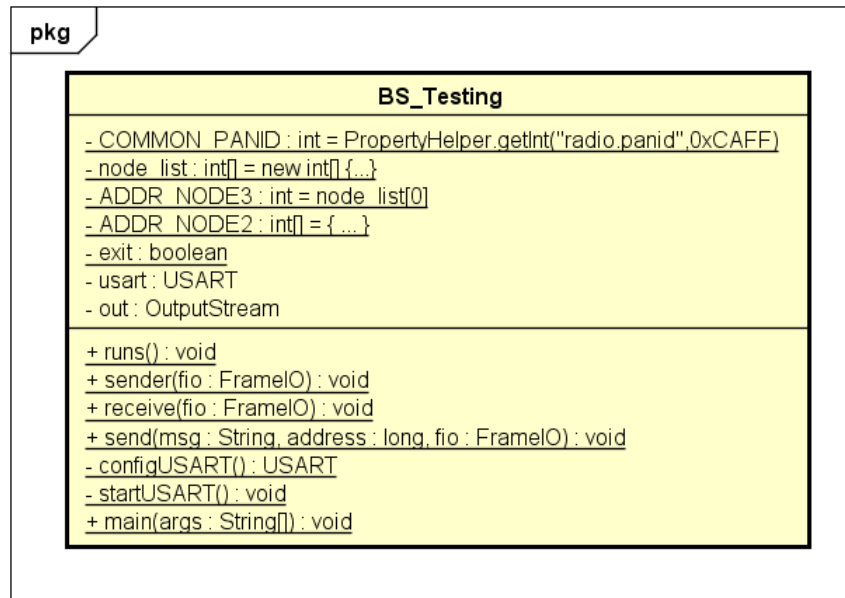
Metode-metode pada kelas ini adalah sebagai berikut:

- * `public void runs()`
Metode ini berisi inisialisasi dari radio, transmitter dan memanggil method `send_receive()`.
- * `public void send_receive(Final FrameIO fio)`
Metode ini digunakan node sensor untuk menangani pengiriman dan penerimaan data.
- * `public void send(String message, int source, int destination, FrameIO fio)`
Metode ini digunakan untuk mengirim pesan (*message*) dari dirinya (*source*) kepada tujuan (*destination* dengan `FrameIO (fio)`.
- * `public static void main(String[] args)`
Metode ini digunakan sebagai metode utama dari kelas ini dan memanggil metode `runs()`.

– Kelas **BS_Testing**

Kelas ini menangani fungsi-fungsi yang ada pada *base station* seperti mengirimkan perintah-perintah kepada node sensor dibawahnya dan menerima data dari node sensor. Kelas ini yang akan diunggah kedalam node sensor *base station*. Kelas ini digunakan sebagai *base station* yang tidak menangani *reliability*. Atribut-atribut pada kelas ini adalah sebagai berikut:

- * `private static int COMMON_PANID`
Atribut ini digunakan untuk menyimpan PAN ID dari satu jaringan node sensor.
- * `private static int [] node_list`
Atribut ini digunakan untuk menyimpan daftar alamat node sensor pada satu jaringan.
- * `private static int ADDR_NODE3`
Atribut ini digunakan untuk menyimpan alamat dari *base station*.
- * `private static int[] ADDR_NODE2`
Atribut ini menyimpan alamat node sensor yang terhubung langsung *base station*.
- * `private static boolean exit`
Atribut ini digunakan menyimpan kondisi exit saat mendapatkan input "Exit" dari user.



Gambar 4.10: Diagram Kelas BS_Testing

- * private static USART usart

Atribut ini digunakan untuk inisialisasi USART yang menghubungkan *base station* dengan komputer yang terhubung langsung.

- * private static OutputStream out

Atribut ini digunakan untuk inisialisasi OutputStream yang menulis data dari node sensor ke program komputer agar bisa dilihat oleh pengguna.

Metode-metode pada kelas ini adalah sebagai berikut:

- * public void runs()

Metode ini digunakan untuk memanggil metode `sender()` dan `receiver()`.

- * public static void sender(Final FrameIO fio)

Metode ini digunakan untuk melakukan mengirim perintah-perintah ke node sensor.

- * public static void receiver(Final FrameIO fio)

Metode ini digunakan untuk menerima data dari node sensor.

- * public void send(String msg, long address, FrameIO fio)

Metode ini digunakan untuk mengirimkan pesan (*msg*) dari *base station* ke alamat tujuan (*address*) dengan menggunakan FrameIO (*fio*).

- * public static void main(String[] args)

Metode ini digunakan sebagai metode utama dari kelas ini dan memanggil metode `runs()`.

– Kelas NS_Testing

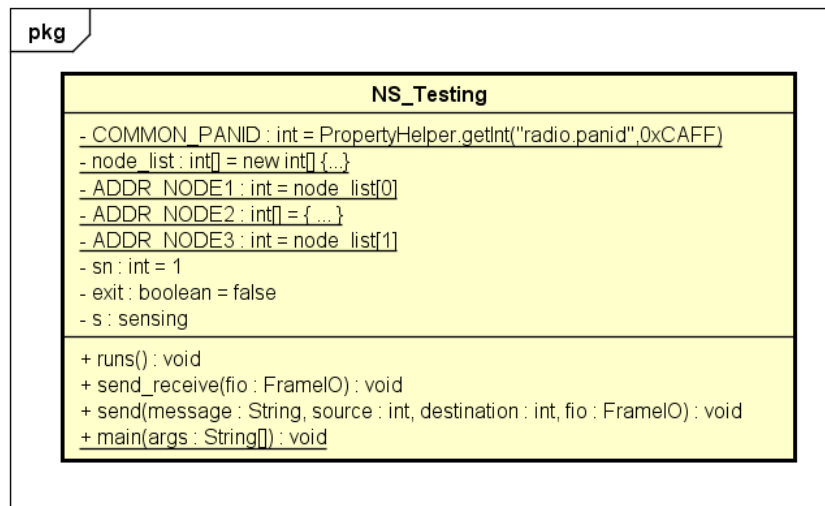
Kelas ini diunggah ke dalam node sensor untuk melakukan *sensing* dan mengirimkan data ke *base station* atau ke node sensor lain. Kelas ini digunakan sebagai node sensor yang tidak menangani *reliability*. Atribut-atribut pada kelas ini adalah sebagai berikut:

- * private static int COMMON_PANID

Atribut ini digunakan untuk menyimpan PAN ID dari satu jaringan node sensor.

- * private static int [] node_list

Atribut ini digunakan untuk menyimpan daftar alamat node sensor pada satu jaringan.



Gambar 4.11: Diagram Kelas NS_Testing

- * private int ADDR_NODE1
Atribut ini digunakan untuk menyimpan alamat dari node sensor diatas node sensor ini.
- * private static int[] ADDR_NODE2
Atribut ini menyimpan array alamat yang terhubung langsung node sensor ini kecuali node *base station*.
- * private static int ADDR_NODE3
Atribut ini digunakan untuk menyimpan alamat dari node sensor ini.
- * private sensing s
Atribut ini digunakan untuk membuat objek dari kelas Sensing.
- * private int sn = 1
Atribut ini digunakan untuk menyimpan *counter* dari *sequence number* pada satu kali pengiriman *frame*.
- * private boolean exit
Atribut ini digunakan sebagai penanda program telah dihentikan dari *base station* dah harus dihentikan juga pada node sensor ini.

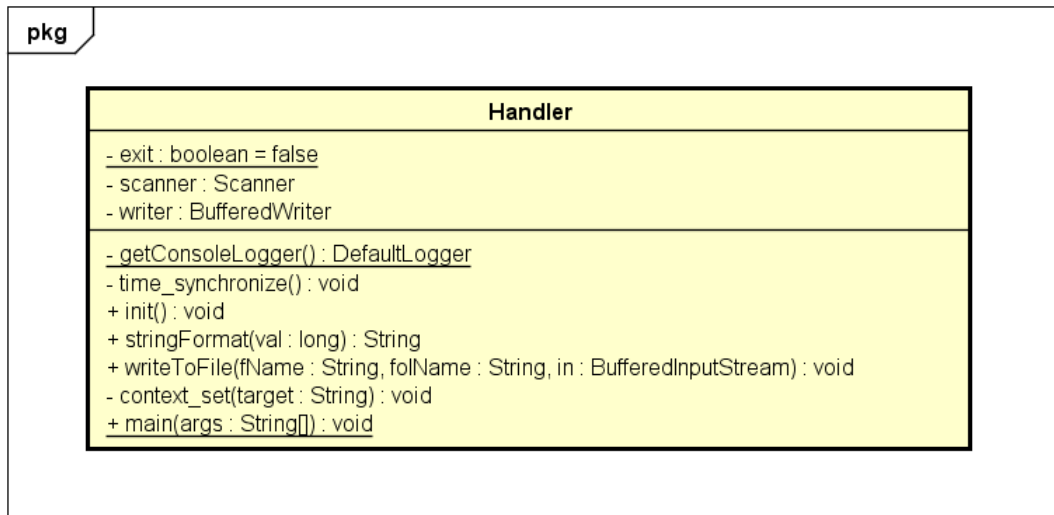
Metode-metode pada kelas ini adalah sebagai berikut:

- * public void runs()
Metode ini berisi inialisai dari radio, transmitter dan memanggil method send_receive().
- * public void send_receive(Final FrameIO fio)
Metode ini digunakan node sensor untuk menangani pengiriman dan penerimaan data.
- * public void send(String message, int source, int destination, FrameIO fio)
Metode ini digunakan untuk mengirim pesan (*message*) dari dirinya (*source*) kepada tujuan (*destination* dengan FrameIO (*fio*)).
- * public static void main(String[] args)
Metode ini digunakan sebagai metode utama dari kelas ini dan memanggil metode runs().

• Package Handler

– Kelas Handler

Kelas ini menangani hubungan antara *base station* dengan komputer pengguna. Saat



Gambar 4.12: Diagram Kelas Handler

menerima data, *base station* akan mengirimkan data yang diterima kepada kelas Handler. Kelas Handler ini akan mengelola data yang diterima dan ditampilkan atau ditulis kedalam file. Atribut-atribut pada kelas ini adalah sebagai berikut:

- * `private volatile static boolean exit = false`
Atribut ini digunakan untuk *variable* exit. Saat pengguna memasukan angka 0, maka exit akan menjadi *true*, dan program akan berhenti.
- * `private Scanner scanner`
Atribut ini digunakan untuk inisialisasi Scanner sebagai masukan pengguna pada CLI.
- * `private BufferedWriter writer`
Atribut ini digunakan untuk membuat objek BufferedWriter yang digunakan untuk menulis data *sensing* kedalam file.

Metode-metode pada kelas ini adalah sebagai berikut:

- * `private static DefaultLogger getConsoleLogger()`
Metode ini digunakan untuk memunculkan tulisan pada *console*.
- * `private void time_synchronize()`
Metode ini digunakan untuk melakukan sinkronisasi waktu *base station* sesuai dengan waktu pada komputer saat itu.
- * `public void init()`
Metode ini digunakan untuk membangun koneksi antara program pada komputer dengan program pada *base station*. Pada metode ini juga terdapat pilihan yang dapat dipilih pengguna dan mengirimkan masukan tersebut kepada *base station*.
- * `public String stringFormat(long val)`
Metode ini digunakan untuk mengubah format dari waktu yang didapat dari node sensor sehingga dapat dibaca oleh pengguna.
- * `public void writeToFile(String fName, String folName, BufferedInputStream in)`
Metode ini digunakan untuk menulis data sensing kedalam file.
- * `private void context_set(String target)`
Metode ini digunakan untuk memilih *context* yang akan digunakan.
- * `public static void main(String[] args)`
Metode ini digunakan sebagai metode utama untuk menjalankan kelas ini. Metode ini memanggil metode `context_set()`, `time_synchronize()`, dan `init()`.

4.3 Perancangan Masukan dan Keluaran

Aplikasi transfer data yang dibangun menggunakan *command line interface* sebagai media penerima masukan dan mengeluarkan keluaran. Aplikasi ini dapat menerima masukan dengan tipe *integer*. Akan tersedia 5 buah pilihan, pengguna dapat memasukkan nilai 1-4 dan 0 sebagai masukan. Masukan yang diberikan oleh pengguna antara lain adalah :

- **Masukan dengan nilai 1** digunakan untuk mengetahui node mana saja yang sedang menyala. Keluaran dari masukan ini adalah nama node diikuti dengan status *online*.
- **Masukan dengan nilai 2** digunakan untuk melakukan sinkronisasi waktu setiap node sensor sesuai dengan node waktu pada *base station*. Keluaran dari masukan ini adalah pesan berhasil melakukan sinkronisasi waktu.
- **Masukan dengan nilai 3** digunakan untuk mendapatkan waktu dari setiap node sensor. Keluaran dari masukan ini adalah nama node sensor diikuti dengan waktu pada node sensor tersebut.
- **Masukan dengan nilai 4** digunakan untuk memulai *sensing* dari setiap node sensor. Keluaran dari masukan ini adalah pesan mulai melakukan *sensing* dan setiap data hasil sensing tersebut ditulis kedalam file text.
- **Masukan dengan nilai 0** digunakan untuk menghentikan dan keluar dari aplikasi. Saat pengguna memasukkan nilai 0, sistem memperlihatkan pesan keluar dari aplikasi dan berhenti menerima masukan dari pengguna.
- Apabila pengguna memasukkan nilai **masukan yang tidak valid**, sistem akan memberikan keluaran pesan bahwa pesan tidak *valid* serta menampilkan kembali pilihan.

4.4 Perancangan Pseudocode Aplikasi Transfer Data Yang Reliable

Pada subbab ini dirancang *pseudocode* untuk membuat aplikasi transfer data yang *reliable*. *Pseudocode* yang dibuat digunakan pada program *base station* dan *node sensor* terkait dengan pengiriman pesan atau data hasil *sensing*.

4.4.1 Base Station

Untuk melakukan pengiriman dan penerimaan pesan, *base station* menggunakan 2 metode yang terpisah. Metode sender digunakan untuk mengirim perintah kepada node sensor. Sedangkan metode receiver digunakan untuk menerima data dari node sensor.

4.4.2 Node Sensor

Berbeda dengan *base station*, node sensor hanya menggunakan 1 metode untuk menangani pengiriman dan penerimaan pesan yaitu metode `send_receive`.

Algorithm 1 Metode sender

```

1: function SENDER
2:   thread  $\leftarrow$  thread baru do
3:     function RUN
4:       while true do
5:         temp  $\leftarrow$  membuat variable lokal
6:         try
7:           temp  $\leftarrow$  usart.read() // membaca masukan dari pengguna
8:         catch USARTException
9:         end try
10:        if temp = 0 then
11:          mengirim pesan "EXIT" kepada semua node ADDR_NODE2
12:        else if temp = 1 then
13:          mengirim pesan "ON" kepada semua node ADDR_NODE2
14:        else if temp = 2 then
15:          mengirim pesan ("Q" + currTime) kepada semua node ADDR_NODE2
16:        else if temp = 3 then
17:          mengirim pesan "WAKTU" kepada semua node ADDR_NODE2
18:        else if temp = 4 then
19:          mengirim pesan "DETECT" kepada semua node ADDR_NODE2
20:        end if
21:      end while
22:    end function
23:  end thread
24:  thread.start()
25: end function

```

Algorithm 2 Metode receive

```

1: function RECEIVE
2:   thread receive  $\leftarrow$  thread baru do
3:     function RUN
4:       frame  $\leftarrow$  membuat objek frame baru
5:       while true do
6:         try
7:           fio.receive(frame) // menerima frame
8:           dg  $\leftarrow$  mendapatkan isi dari payload sebuah frame
9:           str  $\leftarrow$  mengubah dg menjadi string
10:          if str.charAt(str.length()-1) = 'E' then
11:            merupakan pesan status menyala dari node sensor
12:            try
13:              menulis pesan agar dapat dibaca pengguna
14:              out.write(msg.getBytes(), 0, msg.length())
15:              usart.flush()
16:            catch Exception
17:          end try
18:          else if str.charAt(0) = 'T' then
19:            merupakan pesan waktu node sensor
20:            try
21:              menulis pesan agar dapat dibaca pengguna
22:              out.write(msg.getBytes(), 0, msg.length())
23:              usart.flush()
24:            catch Exception
25:          end try
26:          else if str.startsWith("SENSE") then
27:            merupakan pesan data hasil sensing dari node sensor
28:            node  $\leftarrow$  alamat node sensor yang mengirim data
29:            sn  $\leftarrow$  sequence number dari data yang diterima
30:            if hmapSN.get(node)=sn then
31:              apakah sequence number data yang diterima sesuai urutan
32:              try
33:                menulis pesan agar dapat dibaca pengguna
34:                out.write(msg.getBytes(), 0, msg.length())
35:                usart.flush()
36:              catch Exception
37:            end try
38:            hmapSN.put(node, sn+1)
39:          end if
40:          base station akan mengirimkan ACK setiap kali menerima data hasil sensing
41:        end if
42:      catch Exception
43:    end try
44:  end while
45: end function
46: end thread
47: receive.start()
48: end function

```

Algorithm 3 Metode send_receive

```

1: function SEND_RECEIVE
2:   thread thread  $\leftarrow$  thread baru do
3:     function RUN
4:       frame  $\leftarrow$  membuat objek frame baru
5:       while true do
6:         try
7:           fio.receive(frame) // menerima frame
8:           dg  $\leftarrow$  mendapatkan payload dari frame
9:           str  $\leftarrow$  mengubah dg menjadi string
10:          if str.charAt(0) == 'Q' then
11:            merupakan pesan dari base station untuk sinkronisasi waktu
12:            Time.setCurrentTimeMillis(currTime)
13:            if ADDR_NODE2.length > 0 then
14:              meneruskan waktu base station kepada semua node ADDR_NODE2
15:            end if
16:          else if str.charAt(0) == 'T' then
17:            merupakan pesan berisi waktu dari node ADDR_NODE2
18:            pesan diteruskan kepada node ADDR_NODE1
19:          else if str.equalsIgnoreCase("EXIT") then
20:            merupakan pesan untuk menghentikan program dari base station
21:            isSensing  $\leftarrow$  false
22:            exit  $\leftarrow$  true
23:            if ADDR_NODE2.length > 0 then
24:              meneruskan pesan "EXIT" kepada semua node ADDR_NODE2
25:            end if
26:          else if str.equalsIgnoreCase("WAKTU") then
27:            merupakan pesan untuk meminta waktu dari setiap node sensor
28:            msg  $\leftarrow$  "Time "ADDR_NODE3 + " " + Time.currentTimeMillis()
29:            node sensor mengirimkan msg kepada ADDR_NODE1
30:            if ADDR_NODE2.length > 0 then
31:              meneruskan pesan "WAKTU" kepada semua node ADDR_NODE2
32:            end if
33:          else if str.equalsIgnoreCase("ON") then
34:            merupakan pesan untuk meminta status menyala dari setiap node sensor
35:            msg  $\leftarrow$  pesan bahwa node tersebut online
36:            mengirim msg kepada node ADDR_NODE1
37:            if ADDR_NODE2.length > 0 then
38:              meneruskan pesan "ON" kepada semua node ADDR_NODE2
39:            end if

```

```

40:         else if str.charAt(str.length()-1) = 'E' then
41:             meneruskan status menyala dari node sensor lain kepada ADDR_NODE1.
42:         else if str.equalsIgnoreCase("DETECT") then
43:             menerima perintah melakukan sensing.
44:             message ← membuat format pesan pengiriman data hasil sensing
45:             myTemp ← message
46:             sn++
47:             if ADDR_NODE2.length > 0 then
48:                 meneruskan pesan "DETECT" kepada semua node ADDR_NODE2
49:             end if
50:             mengirim myTemp kepada ADDR_NODE1
51:             end ← mengatur timer batas menunggu ACK
52:             isSensing ← true
53:         else if str.charAt(0) = 'S' then
54:             mendapatkan data hasil sensing dari node sensor lain
55:             meneruskan data hasil sensing kepada node ADDR_NODE1
56:         else if str.startsWith("ACK") then
57:             menerima ACK
58:             node ← node yang ada pada pesan ACK
59:             if node = ADDR_NODE3 then
60:                 se ← sequence number pada pesan ACK
61:                 if se = sn-1 then
62:                     isSensing ← false
63:                     message ← melakukan sensing dan membuat format pesan data
sensing
64:                     myTemp ← message
65:                     sn++
66:                     mengirim data sensing kepada node ADDR_NODE1
67:                     end ← mengatur timer batas menunggu ACK
68:                     isSensing ← true
69:                 else
70:                     mengirim myTemp kepada ADDR_NODE1
71:                     end ← mengatur timer batas menunggu ACK
72:                 end if
73:             else
74:                 meneruskan pesan ACK kepada semua node ADDR_NODE2.
75:             end if
76:         end if
77:     catch Exception
78:     end try
79: end while
80: end function
81: end thread
82: thread.start()
83: while thread.isAlive() do
84:     if isSensing = true and exit = false then
85:         if Time.currentTimeMillis() > end then
86:             jika sudah melewati batas timer maka akan dikirim ulang myTemp
87:             end ← mengatur ulang timer batas menunggu ACK
88:         end if
89:     end if
90: end while
91: end function

```

4.5 Perancangan *Routing* Pada Aplikasi Transfer Data

Pada subbab 3.1.1 telah dijelaskan bahwa aplikasi transfer data membutuhkan alamat tujuan untuk mengirim data baik data hasil *sensing* maupun pesan perintah untuk melakukan sesuatu. Pada subbab ini dijelaskan lebih detail mengenai *routing* yang digunakan pada aplikasi transfer data.

Pada aplikasi transfer data yang dibangun, menggunakan 5 buah node sensor dengan 1 node sensor sebagai *base station* dan 4 node sensor untuk melakukan *sensing*. *Routing* pada *single-hop* dan *multi-hop* memiliki perbedaan pada tujuan pengiriman data setiap node sensor. Tabel 4.1 dan Tabel 4.2 adalah tabel *routing* yang digunakan pada aplikasi *single-hop* dan *multi-hop*.

Tabel 4.1: Tabel *routing* pada komunikasi *single-hop*.

Node	Tujuan Pengiriman Data
ABFE	DAAA, DAAB, DAAC, DAAD
DAAA	ABFE
DAAB	ABFE
DAAC	ABFE
DAAD	ABFE

Pada aplikasi transfer data dengan komunikasi *single-hop*, node ABFE memiliki peran sebagai *base station* yang akan mengirim data atau perintah kepada node sensor DAAA, DAAB, DAAC, dan DAAD. Sedangkan pada *multi-hop* terdapat 5 tipe topologi yang digunakan dengan node ABFE sebagai *base station* dan node DAAA, DAAB, DAAC, DAAD sebagai node yang melakukan *sensing*.

4.6 Perancangan Format Pesan

Pada subbab 3.2 telah dijelaskan bahwa dalam menjalankan setiap fitur harus ditentukan format pesan yang dikirimkan. Untuk mendapatkan data hasil *sensing*, *base station* mengirimkan pesan "DETECT" kepada node sensor yang terhubung dengan *base station* tersebut. Kemudian node sensor akan mengirimkan data hasil *sensing* dengan format tertentu agar dapat diproses hingga ditulis ke dalam file text. Format pesan data hasil *sensing* terdiri dari:

1. Kata awal "SENSE". Kata ini digunakan sebagai penanda bahwa pesan yang dikirim adalah data hasil *sensing*.
2. Nama node. Nama node diperlukan untuk mengetahui data hasil *sensing* didapat dari node sensor yang mana.
3. *Sequence Number*. Setelah nama node sensor terdapat *sequence number*.
4. Waktu atau *timestamp*. Waktu pada pesan ini adalah waktu saat node sensor melakukan *sensing* dalam format *long*.
5. Suhu. Data untuk suhu diawali dengan huruf 'T'. Suhu yang didapatkan merupakan suhu dalam celsius, sehingga diakhir terdapat huruf '[C]'.
6. *Acceleration*. Data untuk *acceleration* diawali dengan huruf 'A'. *Acceleration* ini terdapat 3 nilai yaitu x,y, dan z.
7. Kelembaban. Data untuk kelembaban diawali dengan huruf 'H'.

Saat disatukan pesan tersebut akan menjadi "SENSE <Nama Node> <Sequence Number> <Timestamp> <Data Sensing>". Contoh pesan data hasil *sensing* adalah

Tabel 4.2: Tabel *routing* pada komunikasi *multi-hop*.

Tipe	Node	Tujuan Pengiriman Perintah	Tujuan Pengiriman Data Sensing
1	ABFE	DAAA	-
	DAAA	DAAB	ABFE
	DAAB	DAAC	DAAA
	DAAC	DAAD	DAAB
	DAAD	-	DAAC
2	ABFE	DAAA DAAC	-
	DAAA	DAAB	ABFE
	DAAB	-	DAAA
	DAAC	DAAD	ABFE
	DAAD	-	DAAC
3	ABFE	DAAA DAAB	-
	DAAA	-	ABFE
	DAAB	DAAC DAAD	DAAA
	DAAC	-	DAAB
	DAAD	-	DAAB
4	ABFE	DAAA	-
	DAAA	DAAB	ABFE
	DAAB	DAAC DAAD	DAAA
	DAAC	-	DAAB
	DAAD	-	DAAB
5	ABFE	DAAA DAAB DAAC	-
	DAAA	-	ABFE
	DAAB	-	ABFE
	DAAC	DAAD	ABFE
	DAAD	-	DAAC

- SENSE DAAB 1 1557929450 T: 29.01599884033203 [C]; A:[0, 0, 0]; H: 72.45306396484375
- SENSE DAAA 2 1557929455 T: 26.64480018615722 [C]; A:[118, 78, 208]; H: 77.19854736328125

Aplikasi yang dibangun juga memiliki fitur untuk mengetahui node mana yang menyala (*online*). Pada fitur ini *base station* akan mengirimkan pesan "ON" kepada node yang terhubung dengan *base station* tersebut. Kemudian node sensor yang menerima pesan tersebut akan membangun pesan untuk mengetahui node yang menyala. Format pesan untuk mengetahui node yang menyala terdiri dari:

1. Kata "Node"
2. Nama node.
3. Kata "ONLINE".

Saat disatukan pesan tersebut akan menjadi "Node <Nama Node> ONLINE". Contoh pesan node yang online adalah

- "Node daaa ONLINE"
- "Node daab ONLINE"
- "Node daac ONLINE"

Fitur untuk melakukan sinkronisasi waktu juga memerlukan format pesan yang harus dikirimkan. Pesan melakukan sinkronisasi waktu ini dikirimkan oleh *base station*. Format pesan melakukan sinkronisasi waktu terdiri dari:

1. Huruf 'Q'.
2. Waktu dari *base station*.

Saat disatukan pesan tersebut akan menjadi "Q<Waktu *base station*>". Contoh pesan yang *base station* kirim adalah "Q1557929422"

Fitur lain yang dimiliki aplikasi ini adalah mengetahui waktu setiap node sensor. Awalnya *base station* akan mengirimkan pesan "WAKTU" untuk meminta waktu dari setiap node sensor. Node sensor yang menerima pesan tersebut akan membuat pesan yang terdiri dari:

1. Kata "Time"
2. Nama node
3. Waktu yang terdiri dari tanggal dan jam.

Saat disatukan pesan tersebut akan menjadi "Time <Nama Node> <Waktu Node Sensor>". Waktu node sensor yang dikirimkan menggunakan format *long*. Contoh pesan mendapatkan waktu adalah

- "Time DAAA 1557929450"
- "Time DAAB 1557925150"

Untuk memastikan data telah diterima dengan benar oleh *base station*, maka *base station* akan mengirimkan pesan ACK yang ditujukan kepada node yang terhubung dengan *base station*. Isi pesan tersebut adalah sebagai berikut :

1. Kata "ACK"
2. Nama node
3. Sequence number dari data tersebut

Saat disatukan pesan tersebut akan menjadi "ACK <Nama Node> <Sequence Number>". Sehingga pesan yang dikirimkan dari *base station* adalah

- "ACK DAAA 12"
- "ACK DAAB 15"
- "ACK DAAC 14"

BAB 5

IMPLEMENTASI DAN PENGUJIAN

Bab ini terdiri atas implementasi, pengujian, dan masalah yang dihadapi. Pada bagian implementasi dijelaskan mengenai lingkungan implementasi dan hasil dari implementasi. Pada bagian pengujian berisi hasil dari pengujian. Pada bagian masalah yang dihadapi dijelaskan masalah-masalah yang dihadapi pada saat implementasi.

5.1 Implementasi

5.1.1 Lingkungan Implementasi

Berikut adalah spesifikasi *laptop* yang digunakan untuk implementasi:

1. *Processor* : Intel Core i3-4030U 1.9 Ghz
2. *Memory* : 6144MB Ram
3. *Storage* : 500GB HDD dan 250GB SSD
4. *VGA* : NVIDIA GeForce 840M + Intel HD Graphics Family
5. *Operating System* : Windows 10 64-bit

Berikut adalah spesifikasi perangkat lunak yang digunakan untuk implementasi:

1. IDE : Eclipse IDE Photon Release (4.8.0)
2. Bahasa Pemrograman : Java
3. *Java Library* : Java 1.8.0_181

Berikut adalah spesifikasi node sensor yang digunakan untuk implementasi:

1. Nama : Preon32
2. *Processor* : Cortex-M3
3. *Operating System* : PreonVM
4. Penyimpanan Sistem : 64 kByte SRAM
5. Penyimpanan Data : 256 kByte Flash
6. Pita frekuensi : 2400.0 - 2483.5 MHz
7. Jangkauan : 250 meter (luar ruangan) dan 30 meter (dalam ruangan)
8. Sensor-sensor : Sensor suhu, sensor cahaya, sensor kelembaban, sensor tekanan udara, dan sensor getaran.

5.1.2 Hasil Implementasi

Hasil Implementasi ini adalah aplikasi sesuai dengan perancangan pada Bab 4. Implementasi aplikasi ini menggunakan Bahasa Pemrograman Java. Terdapat kelas-kelas yang akan diunggah ke dalam node sensor yaitu kelas BS dan kelas NS. Selain kelas-kelas yang diunggah ke dalam node sensor, terdapat juga kelas yang akan menghubungkan *base station* dengan komputer pengguna menggunakan *command line interface* yaitu kelas Handler. Untuk mendapatkan data diperlukan juga kelas-kelas untuk menangani setiap sensor pada Preon32.

Kelas AccelerationSensor

Kode program kelas ini dapat dilihat pada Lampiran A listing A.1. Pada kelas ini terdapat metode *run*. Di dalam metode *run* terdapat variabel lokal *spi* dengan tipe NativeSPI yang berfungsi untuk *driver bus*. Sebelum dapat melakukan *sensing*, harus dinyalakan dahulu driver NativeSPI dan GPIO. Nilai yang didapat dari sensor ini terdiri dari 3 nilai dan disimpan ke dalam variabel *values*. Nilai *values* ini kemudian akan disimpan ke dalam atribut *temp* dengan format "A: [x,y,z]". Pada kelas ini juga terdapat metode *getTemp* untuk mengembalikan nilai dari atribut *temp*.

Kelas HumiditySensor

Kode program kelas ini dapat dilihat pada Lampiran A listing A.2. Pada kelas ini terdapat metode *run* yang menyalakan driver SHT21 tersebut dan membuat format "H: + rh" dan disimpan ke dalam atribut *temp*. Pada kelas ini juga terdapat metode *getTemp* yang berfungsi untuk mengembalikan nilai dari atribut *temp*.

Kelas TemperatureSensor

Kode program untuk kelas ini dapat dilihat pada Lampiran A listing A.3. Pada kelas ini terdapat metode *run*. Pada metode *run*, driver untuk sensor suhu tersebut dinyalakan. Suhu yang didapat adalah suhu dalam celsius dan disimpan ke dalam variabel lokal *celsius*. Kemudian atribut *temp* akan diisi dengan format "T: + celsius + [C]". Pada kelas ini juga terdapat metode *getTemp* yang mengembalikan nilai dari atribut *temp*.

Kelas sensing

Kode program kelas ini dapat dilihat pada Lampiran A listing A.4. Kelas ini digunakan untuk menggabungkan ketiga sensor yang digunakan menjadi sebuah string. String ini yang akan disimpan nantinya ke dalam file. Terdapat metode *sense* dengan tipe String yang mengembalikan nilai gabungan dari ketiga sensor tersebut. Pertama-tama harus dinyalakan dahulu driver *NativeI2C*. Kemudian memanggil metode *run* dari setiap objek sensor yang telah dibuat. Metode *sense* akan mengembalikan String.

Kelas BS

Kode program kelas ini dapat dilihat pada Lampiran A listing A.5. Saat dijalankan, kelas ini akan menginisialisasi atribut-atribut yang digunakan seperti menginisialisasi *hmapSN* dengan *key* adalah semua node pada *node_list*, dan *value* 1. Setelah itu akan dipanggil metode *startUSART*. Metode *startUSART* ini digunakan untuk melakukan inisialisasi USART yang digunakan untuk mengirim pesan kepada komputer pengguna. Setelah itu akan dipanggil metode *runs*.

Metode *runs* digunakan untuk mengaktifkan radio, *tranceiver*, dan membuat objek *FrameIO*. Di dalam metode ini juga akan dipanggil metode *sender* dan *receiver* di dalam sebuah *thread* yang terus berjalan.

Metode *sender* digunakan untuk mengirimkan perintah kepada semua node ADDR_NODE2. Pesan ini bergantung pada masukan pengguna melalui program pengguna (Handler). Pada metode

ini menggunakan USART untuk menerima pesan tersebut. Jika masukan tersebut adalah '0', maka yang dikirim adalah pesan "EXIT". Jika masukan tersebut adalah '1', maka yang dikirim adalah pesan "ON". Jika masukan tersebut adalah '2', maka yang dikirim adalah pesan "Q" + Waktu saat ini sesuai dengan waktu komputer pengguna. Jika masukan tersebut adalah '3', maka pesan yang dikirim adalah pesan "WAKTU". Jika masukan tersebut adalah '4', maka pesan yang dikirim adalah pesan "DETECT". Implementasi dari metode *sender* dapat dilihat pada listing 5.1

Listing 5.1: Metode sender pada kelas BS

```
public static void sender(final FrameIO fio) throws Exception {
    new Thread() {
        public void run() {
            while (true) {
                int temp = 0;
                try {
                    temp = usart.read();
                } catch (USARTException e1) {
                    e1.printStackTrace();
                }
                if (temp == 0) {
                    try {
                        for (int i = 0; i < ADDR_NODE2.length; i++) {
                            send("EXIT", ADDR_NODE2[i], fio);
                        }
                    } catch (Exception e1) {
                        e1.printStackTrace();
                    }
                    exit = true;
                    firstSense = false;

                    break;
                } else if (temp == 1) {
                    try {
                        for (int i = 0; i < ADDR_NODE2.length; i++) {
                            send("ON", ADDR_NODE2[i], fio);
                        }
                    } catch (Exception e1) {
                        e1.printStackTrace();
                    }
                } else if (temp == 2) {
                    long currTime = Time.currentTimeMillis();
                    try {
                        for (int i = 0; i < ADDR_NODE2.length; i++) {
                            send(("Q" + currTime), ADDR_NODE2[i], fio);
                        }
                    } catch (Exception e1) {
                        e1.printStackTrace();
                    }
                } else if (temp == 3) {
                    try {
                        for (int i = 0; i < ADDR_NODE2.length; i++) {
                            send("WAKTU", ADDR_NODE2[i], fio);
                        }
                    } catch (Exception e1) {
                        e1.printStackTrace();
                    }
                } else if (temp == 4) {
                    firstSense = true;
                    try {
                        for (int i = 0; i < ADDR_NODE2.length; i++) {
                            send("DETECT", ADDR_NODE2[i], fio);
                        }
                    } catch (Exception e1) {
                        e1.printStackTrace();
                    }
                }
            }
        }
    }.start();
}
```

Metode *receiver* digunakan untuk menerima *Frame* dari node ADDR_NODE2. *Frame* tersebut berisi pesan atau *payload*. Jika pesan diakhiri dengan huruf 'E' atau diawali dengan huruf 'T' maka pesan tersebut akan ditambahkan dengan awalan dan akhiran '#' kemudian dikirimkan melalui USART.

Jika pesan yang diterima diawali dengan kata "SENSE" berarti pesan tersebut adalah data hasil *sensing*. Disini pesan tersebut akan dipecah untuk mendapatkan alamat utama node pengirim pesan, dan *sequence number*. Untuk menghindari duplikasi harus dilakukan pengecekan *sequence number* tersebut. Setelah itu *base station* akan mengirimkan ACK kepada alamat yang terdapat pada *frame* tersebut. Implementasi dari metode *receiver* dapat dilihat pada listing 5.2

Listing 5.2: Metode receiver pada kelas BS

```
public static void receive(final FrameIO fio) throws Exception {
    Thread receive = new Thread() {
```

```

public void run() {
    Frame frame = new Frame();
    while (true) {
        try {
            fio.receive(frame);
            byte[] dg = frame.getPayload();
            String str = new String(dg, 0, dg.length);
            if (str.charAt(str.length() - 1) == 'E') {
                String msg = "#" + str + "#";
                try {
                    out.write(msg.getBytes(), 0, msg.length());
                    usart.flush();
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
            else if (str.charAt(0) == 'T') {
                String msg = "#" + str + "#";
                try {
                    out.write(msg.getBytes(), 0, msg.length());
                    usart.flush();
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
            else if (str.startsWith("SENSE")) {
                int beginNode = str.indexOf('<');
                int beginSN = str.indexOf('>');
                int endSN = str.indexOf('?');
                int node = Integer.parseInt(str.substring(beginNode + 1, beginSN));
                int sn = Integer.parseInt(str.substring(beginSN + 1, endSN));
                if (hmapSN.get(node) == sn) {
                    String msg = "#" + str + "#";
                    try {
                        out.write(msg.getBytes(), 0, msg.length());
                        usart.flush();
                        Thread.sleep(50);
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                    hmapSN.put(node, sn+1);
                }
                send("ACK" + node+"."+sn, frame.getSrcAddr(), fio);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
};
receive.start();
}

```

Pada kelas ini juga terdapat metode *send* untuk mengirimkan pesan kepada node sensor menggunakan *FrameIO*.

Kelas NS

Kode program kelas ini dapat dilihat pada Lampiran A listing A.6. Saat dijalankan, kelas ini akan menginisialisasi atribut-atribut yang digunakan dan memanggil metode *runs*. Metode *runs* ini akan mengaktifkan radio, *transceiver* dan membuat *FrameIO* dari *transceiver* tersebut. Setelah itu metode *runs* akan memanggil metode *send_receive* untuk menangani penerimaan dan pengiriman pesan.

Pada metode *send_receive* terdapat *thread* agar pengiriman dan penerimaan pesan dapat berjalan secara paralel. Di dalam *thread* ini akan menunggu *frame* yang masuk. Jika ada *frame* yang masuk, akan dilihat *payload* dari *frame* tersebut. Jika *payload* diawali dengan huruf 'Q' berarti pesan tersebut adalah waktu dari node sebelumnya. Node sensor ini akan mengatur waktu sesuai dengan waktu yang ada pada pesan tersebut. Kemudian node sensor akan meneruskan pesan tersebut kepada node lain yang berada pada atribut *ADDR_NODE2*.

Jika pesan diawali dengan huruf 'T' berarti pesan tersebut berisi waktu dari node yang terdaftar pada *ADDR_NODE2*. Node sensor akan langsung meneruskan pesan tersebut kepada *ADDR_NODE1*. Jika pesan tersebut adalah kata "EXIT" maka node sensor akan meneruskan pesan "EXIT" tersebut kepada *ADDR_NODE2* dan program yang sedang berjalan akan dihentikan. Jika pesan adalah kata "ON", node sensor akan membuat pesan "Node + *ADDR_NODE3* + ONLINE" untuk memberitahu node *ADDR_NODE1* bahwa dirinya menyala. Setelah itu node sensor akan meneruskan pesan "ON" kepada node *ADDR_NODE2*.

Jika pesan yang diterima diakhiri dengan huruf 'E', berarti pesan tersebut didapat dari node *ADDR_NODE2* yang berisi status node *ADDR_NODE2* yang online. Node sensor yang mendapat

pesan ini akan meneruskan langsung pesan tersebut kepada ADDR_NODE1. Jika pesan yang diterima diawali dengan huruf 'S' berarti pesan tersebut adalah data yang didapatkan dari node ADDR_NODE2. Node sensor yang mendapat pesan ini akan meneruskan langsung pesan tersebut kepada node ADDR_NODE1.

Jika pesan yang diterima adalah kata "DETECT" berarti pesan tersebut adalah perintah untuk mulai melakukan *sensing*. Kelas ini akan mengatur waktu *timer* (atribut *end*) yaitu waktu saat ini ditambah 4 detik. Setelah itu kelas ini akan membuat pesan yang berisi "SENSE<" + ADDR_NODE3 + ">" + *sequence number*(sn) + "?" + waktu saat ini + " "+s.sense(). Contoh pesan tersebut adalah "SENSE<daaa>12?1556448762472 T: 24.8351993560791 [C]; A: [-10, -36, 246]; H: 67.2421875". Setelah itu menyimpan pesan tersebut ke dalam variabel *myTemp*. setelah itu menambah *sn* sebanyak 1 angka dan mengirim *myTemp* kepada node ADDR_NODE1. Pesan "DETECT" juga diteruskan kepada semua node ADDR_NODE2. Terakhir adalah mengubah nilai dari atribut *isSensing* menjadi *true*.

Jika pesan yang diterima diawali dengan kata "ACK", maka pesan tersebut adalah ACK yang dikirim dari *base station*. Pesan ACK yang diterima contohnya adalah "ACK55123.12". "55123" adalah alamat node, dan 12 adalah *sequence number* dari data yang telah diterima oleh *base station*. Jika alamat node tersebut sama dengan ADDR_NODE3 maka akan dilihat apakah *sequence number* yang diterima tersebut sama dengan atribut *sn-1*. Jika sama, maka ACK tersebut sudah tepat. Lalu atribut *isSensing* tersebut diubah menjadi *false* agar *thread.isAlive()* tidak terus berjalan. Setelah itu akan dilakukan *sensing* dan menyimpan pesan ke dalam atribut *myTemp*. Kemudian menambah atribut *sn* sebanyak 1 dan mengirim *myTemp* ke node ADDR_NODE1 dan mengubah status *isSensing* menjadi *true* untuk menjalankan fungsi *retransmission* melewati saat *timer*. Jika *sequence number* pada pesan ACK tidak sama dengan atribut *sn-1*, maka akan langsung dikirim ulang pesan dari *myTemp* dan mengatur ulang waktu *timer* (atribut *end*). Terakhir jika alamat node pada pesan ACK tidak sama dengan ADDR_NODE3, maka pesan langsung diteruskan ke semua node ADDR_NODE2.

Selama *thread* berjalan akan ada pengecekan atribut *isSensing* dan *exit*. Jika nilai *isSensing* adalah *true* dan nilai *exit* adalah *false*, maka akan dilihat waktu saat ini (*Time.currentTimeMillis*) sudah melebihi waktu pada atribut *end* atau belum. Jika sudah melewati, maka akan dilakukan transfer ulang data dari *myTemp* dan mengatur ulang waktu *end*. Implementasi metode *send_receive* pada node sensor dapat dilihat pada listing 5.3.

Listing 5.3: Metode *send_receive* pada kelas NS

```
public static void send_receive(final FrameIO fio) throws Exception {
    Thread thread = new Thread() {
        public void run() {
            Frame frame = new Frame();
            while (true) {
                try {
                    fio.receive(frame);
                    byte[] dg = frame.getPayload();
                    String str = new String(dg, 0, dg.length);
                    if (str.charAt(0) == 'Q') {
                        String tm = str.substring(1);
                        long currTime = Long.parseLong(tm);
                        Time.setCurrentTimeMillis(currTime);
                        if (ADDR_NODE2.length > 0) {
                            for (int i = 0; i < ADDR_NODE2.length; i++) {
                                String message = "Q" + Time.currentTimeMillis();
                                send(message, ADDR_NODE3, ADDR_NODE2[i], fio);
                                Thread.sleep(50);
                            }
                        }
                    } else if (str.charAt(0) == 'T') {
                        send(str, ADDR_NODE3, ADDR_NODE1, fio);
                    }
                    else if (str.equalsIgnoreCase("EXIT")) {
                        isSensing = false;
                        exit = true;
                        if (ADDR_NODE2.length > 0) {
                            for (int i = 0; i < ADDR_NODE2.length; i++) {
                                String message = "EXIT";
                                send(message, ADDR_NODE3, ADDR_NODE2[i], fio);
                                Thread.sleep(50);
                            }
                        }
                        break;
                    }
                    else if (str.equalsIgnoreCase("WAKTU")) {
                        String msg = "Time_" + Integer.toHexString(ADDR_NODE3) + "_" + Time.currentTimeMillis();
```

```

        send(msg, ADDR_NODE3, ADDR_NODE1, fio);
        if (ADDR_NODE2.length > 0) {
            for (int i = 0; i < ADDR_NODE2.length; i++) {
                String message = "WAKTU";
                send(message, ADDR_NODE3, ADDR_NODE2[i], fio);
                Thread.sleep(50);
            }
        }
        else if (str.equalsIgnoreCase("ON")) {
            String msg = "Node_" + Integer.toHexString(ADDR_NODE3) + "_ONLINE";
            send(msg, ADDR_NODE3, ADDR_NODE1, fio);
            if (ADDR_NODE2.length > 0) {
                for (int i = 0; i < ADDR_NODE2.length; i++) {
                    send("ON", ADDR_NODE3, ADDR_NODE2[i], fio);
                    Thread.sleep(50);
                }
            }
        }
        else if (str.charAt(str.length() - 1) == 'E') {
            send(str, ADDR_NODE3, ADDR_NODE1, fio);
        }
        else if (str.equalsIgnoreCase("DETECT")) {
            String message = "SENSE<" + ADDR_NODE3 + ">" + sn + "?" + Time.currentTimeMillis() + " "
                + s.sense();
            myTemp = message;
            Thread.sleep(50);
            sn++;
            if (ADDR_NODE2.length > 0) {
                for (int i = 0; i < ADDR_NODE2.length; i++) {
                    send("DETECT", ADDR_NODE3, ADDR_NODE2[i], fio);
                    Thread.sleep(50);
                }
            }
            send(myTemp, ADDR_NODE3, ADDR_NODE1, fio);
            end = Time.currentTimeMillis() + 4000;
            Thread.sleep(50);
            isSensing = true;
        }
        else if (str.charAt(0) == 'S') {
            send(str, ADDR_NODE3, ADDR_NODE1, fio);
        }
        else if (str.startsWith("ACK")) {
            int indexDot = str.indexOf(".");
            int node = Integer.parseInt(str.substring(3, indexDot));
            if (node == ADDR_NODE3) {
                int se = Integer.parseInt(str.substring(indexDot + 1));
                if (se == sn - 1) {
                    isSensing = false;
                    String message = "SENSE<" + ADDR_NODE3 + ">" + sn + "?" + Time.currentTimeMillis()
                        + " " + s.sense();
                    myTemp = message;
                    Thread.sleep(50);
                    sn++;
                    send(myTemp, ADDR_NODE3, ADDR_NODE1, fio);
                    end = Time.currentTimeMillis() + 4000;
                    Thread.sleep(50);
                    isSensing = true;
                }
                else {
                    send(myTemp, ADDR_NODE3, ADDR_NODE1, fio);
                    end = Time.currentTimeMillis() + 4000;
                }
            }
            else {
                for (int i = 0; i < ADDR_NODE2.length; i++) {
                    send(str, ADDR_NODE3, ADDR_NODE2[i], fio);
                }
            }
        }
    }
    catch (Exception e) {
        e.printStackTrace();
    }
}
};
thread.start();

while (thread.isAlive()) {
    if (isSensing == true && exit == false) {
        if (Time.currentTimeMillis() > end) {
            send(myTemp, ADDR_NODE3, ADDR_NODE1, fio);
            end = Time.currentTimeMillis() + 4000;
        }
    }
}
}
}
}

```

Pada kelas ini terdapat metode *send*. Karena banyak melakukan pengiriman pesan, dibuat metode sendiri yang akan selalu dipanggil saat akan mengirim pesan. Metode ini menggunakan 3 buah parameter yaitu *message*, *source*, dan *destination*. *Message* adalah pesan yang akan dikirim, *source* adalah asal pesan ini. Asal pesan ini biasanya akan diisi oleh ADDR_NODE3. *Destination* adalah tujuan dari pesan ini, dapat diisi dengan ADDR_NODE1 atau ADDR_NODE2.

Kelas BS_Testing

Kode program kelas ini dapat dilihat pada Lampiran A listing A.7. Saat dijalankan, kelas ini akan menginisialisasi atribut-atribut yang digunakan dan mengaktifkan USART dengan memanggil metode *startUSART*. Setelah itu metode *runs* akan dipanggil.

Metode *runs* digunakan untuk mengaktifkan radio, *transceiver*, dan membuat objek *FrameIO*. Di dalam metode ini juga akan dipanggil metode *sender* dan *receiver* dalam *thread* yang berjalan.

Metode *sender* digunakan untuk mengirimkan pesan kepada node ADDR_NODE2. Pesan ini bergantung pada masukan dari pengguna yang dikirimkan dan dibaca melalui USART pada *base station*. Jika masukan tersebut adalah '0', maka yang dikirimkan adalah pesan "EXIT". Jika masukan tersebut adalah '1', maka yang dikirimkan adalah pesan "ON". Jika masukan tersebut adalah '2', maka yang dikirimkan adalah pesan "T + Waktu saat ini". Jika masukan tersebut adalah '3', maka yang dikirimkan adalah pesan "WAKTU". Jika masukan tersebut adalah '4', maka yang dikirimkan adalah pesan "DETECT".

Metode *receive* digunakan untuk menerima *Frame* dari node ADDR_NODE2. Jika pesan tersebut diakhiri dengan huruf 'E' atau jika pesan tersebut diawali dengan huruf 'T' atau jika pesan tersebut diawali dengan huruf 'S', maka pesan tersebut akan ditambahkan tanda '#' diawal dan akhir pesan tersebut. Pesan tersebut akan dikirim kepada program lain melalui USART.

Pada kelas ini juga terdapat metode *send* yang digunakan untuk mengirimkan pesan (*message*) dengan tujuan (*address*) melalui *FrameIO* (*fio*).

Kelas NS_Testing

Kode program kelas ini dapat dilihat pada Lampiran A listing A.8. Saat dijalankan, kelas ini akan menginisialisasi atribut-atribut yang digunakan dan memanggil metode *runs*. Metode ini berfungsi untuk mengaktifkan radio, *transceiver* dan membuat *FrameIO* dari *transceiver* tersebut. Di dalam metode ini akan dipanggil metode *send_receive* untuk menangani penerimaan dan pengiriman pesan.

Pada metode *send_receive* terdapat *thread* agar pengiriman dan penerimaan pesan dapat berjalan secara paralel. Di dalam *thread* ini akan menunggu *frame* yang masuk. Jika ada *frame* yang masuk, akan dilihat *payload* dari *frame* tersebut.

Jika pesan tersebut diawali dengan huruf 'T', node sensor akan mengatur waktu sesuai dengan waktu yang diterima. Kemudian node sensor akan meneruskan pesan tersebut kepada semua node sensor ADDR_NODE2. Jika pesan tersebut adalah kata "EXIT" berarti program harus dihentikan dan meneruskan pesan "EXIT" tersebut kepada node sensor ADDR_NODE2.

Jika pesan tersebut adalah kata "WAKTU", maka node sensor akan mengirim waktu node sensor kepada ADDR_NODE1 dan meneruskan kata "WAKTU" kepada ADDR_NODE2. Jika pesan tersebut adalah kata "ON", maka node sensor akan membuat pesan yang berisi "Node + ADDR_NODE3 + Online" dan mengirimkan ke ADDR_NODE1. Kemudian node sensor ini akan meneruskan pesan "ON" kepada node sensor ADDR_NODE2.

Jika pesan yang diterima memiliki huruf akhir 'E', berarti node sensor ini mendapatkan pesan dari node sensor ADDR_NODE2 dan diteruskan ke node sensor ADDR_NODE1. Jika pesan yang diterima memiliki huruf pertama 'S', berarti node sensor ini mendapatkan pesan yang berisi data hasil *sensing* dari node sensor lain dan meneruskan pesan tersebut ke node ADDR_NODE1.

Jika pesan tersebut adalah kata "DETECT" berarti pesan tersebut adalah perintah untuk melakukan *sensing*. Node sensor akan membuat pesan yang berisi data *sensing* sesuai format yang ditentukan dan mengirim pesan tersebut kepada node ADDR_NODE1. Setelah itu node sensor akan meneruskan kata "DETECT" kepada node ADDR_NODE2.

Pada kelas ini juga terdapat metode *send* untuk mengirimkan pesan dari node sensor kepada node ADDR_NODE1 atau node ADDR_NODE2. Metode ini memiliki parameter *message*, *source*, dan *destination*. Parameter *message* adalah pesan yang akan dikirim, *source* adalah ADDR_NODE3, *destination* adalah tujuan pengiriman pesan, dan *fio* adalah *FrameIO*.

Kelas Handler

Kelas Handler ini adalah proyek lain yang digunakan pada komputer pengguna sebagai pengendali dari jaringan WSN. Kode program kelas ini dapat dilihat pada Lampiran A listing A.9. Kelas Handler ini akan terhubung dengan *base station* dan *base station* akan terhubung pada komputer pengguna melalui *port* USB.

Saat dijalankan akan dipanggil metode *context_set* untuk berpindah *context* secara otomatis. Handler juga berfungsi untuk mengatur waktu *base station* sesuai dengan waktu pada komputer pengguna. Kelas ini akan menampilkan pilihan yang dapat dipilih oleh pengguna. Jika pengguna memilih pilihan yang disediakan maka pilihan tersebut akan dikirimkan melalui *connection* yang dibuat antara Handler dengan *base station*.

Jika pengguna memasukkan angka '0', berarti berhenti dari aplikasi. Jika pengguna memasukkan angka '1', maka aplikasi akan menampilkan node yang menyala. Jika pengguna memasukkan angka '2', maka aplikasi akan menampilkan pesan "Done Synchronize". Jika pengguna memasukkan angka '3', maka aplikasi akan menampilkan waktu dari setiap node sensor. Saat diterima sebenarnya waktu masih menggunakan format long, sehingga perlu diubah ke dalam bahasa yang dapat dibaca oleh pengguna menggunakan metode *stringFormat*. Jika pengguna memasukkan angka '4', maka aplikasi akan mulai melakukan *sensing* dan memanggil metode untuk menulis ke file.

Pada metode ini harus ditentukan dahulu *port* yang digunakan sebagai *base station* dan nama modul yang telah diunggah ke dalam *base station* tersebut. Data yang diterima dari *base station* berupa *byte*, jadi harus dilakukan konversi ke dalam *string* agar dapat dibaca oleh pengguna.

Metode *stringFormat* akan mengembalikan string yang berisi waktu dengan format "dd-MM-yyyy HH:mm:ss.SSS" dengan 'dd' adalah tanggal, 'MM' adalah bulan, 'yyyy' adalah tahun, 'HH' adalah jam, mm adalah menit, 'ss' adalah detik, dan 'SSS' adalah *mikrodetik*.

Untuk menulis data *sensing* ke *file*, menggunakan metode *writeToFile*. Metode ini akan membuat nama *file* dan *folder* sesuai dengan keperluan pengguna. Jika *file* sudah ada maka yang dilakukan adalah menimpa (*overwrite*) *file* tersebut. Data *sensing* yang ditulis adalah data dengan kata awal "SENSE". Data akan ditulis ke dalam *file* setiap 10 data yang diterima.

Kelas USARTConstants

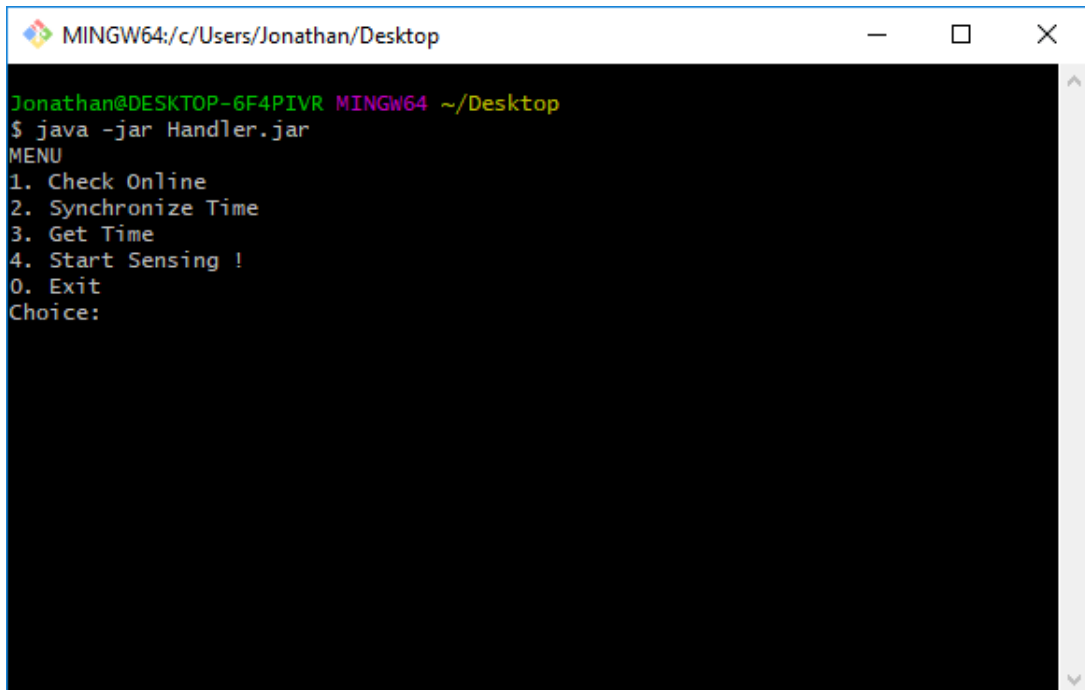
Kelas ini digunakan untuk mengatur frekuensi yang digunakan untuk mengirim pesan melalui USART kepada komputer pengguna. Kode program untuk kelas ini dapat dilihat pada Lampiran A listing A.10. Kelas ini sudah disediakan oleh *virtenio* dan pengguna hanya perlu mengatur frekuensi yang digunakan sesuai dengan kebutuhannya.

5.2 Pengujian

Pengujian dilakukan dengan menggunakan dua buah metode yaitu pengujian fungsional dan pengujian eksperimental. Pengujian fungsional bertujuan untuk menguji fitur-fitur yang disediakan pada aplikasi. Pengujian Eksperimental bertujuan untuk menguji tingkat *reliability* transfer data.

5.2.1 Pengujian Fungsional

Pengujian Fungsional dilakukan dengan menguji fitur yang dijalankan melalui *Command Line Interface* (CLI). Gambar 5.1 adalah tampilan utama pada aplikasi yang dibangun.



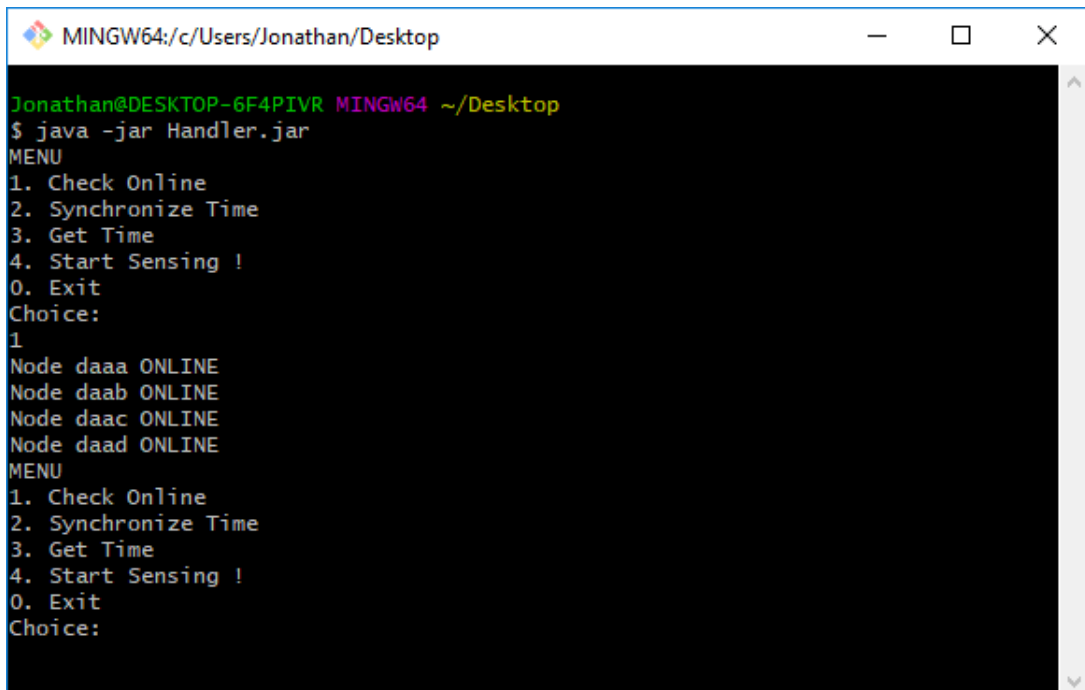
```
MINGW64:/c/Users/Jonathan/Desktop
Jonathan@DESKTOP-6F4PIVR MINGW64 ~/Desktop
$ java -jar Handler.jar
MENU
1. Check Online
2. Synchronize Time
3. Get Time
4. Start Sensing !
0. Exit
Choice:
```

Gambar 5.1: Tampilan Utama Aplikasi

Fitur-fitur yang disediakan pada aplikasi ini adalah sebagai berikut:

1. Check Online

Fitur ini berfungsi untuk mengetahui node yang menyala pada satu jaringan. Untuk menjalankan fitur ini, pengguna harus memasukkan angka "1". Setelah pengguna telah memasukkan angka "1" maka sistem akan menampilkan node yang menyala seperti pada Gambar 5.2.

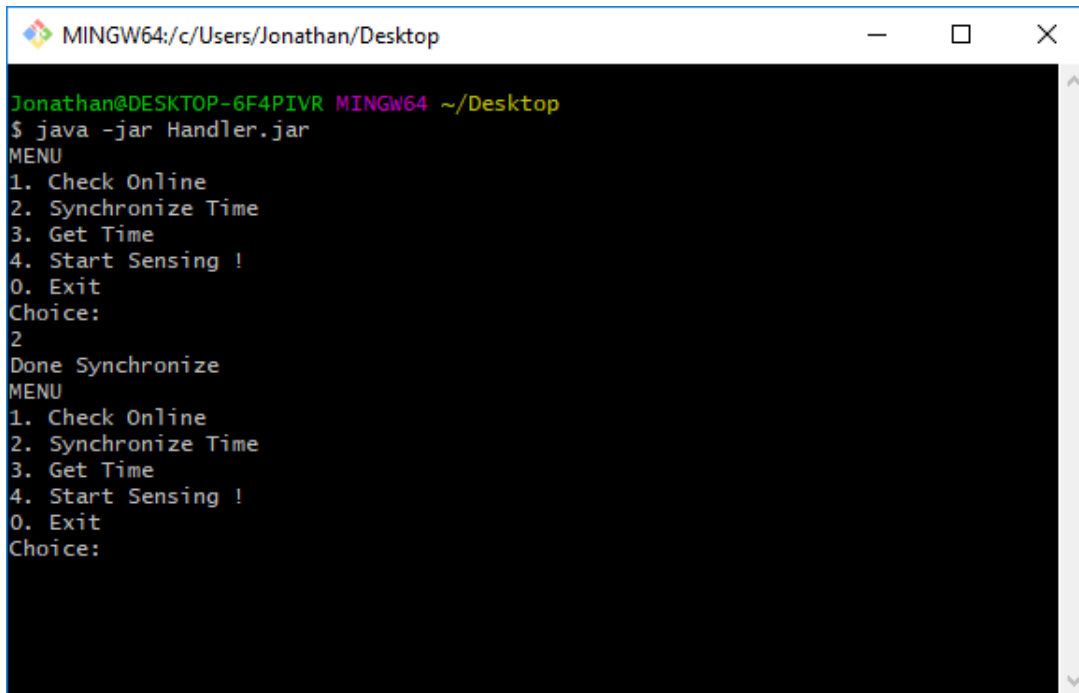


```
MINGW64:/c/Users/Jonathan/Desktop
Jonathan@DESKTOP-6F4PIVR MINGW64 ~/Desktop
$ java -jar Handler.jar
MENU
1. Check Online
2. Synchronize Time
3. Get Time
4. Start Sensing !
0. Exit
Choice:
1
Node daaa ONLINE
Node daab ONLINE
Node daac ONLINE
Node daad ONLINE
MENU
1. Check Online
2. Synchronize Time
3. Get Time
4. Start Sensing !
0. Exit
Choice:
```

Gambar 5.2: Tampilan Check Online

2. Synchronize Time

Fitur ini digunakan untuk melakukan sinkronisasi waktu setiap node sensor sesuai dengan waktu pada *base station*. *Base station* akan mengirimkan waktu yang didapat dari komputer pengguna kepada node sensor. Jika pengguna memasukkan angka "2" maka aplikasi akan menampilkan pesan "Done Synchronize" seperti pada Gambar 5.3.

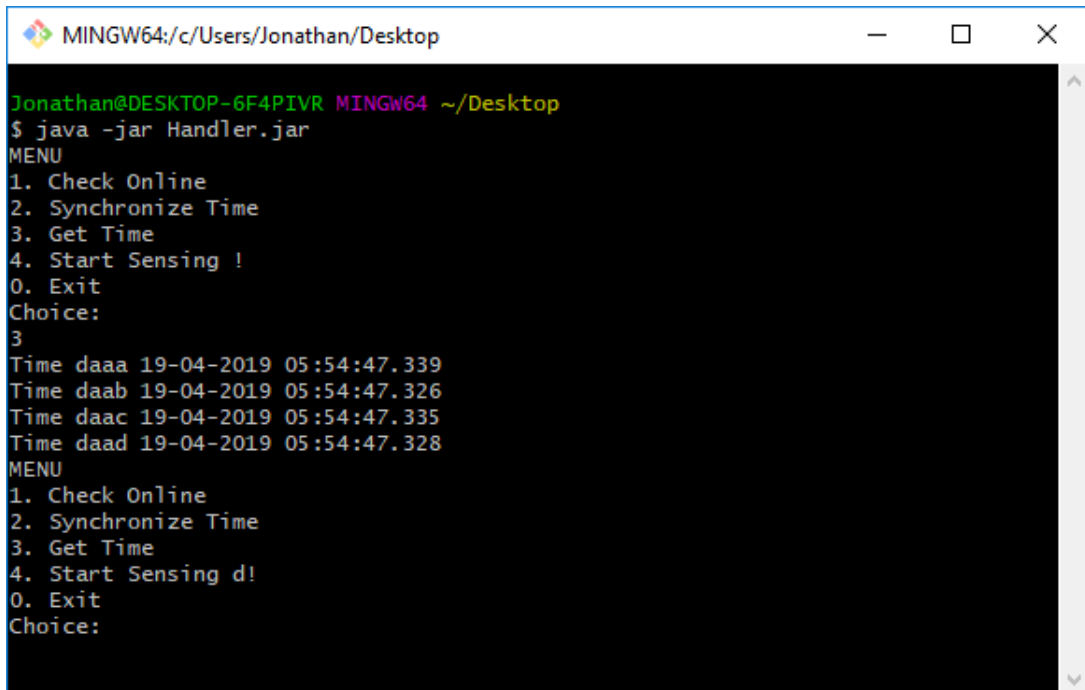


```
MINGW64:/c/Users/Jonathan/Desktop
Jonathan@DESKTOP-6F4PIVR MINGW64 ~/Desktop
$ java -jar Handler.jar
MENU
1. Check Online
2. Synchronize Time
3. Get Time
4. Start Sensing !
0. Exit
Choice:
2
Done Synchronize
MENU
1. Check Online
2. Synchronize Time
3. Get Time
4. Start Sensing !
0. Exit
Choice:
```

Gambar 5.3: Tampilan Synchronize Time

3. Get Time

Fitur ini digunakan untuk mengetahui waktu dari setiap node sensor. Gambar 5.4 adalah tampilan setelah pengguna memasukkan angka "3" untuk mengetahui waktu setiap node sensor.



```
MINGW64:/c/Users/Jonathan/Desktop
Jonathan@DESKTOP-6F4PIVR MINGW64 ~/Desktop
$ java -jar Handler.jar
MENU
1. Check Online
2. Synchronize Time
3. Get Time
4. Start Sensing !
0. Exit
Choice:
3
Time daaa 19-04-2019 05:54:47.339
Time daab 19-04-2019 05:54:47.326
Time daac 19-04-2019 05:54:47.335
Time daad 19-04-2019 05:54:47.328
MENU
1. Check Online
2. Synchronize Time
3. Get Time
4. Start Sensing d!
0. Exit
Choice:
```

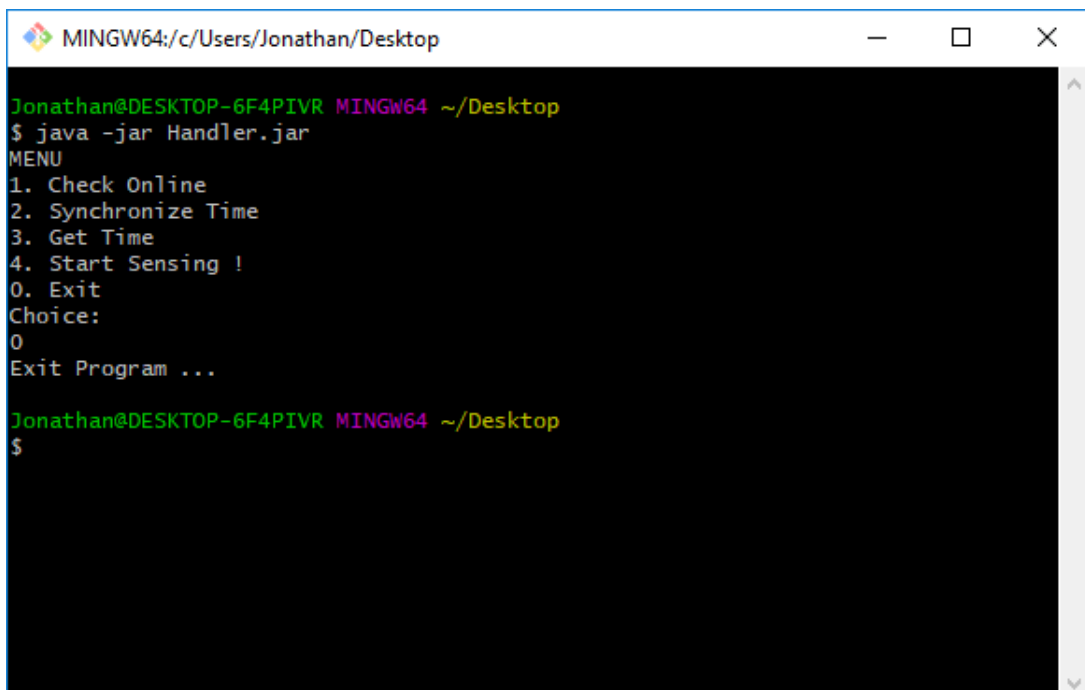
Gambar 5.4: Tampilan Get Time

4. Start Sensing

Fitur ini digunakan sebagai fitur utama aplikasi yang telah dibangun. Aplikasi akan membuat node sensor melakukan *sensing* dan hasilnya akan disimpan langsung ke dalam *file text* untuk dilakukan analisis.

5. Exit

Fitur ini digunakan untuk berhenti dan keluar dari aplikasi. Pengguna harus memasukkan angka "0" untuk berhenti dan keluar dari aplikasi ini seperti pada Gambar 5.5.



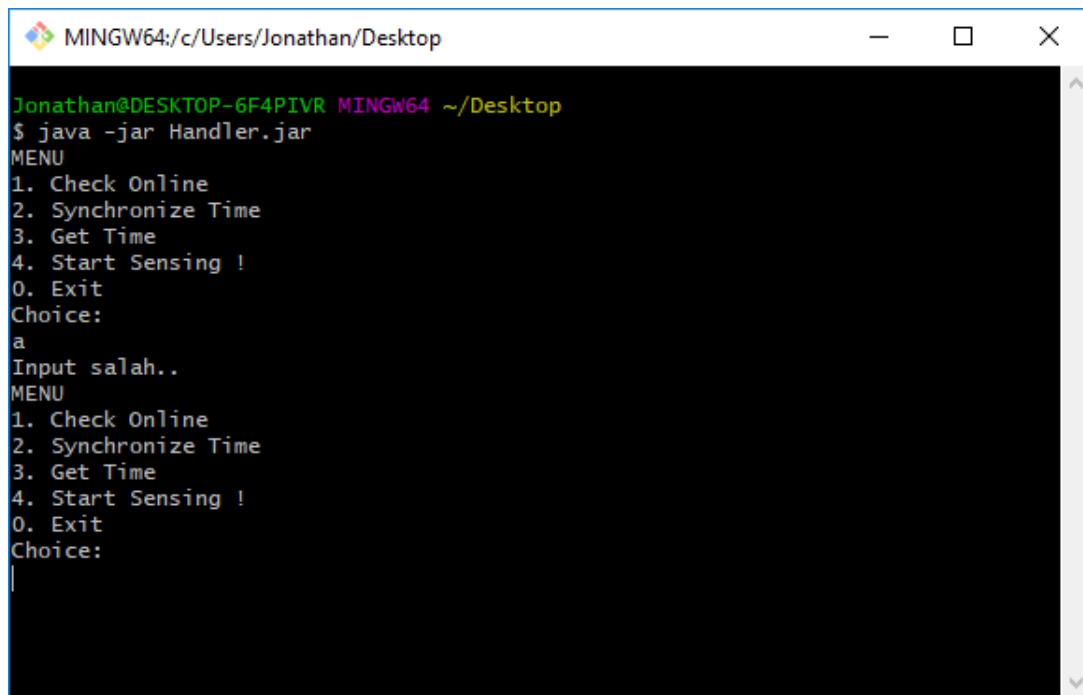
```
MINGW64:/c/Users/Jonathan/Desktop
Jonathan@DESKTOP-6F4PIVR MINGW64 ~/Desktop
$ java -jar Handler.jar
MENU
1. Check Online
2. Synchronize Time
3. Get Time
4. Start Sensing !
0. Exit
Choice:
0
Exit Program ...

Jonathan@DESKTOP-6F4PIVR MINGW64 ~/Desktop
$
```

Gambar 5.5: Tampilan Exit

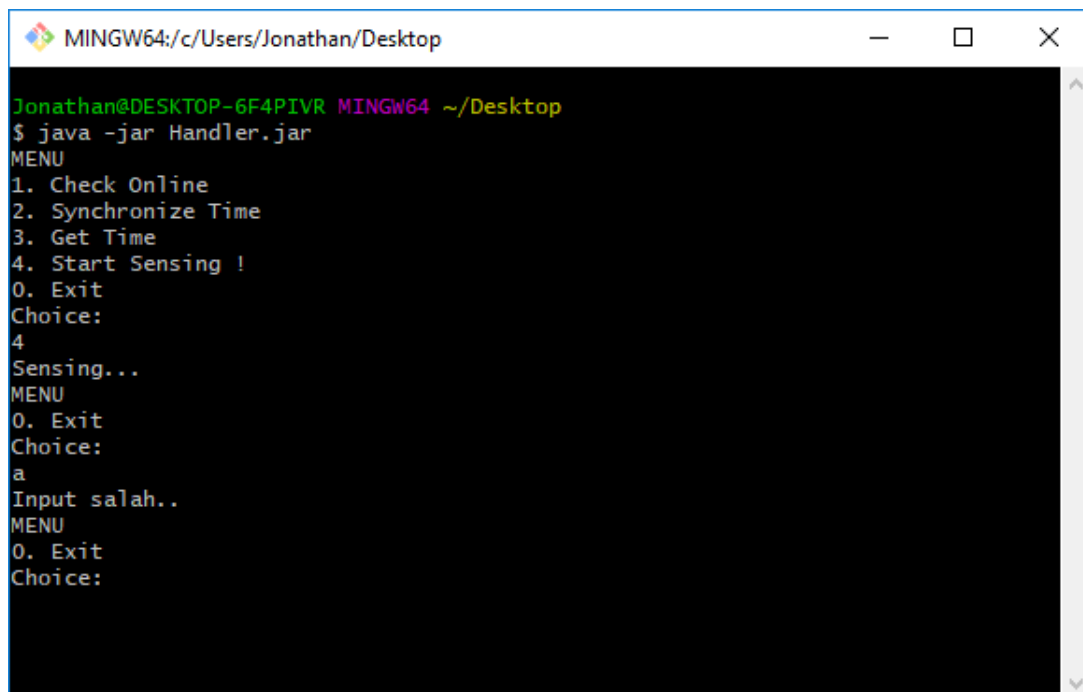
6. Kesalahan Masukkan.

Masukkan yang tidak sesuai akan menampilkan pesan "Input Salah..." Gambar 5.6 dan Gambar 5.7 adalah tampilan saat ada kesalahan masukan.



```
MINGW64:/c/Users/Jonathan/Desktop
Jonathan@DESKTOP-6F4PIVR MINGW64 ~/Desktop
$ java -jar Handler.jar
MENU
1. Check Online
2. Synchronize Time
3. Get Time
4. Start Sensing !
0. Exit
Choice:
a
Input salah..
MENU
1. Check Online
2. Synchronize Time
3. Get Time
4. Start Sensing !
0. Exit
Choice:
|
```

Gambar 5.6: Kesalahan Input 1



```
MINGW64:/c/Users/Jonathan/Desktop
Jonathan@DESKTOP-6F4PIVR MINGW64 ~/Desktop
$ java -jar Handler.jar
MENU
1. Check Online
2. Synchronize Time
3. Get Time
4. Start Sensing !
0. Exit
Choice:
4
Sensing...
MENU
0. Exit
Choice:
a
Input salah..
MENU
0. Exit
Choice:
```

Gambar 5.7: Kesalahan Input 2

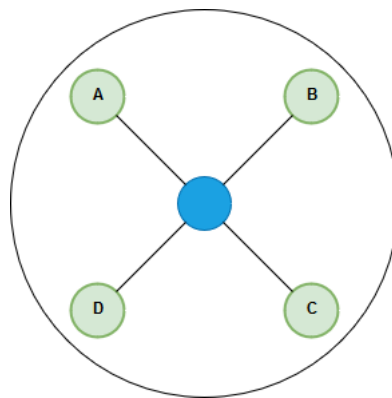
5.2.2 Pengujian Eksperimental

Pengujian eksperimental dilakukan di *Rooftop* Gedung 10 Universitas Katholik Parahyangan sebagai ruang terbuka dan di dalam ruang tertutup. Arsitektur WSN yang digunakan saat pengujian adalah flat dengan *single-hop* dan *multi-hop*. Pengujian dilakukan dengan membandingkan data hasil *sensing* aplikasi transfer yang *reliable* dengan aplikasi transfer data yang tidak *reliable*. Hal yang dibandingkan adalah *sequence number* dan jumlah data yang diterima.

Pengujian dilakukan dengan menggunakan 5 node sensor yang terdiri dari 1 node sensor sebagai *base station* dan 4 node sensor untuk melakukan *sensing*. Pengujian dilakukan sebanyak 1 kali untuk aplikasi transfer data *reliable* dan 3 kali untuk aplikasi transfer data tidak *reliable*. Pada setiap pengujian, diambil data *sample* dengan *sequence number* dari 1 sampai 1000 setiap node sensor untuk dilihat berapa data yang diterima.

Hasil dari setiap pengujian adalah tabel yang berisi jumlah data diterima. Tabel tersebut dibuat menjadi grafik untuk membandingkan jumlah data yang diterima dari aplikasi transfer data *reliable* dalam ruangan (IN R), *reliable* pada ruang terbuka (OUT R), tidak *reliable* dalam ruangan (IN NR), dan tidak *reliable* pada ruang terbuka (OUT NR).

Single Hop

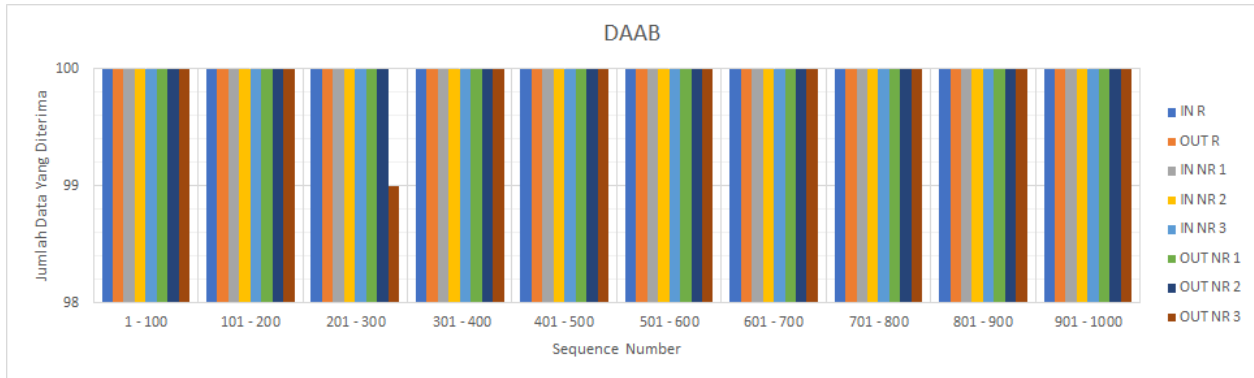


Gambar 5.8: Arsitektur flat *single-hop*

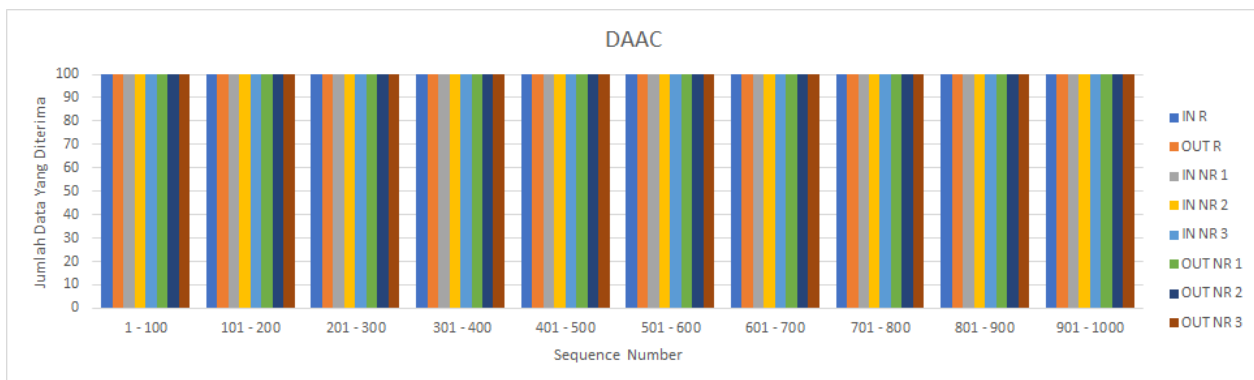
Gambar 5.8 adalah topologi yang penulis gunakan untuk melakukan pengujian pada arsitektur flat dengan *single hop*. Hasil pengujian topologi *single-hop* dapat dilihat pada Lampiran B. Hasil yang didapat adalah tabel untuk setiap node sensor yang berisi jumlah data yang diterima pada setiap skenario. Gambar 5.9 sampai Gambar 5.12 adalah grafik hasil setiap node sensor pada topologi *single-hop*.



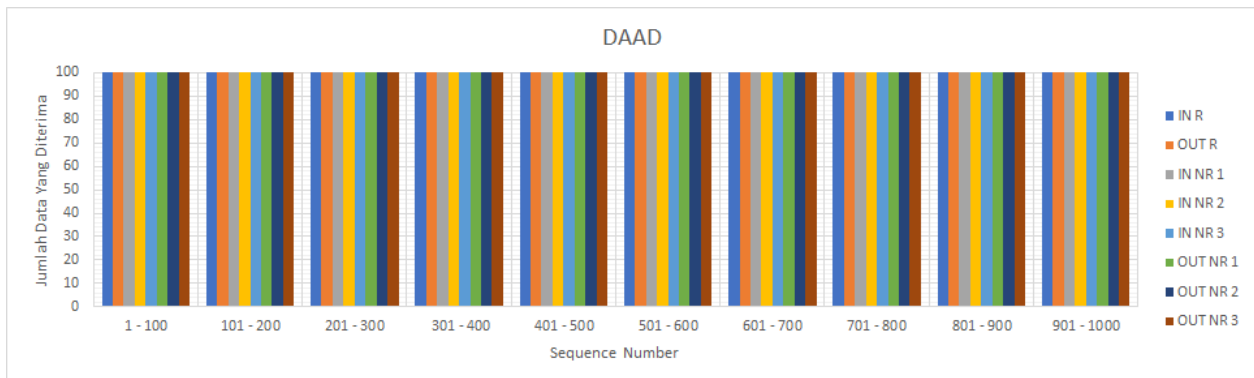
Gambar 5.9: Grafik node DAAA pada *single-hop*



Gambar 5.10: Grafik node DAAB pada single-hop



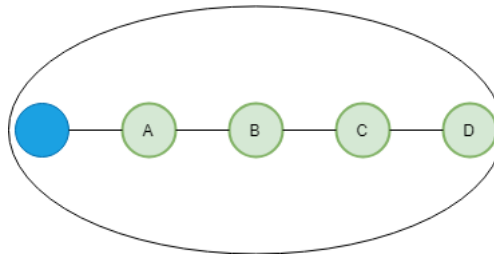
Gambar 5.11: Grafik node DAAC pada single-hop



Gambar 5.12: Grafik node DAAD pada single-hop

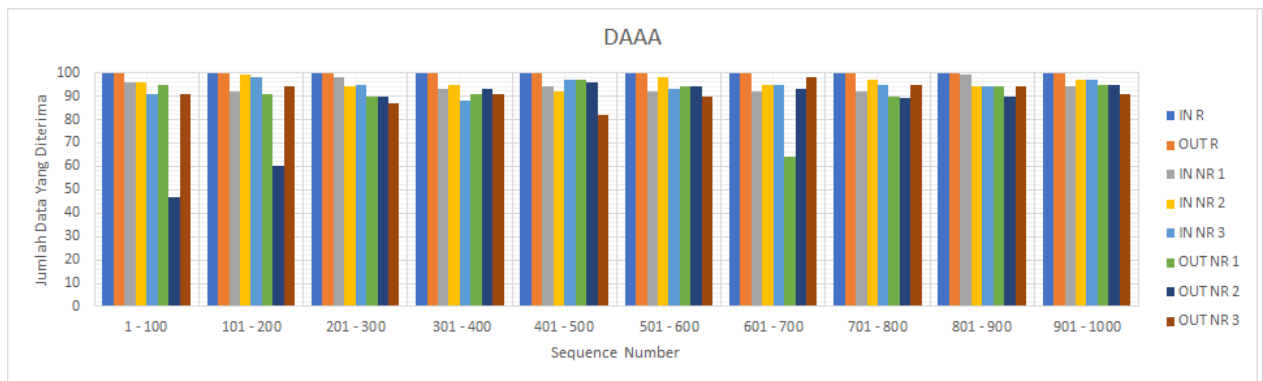
Dari data yang diperoleh dapat dilihat bahwa pada topologi *single-hop* untuk skenario dalam ruangan tidak terdapat *loss* sedangkan untuk skenario ruang terbuka terdapat *loss* hingga 2 data pada salah satu node sensor. Aplikasi transfer *reliable* dan tidak *reliable* tidak banyak perbedaan jumlah *loss*. Hal ini karena pada *layer network* sudah ditangani *auto retry* jika tidak mendapatkan ACK.

Multi Hop

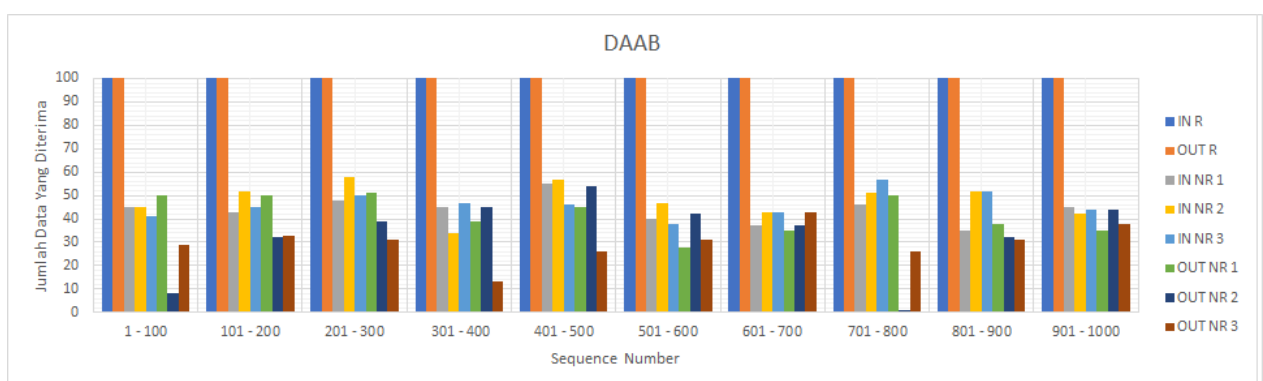


Gambar 5.13: Arsitektur flat *multi-hop* tipe 1

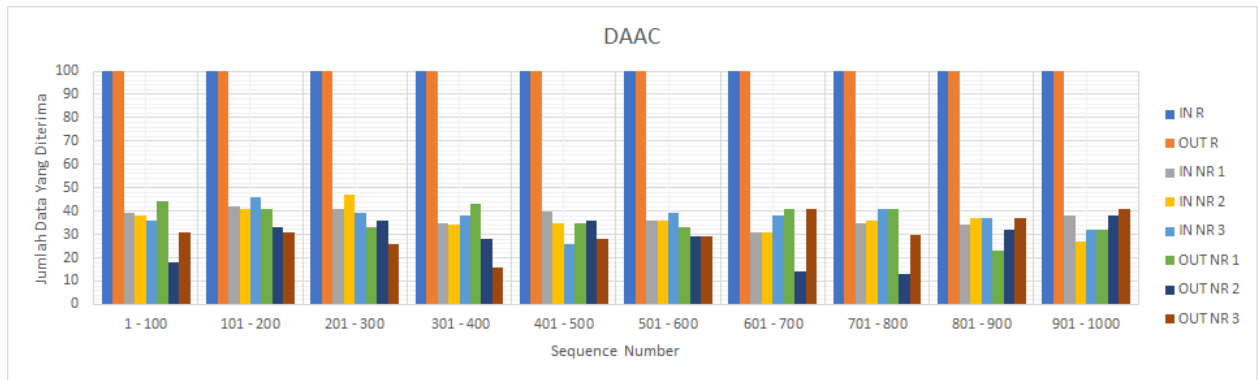
Gambar 5.13 adalah topologi pertama yang penulis gunakan untuk melakukan pengujian pada arsitektur flat dengan *multi-hop*. Hasil pengujian topologi *multi-hop* tipe 1 dapat dilihat pada Lampiran C. Hasil yang didapat adalah tabel setiap node sensor yang berisi jumlah data yang diterima pada setiap skenario. Gambar 5.14 sampai Gambar 5.17 adalah grafik hasil setiap node sensor pada topologi *multi-hop* tipe 1.



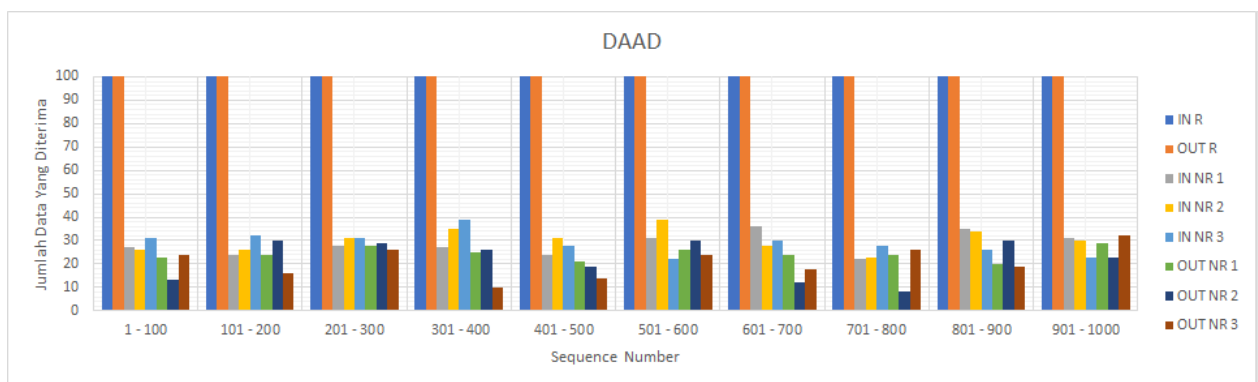
Gambar 5.14: Grafik node DAAA pada multi-hop tipe 1



Gambar 5.15: Grafik node DAAB pada multi-hop tipe 1



Gambar 5.16: Grafik node DAAC pada multi-hop tipe 1

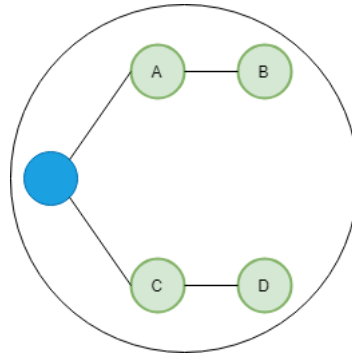


Gambar 5.17: Grafik node DAAD pada multi-hop tipe 1

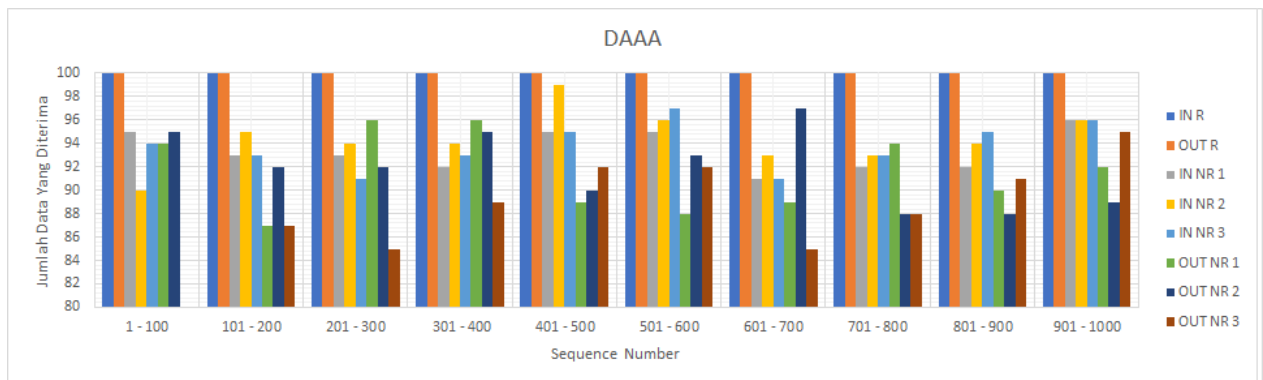
Dari Gambar 5.14 sampai Gambar 5.17 didapat hasil aplikasi transfer yang tidak *reliable* mendapatkan jumlah data yang berbeda-beda.

Untuk aplikasi tidak *reliable* node DAAA dalam ruangan (IN NR) data yang diterima setiap 100 *sequence number* rata-rata sebanyak 94 dan luar ruangan (OUT NR) sebanyak 88. Node DAAB dalam ruangan (IN NR) data yang diterima setiap 100 *sequence number* rata-rata sebanyak 46 dan luar ruangan (OUT NR) sebanyak 35. Node DAAC dalam ruangan (IN NR) data yang diterima setiap 100 *sequence number* rata-rata sebanyak 36 dan luar ruangan (OUT NR) sebanyak 31. Node DAAD dalam ruangan (IN NR) data yang diterima setiap 100 *sequence number* rata-rata sebanyak 29 sedangkan luar ruangan (OUT NR) sebanyak 22.

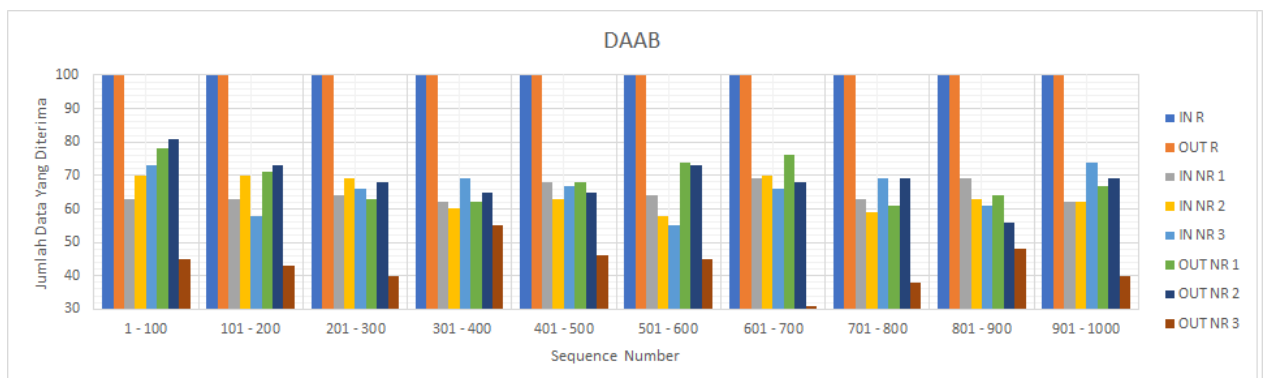
Pada tipe ini semakin banyak hop yang harus dilalui sebuah data, maka semakin sedikit juga data yang diterima *base station*. Sedangkan pada aplikasi transfer data yang *reliable* data yang diterima adalah 100 di ruang terbuka dan di dalam ruangan. Aplikasi memastikan data sampai ke *base station* dengan lengkap.

Gambar 5.18: Arsitektur flat *multi-hop* tipe 2

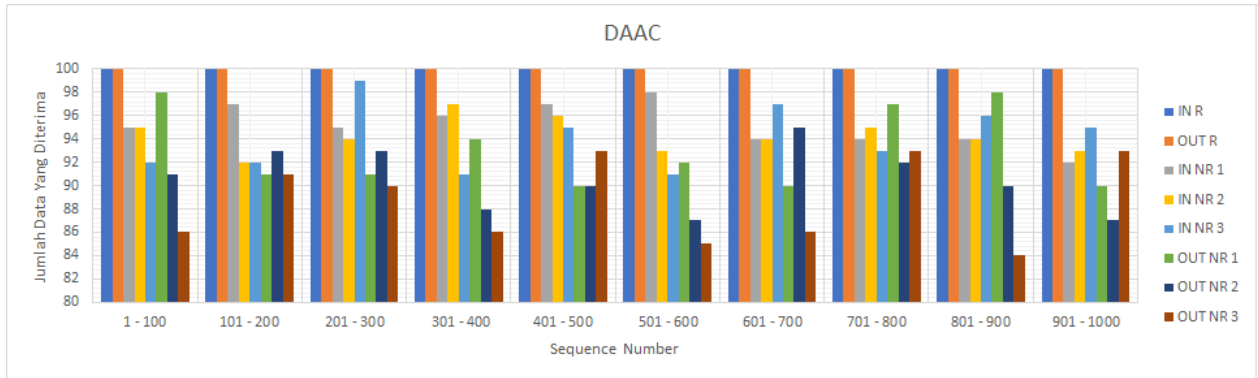
Gambar 5.18 adalah topologi kedua yang penulis gunakan untuk melakukan pengujian pada arsitektur flat dengan *multi-hop*. Hasil pengujian topologi *multi-hop* tipe 2 dapat dilihat pada Lampiran D. Hasil yang didapat adalah tabel untuk setiap node sensor yang berisi jumlah data yang diterima pada setiap skenario. Topologi *multi-hop* tipe 2 ini memiliki perbedaan pada node yang terhubung langsung pada *base station*. Hanya terdapat 2 node yang terhubung langsung dan node tersebut memiliki masing-masing satu node yang terhubung selain *base station*. Gambar 5.19 sampai Gambar 5.22 adalah grafik hasil setiap node sensor pada topologi *multi-hop* tipe 2.



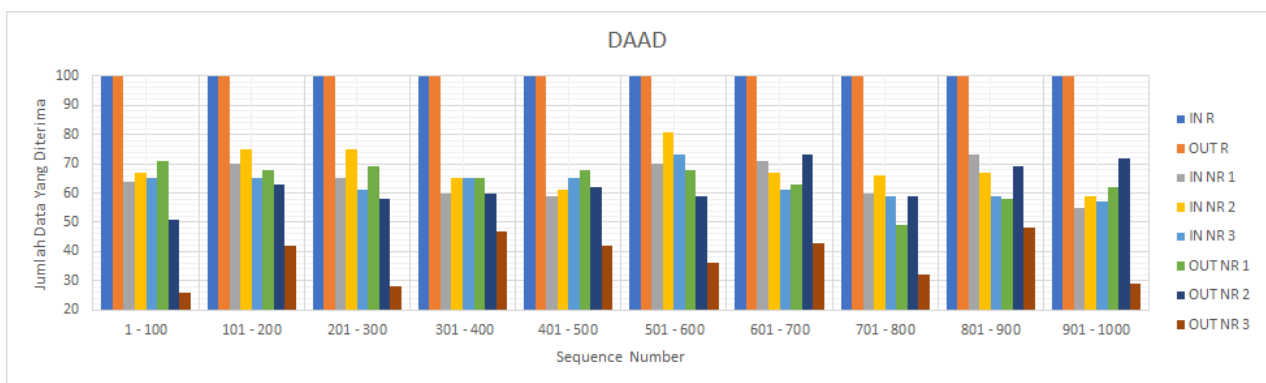
Gambar 5.19: Grafik node DAAA pada multi-hop tipe 2



Gambar 5.20: Grafik node DAAB pada multi-hop tipe 2



Gambar 5.21: Grafik node DAAC pada multi-hop tipe 2

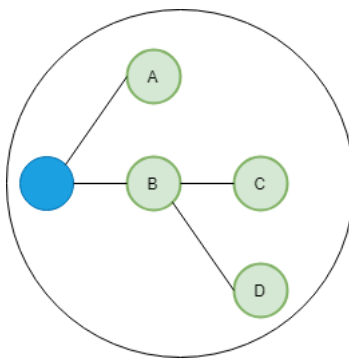


Gambar 5.22: Grafik node DAAD pada multi-hop tipe 2

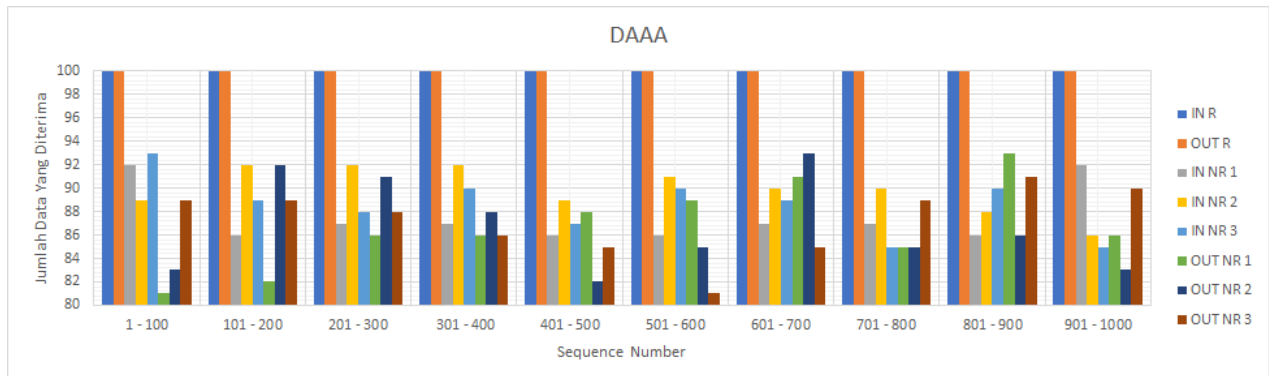
Pada topologi *multi-hop* tipe 2, node yang terhubung langsung pada *base station* adalah node DAAA dan DAAC. Dari kedua node ini data yang diterima cenderung lebih banyak dari setiap skenario dibandingkan dengan node DAAB dan DAAD.

Untuk aplikasi tidak *reliable* node DAAA dalam ruangan (IN NR) data yang diterima setiap 100 *sequence number* rata-rata sebanyak 93 dan luar ruangan (OUT NR) sebanyak 90. Node DAAB dalam ruangan (IN NR) data yang diterima setiap 100 *sequence number* rata-rata sebanyak 64 dan luar ruangan (OUT NR) sebanyak 60. Node DAAC dalam ruangan (IN NR) data yang diterima setiap 100 *sequence number* rata-rata sebanyak 94 dan luar ruangan (OUT NR) sebanyak 90. Node DAAD dalam ruangan (IN NR) data yang diterima setiap 100 *sequence number* rata-rata sebanyak 65 sedangkan luar ruangan (OUT NR) sebanyak 54.

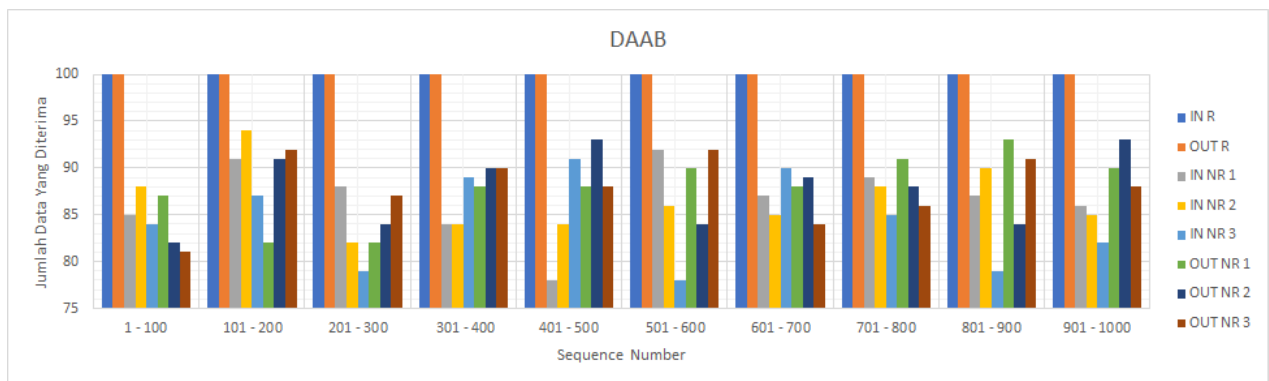
Pada pengujian aplikasi *reliable* baik dalam ruangan maupun ruang terbuka, sama-sama memberikan hasil dengan jumlah *loss* data 0.

Gambar 5.23: Arsitektur flat *multi-hop* tipe 3

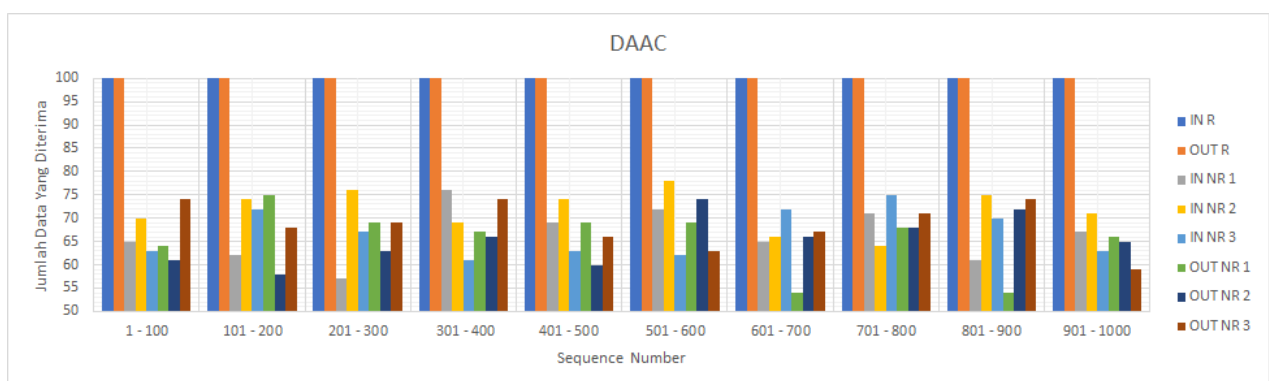
Gambar 5.23 adalah topologi ketiga yang penulis gunakan untuk melakukan pengujian pada arsitektur flat dengan *multi-hop*. Hasil pengujian topologi *multi-hop* tipe 3 dapat dilihat pada Lampiran E.



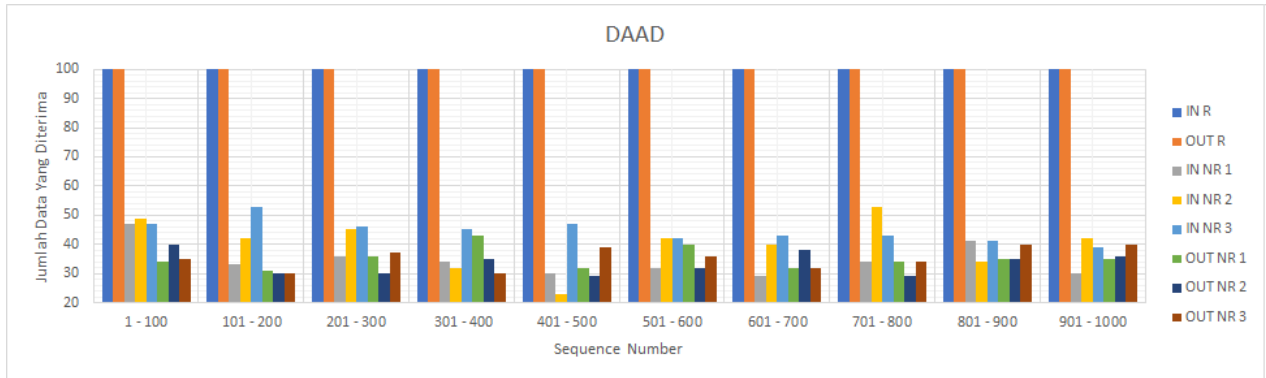
Gambar 5.24: Grafik node DAAA pada multi-hop tipe 3



Gambar 5.25: Grafik node DAAB pada multi-hop tipe 3



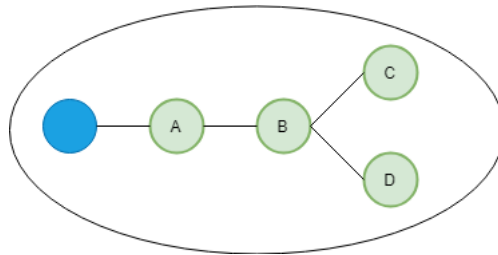
Gambar 5.26: Grafik node DAAC pada multi-hop tipe 3



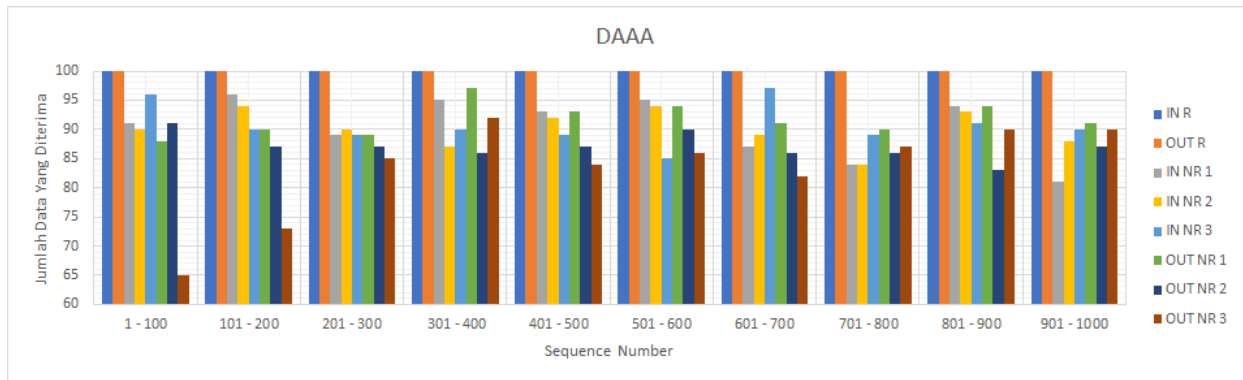
Gambar 5.27: Grafik node DAAD pada multi-hop tipe 3

Gambar 5.24 sampai Gambar 5.27 adalah grafik hasil setiap node sensor pada topologi *multi-hop* tipe 3. Setiap node juga mendapatkan jumlah *loss* yang beragam baik dari pengujian dalam ruangan maupun ruang terbuka. Pada aplikasi transfer data *reliable*, node DAAA, DAAB, DAAC, dan DAAD tidak mendapatkan *loss* pada setiap skenario (jumlah data yang diterima adalah 100 setiap 100 *sequence number*).

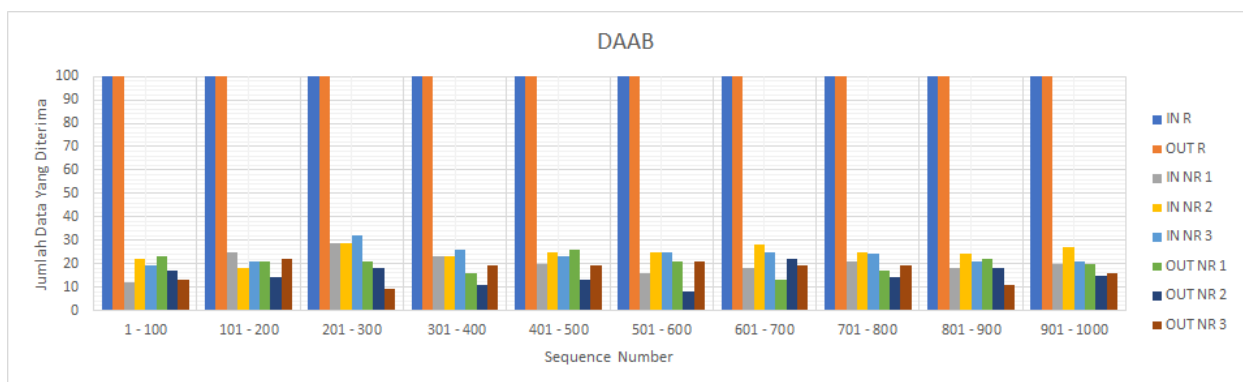
Untuk aplikasi tidak *reliable* node DAAA dalam ruangan (IN NR) data yang diterima setiap 100 *sequence number* rata-rata sebanyak 88 dan luar ruangan (OUT NR) sebanyak 86. Node DAAB dalam ruangan (IN NR) data yang diterima setiap 100 *sequence number* rata-rata sebanyak 85 dan luar ruangan (OUT NR) sebanyak 87. Node DAAC dalam ruangan (IN NR) data yang diterima setiap 100 *sequence number* rata-rata sebanyak 68 dan luar ruangan (OUT NR) sebanyak 66. Node DAAD dalam ruangan (IN NR) data yang diterima setiap 100 *sequence number* rata-rata sebanyak 39 sedangkan luar ruangan (OUT NR) sebanyak 34.

Gambar 5.28: Arsitektur flat *multi-hop* tipe 4

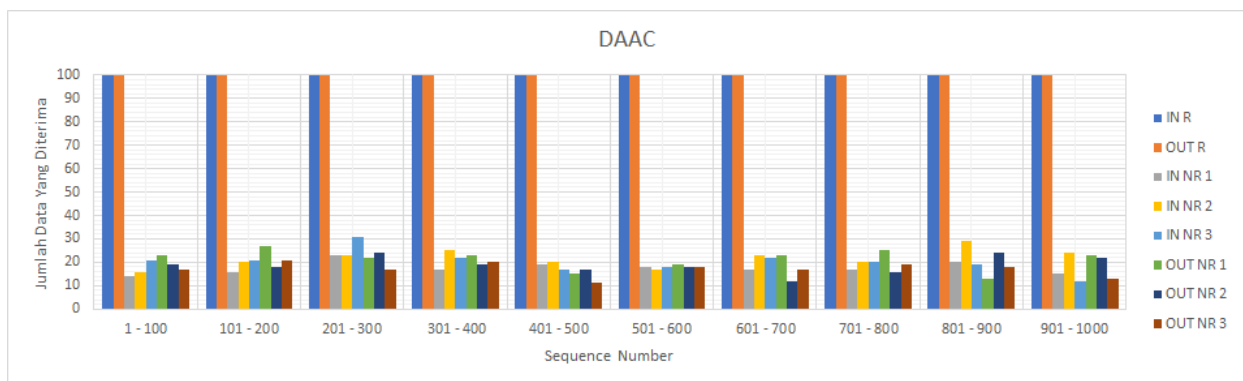
Gambar 5.28 adalah topologi keempat yang penulis gunakan untuk melakukan pengujian pada arsitektur flat dengan *multi-hop*. Hasil pengujian topologi *multi-hop* tipe 4 dapat dilihat pada Lampiran H.



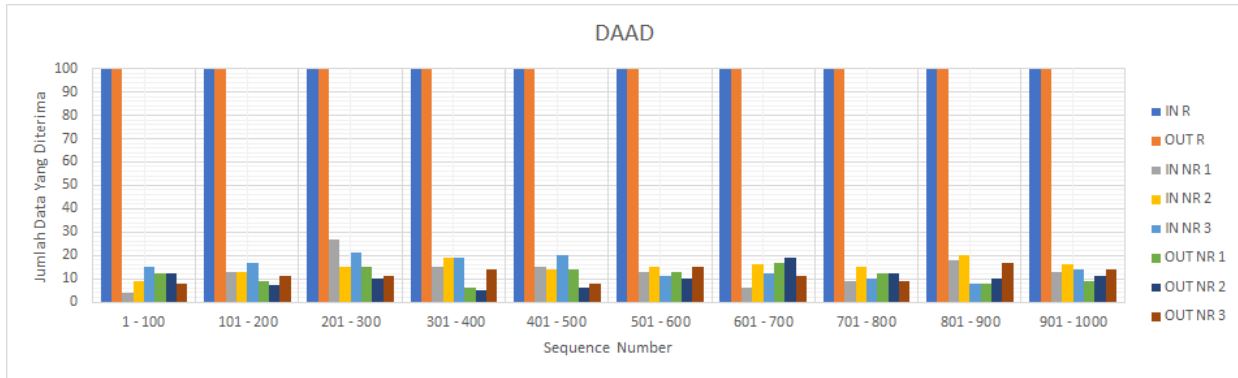
Gambar 5.29: Grafik node DAAA pada multi-hop tipe 4



Gambar 5.30: Grafik node DAAB pada multi-hop tipe 4



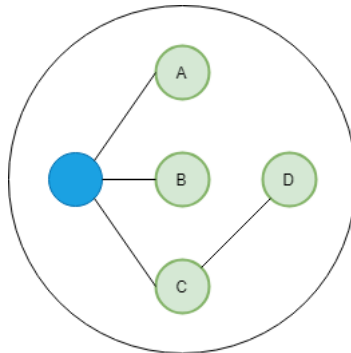
Gambar 5.31: Grafik node DAAC pada multi-hop tipe 4



Gambar 5.32: Grafik node DAAD pada multi-hop tipe 4

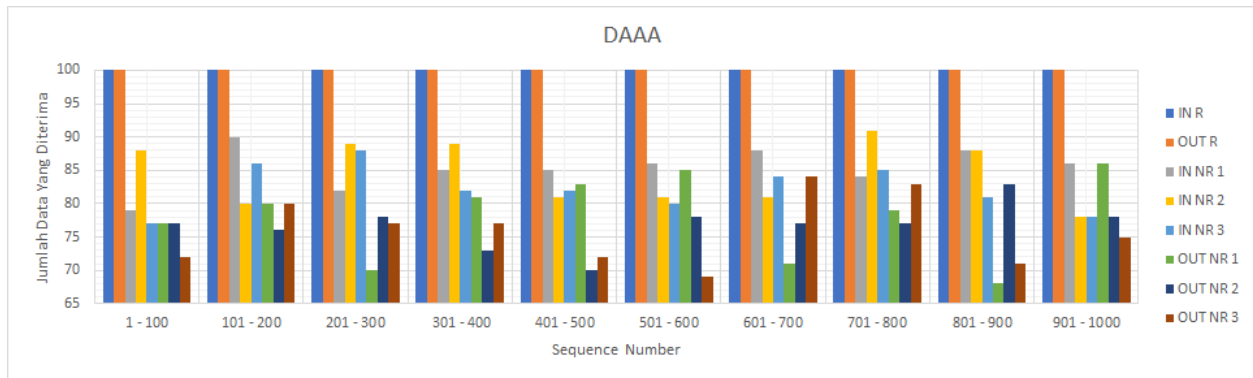
Gambar 5.29 sampai Gambar 5.32 adalah grafik hasil setiap node sensor pada topologi *multi-hop* tipe 4. Pada aplikasi transfer data *reliable* node sensor DAAA, DAAB, DAAC, dan DAAD tidak mendapatkan *loss* pada setiap skenario (jumlah data yang diterima adalah 100 setiap 100 *sequence number*).

Untuk aplikasi tidak *reliable* node DAAA dalam ruangan (IN NR) data yang diterima setiap 100 *sequence number* rata-rata sebanyak 90 dan luar ruangan (OUT NR) sebanyak 87. Node DAAB dalam ruangan (IN NR) data yang diterima setiap 100 *sequence number* rata-rata sebanyak 22 dan luar ruangan (OUT NR) sebanyak 17. Node DAAC dalam ruangan (IN NR) data yang diterima setiap 100 *sequence number* rata-rata sebanyak 19 dan luar ruangan (OUT NR) sebanyak 19. Node DAAD dalam ruangan (IN NR) data yang diterima setiap 100 *sequence number* rata-rata sebanyak 14 sedangkan luar ruangan (OUT NR) sebanyak 11.

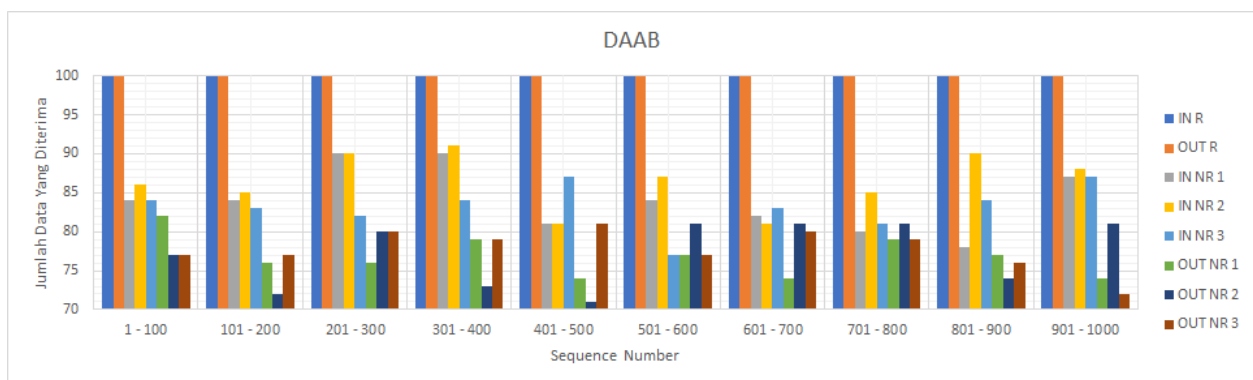


Gambar 5.33: Arsitektur flat *multi-hop* tipe 5

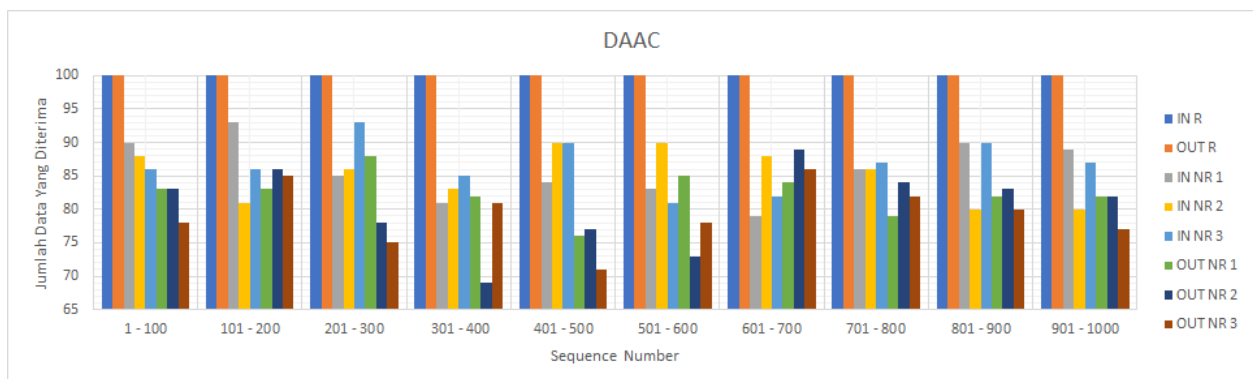
Gambar 5.33 adalah topologi kelima yang penulis gunakan untuk melakukan pengujian pada arsitektur flat dengan *multi-hop*. Hasil pengujian topologi *multi-hop* tipe 5 dapat dilihat pada Lampiran G.



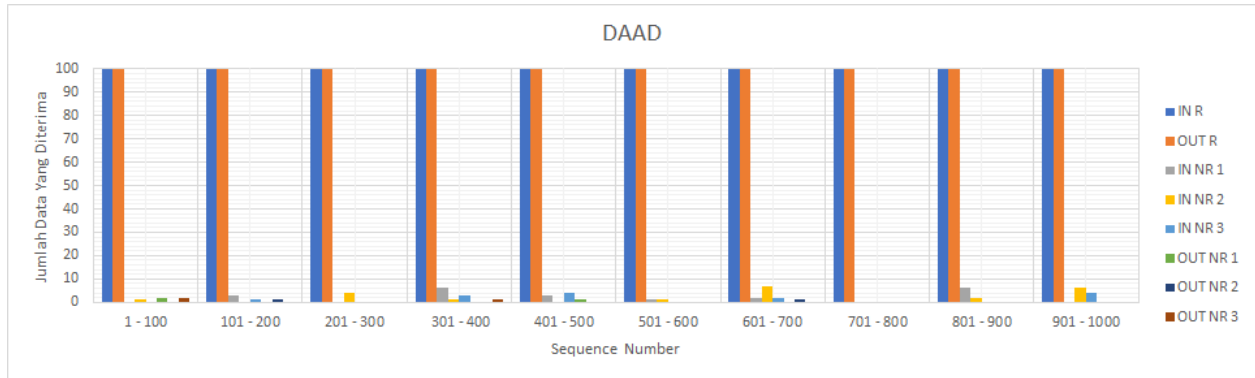
Gambar 5.34: Grafik node DAAA pada multi-hop tipe 5



Gambar 5.35: Grafik node DAAB pada multi-hop tipe 5



Gambar 5.36: Grafik node DAAC pada multi-hop tipe 5



Gambar 5.37: Grafik node DAAD pada multi-hop tipe 5

Gambar 5.34 sampai Gambar 5.37 adalah grafik hasil setiap node sensor pada topologi *multi-hop* tipe 5.

Untuk aplikasi tidak *reliable* node DAAA dalam ruangan (IN NR) data yang diterima setiap 100 *sequence number* rata-rata sebanyak 84 dan luar ruangan (OUT NR) sebanyak 76. Node DAAB dalam ruangan (IN NR) data yang diterima setiap 100 *sequence number* rata-rata sebanyak 84 dan luar ruangan (OUT NR) sebanyak 77. Node DAAC dalam ruangan (IN NR) data yang diterima setiap 100 *sequence number* rata-rata sebanyak 85 dan luar ruangan (OUT NR) sebanyak 80. Node DAAD dalam ruangan (IN NR) data yang diterima setiap 100 *sequence number* rata-rata sebanyak 1 sedangkan luar ruangan (OUT NR) sebanyak 0. Dari node DAAD sedikit data yang berhasil diterima oleh *base station* karena terjadi kepadatan jaringan. Jaringan yang padat menyebabkan data dari DAAD tidak terkirim.

Sedangkan pada aplikasi transfer data *reliable* setiap skenario, semua node sensor tidak mendapatkan *loss*.

5.2.3 Kesimpulan Hasil Eksperimen

Dari setiap hasil eksperimen, kesimpulan yang dapat diambil oleh penulis yaitu dengan aplikasi transfer data *reliable* yang telah dibuat dapat mengirim data tanpa ada data yang *loss* hingga sampai ke *base station*.

Aplikasi transfer data yang *reliable* ini memiliki kelemahan sebagai konsekuensi yang harus diterima. Untuk mendapatkan 1000 data secara utuh dari setiap node sensor pada satu jaringan memerlukan waktu yang lebih lama dibandingkan aplikasi transfer data biasa. Hal ini disebabkan karena aplikasi ini menggunakan mekanisme ACK dan *timer*. *Timer* yang terjadi dapat lebih dari 1 kali yang membuat pengiriman data berikutnya menjadi tertunda. Tabel 5.1 sampai Tabel 5.3 perbandingan waktu yang diperlukan setiap node sensor mendapatkan data. SR adalah *single-hop reliable*, SNR adalah *single-hop* tidak *reliable*, MH1R adalah *multi-hop* tipe 1 *reliable*, MH1NR adalah *multi-hop* tipe 1 tidak *reliable* dan seterusnya. "IN" adalah dalam ruangan dan "OUT" adalah ruang terbuka.

Tabel 5.1: Perbandingan waktu yang diperlukan node sensor mengumpulkan data (Single-Hop dan Multi-Hop tipe 1)

Node	Waktu yang dibutuhkan (Menit)							
	SR IN	SR OUT	SNR IN	SNR OUT	MH1R IN	MH1R OUT	MH1NR IN	MH1NR OUT
A	19.04	8.55	5.16	5.31	12.02	17.26	7.02	6.44
B	19.07	6.43	5.16	5.32	27.05	19.10	8.15	8.19
C	19.1	13.13	5.16	5.33	36.36	24.26	8.53	8.58
D	19.13	9.10	5.16	5.33	44.09	28.55	8.48	8.50

Tabel 5.2: Perbandingan waktu yang diperlukan node sensor mengumpulkan data (Multi-Hop tipe 2 dan Multi-Hop tipe 3)

Node	Waktu yang dibutuhkan (Menit)							
	MH2R IN	MH2R OUT	MH2NR IN	MH2NR OUT	MH3R IN	MH3R OUT	MH3NR IN	MH3NR OUT
A	8.40	8.47	6.46	6.54	11.24	7.17	4.04	3.32
B	13.30	14.32	6.46	4.49	14.54	19.19	9.15	8.48
C	8.38	8.09	6.46	6.49	21.06	19.19	9.11	8.47
D	14.54	14.56	6.43	4.47	22.28	19.19	9.09	6.55

Tabel 5.3: Perbandingan waktu yang diperlukan node sensor mengumpulkan data (Multi-Hop tipe 4 dan Multi-Hop tipe 5)

Node	Waktu yang dibutuhkan (Menit)							
	MH4R IN	MH4R OUT	MH4NR IN	MH4NR OUT	MH5R IN	MH5R OUT	MH5NR IN	MH5NR OUT
A	17.54	7.41	5.57	7.56	9.25	8.36	7.22	8.26
B	19.34	11.20	9.24	12.29	9.13	9.59	7.04	8.26
C	26.13	28.59	9.14	12.35	9.45	9.52	8.43	11.07
D	26.09	29.26	9	12.26	20.12	22.24	16.28	12.07

Dari Tabel 5.1 sampai Tabel 5.3 dapat diambil kesimpulan bahwa aplikasi transfer data yang *reliable* (R) memerlukan waktu yang lebih lama dibandingkan dengan aplikasi transfer data yang tidak *reliable* (NR).

Aplikasi transfer data yang *reliable* ini juga sangat boros dalam menggunakan sumber daya atau energi. Hal ini diakibatkan node sensor harus melakukan pengiriman ulang data yang *loss* berkali-kali hingga node sensor tersebut mendapatkan ACK. Penggunaan energi paling besar pada WSN adalah saat melakukan transfer data. Jadi, untuk mencapai target jumlah data yang harus dikumpulkan diperlukan sumber daya yang banyak juga.

5.3 Masalah yang Dihadapi pada Saat Implementasi

Berikut adalah beberapa masalah yang dihadapi pada saat implementasi:

1. Keterbatasan jumlah alat yang digunakan. Karena node sensor yang digunakan terbatas jadi harus digunakan bersama dengan mahasiswa lain yang menggunakan node sensor juga untuk skripsi mereka.
2. Node sensor ini sangat bergantung pada lingkungan sekitar. Jika terdapat penghalang sedikit saja maka akan memberikan hasil yang berbeda. Hal ini dikarenakan node sensor menggunakan gelombang radio sebagai media komunikasinya. Gelombang radio ini sangat mudah terintervensi oleh lingkungan sekitarnya seperti suhu, tembok, dan hujan.
3. Saat pengujian node sensor dapat mati secara tiba-tiba. Node yang mati bisa bermacam-macam seperti node sensor untuk *sensing*, node sensor perantara, maupun *base station*. Hal ini diakibatkan karena ada masalah pada *power* yang menghubungkan node sensor dengan baterai atau *base station* dengan *port* komputer.

BAB 6

KESIMPULAN DAN SARAN

6.1 Kesimpulan

Berdasarkan hasil penelitian yang dilakukan, diperoleh kesimpulan-kesimpulan sebagai berikut:

1. Aplikasi transfer data yang *reliable* berhasil dibangun dengan menggunakan mekanisme *end-to-end retransmission*. Setelah melakukan pengujian didapatkan hasil bahwa aplikasi transfer data *reliable* berhasil melakukan transfer data dengan tidak ada data yang *loss*.
2. *Wireless Sensor Network* dapat dibangun dengan memastikan *reliability* data dari setiap node sensor, tetapi terdapat beberapa konsekuensi yang harus diterima jika membangun aplikasi transfer data yang *reliable* pada WSN diantaranya:
 - (a) Waktu menerima data cenderung lebih lama dibanding aplikasi transfer data biasa. Hal ini diakibatkan aplikasi transfer data yang *reliable* menggunakan *timeout* dan ACK untuk memastikan data sampai ke *base station*.
 - (b) Penggunaan energi yang lebih banyak dibandingkan aplikasi transfer data biasa. Aplikasi transfer data *reliable* banyak sekali melakukan transfer ulang data. Pada WSN transfer data menggunakan lebih banyak energi dibandingkan memproses data.
 - (c) Karena node sensor memiliki penyimpanan yang kecil, maka perlu diperhatikan berapa banyak data yang akan disimpan pada sebuah node sensor saat menunggu ACK. Data yang terlalu banyak akan menyebabkan *error* pada aplikasi karena kehabisan ruang memori.

6.2 Saran

Berdasarkan hasil penelitian yang dilakukan, berikut adalah beberapa saran untuk pengembangan:

1. Perlu diperhatikan penggunaan energi jika menggunakan aplikasi transfer data *reliable* ini. Dapat digunakan cara seperti membatasi jumlah pengiriman ulang data yang *loss*. Namun hal ini akan mengakibatkan data yang tidak 100% *reliable*.
2. Aplikasi yang telah dibuat ini menggunakan mekanisme *end-to-end* dalam memastikan *reliability* data. Dengan adaptasi protokol RMST sebenarnya dapat juga dibangun aplikasi dengan mekanisme *hop-by-hop* dalam memastikan pengiriman data yang *reliable*.
3. Pengiriman data *reliable* yang dilakukan pada penelitian ini adalah mengirimkan satu data setiap pengiriman. Pengiriman data ini dapat dilakukan juga dengan cara mengirimkan beberapa data pada satu waktu secara bersamaan (paket data) pada setiap pengiriman.

DAFTAR REFERENSI

- [1] Stojmenovic, I. (2005) *Handbook Of Sensor Networks, Algorithms And Architectures*. A John Wiley and Sons, Ltd.
- [2] McGrath, M. J. dan Scanail, C. N. (2013) *Sensor Technologies: Healthcare, Wellness, and Environmental Application*. Apress Open.
- [3] Zheng, J. dan Jamalipour, A. (2009) *Wireless Sensor Networks A Networking Perspective*. A John Wiley and Sons, Ltd.
- [4] Yaghmaee, M.-H. dan Adjeroh, D. (2008) A reliable transport protocol for wireless sensor networks, . 09, pp. 440 – 445.
- [5] S.Prakasm, D. dan S.Lavanya (2014) Reliable techniques for data transfer in wireless sensor networks. *International Journal of Engineering and Computer Science*, **3**.
- [6] Dargie, W. dan Poellabauer, C. (2010) *Fundamentals Of Wireless Sensor Network Theory And Practice*. A John Wiley and Sons, Ltd.
- [7] Karl, H. dan Willig, A. (2005) *Protocol And Architectures For Wireless Sensor Networks*. A John Wiley and Sons, Ltd.
- [8] Farooq, M. O. dan Kunz, T. (2011) Operating systems for wireless sensor networks: A survey. *Sensors*, **11**, 5900–5930.
- [9] Matin, M. dan Islam, M. (2012) *Overview of Wireless Sensor Network, Wireless Sensor Networks - Technology and Protocols*. IntechOpen.
- [10] Kim, S., Fonseca, R., dan Culler, D. E. (2004) Reliable transfer on wireless sensor networks. *Proceedings of the First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, SECON 2004, October 4-7, 2004, Santa Clara, CA, USA*, pp. 449–459.
- [11] Sankarasubramaniam, Y., B. Akan, O., dan Akyildiz, I. (2003) Esrt: event-to-sink reliable transport in wireless sensor networks, . 01 177.
- [12] Stann, F. dan Heidemann, J. (2003) Rmst: Reliable data transport in sensor networks, . 06, pp. 102 – 112.
- [13] Tezcan, N. dan Wang, W. (2007) Art: an asymmetric and reliable transport mechanism for wireless sensor networks. *IJSNet*, **2**, 188–200.
- [14] Zhou, Y., Lyu, M., Liu, J., dan Wang, J. (2005) Port: a price-oriented reliable transport protocol for wireless sensor networks, . 12 10.
- [15] Gungor, V. dan Akan, O. (2006) Dst: Delay sensitive transport in wireless sensor networks, . 01, pp. 116 – 122.

LAMPIRAN A

KODE PROGRAM

Listing A.1: AccelerationSensor.java

```
1 package sensors;
2
3 import java.util.Arrays;
4
5 import com.virtenio.driver.device.ADXL345;
6 import com.virtenio.driver.gpio.GPIO;
7 import com.virtenio.driver.gpio.NativeGPIO;
8 import com.virtenio.driver.spi.NativeSPI;
9
10 public class AccelerationSensor {
11     private ADXL345 accelerationSensor;
12     private GPIO accelCs;
13
14     private String temp;
15
16     public void run() throws Exception {
17         accelCs = NativeGPIO.getInstance(20);
18         NativeSPI spi = NativeSPI.getInstance(0);
19         if(spi.isOpened()) {
20
21         }
22         else {
23             spi.open(ADXL345.SPI_MODE, ADXL345.SPI_BIT_ORDER, ADXL345.SPI_MAX_SPEED);
24         }
25         accelerationSensor = new ADXL345(spi, accelCs);
26         if(accelerationSensor.isOpened()) {
27
28         }
29         else {
30             accelerationSensor.open();
31             accelerationSensor.setDataFormat(ADXL345.DATA_FORMAT_RANGE_2G);
32             accelerationSensor.setDataRate(ADXL345.DATA_RATE_3200HZ);
33             accelerationSensor.setPowerControl(ADXL345.POWER_CONTROL_MEASURE);
34         }
35         short[] values = new short[3];
36         accelerationSensor.getValuesRaw(values, 0);
37         temp = "A:" + Arrays.toString(values);
38     }
39     public String getTemp(){
40         return this.temp;
41     }
42 }
```

Listing A.2: HumiditySensor.java

```
1 package sensors;
2
3 import com.virtenio.driver.device.SHT21;
4 import com.virtenio.driver.i2c.NativeI2C;
5
6 public class HumiditySensor {
7     private SHT21 sht21;
8
9     private String temp;
10
11     public void run(NativeI2C i2c) throws Exception {
12         sht21 = new SHT21(i2c);
13         if(sht21.isOpened()) {
14
15         }
16         else {
17             sht21.open();
18             sht21.setResolution(SHT21.RESOLUTION_RH12_T14);
19         }
20         sht21.startRelativeHumidityConversion();
21         Thread.sleep(100);
22         int rawRH = sht21.getRelativeHumidityRaw();
23         float rh = SHT21.convertRawRHToRHw(rawRH);
24         this.temp = "H:␣" + rh;
25     }
26
27     public String getTemp() {
28         return this.temp;
29     }
30 }
```

Listing A.3: TemperatureSensor.java

```

1 package sensors;
2
3 import com.virtenio.driver.device.ADT7410;
4 import com.virtenio.driver.i2c.NativeI2C;
5
6 public class TemperatureSensor {
7     private ADT7410 temperatureSensor;
8
9     private String temp;
10
11     public void run(NativeI2C i2c) throws Exception {
12         temperatureSensor = new ADT7410(i2c, ADT7410.ADDR_0, null, null);
13         if(temperatureSensor.isOpened()) {
14
15         }
16         else {
17             temperatureSensor.open();
18             temperatureSensor.setMode(ADT7410.CONFIG_MODE_CONTINUOUS);
19         }
20         float celsius = temperatureSensor.getTemperatureCelsius();
21         temp = "T:␣" + celsius + "␣[C]";
22     }
23
24     public String getTemp() {
25         return this.temp;
26     }
27 }

```

Listing A.4: sensing.java

```

1 import sensors.AccelerationSensor;
2 import sensors.HumiditySensor;
3 import sensors.TemperatureSensor;
4
5 import com.virtenio.driver.i2c.I2C;
6 import com.virtenio.driver.i2c.NativeI2C;
7
8 public class sensing extends Thread {
9
10     private TemperatureSensor TS = new TemperatureSensor();
11     private AccelerationSensor AS = new AccelerationSensor();
12     private HumiditySensor HS = new HumiditySensor();
13
14     private NativeI2C i2c = NativeI2C.getInstance(1);
15
16     public String sense() throws Exception{
17         if(i2c.isOpened()) {
18
19         }
20         else {
21             i2c.open(I2C.DATA_RATE_400);
22         }
23         TS.run(i2c);
24         AS.run();
25         HS.run(i2c);
26
27         String message = TS.getTemp()+"␣";
28         message += AS.getTemp()+"␣";
29         message += HS.getTemp();
30         return message;
31     }
32 }

```

Listing A.5: BS.java

```

1
2 import com.virtenio.preon32.examples.common.USARTConstants;
3 import com.virtenio.radio.ieee_802_15_4.Frame;
4 import com.virtenio.vm.Time;
5
6 import com.virtenio.driver.device.at86rf231.AT86RF231;
7 import com.virtenio.driver.device.at86rf231.AT86RF231RadioDriver;
8 import com.virtenio.misc.PropertyHelper;
9 import com.virtenio.preon32.node.Node;
10 import com.virtenio.radio.ieee_802_15_4.FrameIO;
11 import com.virtenio.radio.ieee_802_15_4.RadioDriver;
12 import com.virtenio.radio.ieee_802_15_4.RadioDriverFrameIO;
13
14 import java.io.OutputStream;
15 import java.util.HashMap;
16 import com.virtenio.driver.usart.NativeUSART;
17 import com.virtenio.driver.usart.USART;
18 import com.virtenio.driver.usart.USARTException;
19 import com.virtenio.driver.usart.USARTParams;
20 import com.virtenio.io.Console;
21
22 public class BS extends Thread {
23     private static int COMMON_PANID = PropertyHelper.getInt("radio.panid", 0xCAFF);
24     private static int[] node_list = new int[] { PropertyHelper.getInt("radio.panid", 0xABFE),
25         PropertyHelper.getInt("radio.panid", 0xDAAA), PropertyHelper.getInt("radio.panid", 0xDAAB),
26         PropertyHelper.getInt("radio.panid", 0xDAAC), PropertyHelper.getInt("radio.panid", 0xDAAD),
27         PropertyHelper.getInt("radio.panid", 0xDAAE) };
28
29     private static int ADDR_NODE3 = node_list[0]; // NODE DIRINYA (BS)
30 }

```



```

31 // private static int ADDR_NODE2[] = { PropertyHelper.getInt("radio.panid", 0xDAAA),
32 // PropertyHelper.getInt("radio.panid", 0xDAAB), PropertyHelper.getInt("radio.panid", 0xDAAC),
33 // PropertyHelper.getInt("radio.panid", 0xDAAD) };
34 //=====
35 private static int ADDR_NODE2[] = { PropertyHelper.getInt("radio.panid", 0xDAAA),PropertyHelper.getInt("radio.panid", 0xDAAC)};
36
37 private static HashMap<Integer, Integer> hmapSN = new HashMap<Integer, Integer>();
38 private static USART usart;
39 private static OutputStream out;
40 private static boolean exit;
41 private static boolean firstSense;
42
43 private static Console console;
44
45 public static void runs() {
46     try {
47         AT86RF231 t = Node.getInstance().getTransceiver();
48         t.open();
49         t.setAddressFilter(COMMON_PANID, ADDR_NODE3, ADDR_NODE3, false);
50         final RadioDriver radioDriver = new AT86RF231RadioDriver(t);
51         final FrameIO fio = new RadioDriverFrameIO(radioDriver);
52         Thread thread = new Thread() {
53             public void run() {
54                 try {
55                     sender(fio);
56                     receive(fio);
57                 } catch (Exception e) {
58                     }
59             }
60         };
61         thread.start();
62     } catch (Exception e) {
63         e.printStackTrace();
64     }
65 }
66
67 public static void sender(final FrameIO fio) throws Exception {
68     new Thread() {
69         public void run() {
70             while (true) {
71                 // console = new Console();
72                 // int temp = console.readInt("Input");
73                 int temp = 100;
74                 try {
75                     temp = usart.read();
76                 } catch (USARTException e1) {
77                     e1.printStackTrace();
78                 }
79                 if (temp == 0) {
80                     try {
81                         for (int i = 0; i < ADDR_NODE2.length; i++) {
82                             send("EXIT", ADDR_NODE2[i], fio);
83                         }
84                     } catch (Exception e1) {
85                         e1.printStackTrace();
86                     }
87                     exit = true;
88                     firstSense = false;
89                     break;
90                 } else if (temp == 1) {
91                     try {
92                         for (int i = 0; i < ADDR_NODE2.length; i++) {
93                             send("ON", ADDR_NODE2[i], fio);
94                         }
95                     } catch (Exception e1) {
96                         e1.printStackTrace();
97                     }
98                 } else if (temp == 2) {
99                     long currTime = Time.currentTimeMillis();
100                     try {
101                         for (int i = 0; i < ADDR_NODE2.length; i++) {
102                             send(("Q" + currTime), ADDR_NODE2[i], fio);
103                         }
104                     } catch (Exception e1) {
105                         e1.printStackTrace();
106                     }
107                 } else if (temp == 3) {
108                     try {
109                         for (int i = 0; i < ADDR_NODE2.length; i++) {
110                             send("WAKTU", ADDR_NODE2[i], fio);
111                         }
112                     } catch (Exception e1) {
113                         e1.printStackTrace();
114                     }
115                 } else if (temp == 4) {
116                     firstSense = true;
117                     try {
118                         for (int i = 0; i < ADDR_NODE2.length; i++) {
119                             send("DETECT", ADDR_NODE2[i], fio);
120                         }
121                     } catch (Exception e1) {
122                         e1.printStackTrace();
123                     }
124                 }
125             }
126             // receive(fio);
127         }
128     }.start();
129 }

```

```

130 }
131
132 public static void receive(final FrameIO fio) throws Exception {
133     Thread receive = new Thread() {
134         public void run() {
135             Frame frame = new Frame();
136             while (true) {
137                 try {
138                     fio.receive(frame);
139                     byte[] dg = frame.getPayload();
140                     String str = new String(dg, 0, dg.length);
141                     // System.out.println("ASD"+str);
142                     // DPT NODE YANG ONLINE
143                     if (str.charAt(str.length() - 1) == 'E') {
144                         // System.out.println(str);
145                         String msg = "#" + str + "#";
146                         try {
147                             System.out.println(msg);
148                             Thread.sleep(500);
149                             out.write(msg.getBytes(), 0, msg.length());
150                             usart.flush();
151                         } catch (Exception e) {
152                             e.printStackTrace();
153                         }
154                     }
155                     // DPT WAKTU DR SETIAP NODE
156                     else if (str.charAt(0) == 'T') {
157                         // System.out.println(str);
158                         String msg = "#" + str + "#";
159                         try {
160                             System.out.println(msg);
161                             Thread.sleep(500);
162                             out.write(msg.getBytes(), 0, msg.length());
163                             usart.flush();
164                         } catch (Exception e) {
165                             e.printStackTrace();
166                         }
167                     } else if (str.startsWith("SENSE")) {
168                         int beginNode = str.indexOf('<');
169                         int beginSN = str.indexOf('>');
170                         int endSN = str.indexOf('?');
171                         int node = Integer.parseInt(str.substring(beginNode + 1, beginSN));
172                         int sn = Integer.parseInt(str.substring(beginSN + 1, endSN));
173                         // System.out.println(node + " " + (int) frame.getSrcAddr());
174                         // System.out.println(node + " " + sn);
175                         // System.out.println(hmapSN.get(node));
176                         // Nulis sekali.. biar ga duplikat data
177                         if (hmapSN.get(node) == sn) {
178                             // System.out.println("Here!");
179                             // hmapSN.put(node, sn);
180                             // System.out.println(str);
181
182                             String msg = "#" + str + "#";
183                             try {
184                                 out.write(msg.getBytes(), 0, msg.length());
185                                 usart.flush();
186                                 Thread.sleep(50);
187                             } catch (Exception e) {
188                                 e.printStackTrace();
189                             }
190
191                             hmapSN.put(node, sn+1);
192                         }
193                         send("ACK" + node+"."+sn, frame.getSrcAddr(), fio);
194                     }
195                 } catch (Exception e) {
196                 }
197             }
198         }
199     };
200     receive.start();
201 }
202
203 public static void send(String msg, long address, final FrameIO fio) throws Exception {
204     int frameControl = Frame.TYPE_DATA | Frame.DST_ADDR_16 | Frame.INTRA_PAN | Frame.ACK_REQUEST
205     | Frame.SRC_ADDR_16;
206     final Frame testFrame = new Frame(frameControl);
207     testFrame.setDestPanId(COMMON_PANID);
208     testFrame.setDestAddr(address);
209     testFrame.setSrcAddr(ADDR_NODE3);
210     testFrame.setPayload(msg.getBytes());
211     try {
212         fio.transmit(testFrame);
213         Thread.sleep(50);
214     } catch (Exception e) {
215     }
216 }
217
218 private static USART configUSART() {
219     USARTParams params = USARTConstants.PARAMS_115200;
220     NativeUSART usart = NativeUSART.getInstance(0);
221     try {
222         usart.close();
223         usart.open(params);
224         return usart;
225     } catch (Exception e) {
226         return null;
227     }
228 }

```

Listing A.6: NS.java

[illegible]

```

77         Time.setCurrentTimeMillis(currTime);
78         if (ADDR_NODE2.length > 0) {
79             for (int i = 0; i < ADDR_NODE2.length; i++) {
80                 String message = "Q" + Time.currentTimeMillis();
81                 send(message, ADDR_NODE3, ADDR_NODE2[i], fio);
82                 Thread.sleep(50);
83             }
84         }
85     } else if (str.charAt(0) == 'T') {
86         send(str, ADDR_NODE3, ADDR_NODE1, fio);
87         System.out.println(str);
88     }
89     // Kalau dpt 'EXIT' stop dirinya dan kirim 'EXIT' ke node di bwhnya
90     else if (str.equalsIgnoreCase("EXIT")) {
91         isSensing = false;
92         exit = true;
93         if (ADDR_NODE2.length > 0) {
94             for (int i = 0; i < ADDR_NODE2.length; i++) {
95                 String message = "EXIT";
96                 send(message, ADDR_NODE3, ADDR_NODE2[i], fio);
97                 Thread.sleep(50);
98             }
99         }
100         break;
101     }
102     // Kalau dpt 'WAKTU', kirim waktu dirinya ke node diatasnya, dan kirim 'WAKTU'
103     // ke node di bwhnya.
104     else if (str.equalsIgnoreCase("WAKTU")) {
105         String msg = "Time_" + Integer.toHexString(ADDR_NODE3) + "_" + Time.currentTimeMillis();
106         send(msg, ADDR_NODE3, ADDR_NODE1, fio);
107         if (ADDR_NODE2.length > 0) {
108             for (int i = 0; i < ADDR_NODE2.length; i++) {
109                 String message = "WAKTU";
110                 send(message, ADDR_NODE3, ADDR_NODE2[i], fio);
111                 Thread.sleep(50);
112             }
113         }
114         System.out.println(msg);
115     }
116     // Kalau dpt 'ON' kirim status ke node diatasnya dan kirim "ON" ke node di
117     // bwhnya
118     else if (str.equalsIgnoreCase("ON")) {
119         String msg = "Node_" + Integer.toHexString(ADDR_NODE3) + "_ONLINE";
120         send(msg, ADDR_NODE3, ADDR_NODE1, fio);
121         System.out.println("Dirinya:_" + msg);
122         if (ADDR_NODE2.length > 0) {
123             for (int i = 0; i < ADDR_NODE2.length; i++) {
124                 send("ON", ADDR_NODE3, ADDR_NODE2[i], fio);
125                 Thread.sleep(50);
126             }
127         }
128     }
129     // Kalau dpt akhiran 'E' (status online dr node di bwhnya) terusin ke node
130     // diatasnya.
131     else if (str.charAt(str.length() - 1) == 'E') {
132         Thread.sleep(200);
133         System.out.println("Node_bwh:_" + str);
134         send(str, ADDR_NODE3, ADDR_NODE1, fio);
135     }
136     // kalau dpt 'Detect', dia set end, sensing, sn++, simpen ke myTemp, kirim ke
137     // node diatasnya
138     // kirim juga END+ ADDR_NODE3
139     // kirim 'DETECT' ke node di bwhnya
140     else if (str.equalsIgnoreCase("DETECT")) {
141         System.out.println("DETECT");
142         end = Time.currentTimeMillis() + 4000;
143         String message = "SENSE<" + ADDR_NODE3 + ">" + sn + "?" + Time.currentTimeMillis() + "_"
144             + s.sense();
145         myTemp = message;
146         Thread.sleep(50);
147         System.out.println("MY_SENSE");
148         System.out.println(myTemp);
149         System.out.println("=====");
150         sn++;
151         // Send to anak-anaknya
152         if (ADDR_NODE2.length > 0) {
153             System.out.println("Send_DETECT_ke_ADDR_NODE2[]");
154             for (int i = 0; i < ADDR_NODE2.length; i++) {
155                 send("DETECT", ADDR_NODE3, ADDR_NODE2[i], fio);
156                 Thread.sleep(50);
157             }
158         }
159         System.out.println("SEND_DATA_&_END_TO_ADDR_NODE1");
160         send(myTemp, ADDR_NODE3, ADDR_NODE1, fio);
161         Thread.sleep(50);
162         isSensing = true;
163     } else if (str.charAt(0) == 'S') {
164         System.out.println("Receive_SENSE");
165         System.out.println(str);
166         send(str, ADDR_NODE3, ADDR_NODE1, fio);
167         System.out.println("SEND_" + str);
168     } else if (str.startsWith("ACK")) {
169         int indexDot = str.indexOf(".");
170         int node = Integer.parseInt(str.substring(3, indexDot));
171         System.out.println(node);
172         if (node == ADDR_NODE3) {
173
174             int se = Integer.parseInt(str.substring(indexDot + 1));
175             if (se == sn - 1) {

```

```

176         System.out.println("RECEIVE_ACK");
177         isSensing = false;
178         end = Time.currentTimeMillis() + 4000;
179         String message = "SENSE<" + ADDR_NODE3 + ">" + sn + "?" + Time.currentTimeMillis()
180             + "." + s.sense();
181         myTemp = message;
182         Thread.sleep(50);
183         System.out.println("MY_SENSE");
184         System.out.println(myTemp);
185         System.out.println("=====");
186         sn++;
187         send(myTemp, ADDR_NODE3, ADDR_NODE1, fio);
188         Thread.sleep(50);
189         isSensing = true;
190     }
191     else {
192         send(myTemp, ADDR_NODE3, ADDR_NODE1, fio);
193
194         end = Time.currentTimeMillis() + 4000;
195     }
196     else {
197         for (int i = 0; i < ADDR_NODE2.length; i++) {
198             send(str, ADDR_NODE3, ADDR_NODE2[i], fio);
199         }
200     }
201 }
202 } catch (Exception e) {
203     e.printStackTrace();
204 }
205 }
206 }
207 }
208 thread.start();
209
210 while (thread.isAlive()) {
211     if (isSensing == true && exit == false) {
212         if (Time.currentTimeMillis() > end) {
213             System.out.println("Timeout");
214
215             send(myTemp, ADDR_NODE3, ADDR_NODE1, fio);
216             System.out.println(myTemp);
217             end = Time.currentTimeMillis() + 4000;
218         }
219     }
220 }
221 }
222
223 public static void send(String message, int source, int destination, final FrameIO fio) {
224     int frameControl = Frame.TYPE_DATA | Frame.DST_ADDR_16 | Frame.INTRA_PAN | Frame.ACK_REQUEST
225         | Frame.SRC_ADDR_16;
226     final Frame testFrame = new Frame(frameControl);
227     testFrame.setDestPanId(COMMON_PANID);
228     testFrame.setDestAddr(destination);
229     testFrame.setSrcAddr(source);
230     testFrame.setPayload(message.getBytes());
231     try {
232         fio.transmit(testFrame);
233         Thread.sleep(50);
234     } catch (Exception e) {
235     }
236 }
237
238 public static void main(String[] args) throws Exception {
239     exit = false;
240     runs();
241 }
242 }

```

Listing A.7: BS_Testing.java

```

1
2 import com.virtenio.preon32.examples.common.USARTConstants;
3 import com.virtenio.radio.ieee_802_15_4.Frame;
4 import com.virtenio.vm.Time;
5
6 import com.virtenio.driver.device.at86rf231.AT86RF231;
7 import com.virtenio.driver.device.at86rf231.AT86RF231RadioDriver;
8 import com.virtenio.misc.PropertyHelper;
9 import com.virtenio.preon32.node.Node;
10 import com.virtenio.radio.ieee_802_15_4.FrameIO;
11 import com.virtenio.radio.ieee_802_15_4.RadioDriver;
12 import com.virtenio.radio.ieee_802_15_4.RadioDriverFrameIO;
13
14 import java.io.OutputStream;
15 import com.virtenio.driver.usart.NativeUSART;
16 import com.virtenio.driver.usart.USART;
17 import com.virtenio.driver.usart.USARTException;
18 import com.virtenio.driver.usart.USARTParams;
19
20 public class BS_Testing extends Thread {
21
22     private static int COMMON_PANID = PropertyHelper.getInt("radio.panid", 0xCAFF);
23     private static int[] node_list = new int[] { PropertyHelper.getInt("radio.panid", 0xABFE),
24         PropertyHelper.getInt("radio.panid", 0xDAAA), PropertyHelper.getInt("radio.panid", 0xDAAB),
25         PropertyHelper.getInt("radio.panid", 0xDAAE), PropertyHelper.getInt("radio.panid", 0xDAAD),
26         PropertyHelper.getInt("radio.panid", 0xDAAE) };
27
28     private static int ADDR_NODE3 = node_list[0]; // NODE DIRINYA (BS)

```

```

29 // private static int ADDR_NODE2[] = { PropertyHelper.getInt("radio.panid", 0xDAAA),
30 // PropertyHelper.getInt("radio.panid", 0xDAAB), PropertyHelper.getInt("radio.panid", 0xDAAc),
31 // PropertyHelper.getInt("radio.panid", 0xDAAD) };
32 //=====
33 private static int ADDR_NODE2[] = { PropertyHelper.getInt("radio.panid", 0xDAAA),PropertyHelper.getInt("radio.panid", 0xDAAc)};
34 //=====
35
36 private static USART usart;
37 private static OutputStream out;
38 private static boolean exit;
39
40 public static void runs() {
41     try {
42         AT86RF231 t = Node.getInstance().getTransceiver();
43         t.open();
44         t.setAddressFilter(COMMON_PANID, ADDR_NODE3, ADDR_NODE3, false);
45         final RadioDriver radioDriver = new AT86RF231RadioDriver(t);
46         final FrameIO fio = new RadioDriverFrameIO(radioDriver);
47         Thread thread = new Thread() {
48             public void run() {
49                 try {
50                     receive(fio);
51                     sender(fio);
52                 } catch (Exception e) {
53                     }
54             }
55         };
56         thread.start();
57     } catch (Exception e) {
58         e.printStackTrace();
59     }
60 }
61
62 public static void sender(final FrameIO fio) throws Exception {
63     while (true) {
64         int temp = 100;
65         try {
66             temp = usart.read();
67         } catch (USARTException e1) {
68             e1.printStackTrace();
69         }
70         if (temp == 0) {
71             try {
72                 for (int i = 0; i < ADDR_NODE2.length; i++) {
73                     send("EXIT", ADDR_NODE2[i], fio);
74                 }
75             } catch (Exception e1) {
76                 e1.printStackTrace();
77             }
78             exit = true;
79             break;
80         } else if (temp == 1) {
81             try {
82                 for (int i = 0; i < ADDR_NODE2.length; i++) {
83                     send("ON", ADDR_NODE2[i], fio);
84                 }
85             } catch (Exception e1) {
86                 e1.printStackTrace();
87             }
88         } else if (temp == 2) {
89             long currTime = Time.currentTimeMillis();
90             try {
91                 for (int i = 0; i < ADDR_NODE2.length; i++) {
92                     send(("T" + currTime), ADDR_NODE2[i], fio);
93                 }
94             } catch (Exception e1) {
95                 e1.printStackTrace();
96             }
97         } else if (temp == 3) {
98             try {
99                 for (int i = 0; i < ADDR_NODE2.length; i++) {
100                     send("WAKTU", ADDR_NODE2[i], fio);
101                 }
102             } catch (Exception e1) {
103                 e1.printStackTrace();
104             }
105         } else if (temp == 4) {
106             while (exit != true) {
107                 try {
108                     for (int i = 0; i < ADDR_NODE2.length; i++) {
109                         send("DETECT", ADDR_NODE2[i], fio);
110                     }
111                 } catch (Exception e1) {
112                     e1.printStackTrace();
113                 }
114                 Thread.sleep(50);
115             }
116         }
117     }
118 }
119
120 public static void receive(final FrameIO fio) throws Exception {
121     Thread receive = new Thread() {
122         public void run() {
123             Frame frame = new Frame();
124             while (true) {
125                 try {
126                     fio.receive(frame);
127                     byte[] dg = frame.getPayload();

```

```

128         String str = new String(dg, 0, dg.length);
129         // DPT NODE YANG ONLINE
130         if (str.charAt(str.length() - 1) == 'E') {
131             String msg = "#" + str + "#";
132             try {
133                 Thread.sleep(200);
134                 out.write(msg.getBytes(), 0, msg.length());
135                 usart.flush();
136             } catch (Exception e) {
137                 e.printStackTrace();
138             }
139         }
140         // DPT WAKTU DR SETIAP NODE
141         else if (str.charAt(0) == 'T') {
142             String msg = "#" + str + "#";
143             try {
144                 out.write(msg.getBytes(), 0, msg.length());
145                 usart.flush();
146                 Thread.sleep(200);
147             } catch (Exception e) {
148                 e.printStackTrace();
149             }
150         } else if (str.charAt(0) == 'S') {
151             String msg = "#" + str + "#";
152             try {
153                 out.write(msg.getBytes(), 0, msg.length());
154                 usart.flush();
155                 Thread.sleep(200);
156             } catch (Exception e) {
157                 e.printStackTrace();
158             }
159         } catch (Exception e) {
160             e.printStackTrace();
161         }
162     }
163 }
164 receive.start();
165 }
166
167 public static void send(String msg, long address, final FrameIO fio) throws Exception {
168     int frameControl = Frame.TYPE_DATA | Frame.DST_ADDR_16 | Frame.INTRA_PAN | Frame.ACK_REQUEST
169         | Frame.SRC_ADDR_16;
170     final Frame testFrame = new Frame(frameControl);
171     testFrame.setDestPanId(COMMON_PANID);
172     testFrame.setDestAddr(address);
173     testFrame.setSrcAddr(ADDR_NODE3);
174     testFrame.setPayload(msg.getBytes());
175     try {
176         fio.transmit(testFrame);
177         Thread.sleep(50);
178     } catch (Exception e) {
179         e.printStackTrace();
180     }
181 }
182
183 private static USART configUSART() {
184     USARTParams params = USARTConstants.PARAMS_115200;
185     NativeUSART usart = NativeUSART.getInstance(0);
186     try {
187         usart.close();
188         usart.open(params);
189         return usart;
190     } catch (Exception e) {
191         return null;
192     }
193 }
194
195 private static void startUSART() {
196     usart = configUSART();
197 }
198
199 public static void main(String[] args) throws Exception {
200     try {
201         startUSART();
202         out = usart.getOutputStream();
203     } catch (Exception e) {
204         e.printStackTrace();
205     }
206     runs();
207 }

```

Listing A.8: NS_Testing.java

```

1 import com.virtenio.radio.ieee_802_15_4.Frame;
2 import com.virtenio.vm.Time;
3 import com.virtenio.misc.PropertyHelper;
4 import com.virtenio.driver.device.at86rf231.AT86RF231;
5 import com.virtenio.driver.device.at86rf231.AT86RF231RadioDriver;
6 import com.virtenio.preon32.node.Node;
7 import com.virtenio.radio.ieee_802_15_4.FrameIO;
8 import com.virtenio.radio.ieee_802_15_4.RadioDriver;
9 import com.virtenio.radio.ieee_802_15_4.RadioDriverFrameIO;
10
11 public class NS_Testing {
12     private static int COMMON_PANID = PropertyHelper.getInt("radio.panid", 0xCAFF);
13     private static int[] node_list = new int[] { PropertyHelper.getInt("radio.panid", 0xABFE),
14         PropertyHelper.getInt("radio.panid", 0xDAAA), PropertyHelper.getInt("radio.panid", 0xDAAB),
15         PropertyHelper.getInt("radio.panid", 0xDAAAC), PropertyHelper.getInt("radio.panid", 0xDAAD),

```

```

16         PropertyHelper.getInt("radio.panid", 0xDAAE) );
17
18 //private static int ADDR_NODE1 = node_list[0];
19 //private static int ADDR_NODE2[] = new int[0];
20 //private static int ADDR_NODE3 = node_list[4];
21 // =====
22 private static int ADDR_NODE1 = node_list[0];
23 private static int ADDR_NODE2[] = { PropertyHelper.getInt("radio.panid", 0xDAAB) };
24 private static int ADDR_NODE3 = node_list[1];
25
26 //private static int ADDR_NODE1 = node_list[1];
27 //private static int ADDR_NODE2[] = new int[0];
28 //private static int ADDR_NODE3 = node_list[2];
29
30 //private static int ADDR_NODE1 = node_list[0];
31 //private static int ADDR_NODE2[] = { PropertyHelper.getInt("radio.panid", 0xDAAD) };
32 //private static int ADDR_NODE3 = node_list[3];
33
34 //private static int ADDR_NODE1 = node_list[3];
35 //private static int ADDR_NODE2[] = new int[0];
36 //private static int ADDR_NODE3 = node_list[4];
37
38 private sensing s = new sensing();
39 private int sn = 1; // sequence number
40
41 private boolean exit = false;
42
43 public void runs() {
44     try {
45         AT86RF231 t = Node.getInstance().getTransceiver();
46         t.open();
47         t.setAddressFilter(COMMON_PANID, ADDR_NODE3, ADDR_NODE3, false);
48         final RadioDriver radioDriver = new AT86RF231RadioDriver(t);
49         final FrameIO fio = new RadioDriverFrameIO(radioDriver);
50         send_receive(fio);
51     } catch (Exception e) {
52         e.printStackTrace();
53     }
54 }
55
56 public void send_receive(final FrameIO fio) throws Exception {
57     Thread thread = new Thread() {
58         public void run() {
59             Frame frame = new Frame();
60             while (true) {
61                 try {
62                     fio.receive(frame);
63                     byte[] dg = frame.getPayload();
64                     String str = new String(dg, 0, dg.length);
65                     if (str.charAt(0) == 'T') {
66                         String tm = str.substring(1);
67                         long currTime = Long.parseLong(tm);
68                         Time.setCurrentTimeMillis(currTime);
69                         if (ADDR_NODE2.length > 0) {
70                             for (int i = 0; i < ADDR_NODE2.length; i++) {
71                                 String message = "T" + Time.currentTimeMillis();
72                                 send(message, ADDR_NODE3, ADDR_NODE2[i], fio);
73                                 Thread.sleep(50);
74                             }
75                         }
76                     } else if (str.equalsIgnoreCase("EXIT")) {
77                         exit = true;
78                         if (ADDR_NODE2.length > 0) {
79                             for (int i = 0; i < ADDR_NODE2.length; i++) {
80                                 String message = "EXIT";
81                                 send(message, ADDR_NODE3, ADDR_NODE2[i], fio);
82                                 Thread.sleep(50);
83                             }
84                         }
85                         break;
86                     } else if (str.equalsIgnoreCase("WAKTU")) {
87                         String msg = "Time_" + Integer.toHexString(ADDR_NODE3) + "_" + Time.currentTimeMillis();
88                         send(msg, ADDR_NODE3, ADDR_NODE1, fio);
89                         if (ADDR_NODE2.length > 0) {
90                             for (int i = 0; i < ADDR_NODE2.length; i++) {
91                                 String message = "WAKTU";
92                                 send(message, ADDR_NODE3, ADDR_NODE2[i], fio);
93                                 Thread.sleep(50);
94                             }
95                         }
96                         System.out.println(msg);
97                     } else if (str.equalsIgnoreCase("ON")) {
98                         String msg = "Node_" + Integer.toHexString(ADDR_NODE3) + "_ONLINE";
99                         System.out.println("My_Node");
100                         System.out.println(msg);
101                         send(msg, ADDR_NODE3, ADDR_NODE1, fio);
102                         if (ADDR_NODE2.length > 0) {
103                             for (int i = 0; i < ADDR_NODE2.length; i++) {
104                                 send("ON", ADDR_NODE3, ADDR_NODE2[i], fio);
105                                 Thread.sleep(50);
106                             }
107                         }
108                     } else if (str.charAt(str.length() - 1) == 'E') {
109                         System.out.println("Node_bawah");
110                         System.out.println(str);
111                         send(str, ADDR_NODE3, ADDR_NODE1, fio);
112                     } else if (str.equalsIgnoreCase("DETECT")) {
113                         System.out.println("DETECT");
114                     }

```



```

115         Thread.sleep(200);
116         String message = "SENSE<" + ADDR_NODE3 + ">" + sn + "?" + Time.currentTimeMillis() + "␣"
117             + s.sense();
118         send(message, ADDR_NODE3, ADDR_NODE1, fio);
119         System.out.println("MY_SENSE");
120         System.out.println(message);
121         System.out.println("=====");
122         sn++;
123         if (ADDR_NODE2.length > 0) {
124             for (int i = 0; i < ADDR_NODE2.length; i++) {
125                 send("DETECT", ADDR_NODE3, ADDR_NODE2[i], fio);
126                 Thread.sleep(100);
127             }
128         }
129         } else if (str.charAt(0) == 'S') {
130             System.out.println("Receive");
131             System.out.println(str);
132             send(str, ADDR_NODE3, ADDR_NODE1, fio);
133             Thread.sleep(100);
134         }
135     }
136     } catch (Exception e) {
137         e.printStackTrace();
138     }
139 }
140 }
141 };
142 thread.start();
143 }
144
145 public void send(String message, int source, int destination, final FrameIO fio) {
146     int frameControl = Frame.TYPE_DATA | Frame.DST_ADDR_16 | Frame.INTRA_PAN | Frame.ACK_REQUEST
147         | Frame.SRC_ADDR_16;
148     final Frame testFrame = new Frame(frameControl);
149     testFrame.setDestPanId(COMMON_PANID);
150     testFrame.setDestAddr(destination);
151     testFrame.setSrcAddr(source);
152     testFrame.setPayload(message.getBytes());
153     try {
154         fio.transmit(testFrame);
155         Thread.sleep(50);
156     } catch (Exception e) {
157     }
158 }
159
160 public static void main(String[] args) throws Exception {
161     new NS_Testing().runs();
162 }
163
164 }

```

Listing A.9: Handler.java

```

1 import com.virtenio.commander.io.*;
2 import com.virtenio.commander.toolsets.preon32.Preon32Helper;
3
4 import java.io.BufferedReader;
5 import java.io.BufferedWriter;
6 import java.io.File;
7 import java.io.FileWriter;
8 import java.util.Arrays;
9 import java.util.Scanner;
10
11 import org.apache.tools.ant.*;
12 import java.text.SimpleDateFormat;
13 import java.util.Date;
14 import java.util.InputMismatchException;
15
16 public class Handler {
17
18     private Scanner scanner;
19     private volatile static boolean exit = false;
20     private BufferedWriter writer;
21     private static boolean isSensing;
22
23     private static DefaultLogger getConsoleLogger() {
24         DefaultLogger consoleLogger = new DefaultLogger();
25         consoleLogger.setErrorPrintStream(System.err);
26         consoleLogger.setOutputPrintStream(System.out);
27         consoleLogger.setMessageOutputLevel(Project.MSG_INFO);
28
29         return consoleLogger;
30     }
31
32     private void time_synchronize() throws Exception {
33         DefaultLogger consoleLogger = getConsoleLogger();
34         File buildFile = new File("E:\\Sandbox\\build.xml");
35         Project antProject = new Project();
36         antProject.setUserProperty("ant.file", buildFile.getAbsolutePath());
37         antProject.addBuildListener(consoleLogger);
38
39         try {
40             antProject.fireBuildStarted();
41             antProject.init();
42             ProjectHelper helper = ProjectHelper.getProjectHelper();
43             antProject.addReference("ant.ProjectHelper", helper);
44             helper.parse(antProject, buildFile);
45             String target = "cmd.time.synchronize";

```

```

1 antProject.executeTarget(target);
2 antProject.fireBuildFinished(null);
3 } catch (BuildException e) {
4     e.printStackTrace();
5 }
6
7 public void init() throws Exception {
8     try {
9         Preon32Helper nodeHelper = new Preon32Helper("COM8", 115200);
10        DataConnection conn = nodeHelper.runModule("basestation");
11        BufferedInputStream in = new BufferedInputStream(conn.getInputStream());
12
13        int choiceentry = -1;
14        String s;
15        Scanner scanner = new Scanner(System.in);
16        conn.flush();
17        System.out.println("MENU");
18        System.out.println("1._Check_Online");
19        System.out.println("2._Synchronize_Time");
20        System.out.println("3._Get_Time");
21        System.out.println("4._Start_Sensing!");
22        System.out.println("0._Exit");
23        System.out.println("Choice:_");
24        do {
25            try {
26                choiceentry = scanner.nextInt();
27                conn.write(choiceentry);
28                Thread.sleep(200);
29                switch (choiceentry) {
30                    case 0: {
31                        System.out.println("Exit_Program...");
32                        exit = true;
33                        break;
34                    }
35                    case 1: {
36                        if (isSensing == false) {
37                            byte[] buffer = new byte[1024];
38                            while (in.available() > 0) {
39                                in.read(buffer);
40                                conn.flush();
41                                s = new String(buffer);
42                                String[] ss = s.split("#");
43                                for (String res : ss) {
44                                    if (res.startsWith("Node")) {
45                                        System.out.println(res);
46                                    }
47                                }
48                                Thread.sleep(1000);
49                            }
50                            System.out.println("MENU");
51                            System.out.println("1._Check_Online");
52                            System.out.println("2._Synchronize_Time");
53                            System.out.println("3._Get_Time");
54                            System.out.println("4._Start_Sensing!");
55                            System.out.println("0._Exit");
56                            System.out.println("Choice:_");
57                        } else {
58                            System.out.println("MENU");
59                            System.out.println("0._Exit");
60                            System.out.println("Choice:_");
61                        }
62                        break;
63                    }
64                    case 2: {
65                        Thread.sleep(500);
66                        if (isSensing == false) {
67                            System.out.println("Done_Synchronize");
68                            System.out.println("MENU");
69                            System.out.println("1._Check_Online");
70                            System.out.println("2._Synchronize_Time");
71                            System.out.println("3._Get_Time");
72                            System.out.println("4._Start_Sensing!");
73                            System.out.println("0._Exit");
74                            System.out.println("Choice:_");
75                        } else {
76                            System.out.println("MENU");
77                            System.out.println("0._Exit");
78                            System.out.println("Choice:_");
79                        }
80                        break;
81                    }
82                    case 3: {
83                        if (isSensing == false) {
84                            byte[] buffer = new byte[1024];
85                            while (in.available() > 0) {
86                                in.read(buffer);
87                                conn.flush();
88                                s = new String(buffer);
89                                String[] ss = s.split("#");
90                                for (String res : ss) {
91                                    if (res.startsWith("Time")) {
92                                        String[] fin = res.split("_");
93                                        System.out.println(res);
94                                        long time = Long.parseLong(fin[2]);
95                                        System.out.println(fin[0] + "_" + fin[1] + "_" + String.format(time));
96                                    }
97                                }
98                            }
99                        }
100                    }
101                }
102            } catch (Exception e) {
103                e.printStackTrace();
104            }
105        } while (exit == false);
106    }
107 }
108
109 //
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144

```

```

145         Thread.sleep(1000);
146     }
147     System.out.println("MENU");
148     System.out.println("1._Check_Online");
149     System.out.println("2._Synchronize_Time");
150     System.out.println("3._Get_Time");
151     System.out.println("4._Start_Sensing_!");
152     System.out.println("0._Exit");
153     System.out.println("Choice:_");
154 } else {
155     System.out.println("MENU");
156     System.out.println("0._Exit");
157     System.out.println("Choice:_");
158 }
159 break;
160 }
161 case 4: {
162     if (isSensing == false) {
163         System.out.println("Sensing...");
164         String fName = System.currentTimeMillis() + "";
165         fName = "Testing_" + fName + ".txt";
166         writeToFile(fName, "Tester", in);
167         isSensing = true;
168         System.out.println("MENU");
169         System.out.println("0._Exit");
170         System.out.println("Choice:_");
171     } else {
172         System.out.println("Already_Sensing...");
173         System.out.println("MENU");
174         System.out.println("0._Exit");
175         System.out.println("Choice:_");
176     }
177     break;
178 }
179 }
180 } catch (InputMismatchException e) {
181     String input = scanner.next();
182     System.out.println("Input_salah..");
183     if (isSensing == false) {
184         System.out.println("MENU");
185         System.out.println("1._Check_Online");
186         System.out.println("2._Synchronize_Time");
187         System.out.println("3._Get_Time");
188         System.out.println("4._Start_Sensing_!");
189         System.out.println("0._Exit");
190         System.out.println("Choice:_");
191     } else {
192         System.out.println("MENU");
193         System.out.println("0._Exit");
194         System.out.println("Choice:_");
195     }
196     continue;
197 }
198 } while (choiceentry != 0);
199 } catch (Exception e) {
200 }
201 }
202
203 public String stringFormat(long val) {
204     Date date = new Date(val);
205     SimpleDateFormat df = new SimpleDateFormat("dd-MM-yyyy_HH:mm:ss.SSS");
206     String dateText = df.format(date);
207     return dateText;
208 }
209
210 public void writeToFile(String fName, String folName, BufferedInputStream in) throws Exception {
211     Thread t = new Thread() {
212         byte[] buffer = new byte[2048];
213         String s;
214         long count = 0;
215         File newFolder = new File(folName);
216
217         public void run() {
218             if (!newFolder.exists())
219                 newFolder.mkdir();
220             String path = folName + "/" + fName;
221             try {
222                 FileWriter fw = new FileWriter(path);
223                 writer = new BufferedWriter(fw);
224             } catch (Exception e) {
225                 e.printStackTrace();
226             }
227             while (!exit) {
228                 try {
229                     if (in.available() > 0) {
230                         in.read(buffer);
231                         s = new String(buffer);
232                         String[] subStr = s.split("#");
233                         for (String w : subStr) {
234                             if (w.startsWith("SENSE")) {
235                                 String temp = w.replace('<', '_');
236                                 String temp2 = temp.replace('>', '_');
237                                 String temp3 = temp2.replace('?', '_');
238                                 String[] ss = temp3.split("_");
239                                 long val = Long.parseLong(ss[3]);
240                                 String newString = ss[0] + "_" + Integer.toHexString(Integer.parseInt(ss[1])) + "_"
241                                     + ss[2] + "_" + stringFormat(val) + "_" + ss[4] + "_" + ss[5] + "_" + ss[6]
242                                     + "_" + ss[7] + "_" + ss[8] + "_" + ss[9] + "_" + ss[10] + "_" + ss[11];
243                                 writer.write(newString, 0, newString.length());

```

```

244         writer.newLine();
245         // Thread.sleep(200);
246         count++;
247         if (count == 10) {
248             writer.close();
249             FileWriter fw = new FileWriter(path, true);
250             writer = new BufferedWriter(fw);
251             count = 0;
252         }
253     }
254 }
255 // count++;
256 // if (count == 10) {
257 //     writer.close();
258 //     FileWriter fw = new FileWriter(path, true);
259 //     writer = new BufferedWriter(fw);
260 //     count = 0;
261 // }
262 } catch (Exception e) {
263 }
264 }
265 Arrays.fill(buffer, (byte) 0);
266 }
267 try {
268     writer.close();
269 } catch (Exception e) {
270     e.printStackTrace();
271 }
272 }
273 };
274 t.start();
275 }
276
277 private void context_set(String target) throws Exception {
278     DefaultLogger consoleLogger = getConsoleLogger();
279     File buildFile = new File("E:\\Sandbox\\buildUser.xml");
280     Project antProject = new Project();
281     antProject.setUserProperty("ant.file", buildFile.getAbsolutePath());
282     antProject.addBuildListener(consoleLogger);
283
284     try {
285         antProject.fireBuildStarted();
286         antProject.init();
287         ProjectHelper helper = ProjectHelper.getProjectHelper();
288         antProject.addReference("ant.ProjectHelper", helper);
289         helper.parse(antProject, buildFile);
290
291         antProject.executeTarget(target);
292         antProject.fireBuildFinished(null);
293     } catch (BuildException e) {
294         e.printStackTrace();
295     }
296 }
297
298 public static void main(String[] args) throws Exception {
299     Handler handler = new Handler();
300     isSensing = false;
301     handler.context_set("context.set.1");
302     handler.time_synchronize();
303     handler.init();
304 }
305 }

```

Listing A.10: USARTConstants.java

```

1  /*
2  * Copyright (c) 2011., Virtenio GmbH
3  * All rights reserved.
4  *
5  * Commercial software license.
6  * Only for test and evaluation purposes.
7  * Use in commercial products prohibited.
8  * No distribution without permission by Virtenio.
9  * Ask Virtenio for other type of license at info@virtenio.de
10 *
11 * Kommerzielle Softwarelizenz.
12 * Nur zum Test und Evaluierung zu verwenden.
13 * Der Einsatz in kommerziellen Produkten ist verboten.
14 * Ein Vertrieb oder eine Veröffentlichung in jeglicher Form ist nicht ohne Zustimmung von Virtenio erlaubt.
15 * Für andere Formen der Lizenz nehmen Sie bitte Kontakt mit info@virtenio.de auf.
16 */
17
18 package com.virtenio.preon32.examples.common;
19
20 import com.virtenio.driver.usart.USART;
21 import com.virtenio.driver.usart.USARTParams;
22
23 /** Beispiele einer Konfiguration der USART Schnittstelle. */
24 public class USARTConstants {
25
26     public final static int PORT_EXT = 0;
27     public final static int PORT_CP2103 = 1;
28
29     /** Definition für die Konfiguration mit 9600 Baud */
30     public final static USARTParams PARAMS_09600 = new USARTParams(9600, USART.DATA_BITS_8,
31         USART.STOP_BITS_1, USART.PARITY_NONE);
32
33     /** Definition für die Konfiguration mit 19200 Baud */

```

```
34 |
35 | public final static USARTParams PARAMS_19200 = new USARTParams(19200, USART.DATA_BITS_8,
36 |     USART.STOP_BITS_1, USART.PARITY_NONE);
37 |
38 | /** Definition für die Konfiguration mit 38400 Baud */
39 |
40 | public final static USARTParams PARAMS_38400 = new USARTParams(38400, USART.DATA_BITS_8,
41 |     USART.STOP_BITS_1, USART.PARITY_NONE);
42 |
43 | /** Definition für die Konfiguration mit 115200 Baud */
44 |
45 | public final static USARTParams PARAMS_115200 = new USARTParams(115200, USART.DATA_BITS_8,
46 |     USART.STOP_BITS_1, USART.PARITY_NONE);
47 |
48 | /** Definition für die Konfiguration mit 250000 Baud */
49 |
50 | public final static USARTParams PARAMS_250000 = new USARTParams(250000, USART.DATA_BITS_8,
51 |     USART.STOP_BITS_1, USART.PARITY_NONE);
52 |
53 | }
```


LAMPIRAN B

HASIL EKSPERIMEN SINGLE HOP

Tabel B.1: Jumlah Data Yang Diterima Node DAAA Single Hop

Node	Skenario	Jumlah Data Yang Diterima Pada <i>Sequence Number</i> Ke-										Total
		1- 100	101- 200	201- 300	301- 400	401- 500	501- 600	601- 700	701- 800	801- 900	901- 1000	
DAAA	IN R	100	100	100	100	100	100	100	100	100	100	1000
	OUT R	100	100	100	100	100	100	100	100	100	100	1000
	IN NR 1	100	100	100	100	100	100	100	100	100	100	1000
	IN NR 2	100	100	100	100	100	100	100	100	100	100	1000
	IN NR 3	100	100	100	100	100	100	100	100	100	100	1000
	OUT NR 1	100	100	100	100	100	100	100	100	100	100	1000
	OUT NR 2	100	100	100	100	100	100	100	99	100	100	999
	OUT NR 3	99	100	100	100	100	99	100	100	100	100	998

Tabel B.2: Jumlah Data Yang Diterima Pada Node DAAB Single Hop

Node	Skenario	Jumlah Data Yang Diterima Pada <i>Sequence Number</i> Ke-										Total
		1- 100	101- 200	201- 300	301- 400	401- 500	501- 600	601- 700	701- 800	801- 900	901- 1000	
DAAB	IN R	100	100	100	100	100	100	100	100	100	100	1000
	OUT R	100	100	100	100	100	100	100	100	100	100	1000
	IN NR 1	100	100	100	100	100	100	100	100	100	100	1000
	IN NR 2	100	100	100	100	100	100	100	100	100	100	1000
	IN NR 3	100	100	100	100	100	100	100	100	100	100	1000
	OUT NR 1	100	100	100	100	100	100	100	100	100	100	1000
	OUT NR 2	100	100	100	100	100	100	100	100	100	100	1000
	OUT NR 3	100	100	99	100	100	100	100	100	100	100	999

LAMPIRAN C

HASIL EKSPERIMEN MULTI HOP TIPE 1

Tabel C.1: Jumlah Data Yang Diterima Pada Node DAAA Multi Hop Tipe 1

Node	Skenario	Jumlah Data Yang Diterima Pada <i>Sequence Number</i> Ke-										Total
		1- 100	101- 200	201- 300	301- 400	401- 500	501- 600	601- 700	701- 800	801- 900	901- 1000	
DAAA	IN R	100	100	100	100	100	100	100	100	100	100	1000
	OUT R	100	100	100	100	100	100	100	100	100	100	1000
	IN NR 1	96	92	98	93	94	92	92	92	99	94	942
	IN NR 2	96	99	94	95	92	98	95	97	94	97	957
	IN NR 3	91	98	95	88	97	93	95	95	94	97	943
	OUT NR 1	95	91	90	91	97	94	64	90	94	95	901
	OUT NR 2	47	60	90	93	96	94	93	89	90	95	847
	OUT NR 3	91	94	87	91	82	90	98	95	94	91	913

Tabel C.2: Jumlah Data Yang Diterima Pada Node DAAB Multi Hop Tipe 1

Node	Skenario	Jumlah Data Yang Diterima Pada <i>Sequence Number</i> Ke-										Total
		1- 100	101- 200	201- 300	301- 400	401- 500	501- 600	601- 700	701- 800	801- 900	901- 1000	
DAAB	IN R	100	100	100	100	100	100	100	100	100	100	1000
	OUT R	100	100	100	100	100	100	100	100	100	100	1000
	IN NR 1	45	43	48	45	55	40	37	46	35	45	439
	IN NR 2	45	52	58	34	57	47	43	51	52	42	481
	IN NR 3	41	45	50	47	46	38	43	57	52	44	463
	OUT NR 1	50	50	51	39	45	28	35	50	38	35	421
	OUT NR 2	8	32	39	45	54	42	37	1	32	44	334
	OUT NR 3	29	33	31	13	26	31	43	26	31	38	301

Tabel C.3: Jumlah Data Yang Diterima Pada Node DAAC Multi Hop Tipe 1

Node	Skenario	Jumlah Data Yang Diterima Pada <i>Sequence Number</i> Ke-										Total
		1-100	101-200	201-300	301-400	401-500	501-600	601-700	701-800	801-900	901-1000	
DAAC	IN R	100	100	100	100	100	100	100	100	100	100	1000
	OUT R	100	100	100	100	100	100	100	100	100	100	1000
	IN NR 1	39	42	41	35	40	36	31	35	34	38	371
	IN NR 2	38	41	47	34	35	36	31	36	37	27	362
	IN NR 3	36	46	39	38	26	39	38	41	37	32	372
	OUT NR 1	44	41	33	43	35	33	41	41	23	32	366
	OUT NR 2	18	33	36	28	36	29	14	13	32	38	277
	OUT NR 3	31	31	26	16	28	29	41	30	37	41	310

Tabel C.4: Jumlah Data Yang Diterima Pada Node DAAD Multi Hop Tipe 1

Node	Skenario	Jumlah Data Yang Diterima Pada <i>Sequence Number</i> Ke-										Total
		1-100	101-200	201-300	301-400	401-500	501-600	601-700	701-800	801-900	901-1000	
DAAD	IN R	100	100	100	100	100	100	100	100	100	100	1000
	OUT R	100	100	100	100	100	100	100	100	100	100	1000
	IN NR 1	27	24	28	27	24	31	36	22	35	31	285
	IN NR 2	26	26	31	35	31	39	28	23	34	30	303
	IN NR 3	31	32	31	39	28	22	30	28	26	23	290
	OUT NR 1	23	24	28	25	21	26	24	24	20	29	244
	OUT NR 2	13	30	29	26	19	30	12	8	30	23	220
	OUT NR 3	24	16	26	10	14	24	18	26	19	32	209

LAMPIRAN D

HASIL EKSPERIMEN MULTI HOP TIPE 2

Tabel D.1: Jumlah Data Yang Diterima Pada Node DAAA Multi Hop Tipe 2

Node	Skenario	Jumlah Data Yang Diterima Pada <i>Sequence Number</i> Ke-										Total
		1- 100	101- 200	201- 300	301- 400	401- 500	501- 600	601- 700	701- 800	801- 900	901- 1000	
DAAA	IN R	100	100	100	100	100	100	100	100	100	100	1000
	OUT R	100	100	100	100	100	100	100	100	100	100	1000
	IN NR 1	95	93	93	92	95	95	91	92	92	96	934
	IN NR 2	90	95	94	94	99	96	93	93	94	96	944
	IN NR 3	94	93	91	93	95	97	91	93	95	96	938
	OUT NR 1	94	87	96	96	89	88	89	94	90	92	915
	OUT NR 2	95	92	92	95	90	93	97	88	88	89	919
	OUT NR 3	80	87	85	89	92	92	85	88	91	95	884

Tabel D.2: Jumlah Data Yang Diterima Pada Node DAAB Multi Hop Tipe 2

Node	Skenario	Jumlah Data Yang Diterima Pada <i>Sequence Number</i> Ke-										Total
		1- 100	101- 200	201- 300	301- 400	401- 500	501- 600	601- 700	701- 800	801- 900	901- 1000	
DAAB	IN R	100	100	100	100	100	100	100	100	100	100	1000
	OUT R	100	100	100	100	100	100	100	100	100	100	1000
	IN NR 1	63	63	64	62	68	64	69	63	69	62	647
	IN NR 2	70	70	69	60	63	58	70	59	63	62	644
	IN NR 3	73	58	66	69	67	55	66	69	61	74	658
	OUT NR 1	78	71	63	62	68	74	76	61	64	67	684
	OUT NR 2	81	73	68	65	65	73	68	69	56	69	687
	OUT NR 3	45	43	40	55	46	45	31	38	48	40	431

Tabel D.3: Jumlah Data Yang Diterima Pada Node DAAC Multi Hop Tipe 2

Node	Skenario	Jumlah Data Yang Diterima Pada <i>Sequence Number</i> Ke-										Total
		1-100	101-200	201-300	301-400	401-500	501-600	601-700	701-800	801-900	901-1000	
DAAC	IN R	100	100	100	100	100	100	100	100	100	100	1000
	OUT R	100	100	100	100	100	100	100	100	100	100	1000
	IN NR 1	95	97	95	96	97	98	94	94	94	92	952
	IN NR 2	95	92	94	97	96	93	94	95	94	93	943
	IN NR 3	92	92	99	91	95	91	97	93	96	95	941
	OUT NR 1	98	91	91	94	90	92	90	97	98	90	931
	OUT NR 2	91	93	93	88	90	87	95	92	90	87	906
	OUT NR 3	86	91	90	86	93	85	86	93	84	93	887

Tabel D.4: Jumlah Data Yang Diterima Pada Node DAAD Multi Hop Tipe 2

Node	Skenario	Jumlah Data Yang Diterima Pada <i>Sequence Number</i> Ke-										Total
		1-100	101-200	201-300	301-400	401-500	501-600	601-700	701-800	801-900	901-1000	
DAAD	IN R	100	100	100	100	100	100	100	100	100	100	1000
	OUT R	100	100	100	100	100	100	100	100	100	100	1000
	IN NR 1	64	70	65	60	59	70	71	60	73	55	647
	IN NR 2	67	75	75	65	61	81	67	66	67	59	683
	IN NR 3	65	65	61	65	65	73	61	59	59	57	630
	OUT NR 1	71	68	69	65	68	68	63	49	58	62	641
	OUT NR 2	51	63	58	60	62	59	73	59	69	72	626
	OUT NR 3	26	42	28	47	42	36	43	32	48	29	373

LAMPIRAN E

HASIL EKSPERIMEN MULTI HOP TIPE 3

Tabel E.1: Jumlah Data Yang Diterima Pada Node DAAA Multi Hop Tipe 3

Node	Skenario	Jumlah Data Yang Diterima Pada <i>Sequence Number</i> Ke-										Total
		1- 100	101- 200	201- 300	301- 400	401- 500	501- 600	601- 700	701- 800	801- 900	901- 1000	
DAAA	IN R	100	100	100	100	100	100	100	100	100	100	1000
	OUT R	100	100	100	100	100	100	100	100	100	100	1000
	IN NR 1	92	86	87	87	86	86	87	87	86	92	876
	IN NR 2	89	92	92	92	89	91	90	90	88	86	899
	IN NR 3	93	89	88	90	87	90	89	85	90	85	886
	OUT NR 1	81	82	86	86	88	89	91	85	93	86	867
	OUT NR 2	83	92	91	88	82	85	93	85	86	83	868
	OUT NR 3	89	89	88	86	85	81	85	89	91	90	873

Tabel E.2: Jumlah Data Yang Diterima Pada Node DAAB Multi Hop Tipe 3

Node	Skenario	Jumlah Data Yang Diterima Pada <i>Sequence Number</i> Ke-										Total
		1- 100	101- 200	201- 300	301- 400	401- 500	501- 600	601- 700	701- 800	801- 900	901- 1000	
DAAB	IN R	100	100	100	100	100	100	100	100	100	100	1000
	OUT R	100	100	100	100	100	100	100	100	100	100	1000
	IN NR 1	85	91	88	84	78	92	87	89	87	86	867
	IN NR 2	88	94	82	84	84	86	85	88	90	85	866
	IN NR 3	84	87	79	89	91	78	90	85	79	82	844
	OUT NR 1	87	82	82	88	88	90	88	91	93	90	879
	OUT NR 2	82	91	84	90	93	84	89	88	84	93	878
	OUT NR 3	81	92	87	90	88	92	84	86	91	88	879

Tabel E.3: Jumlah Data Yang Diterima Pada Node DAAC Multi Hop Tipe 3

Node	Skenario	Jumlah Data Yang Diterima Pada <i>Sequence Number</i> Ke-										Total
		1-100	101-200	201-300	301-400	401-500	501-600	601-700	701-800	801-900	901-1000	
DAAC	IN R	100	100	100	100	100	100	100	100	100	100	1000
	OUT R	100	100	100	100	100	100	100	100	100	100	1000
	IN NR 1	65	62	57	76	69	72	65	71	61	67	665
	IN NR 2	70	74	76	69	74	78	66	64	75	71	717
	IN NR 3	63	72	67	61	63	62	72	75	70	63	668
	OUT NR 1	64	75	69	67	69	69	54	68	54	66	655
	OUT NR 2	61	58	63	66	60	74	66	68	72	65	653
	OUT NR 3	74	68	69	74	66	63	67	71	74	59	685

Tabel E.4: Jumlah Data Yang Diterima Pada Node DAAD Multi Hop Tipe 3

Node	Skenario	Jumlah Data Yang Diterima Pada <i>Sequence Number</i> Ke-										Total
		1-100	101-200	201-300	301-400	401-500	501-600	601-700	701-800	801-900	901-1000	
DAAD	IN R	100	100	100	100	100	100	100	100	100	100	1000
	OUT R	100	100	100	100	100	100	100	100	100	100	1000
	IN NR 1	47	33	36	34	30	32	29	34	41	30	346
	IN NR 2	49	42	45	32	23	42	40	53	34	42	402
	IN NR 3	47	53	46	45	47	42	43	43	41	39	446
	OUT NR 1	34	31	36	43	32	40	32	34	35	35	352
	OUT NR 2	40	30	30	35	29	32	38	29	35	36	334
	OUT NR 3	35	30	37	30	39	36	32	34	40	40	353

LAMPIRAN F

HASIL EKSPERIMEN MULTI HOP TIPE 4

Tabel F.1: Jumlah Data Yang Diterima Pada Node DAAA Multi Hop Tipe 4

Node	Skenario	Jumlah Data Yang Diterima Pada <i>Sequence Number</i> Ke-										Total
		1- 100	101- 200	201- 300	301- 400	401- 500	501- 600	601- 700	701- 800	801- 900	901- 1000	
DAAA	IN R	100	100	100	100	100	100	100	100	100	100	1000
	OUT R	100	100	100	100	100	100	100	100	100	100	1000
	IN NR 1	91	96	89	95	93	95	87	84	94	81	905
	IN NR 2	90	94	90	87	92	94	89	84	93	88	901
	IN NR 3	96	90	89	90	89	85	97	89	91	90	906
	OUT NR 1	88	90	89	97	93	94	91	90	94	91	917
	OUT NR 2	91	87	87	86	87	90	86	86	83	87	870
	OUT NR 3	65	73	85	92	84	86	82	87	90	90	834

Tabel F.2: Jumlah Data Yang Diterima Pada Node DAAB Multi Hop Tipe 4

Node	Skenario	Jumlah Data Yang Diterima Pada <i>Sequence Number</i> Ke-										Total
		1- 100	101- 200	201- 300	301- 400	401- 500	501- 600	601- 700	701- 800	801- 900	901- 1000	
DAAB	IN R	100	100	100	100	100	100	100	100	100	100	1000
	OUT R	100	100	100	100	100	100	100	100	100	100	1000
	IN NR 1	12	25	29	23	20	16	18	21	18	20	202
	IN NR 2	22	18	29	23	25	25	28	25	24	27	246
	IN NR 3	19	21	32	26	23	25	25	24	21	21	237
	OUT NR 1	23	21	21	16	26	21	13	17	22	20	200
	OUT NR 2	17	14	18	11	13	8	22	14	18	15	150
	OUT NR 3	13	22	9	19	19	21	19	19	11	16	168

Tabel F.3: Jumlah Data Yang Diterima Pada Node DAAC Multi Hop Tipe 4

Node	Skenario	Jumlah Data Yang Diterima Pada <i>Sequence Number</i> Ke-										Total
		1-100	101-200	201-300	301-400	401-500	501-600	601-700	701-800	801-900	901-1000	
DAAC	IN R	100	100	100	100	100	100	100	100	100	100	1000
	OUT R	100	100	100	100	100	100	100	100	100	100	1000
	IN NR 1	14	16	23	17	19	18	17	17	20	15	176
	IN NR 2	16	20	23	25	20	17	23	20	29	24	217
	IN NR 3	21	21	31	22	17	18	22	20	19	12	203
	OUT NR 1	23	27	22	23	15	19	23	25	13	23	213
	OUT NR 2	19	18	24	19	17	18	12	16	24	22	189
	OUT NR 3	17	21	17	20	11	18	17	19	18	13	171

Tabel F.4: Jumlah Data Yang Diterima Pada Node DAAD Multi Hop Tipe 4

Node	Skenario	Jumlah Data Yang Diterima Pada <i>Sequence Number</i> Ke-										Total
		1-100	101-200	201-300	301-400	401-500	501-600	601-700	701-800	801-900	901-1000	
DAAD	IN R	100	100	100	100	100	100	100	100	100	100	1000
	OUT R	100	100	100	100	100	100	100	100	100	100	1000
	IN NR 1	4	13	27	15	15	13	6	9	18	13	133
	IN NR 2	9	13	15	19	14	15	16	15	20	16	152
	IN NR 3	15	17	21	19	20	11	12	10	8	14	147
	OUT NR 1	12	9	15	6	14	13	17	12	8	9	115
	OUT NR 2	12	7	10	5	6	10	19	12	10	11	102
	OUT NR 3	8	11	11	14	8	15	11	9	17	14	118

LAMPIRAN G

HASIL EKSPERIMEN MULTI HOP TIPE 5

Tabel G.1: Jumlah Data Yang Diterima Pada Node DAAA Multi Hop Tipe 5

Node	Skenario	Jumlah Data Yang Diterima Pada <i>Sequence Number</i> Ke-										Total
		1-100	101-200	201-300	301-400	401-500	501-600	601-700	701-800	801-900	901-1000	
DAAA	IN R	100	100	100	100	100	100	100	100	100	100	1000
	OUT R	100	100	100	100	100	100	100	100	100	100	1000
	IN NR 1	79	90	82	85	85	86	88	84	88	86	853
	IN NR 2	88	80	89	89	81	81	81	91	88	78	846
	IN NR 3	77	86	88	82	82	80	84	85	81	78	823
	OUT NR 1	77	80	70	81	83	85	71	79	68	86	780
	OUT NR 2	77	76	78	73	70	78	77	77	83	78	767
	OUT NR 3	72	80	77	77	72	69	84	83	71	75	760

Tabel G.2: Jumlah Data Yang Diterima Pada Node DAAB Multi Hop Tipe 5

Node	Skenario	Jumlah Data Yang Diterima Pada <i>Sequence Number</i> Ke-										Total
		1-100	101-200	201-300	301-400	401-500	501-600	601-700	701-800	801-900	901-1000	
DAAB	IN R	100	100	100	100	100	100	100	100	100	100	1000
	OUT R	100	100	100	100	100	100	100	100	100	100	1000
	IN NR 1	84	84	90	90	81	84	82	80	78	87	840
	IN NR 2	86	85	90	91	81	87	81	85	90	88	864
	IN NR 3	84	83	82	84	87	77	83	81	84	87	832
	OUT NR 1	82	76	76	79	74	77	74	79	77	74	768
	OUT NR 2	77	72	80	73	71	81	81	81	74	81	771
	OUT NR 3	77	77	80	79	81	77	80	79	76	72	778

Tabel G.3: Jumlah Data Yang Diterima Pada Node DAAC Multi Hop Tipe 5

Node	Skenario	Jumlah Data Yang Diterima Pada <i>Sequence Number</i> Ke-										Total
		1-100	101-200	201-300	301-400	401-500	501-600	601-700	701-800	801-900	901-1000	
DAAC	IN R	100	100	100	100	100	100	100	100	100	100	1000
	OUT R	100	100	100	100	100	100	100	100	100	100	1000
	IN NR 1	90	93	85	81	84	83	79	86	90	89	860
	IN NR 2	88	81	86	83	90	90	88	86	80	80	852
	IN NR 3	86	86	93	85	90	81	82	87	90	87	867
	OUT NR 1	83	83	88	82	76	85	84	79	82	82	824
	OUT NR 2	83	86	78	69	77	73	89	84	83	82	804
	OUT NR 3	78	85	75	81	71	78	86	82	80	77	793

Tabel G.4: Jumlah Data Yang Diterima Pada Node DAAD Multi Hop Tipe 5

Node	Skenario	Jumlah Data Yang Diterima Pada <i>Sequence Number</i> Ke-										Total
		1-100	101-200	201-300	301-400	401-500	501-600	601-700	701-800	801-900	901-1000	
DAAD	IN R	100	100	100	100	100	100	100	100	100	100	1000
	OUT R	100	100	100	100	100	100	100	100	100	100	1000
	IN NR 1	0	3	0	6	3	1	2	0	6	0	21
	IN NR 2	1	0	4	1	0	1	7	0	2	6	22
	IN NR 3	0	1	0	3	4	0	2	0	0	4	14
	OUT NR 1	2	0	0	0	1	0	0	0	0	0	3
	OUT NR 2	0	1	0	0	0	0	1	0	0	0	2
	OUT NR 3	2	0	0	1	0	0	0	0	0	0	3

LAMPIRAN H

CONTOH HASIL EKSPERIMEN YANG DISIMPAN PADA FILE TEXT

SENSE daaa 1 24-04-2019 06:55:36.336 T: 30.82559967041015 [C]; A: [-36, 62, 200]; H: 68.737548828125
SENSE daaa 2 24-04-2019 06:55:36.688 T: 30.76319885253906 [C]; A: [-28, 60, 190]; H: 68.737548828125
SENSE daaa 3 24-04-2019 06:55:37.048 T: 30.82559967041015 [C]; A: [-28, 48, 196]; H: 68.76043701171875
SENSE daaa 4 24-04-2019 06:55:37.403 T: 30.82559967041015 [C]; A: [-26, 56, 190]; H: 68.737548828125
SENSE daac 1 24-04-2019 06:55:36.784 T: 28.07999992370605 [C]; A: [0, 0, 0]; H: 74.5816650390625
SENSE daaa 5 24-04-2019 06:55:37.838 T: 30.76319885253906 [C]; A: [-28, 56, 192]; H: 68.737548828125
SENSE daad 1 24-04-2019 06:55:37.025 T: 28.39199829101562 [C]; A: [0, 0, 0]; H: 72.88031005859375
SENSE daab 3 24-04-2019 06:55:37.395 T: 29.82719993591308 [C]; A: [-20, -58, 212]; H: 70.934814453125
SENSE daaa 6 24-04-2019 06:55:38.319 T: 30.76319885253906 [C]; A: [-26, 54, 194]; H: 68.737548828125
SENSE daab 4 24-04-2019 06:55:37.857 T: 29.82719993591308 [C]; A: [-16, -60, 214]; H: 70.934814453125
SENSE daaa 7 24-04-2019 06:55:38.744 T: 30.76319885253906 [C]; A: [-34, 62, 188]; H: 68.737548828125
SENSE daac 3 24-04-2019 06:55:37.674 T: 28.07999992370605 [C]; A: [-32, 0, 238]; H: 74.5816650390625
SENSE daaa 8 24-04-2019 06:55:39.174 T: 30.82559967041015 [C]; A: [-30, 60, 196]; H: 68.737548828125
SENSE daad 4 24-04-2019 06:55:38.371 T: 28.39199829101562 [C]; A: [152, 58, 198]; H: 72.88031005859375
SENSE daaa 9 24-04-2019 06:55:39.574 T: 30.82559967041015 [C]; A: [-30, 60, 192]; H: 68.737548828125
SENSE daaa 10 24-04-2019 06:55:39.939 T: 30.82559967041015 [C]; A: [-32, 56, 192]; H: 68.737548828125
SENSE daaa 11 24-04-2019 06:55:40.299 T: 30.82559967041015 [C]; A: [-28, 54, 198]; H: 68.737548828125
SENSE daab 5 24-04-2019 06:55:38.990 T: 29.82719993591308 [C]; A: [-12, -64, 218]; H: 70.89666748046875
SENSE daaa 12 24-04-2019 06:55:40.744 T: 30.82559967041015 [C]; A: [-28, 56, 174]; H: 68.737548828125
SENSE daaa 13 24-04-2019 06:55:41.099 T: 30.82559967041015 [C]; A: [-28, 58, 196]; H: 68.737548828125
SENSE daad 5 24-04-2019 06:55:39.500 T: 28.32959938049316 [C]; A: [156, 56, 192]; H: 72.88031005859375
SENSE daaa 14 24-04-2019 06:55:41.504 T: 30.76319885253906 [C]; A: [-30, 56, 190]; H: 68.737548828125
SENSE daaa 15 24-04-2019 06:55:41.859 T: 30.82559967041015 [C]; A: [-28, 58, 196]; H: 68.737548828125
SENSE daac 6 24-04-2019 06:55:40.041 T: 28.07999992370605 [C]; A: [-28, 0, 230]; H: 74.54351806640625
SENSE daaa 16 24-04-2019 06:55:42.299 T: 30.82559967041015 [C]; A: [-28, 60, 196]; H: 68.76043701171875
SENSE daad 6 24-04-2019 06:55:40.285 T: 28.39199829101562 [C]; A: [150, 56, 196]; H: 72.88031005859375
SENSE daaa 17 24-04-2019 06:55:42.729 T: 30.82559967041015 [C]; A: [-30, 54, 194]; H: 68.737548828125
SENSE daaa 18 24-04-2019 06:55:43.044 T: 30.82559967041015 [C]; A: [-28, 54, 196]; H: 68.76043701171875
SENSE daac 7 24-04-2019 06:55:40.508 T: 28.14239883422851 [C]; A: [-26, -6, 232]; H: 74.54351806640625
SENSE daaa 19 24-04-2019 06:55:43.449 T: 30.82559967041015 [C]; A: [-28, 58, 194]; H: 68.76043701171875
SENSE daac 8 24-04-2019 06:55:40.884 T: 28.14239883422851 [C]; A: [-32, 0, 236]; H: 74.54351806640625
SENSE daad 7 24-04-2019 06:55:40.751 T: 28.39199829101562 [C]; A: [152, 50, 192]; H: 72.857421875
SENSE daab 10 24-04-2019 06:55:41.751 T: 29.82719993591308 [C]; A: [-18, -64, 212]; H: 70.86614990234375
SENSE daaa 20 24-04-2019 06:55:44.030 T: 30.82559967041015 [C]; A: [-34, 58, 194]; H: 68.76043701171875
SENSE daab 11 24-04-2019 06:55:42.117 T: 29.88959884643554 [C]; A: [-14, -58, 218]; H: 70.86614990234375
SENSE daaa 21 24-04-2019 06:55:44.459 T: 30.82559967041015 [C]; A: [-30, 58, 196]; H: 68.76043701171875
SENSE daab 12 24-04-2019 06:55:42.553 T: 29.88959884643554 [C]; A: [-18, -60, 214]; H: 70.86614990234375
SENSE daad 10 24-04-2019 06:55:42.607 T: 28.39199829101562 [C]; A: [152, 56, 200]; H: 72.826904296875
SENSE daaa 22 24-04-2019 06:55:44.924 T: 30.82559967041015 [C]; A: [-28, 58, 190]; H: 68.737548828125