TESIS – EE185401

MEREDUKSI KONSUMSI ENERGI DARI PEMBUATAN RANTAI KOMUNIKASI DALAM RANGKA MEMPERPANJANG *NETWORK LIFETIME* PADA *WIRELESS SENSOR NETWORK* (WSN)

WINDI PUSPITASARI
07111850032002

DOSEN PEMBIMBING
Eko Setijadi, ST., MT., Ph.D.
Dr. Ir. Achmad Affandi, DEA.

PROGRAM MAGISTER
BIDANG KEAHLIAN TEKNIK TELEKOMUNIKASI MULTIMEDIA
DEPARTEMEN TEKNIK ELEKTRO
FAKULTAS TEKNOLOGI ELEKTRO DAN INFORMATIKA CERDAS
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2020

TESIS – EE185401

MEREDUKSI KONSUMSI ENERGI DARI PEMBUATAN RANTAI KOMUNIKASI DALAM RANGKA MEMPERPANJANG *NETWORK LIFETIME* PADA *WIRELESS SENSOR NETWORK* (WSN)

WINDI PUSPITASARI
07111850032002

DOSEN PEMBIMBING
Eko Setijadi, ST., MT., Ph.D.
Dr. Ir. Achmad Affandi, DEA.

PROGRAM MAGISTER
BIDANG KEAHLIAN TEKNIK TELEKOMUNIKASI MULTIMEDIA
DEPARTEMEN TEKNIK ELEKTRO
FAKULTAS TEKNOLOGI ELEKTRO DAN INFORMATIKA CERDAS
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2020

LEMBAR PENGESAHAN TESIS

Tesis disusun untuk memenuhi salah satu syarat memperoleh gelar
Magister Teknik (MT)
di
Institut Teknologi Sepuluh Nopember
Oleh:

WINDI PUSPITASARI
NRP: 07111850032002

Tanggal Ujian: 7 Juli 2020
Periode Wisuda: September 2020

Disetujui oleh:
Pembimbing:

1. Eko Setijadi, ST., MT., Ph.D
   NIP: 197210012003121002 ..... 23/7/2020

2. Dr. Ir. Achmad Affandi, DEA
   NIP: 196510141990021001 .....

Penguji:

1. Prof. Dr. Ir. Gamantyo Hendrantoro, Ph.D
   NIP: 197011111993031002 .....

2. Dr. Ir. Puji Handayani, MT
   NIP: 196605101992032002 ..... 20/7/20

3. Dr. Ir. Wirawan, DEA
   NIP: 196311091989031011 .....

Kepala Departemen Teknik Elektro
Fakultas Teknologi Elektro dan Informatika Cerdas

Dedet Candra Riawan, ST., M. Eng., Ph.D
NIP: 197311192000031001

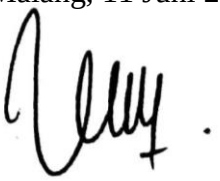
Halaman ini sengaja dikosongkan

PERNYATAAN KEASLIAN TESIS

Dengan ini saya menyatakan bahwa isi keseluruhan Tesis saya dengan judul "**MEREDUKSI KONSUMSI ENERGI DARI PEMBUATAN RANTAI KOMUNIKASI DALAM RANGKA MEMPERPANJANG *NETWORK LIFETIME* PADA *WIRELESS SENSOR NETWORK* (WSN)**" adalah benar-benar hasil karya intelektual mandiri, diselesaikan tanpa menggunakan bahan-bahan yang tidak diijinkan dan bukan merupakan karya pihak lain yang saya akui sebagai karya sendiri.

Semua referensi yang dikutip maupun dirujuk telah ditulis secara lengkap pada daftar pustaka. Apabila ternyata pernyataan ini tidak benar, saya bersedia menerima sanksi sesuai peraturan yang berlaku.

Malang, 11 Juni 2020



Windi Puspitasari
NRP. 07111850032002

Halaman ini sengaja dikosongkan

MEREDUKSI KONSUMSI ENERGI DARI PEMBUATAN RANTAI KOMUNIKASI DALAM RANGKA MEMPERPANJANG *NETWORK LIFETIME* PADA *WIRELESS SENSOR NETWORK* (WSN)

Nama mahasiswa : Windi Puspitasari
NRP : 07111850032002
Pembimbing : 1. Eko Setijadi, ST., MT., Ph.D
2. Dr. Ir. Achmad Affandi, DEA

ABSTRAK

Wireless Sensor Network (WSN) merupakan teknologi yang dapat melakukan monitoring suatu kondisi tertentu. Salah satu permasalahan yang terjadi pada WSN adalah terkendala akan sumber daya yang terbatas. Untuk itulah diperlukan *management* sumber daya yang baik agar sistem dapat berjalan dalam waktu yang cukup lama. *Power-Efficient Gathering in Sensor Information Systems* (PEGASIS) merupakan *routing* protokol yang dapat memperpanjang *Network Lifetime* (NL) dan mengurangi konsumsi energi. Namun, dengan membentuk struktur rantai yang panjang yang dapat menghasilkan kurang efisien dalam penggunaan energi. Maka pada penelitian ini diusulkan *Improved Multi-Hop Routing Protocol based on* PEGASIS (IMHRP-P) yang lebih efisien dalam penggunaan daya Watt dengan jumlah *node* sensor 35 sebesar 0,172 Watt untuk skenario pertama dan 0,171 Watt untuk skenario kedua sedangkan PEGASIS menggunakan daya 0,176 Watt pada skenario pertama serta 0,174 Watt pada skenario kedua.

Kata kunci : WSN, *routing* protokol, PEGASIS, IMHRP-P

Halaman ini sengaja dikosongkan

REDUCE ENERGY CONSUMPTION FROM MAKING OF COMMUNICATION *CHAIN*S to EXTEND *NETWORK LIFETIME* ON *WIRELESS* SENSOR NETWORK (WSN)

Author	: Windi Puspitasari
Student Identity Number	: 07111850032002
Supervisor(s)	: 1. Eko Setijadi, ST., MT., Ph.D
	2. Dr. Ir. Achmad Affandi, DEA

ABSTRACT

Wireless Sensor Network (WSN) is a technology that can monitor certain conditions. One of the problems that occurred at WSN was constrained by limited resources. For this reason, good management of resources is needed so that the system can run for a long time. Power-Efficient Gathering in Sensor Information Systems (PEGASIS) is a *routing* protocol that can extend Network Lifetime (NL) and reduce energy consumption. However, by forming a long chain structure that can produce less efficient use of energy. So in this study it is proposed that Improved Multi-*Hop Routing* Protocol based on PEGASIS (IMHRP-P) is more efficient in the use of Watt power with the number of *node* sensors 35 of 0.172 Watt for the first scenario and 0.171 Watt for the second scenario while the PEGASIS uses 0.176 Watt power at first scenario and 0.174 Watt in the second scenario.

Keywords: WSN, *routing* protocol, PEGASIS, IMHRP-P

Halaman ini sengaja dikosongkan

KATA PENGANTAR

Dengan memanjatkan Puji dan Syukur Kehadirat Allah SWT atas segala Rahmat dan Karunianya pada penulis, akhirnya penulis dapat menyelesaikan penyusunan tesis yang berjudul: "Mereduksi Konsumsi Energi dari Pembuatan Rantai Komunikasi dalam Rangka Memperpanjang *Network Lifetime* pada *Wireless Sensor Network* (WSN) "

Tesis ini disusun untuk memenuhi salah satu persyaratan memperoleh gelar Magister Teknik (M.T.) dalam bidang keahlian Telekomunikasi Multimedia pada program studi Teknik Elektro Institut Teknologi Sepuluh Nopember Surabaya.

Penulis menyadari bahwa tesis dapat diselesaikan berkat dukungan dan bantuan dari berbagai pihak, oleh karena itu penulis berterima kasih kepada semua pihak yang secara langsung maupun tidak langsung memberikan kontribusi dalam menyelesaikan Tesis ini.

Selanjutnya ucapan terima kasih penulis sampaikan kepada:

1. Allah SWT yang telah memberikan kesehatan, kekuatan dan kelancaran dalam proses pengerjaan skripsi ini.
2. Nabi Muhammad SAW, sebagai junjungan yang telah memberikan suri tauladan yang baik kepada umatnya.
3. Kedua Orang tua yang senantiasa memberikan doa restu, kasih sayang, dorongan semangat, serta bantuan moril maupun materil.
4. Bapak. Eko Setijadi, ST.,MT.,Ph.D. dan Bapak Dr. Ir. Achmad Affandi, DEA dengan segala hormat penulis mengucapkan banyak-banyak terima kasih atas bimbingan, arahan dan waktu yang telah diluangkan kepada penulis untuk berdiskusi selama menjadi dosen wali, dosen pembimbing dan perkulihaan.
5. Seluruh Dosen program Pascasarja Teknik Elektro khususnya dosen Telekomunikasi Multimedia yang telah memberikan arahan dan bimbingan untuk mendalami ilmu Telekomunikasi.

6. Balai Pengkajian Teknologi Pertanian Jawa Timur (BPTP Jatim) yang telah memberikan kesempatan untuk melakukan penelitian dan memberikan masukan pada penelitian ini.
7. Serta semua pihak yang tidak dapat penulis sebutkan satu persatu yang telah memberikan dukungan dan dorongan semangat, sehingga tesis ini dapat diselesaikan dengan baik.

Dengan keterbatasan pengalaman, ilmu maupun pustaka yang ditinjau, penulis menyadari bahwa tesis ini masih banyak kekurangan dan pengembangan lanjut agar benar benar bermanfaat. Oleh sebab itu, penulis sangat mengharapkan kritik dan saran agar tesis ini lebih sempurna serta sebagai masukan bagi penulis untuk penelitian dan penulisan karya ilmiah di masa yang akan paket datang.

Akhir kata, penulis berharap tesis ini memberikan manfaat bagi kita semua terutama untuk pengembangan ilmu pengetahuan Telekomunikasi Multimedia.

Malang, 11 Juni 2020
Penulis

DAFTAR ISI

Halaman ini sengaja dikosongkan

DAFTAR GAMBAR

DAFTAR TABEL

Halaman ini sengaja dikosongkan

BAB 1
PENDAHULUAN

1.1 Latar Belakang

Survei Penduduk Antar Sensus (SUPAS) merupakan survei yang tujuan utamanya adalah mengestimasi jumlah penduduk dan indikator demografi diantara dua waktu sensus penduduk yang telah dilakukan oleh Badan Pusat Statistik (BPS). Berdasarkan hasil proyeksi yang telah dilakukan SUPAS pada tahun 2015 dimana mencakup periode 2015-2045 menunjukkan bahwa terjadinya peningkatan jumlah penduduk. Jumlah penduduk pada tahun 2015 sebanyak 255,6 juta sedangkan pada tahun 2045 jumlah penduduk diprediksikan mencapai 319 juta[1]. Dengan meningkatknya jumlah penduduk, maka otomatis kebutuhan bahan pangan akan meningkat. Untuk menunjang kebutuhan bahan pangan penduduk yang semakin meningkat, dibutuhkan teknologi modern yang diimplementasikan dibidang pertanian salah satunya adalah *Wireless Sensor Network* (WSN). WSN merupakan sebuah sistem yang terdiri dari *transceiver* Radio Frekuensi (RF), sensor, mikrokontroller dan catu daya. Dengan kemampuan WSN yang dapat melakukan *self-organize, self-configure, self diagnosis* dan *self heal* telah menjadikan WSN pilihan tepat untuk diimplementasikan pada *smart agriculture* dan produksi pangan [2]. Pada implementasi *smart agriculture* dan produksi pangan, WSN difokuskan untuk *precision agriculture*, otomasi dan *process control automation* [2-7].

Precision agriculture merupakan sebuah teknik budidaya yang mengacu pada sistem pertanian otomatis, akurat, tertakar dan terukur. *Precision agriculture* dapat diimplementasikan dengan memberikan parameter input yang digunakan pada pertanian, seperti : intensitas air, pupuk, pestisida, kelembaban tanah dan kelembaban udara secara otomatis, akurat, tertakar dan terukur [8]. Adapun keuntungan menerapkan *precision agriculture*, diantaraya : untuk produktivitas, hasil panen semaksimal mungkin, dengan pemanfaatan atau penggunaan sumber daya seefisien, dan seefektif mungkin. Dengan mengimplementasikan WSN pada *precision agriculture*, berfungsi untuk mengoptimalkan hasil pertanian dan

melakukan monitoring kondisi di dalam *greenhouse* [9-10]. Pada implementasinya, masalah utama WSN yaitu mengenai konsumsi energi dan *delay*. Begitu pula jika WSN diimplementasikan untuk sistem monitoring kondisi di dalam *greenhouse*. Dimana pada sistem monitoring diharuskan untuk mengirimkan paket data secara teratur dan terus-menerus, maka jumlah konsumsi energi yang diperlukan akan bertambah. Oleh karena itu, dibutuhkan *routing* protokol untuk menentukan rute dan lebih efisien dalam menggunakan energi. Dimana faktor-faktor tersebut menjadi prioritas utama dalam mendesain WSN [11]. Penelitian-penelitian tentang sistem monitoring telah banyak dilakukan, salah satunya tentang monitoring kualitas tanah tanaman tomat dan terong menggunakan *node* sensor nirkabel. Pada penelitian ini, digunakanlah beberapa *node* sensor dan sensor suhu, kelembaban udara dan *Light Dependent Resistor* (LDR). Data hasil monitoring, ditampilkan pada web sehingga mempermudah pengguna (*user*) dalam melakukan monitoring kondisi didalam *greenhouse*. Dari penelitian yang telah dilakukan, menunjukkan bahwa kualitas dan produktivitas tanaman jauh lebih baik daripada tanaman tanpa dilengkapi dengan sistem monitoring dan kontrol [12]. Namun pada penelitian ini belum dilengkapi dengan *routing* protokol sehingga penggunaan energi dapat digunakan lebih efisien. Penelitian selanjutnya mengenai sistem monitoring cerdas *greenhouse* dan konsentrasi kadar CO2. Pada penelitian ini dilengkapi dengan modul kontrol, modul akuisisi kemudian paket data diterima oleh perangkat STM32. Dari STM32, paket data dirikirimkan melalui komunikasi serial RS485 kemudian paket data dapat ditampilkan melalui modul kontrol. Data yang ditampilkan meliputi : paket data *real-time*, kueri data, kueri kurva dan alarm. Dari hasil penelitian yang telah dilakukan, didapatkan hasil bahwa sistem yang diusulkan berjalan dengan stabil, biaya lebih murah dan tingkat akurasi memenuhi persyaratan *greenhouse* yang ditanami buah anggur [13]. Akan tetapi pada penelitian ini belum dilengkapi dengan teknik komunikasi jarak jauh sehingga tidak bisa dilakukan monitoring kondisi *greenhouse* dari jarak jauh. Kemudian penelitian selanjutnya tentang desain sistem monitoring *greenhouse* menggunakan *fuzzy*. *Fuzzy* berfungsi untuk melakukan kontrol terhadap suhu, kelembaban dan membuat lingkungan menjadi stabil. Dari hasil penelitian yang telah dilakukan, memunjukkan bahwa sistem yang telah dirancang memiliki kehandalan dan memberikan langkah praktis.

Selain itu, sistem kontrol memiliki tingkat akurasi kontrol ang tinggi dan memiliki karakteristik penyesuaian yang cepat[14]. Namun, pada *fuzzy* masih membutuhkan parameter-parameter pengetahuan (*knowledge*). *Knowledge* tersebut bisa berasal dari pakar maupun paket data training. Pemilihan paket data training pun bisa menjadi kesulitan tersendiri, sebab paket data yang digunakan harus merepresentasikan data yang sebenarnya. Selain itu, pemilihan data training akan sangat menentukan *knowledge* dan akurasi dari *fuzzy* yang dihasilkan.

PEGASIS (*Power Efficient Gathering in Sensor Information Systems*) merupakan protokol hierarki *chain-based* (berdasarkan rantai) dan pengembangan dari algoritma LEACH. Pada *routing* protokol PEGASIS menggunakan algoritma *Greedy* yang berfungsi untuk menentukan *node* sensor tetangga terdekatnya [15]. Hal ini dapat dapat mengurangi sejumlah energi yang digunakan dalam setiap *round*. Dimana *round* merupakan waktu yang dibutuhkan untuk *node* sensor memulai pembentukan rantai sampai data selesai dikirim menuju *Base Station* (BS). Dalam algoritma *Greedy* dilengkapi pula dengan database *node* sensor beserta posisinya [16]. Sehingga cocok untuk diimplementasikan pada studi kasus yang membutuhkan informasi tentang lokasi dari setiap *node* sensor. Namun, algoritma *Greedy* memiliki kelemahan diantaranya adalah tidak selamanya memberikan solusi yang optimal, dikarenakan pencarian *local maximum* pada setiap langkahnya, tanpa memperhatikan solusi secara keseluruhan [17]. Selain itu di dalam PEGASIS ada kemungkinan membentuk struktur rantai yang panjang sehingga dapat menghasilkan *delay* transmisi yang besar[18]. Penelitian tentang *routing* protokol PEGASIS telah dilakukan, salah satunya adalah melakukan evaluasi performa PEGASIS untuk energi efisiensi pada *Wireless Sensor Network* (WSN). Pada penelitian ini dilakukan perbandingan antara algoritma LEACH dan PEGASIS [19]. Namun, pada penelitian ini masih menggunakan *software* simulasi MATLAB yang tidak dapat merepresentasikan secara visual proses pengiriman data satu *node* sensor ke *node* sensor yang lain. Pada penelitian ini dilakukan perbandingan *routing* protokol PEGASIS dan EE-PEGASIS yang disimulasikan menggunakan *software* simulator MATLAB. Pada EE-PEGASIS menyediakan jarak pendek antara *node* sensor dan *Leader Node* (LN). Hal ini lah yang membuat nilai *delay* rendah, konsumsi daya rendah, *lifetime* bagus, skalabilitas bagus dan memberikan

performa 55% lebih baik daripada PEGASIS[20]. Namun pada penelitian ini belum disertakan informasi tentang pengimplementasian dari *routing* protokol yang diusulkan. Selain itu, pada EE-PEGASIS masih membentuk struktur rantai (*chain*) yang panjang sehingga menghasilkan *delay* transmisi yang besar. Penelitian selanjutnya mengenai pengembangan *routing* protokol PEGASIS, yaitu Distributed PEGASIS (DPEGASIS) yang melakukan perbaikan dalam pembentukan rantai pada *routing* protokol PEGASIS oleh algoritma Greedy. Pada DPEGASIS dilakukan pembentukan rantai tanpa memberikan informasi tentang jarak semua *node* sensor ke *sink*. Hal ini akan lebih efisien untuk diimplementasikan pada area yang luas [21]. Akan tetapi pada penelitian ini masih belum diimplementasikan pada studi kasus tertentu sehingga tidak adanya informasi mengenai jenis data yang dikirimkan menuju sink. Selain itu, belum dilengkapi dengan parameter pengujian *delay* sehingga tidak dapat mengetahui *delay* dari *routing* protokol yang diusulkan. Penelitian selanjutnya tentang perbandingan studi literatur dari beberapa *routing* protokol berbasis PEGASIS seperti: PEGASIS, EEPB, IEEPB, PDCH, PEG-Ant, PEGASIS-PBCA, PEGASIS-IBCA, MH-PEGASIS, Multi-*chain* PEGASIS dan Modified-PEGASIS[22]. Akan tetapi penggunaan algoritma Greedy memiliki kelemahan, yaitu kurang efektif dalam pembuatan rantai (*chain*). Hal ini dikarenakan pada algoritma Greedy tidak selamanya memberikan solusi yang optimal, dikarenakan pencarian *local maximum* pada setiap langkahnya, tanpa memperhatikan solusi secara keseluruhan. Selain itu, dibutuhkan sebuah teknik untuk lebih efisien dalam penggunaan energi.

Pada penelitian ini diusulkan sebuah algoritma *routing* protokol *Improved Multi-Hop Routing Protocol based on* PEGASIS (IMHRP-P) yang berfungsi untuk Memperpendek struktur rantai dalam pemodelan *routing* protokol PEGASIS. Sehingga paket data cepat sampai menuju ke BS. Selain itu, lebih efisien dalam penggunaan daya dan lebih optimal dalam memperpanjang *Network Lifetime* (NL) pada WSN. Pada *routing* protokol ini, *node* sensor dibagi menjadi 2 klasifikasi, yaitu *node* sensor genap dan ganjil. Untuk *node* sensor genap, akan mengirimkan paket data ke *node* sensor terjauh ke 2 menuju BS menggunakan teknik *multi-hop routing*. Sedangkan untuk *node* sensor ganjil akan mengirimkan paket data langsung menuju BS menggunakan teknik *single hop*. Dengan menggunakan

metode *routing* protokol seperti ini dapat lebih efisien dalam penggunaan energi pada pengiriman paket data *node* sensor menuju BS.

1.2 Rumusan Masalah

Permasalahan yang akan dikaji dalam laporan tesis ini adalah :

1. Bagaimana memperbaiki *routing* protokol PEGASIS dengan memperpendek rantai (*chain)* menggunakan *routing* protokol IMHRP-P ?
2. Bagaimana perbandingan energi dari pemodelan *routing* protokol PEGASIS dan IMHRP-P?
3. Bagaimana pengaruh jumlah *node* sensor dan luas area terhadap energi yang digunakan pada pemodelan *routing* protokol PEGASIS dan IMHRP-P?

1.3 Tujuan

Adapun tujuan dari penelitian ini adalah :

1. Mampu merancang *routing* protokol yang baru dimana dapat meminimalisir kekurangan yang dimiliki *routing* protokol PEGASIS.
2. Mampu membandingkan konsumsi energi *routing* protokol PEGASIS dan IMHRP-P.

1.4 Batasan Masalah

Batasan masalah dari proposal tesis ini adalah :

1. Luas area *greenhouse* yang digunakan 225 m^2, jumlah *node* sensor sebanyak 5 dan 1 BS. Selain itu dilakukan simulasi dengan luas area 100 m x 100 m dengan banyak *node* sensor yang digunakan 20, 35 dan menggunakan 1 BS.
2. *Greenhouse* hanya ditanami 1 jenis varietas tanaman, yaitu buah tomat.
3. Jumlah paket data yang dikirimkan sebesar 329 bit.
4. Diasumsikan area dalam keadaan *Line of Sight* (LOS).
5. *Node* sensor dan BS dalam keadaan tidak bergerak.
6. Melakukan simulasi pemodelan *routing* protokol menggunakan aplikasi NS3.
7. Hanya melakukan penelitian tentang efisiensi energi *routing* protokol.

8. Hanya menggunakan sensor suhu dan kelembaban udara DHT11, sensor kelembaban tanah (*soil moisture*).

1.5 Kontribusi

Pada penelitian ini diharapkan dapat memberikan kontribusi bagi dunia keilmuan dan teknologi di bidang WSN yang bertujuan untuk efisiensi energi dan memperpanjang *Network Lifetime* (NL) dalam sebuah jaringan. NL merupakan sebuah rentang waktu sejak dimulainya transmisi paket data yang pertama hingga *node* sensor terakhir mati atau kehabisan energi. Dimana dalam penelitian ini membandingkan antara *routing* protokol PEGASIS dan IMHRP-P yang akan disimulasikan melalui *software* NS3. Pada *routing* protokol PEGASIS masing-masing *node* sensor mengirimkan paket data ke *node* sensor terdekatnya sampai paket data diterima oleh *Base Station* (BS) yang membentuk struktur rantai (*chain*). Hal ini dapat mengurangi konsumsi energi yang digunakan pada masing-masing *node* sensor per *round*. *Round* merupakan waktu yang dibutuhkan untuk *node* sensor memulai pembentukan *chain* sampai paket data selesai dikirim menuju BS. Namun, PEGASIS memiliki kekurangan, yaitu terbentuknya struktur rantai yang panjang sehingga kurang efisien dalam penggunaan energi dan kurang optimal dalam memperpanjang NL dalam jaringan. Dari kekurangan yang dimiliki oleh PEGASIS, maka diusulkanlah *Improved Multi-Hop Routing Protocol based on* PEGASIS (IMHRP-P). Dimana *routing* protokol ini menggunakan teknik *single* dan *multi hop* untuk mengirimkan paket data menuju BS dengan dilakukan penjadwalan (*scheduling)* dalam mengirimkan paket data. Dengan adanya penjadwalan (*scheduling*) berfungsi untuk meminimalisir terjadinya tabrakan paket data.

BAB 2
KAJIAN PUSTAKA

2.1 Kajian Penelitian Terkait

Kajian penelitian terkait ini menjadi salah satu bahan acuan penulis dalam melakukan penelitian sehingga penulis dapat memperkaya teori yang digunakan dalam mengkaji penelitian yang dilakukan. Dari penelitian terdahulu, penulis tidak menemukan penelitian dengan judul yang sama seperti judul penulis. Namun penulis mengangkat beberapa penelitian sebagai referensi dalam mengumpulkan bahan kajian pada penelitian penulis. Untuk mengetahui deskripsi dari nama, sumber, judul dan hasil penelitian dapat dilihat pada Tabel 2.1.

Tabel 2.1 Kajian Penelitian Terkait

No	Nama	Judul	Sumber
1.	Mr. D. Shinde dan Prof. N. Siddiqui	IOT Based Environment change Monitoring & Controlling in *Greenhouse* using WSN	International Conference on Information, Communication, Engineering and Technology (ICICET) 2018
2.	S. Li	Research on Intelligent Monitoring System of *Greenhouse* Intensity and CO2 Concentration Basedon STM32	IEEE International Conference on Mechatronics and Automation (ICMA) 2018
3.	Z. Xinrong, ChangBo dan Jiang Mingxin	Design of *Wireless* Monitoring System for *Greenhouse* Environmental Parameters Based on Fuzzy Control	International Conference on Computer Technology, Electronics and Communication (ICCTEC) 2017
4.	M. R. Mufid, M. U. H. A. Rasyid dan I. Syarif	Performance Evaluation of PEGASIS Protocol for Energy Efficiency	International Electronics Symposium on Engineering Technology and Applications (IES-ETA 2018.

Tabel 2.1 Kajian Penelitian Terkait (Lanjutan)

No	Nama	Judul	Sumber
5.	V. Kumar dan Ajay Khunteta	Energy EfficientPEGASIS *Routing* Protocol for *Wireless* Sensor Networks	2nd International Conference on Micro-Electronics and Telecommunication Engineering 2018
6.	J. Kulshrestha dan M. K. Mishra	DPEGASIS: Distributed PEGASIS for *chain* construction by the *node* sensors in the network or in a zone without having global network topology information	International Conference on Multimedia, Signal Processing and Communication Technologies (IMPACT) 2017
7.	R. K. Yadav dan A. Singh	Comparative Study of PEGASIS based Protocols in *Wireless* Sensor Networks	1st India International Conference on Information Processing (IICIP) 2016

Pada Tabel 2.1 menampilkan kajian penelitian terkait, terdapat beberapa penelitian tentang sistem monitoring dan *routing* protokol yang diantaranya terdapat pada referensi [12]. Penelitian ini membuat sistem monitoring kualitas tanah tanaman tomat dan terong menggunakan *node* sensor *wireless*. Pada sistem monitoring ini menggunakan beberapa *node* sensor, diantaranya : suhu, kelembaban udara, kelembaban tanah dan LDR. Dimana sensor suhu, kelembaban udara dan kelembaban tanah berfungsi untuk melakukan monitoring terhadap lahan perkebunan dalam kondisi kering atau basah. Sedangkan untuk sensor LDR berfungsi untuk memantau intensitas cahaya lahan tersebut. Dari data yang dihasilkan oleh beberapa sensor tersebut, dikirimkan menuju server. Dimana dari server, paket data tersebut disimpan di *cloud*. Kemudian pengguna (*user*) dapat memantau kondisi didalam *greenhouse* melalui web, dimana web tersebut sudah tersambung dengan *cloud*. Dari penelitian yang telah dilakukan, menunjukkan bahwa kualitas dan produktivitas tanaman jauh lebih baik daripada tanaman tanpa dilengkapi dengan sistem monitoring dan kontrol. Namun pada penelitian ini belum dilengkapi dengan *routing* protokol sehingga penggunaan energi dapat digunakan lebih efisien.

Pada referensi [13] melakukan penelitian tentang sistem monitoring yang dilengkapi dengan modul kontrol, modul akuisisi kemudian paket data diterima

oleh perangkat STM32. Dari STM32, data dirikirimkan melalui komunikasi serial RS485 kemudian data dapat ditampilkan melalui modul kontrol. Untuk modul kontrol dan akuisisi terdiri dari sensor CO2 dan LDR. Dimana sensor CO2 mengatur konsentrasi kadar gas CO2 didalam *greenhouse*. Ketika kadar konsentrasi CO2 didalam *greenhouse* kurang, maka perangkat akan melakukan penambahan konsentrasi kadar CO2 sesuai dengan standart yang dibutuhkan oleh tanaman anggur. Sedangkan untuk sensor LDR berfungsi untuk melakukan monitoring intensitas cahaya yang masuk dalam *greenhouse* dan dapat melakukan pengaturan terhadap intensitas cahaya yang dibutuhkan oleh tanaman anggur. Ketika intensitas cahaya lebih tinggi, maka sistem akan mulai mengurangi jumlah intensitas cahaya pada *greenhouse*. Kemudian ketika intensitas cahaya lebih rendah, maka sistem akan meningkatkan jumlah intensitas cahaya dengan cara menyalakan lampu yang sudah tersedia. Paket data yang ditampilkan pada komputer meliputi : data *real-time,* kueri data, kueri kurva dan alarm. Dari hasil penelitian yang telah dilakukan, didapatkan hasil bahwa sistem yang diusulkan berjalan dengan stabil, biaya lebih murah dan tingkat akurasi memenuhi persyaratan *greenhouse* yang ditanami buah anggur. Akan tetapi pada penelitian ini belum dilengkapi dengan teknik komunikasi jarak jauh sehingga tidak bisa dilakukan monitoring kondisi *greenhouse* dari jarak jauh.

Pada referensi [14] melakukan penelitian tentang sistem monitoring yang menggunakan 4 *node* sensor dan modul *wireless* NRF905. Dimana modul *wireless* berfungsi untuk mengirimkan paket data antar *node* sensor. Pada proses pengiriman data, menggunakan teknik *multi hop routing* yang efektif digunakan dalam mengirimkan data untuk area monitoring yang luas. Selain itu, sistem monitoring menerapkan penyimpanan data secara terpusat (*central*) yang dapat menyimpan, menampilkan data dan menyediakan layanan seperti perkiraan dan permintaan. Pada sistem monitoring ini, dilengkapi dengan sistem kontrol sehingga dapat membuat pengaturan suhu dan kelembaban menggunakan *fuzzy*. Dari hasil penelitian yang telah dilakukan, menunjukkan bahwa hasil monitoring dapat menunjukkan data secara *real-time* dan menampilkan data dalam bentuk tabel sehingga mudah dalam melakukan sebuah analisa. Dengan menggunakan *fuzzy*, dapat meningkatkan frekuensi dari pengumpulan paket data, memperpanjang

periode pengiriman data dan mengurangi tingkat kehilangan data. Namun, pada *fuzzy* masih membutuhkan parameter-parameter pengetahuan (*knowledge*). *Knowledge* tersebut bisa berasal dari pakar maupun paket data training. Pemilihan paket data training pun bisa menjadi kesulitan tersendiri, sebab data yang digunakan harus merepresentasikan data yang sebenarnya. Selain itu, pemilihan data training akan sangat menentukan *knowledge* dan akurasi dari *fuzzy* yang dihasilkan.

Pada referensi [19] melakukan penelitian tentang perbandingan antara algoritma LEACH dan PEGASIS yang menggunakan *software* MATLAB. Di mana skenario penempatan *node* sensor akan dibagi menjadi 2, yaitu statis dan acak. Selain itu, posisi BS juga akan dibagi menjadi beberapa skenario. Diantaranya adalah BS ditempatkan di tengah jaringan, di luar jaringan, dan di sudut jaringan. Dengan luas area yang digunakan adalah 100 meter x 100 meter, energi awal *node* sensor 0,25 J, jumlah *node* sensor adalah 100, serta maksismum *round* sebanyak 1200. Kemudian dilakukan analisa perbandingan *routing* protokol antara PEGASIS dan Low-Energy Adaptive Clustering Hierarchy (LEACH) terkait jumlah *node* sensor hidup, jumlah *node* sensor mati, dan sisa energi. Namun, pada penelitian ini masih belum melakukan parameter pengujian tentang *delay* dimana parameter *delay* berfungsi untuk mengetahui waktu yang dibutuhkan pada saat mengirim data sampai ke tujuan. Selain itu, masih menggunakan *software* simulasi MATLAB yang tidak dapat merepresentasikan secara visual proses pengiriman satu *node* sensor ke *node* sensor yang lain.

Pada referensi [20] melakukan penelitian tentang perbandingan *routing* protokol PEGASIS dan EE-PEGASIS yang disimulasikan menggunakan *software* simulator MATLAB. Adapun luas area yang digunakan dalam simulasi ini adalah 100 m x 100 m dan jumlah *node* sensor sebanyak 100, energi awal *node* sensor sebesar 0.5 J, dengan ukuran data sebesar 2000 bit, jumlah maksimum *round* adalah 4500, serta posisi *node* sensor dalam keadaan acak. Dari hasil penelitian menunjukkan bahwa EE-PEGASIS menyediakan jarak pendek antara *node* sensor dan LN . Hal ini lah yang membuat nilai *delay* rendah, konsumsi daya rendah, *lifetime* bagus, skalabilitas bagus dan memberikan performa 55% lebih baik daripada PEGASIS[20]. Namun pada penelitian ini belum disertakan informasi tentang pengimplementasian dari *routing* protokol yang diusulkan. Selain itu, pada

EE-PEGASIS masih membentuk struktur rantai (*chain*) yang panjang sehingga menghasilkan *delay* transmisi yang besar.

Pada referensi [21] melakukan penelitian dengan *node* sensor dalam keadaan tidak bergerak. Selain itu, untuk mengetahui jarak *node* sensor dengan sink *node* sensor dapat digunakan RSSI sinyal atau modul GPS. Untuk proses pembentukan rantai (*chain*) dimulai dari *node* sensor yang berada di luar dari luas area yang sudah ditentukan dan dalam jarak tertentu, rantai (*chain*) mulai dibentuk. Selain itu, pada pembentukan rantai (*chain*) DPEGASIS tanpa memberikan informasi tentang jarak semua *node* sensor ke *sink node* sensor. Pada simulasi ini menggunakan luas area sebesar 100 m x 100 m, jumlah *node* sensor sebanyak 50, 100 dan 500 *node* sensor, energi awal 50 nJ. Akan tetapi pada penelitian ini masih belum diimplementasikan pada studi kasus tertentu sehingga tidak adanya informasi mengenai jenis data yang dikirimkan menuju *sink*. Selain itu, belum dilengkapi dengan parameter pengujian *delay* sehingga tidak dapat mengetahui performa dari *routing* protokol yang diusulkan.

Pada referensi [22] melakukan perbandingan berdasarkan studi literatur dari beberapa *routing* protokol berbasis PEGASIS seperti: PEGASIS, EEPB, IEEPB, PDCH, PEG-Ant, PEGASIS-PBCA, PEGASIS-IBCA, MH-PEGASIS, Multi-*chain* PEGASIS dan Modified-PEGASIS. Perbandingan ini dilakukan bertujuan untuk mengetahui struktur pembuatan rantai (*chain*) berbasis PEGASIS dan keseimbangan konsumsi energi dalam mengirimkan data dari satu *node* sensor ke *node* sensor yang lain. Untuk *routing* protokol IEEPB, PEG-Ant dan ModifiedPEGASIS memiliki sambungan yang panjang antara *node* sensor daripada *routing* protokol lainnya. Selain itu, dari *routing* protokol yang sudah dibandingkan terbukti dapat menghemat konsumsi energi dari masing-masing *node* sensor. Akan tetapi penggunaan algoritma Greedy memiliki kelemahan, yaitu kurang efektif dalam pembuatan rantai (*chain*). Hal ini dikarenakan pada algoritma Greedy tidak selamanya memberikan solusi yang optimal, dikarenakan pencarian *local maximum* pada setiap langkahnya, tanpa memperhatikan solusi secara keseluruhan. Selain itu, dibutuhkan sebuah teknik untuk mengurangi waktu *delay*.

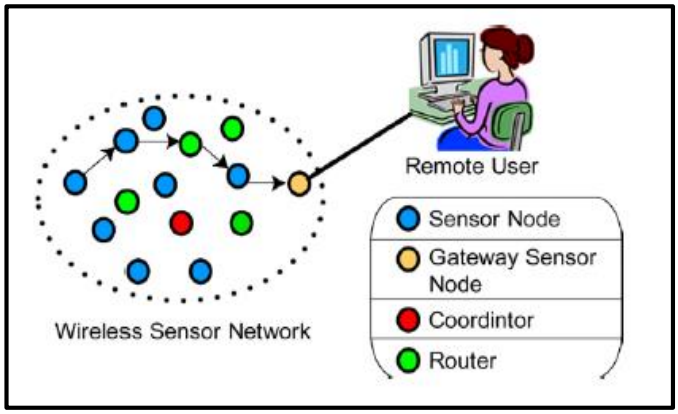
2.2 *Wireless Sensor Network* (WSN)

Terdapat 3 komponen utama WSN yang meliputi sensor, *actuator* dan *tranduser*, maka kini perlu diketahui mengenai definisi dari WSN tersebut. Terdapat beberapa versi menegani definisi dari WSN, yang diberikan oleh sejumlah peneliti dan para ahli di dunia. Beberapa buah definisi tersebut antara lain :

1. Janiver Lopez dari Computer Science Departement University of Malaga Spanyol, menyatakan bahwa WSN merupakan sebuah sistem berbasiskan jaringan *wireless*, yang melakukan pemindaian pada lingkungan nyata ke dalam bentuk paket data-paket data digital pada dunia komputer.
2. Hani Alzaid dari Information Security Queensland Institute Australia beserta dengan Murthy dan Manoj dari India, menyatakan bahwa WSN merupakan jaringan komputer terdistribusi yang memanfaatkan sejumlah *node* sensor berukuran kecil, dikembangkan dan dikonfigurasikan dalam skala besar untuk membantu pemindaian terhadap lingkungan sekitar, memanfaatkan parameter pengukuran berupa temperatur, tekanan, suhu, gerakan atau entitas lainnya yang diketahui oleh manusia.
3. Jamal, Feng Zhao, Thubaistat dan Khalpana Sharma, mereka sepakat menyatakan bahwa WSN merupakan sebuah jaringan komputer terdistribusi yang didalamnya memanfaatkan *Micro Elektro Mechanical System* (MEMS) dengan sejumlah *node* sensor berukuran kecil dan hemat daya yang mengambil paket data dari lingkungan sekitar melalui pemindaian (*sensing*) dan memiliki memori terbatas di dalamnya.

WSN merupakan salah satu teknologi yang paling menarik saat ini. WSN terdiri dari sejumlah besar perangkat murah yang terhubung melalui komunikasi nirkabel berdaya rendah. Secara umum WSN didefinisikan sebagai salah satu jenis dari jaringan *wireless* (nirkabel) terdistribusi, yang memanfaatkan teknologi *embedded system* (sistem benam) dan seperangkat *node* sensor, untuk melakukan proses sensor, monitoring, pengiriman paket data dan penyajian informasi ke pengguna, melakukan komunikasi. Sensor meliputi banyak jenis, antara lain kelembaban, radiasi, temperatur, tekanan, mekanik, gerakan, getaran, posisi dan lain-lain. Setiap jenis sensor memiliki perangkat lunak (aplikasi,sistem operasi) dan perangkat keras masing-masing, yang kemudian akan digabungkan dan dijalankan

kedalam sistem WSN. Adapun gambaran mengenai bentuk komunikasi WSN ditunjukkan dalam Gambar 2.1.



Gambar 2.1 *Wireless Sensor Network*[23]

2.3 Manajemen Daya dan Efisiensi Energi

Masalah yang dihadapi oleh pada WSN adalah penggunaan konsumsi baterai. Sebagai baterai dianggap sebagai sumber energi dari WSN, banyak teknik yang sudah dikembangkan untuk permasalahan ini. Teknik yang dipergunakan setiap lapisan layer OSI untuk mengurangi konsumsi baterai sebagai berikut:

a. *Application layer* berfungsi melakukan teknik pembagian beban setiap *node* sensor yang diatur pada *node* sensor central.
b. *Transport layer* berfungsi sebagai teknik yang digunakan dalam mengurangi transmisi paket loss.
c. *Wireless layer* berfungsi sebagai teknik yang digunakan adalah mengatur algoritma *routing*. Misalnya, teknik yang paling umum adalah *routing* yang multi-*hop* di mana setiap simpul diasumsikan router ketika paket data dikirim ke tujuan dan melewati rute terpendek. Selain itu ada teknik lain yakni agregasi paket data, pengurangan penggunaan *overhead*, dan penjadwalan waktu *sleep* pada *node* sensor.
d. Paket *data link layer* memiliki *Automatic Repeat Request* (ARQ) dan *Forward Error Correction* (FEC) yang dikenal sebagai teknik yang paling umum untuk mengurangi transmisi *overhead*. Dalam ARQ, *node* sensor *routing* yang secara

otomatis meminta pengiriman ulang paket dari *node* sensor sumber tanpa terlebih dahulu memerlukan *node* sensor penerima untuk mendeteksi bahwa terdapat paket loss.

e. MAC *layer* berfungsi sebagai teknik utama adalah penjadwalan waktu *sleep* untuk mengurangi efek *idle listening*.
f. *Phisical layer* digunakan untuk mengatur komponen *hardware*, misalnya efek kebocoran arus, sehingga diperlukan desain *hardware* yang tepat[24].

2.4 *Routing* Protokol pada WSN

Routing protokol dalam jaringan sensor nirkabel berbeda dari *routing* yang secara konvensional pada jaringan tetap dengan berbagai cara. Tidak ada infrastruktur, *link* nirkabel yang dapat diandalkan, *node* sensor bisa saja mengalami kegagalan, dan *routing* protokol harus memenuhi kriteria pemakaian energi yang efisien. Dalam perkembangannya, banyak algoritma *routing* yang dikembangkan untuk jaringan nirkabel. Semua *Routing* protokol utama yang diusulkan untuk WSN dapat dibagi menjadi tujuh kategori seperti yang ditunjukkan pada Tabel 2.2 [25].

Tabel 2.2 Kategori *Routing* Protokol[25]

No.	Kategori	Representative
1.	*Location Based*	MECN, SMECN, GAF, GEAR, Span, TBF, BVGF, GeRaF
2.	Paket *data-centric Protocols*	SPIN, *Direct*ed Diffusion, Rumor *Routing*, COUGAR, EAD, Information-*Direct Routing*, Gradient-Based *Routing*, Energy-aware *Routing*, Quorum-Based Information Dissemination, Home Agent Based Information Dissemination.
3.	*Hierarchical Protocols*	LEACH, PEGASIS, HEED, TEEN, APTEEN.
4.	*Mobility-based Protocols*	SEAD, TTDD, Joint Mobility and *Routing*, Paket data MULES, Dynamic Proxy Tree-Base Paket data Dissemination.
5.	*Multi-path Protocols*	Sensor-disjoint Multipath, Braided Multipath, N-to-1 Multipath Discovery.

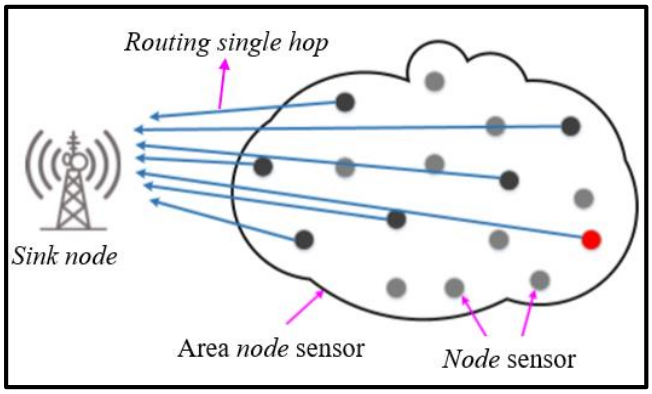
Tabel 2.2 Kategori *Routing* Protokol (Lanjutan) [25]

No.	Kategori	Representative
6.	*Heterogeneity-based Protocols*	IDSQ, CADR, CHR.
7.	*Qos-based Protocols*	SAR, SPEED, Energy-aware *routing*.

Tabel 2.2 menunjukkan kategori *routing* dari WSN. Sebuah jaringan sensor terdiri dari beberapa *node* sensor yang disebar, yang pada umumnya paket data transit melewati beberapa *node* sensor sebelum akhirnya mencapai tujuan. *Routing* merupakan langkah menentukan *path* (rute) yang harus diikuti oleh sebuah paket untuk mencapai tujuan yang diinginkan [24].

2.4.1 Metode *Routing* pada WSN

Pada WSN menggunakan metode *routing* yang dibedakan menjadi tiga jenis yaitu *single hop*, *multi hop*, dan clustering. Awal mulanya pada WSN digunakan *routing* secara *single hop*. Single *hop* ini merupakan sebuah metode sederhana karena setiap *node* sensor hanya perlu mengirimkan paket data hasil pengukuran ke pengumpul paket data yang disebut *sink node* sensor. Oleh karena itu, setiap *sink node* sensor membutuhkan energi yang berbeda-beda tergantung dengan jarak masing-masing *node* sensor tersebut ke *sink node* sensor. *Node* sensor sensor yang letaknya paling jauh dari *sink node* sensor akan membutuhkan energi yang lebih besar untuk mengirimkan paket data informasi dibandingkan dengan *node* sensor sensor yang letaknya lebih dekat dengan *sink node* sensor. Hal tersebut menyebabkan *node* sensor yang letaknya paling jauh dari *sink node* sensor akan mengalami kehabisan energi lebih cepat dibandingkan dengan *node* sensor sensor lainnya. Ilustrasi metode *routing* secara *single hop* dapat dilihat pada Gambar 2.2 dimana *node* sensor yang berwarna merah adalah *node* sensor yang memiliki jarak yang paling jauh. Sehingga akan mengalami kehabisan energi paling cepat[26]



Gambar 2.2 Metode *Routing Single Hop*[26]

2.5 PEGASIS

PEGASIS (*Power-Efficient Gathering in Sensor Information Systems*) adalah pioneer dari protokol hierarki *chain-based* dan pengembangan dari algoritma LEACH. Dalam membentuk algoritma rantai pada PEGASIS, proses pengiriman paket data dimulai dari *node* sensor target dan setiap *node* sensor diasumsikan telah mengetahui lokasi setiap *node* sensor yang ada [21].

Adapun karakteristik dari *routing* protokol PEGASIS adalah sebagai berikut [15]:

1. Proses transmisi menggunakan topologi rantai bukan *cluster*, sehingga semua *node* sensor akan membentuk rantai. Transmisi paket data dilakukan antar *node* sensor dengan jarak 1 *hop* dari setiap *node* sensor dan dikirimkan ke BS.
2. Paket data dikirimkan ke *node* sensor terdekat, sehingga energi yang digunakan untuk mengirimkan paket data lebih sedikit.
3. Hanya membutuhkan satu *node* sensor yang akan mengirimkan informasi ke BS, sehingga lebih hemat energi dibandingkan jika setiap *node* sensor langsung mengirimkan informasi ke BS.

Routing protokol PEGASIS bekerja dalam 3 fase, diantaranya adalah :

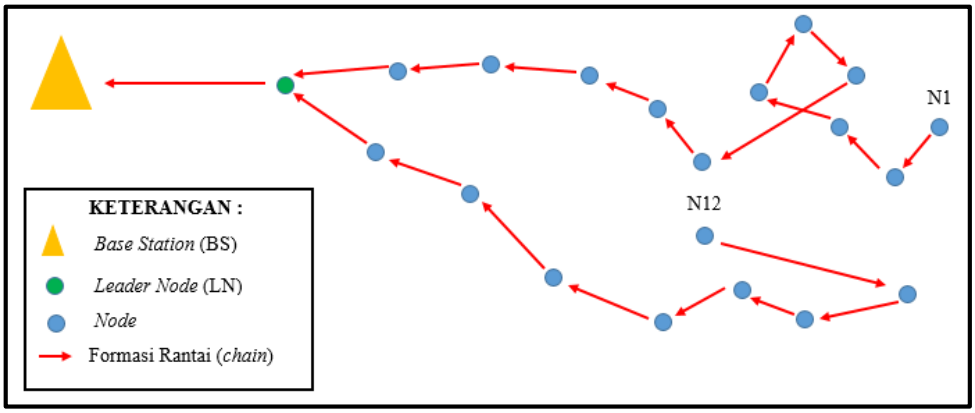
1. Pemilihan LN

Pada fase pertama dilakukan pemilihan LN. Dimana awalnya pemilihan LN dilakukan secara acak. Selama pembentukan rantai, ada kemungkinan beberapa sensor memiliki tetangga yang relatif jauh di sepanjang rantai yang memaksa *node* sensor ini untuk membuang sejumlah energi yang relatif lebih banyak di setiap

round dibandingkan dengan *node* sensor lainnya. Jadi jarak tetangga diperhitungkan saat pemilihan LN. Jika sebuah *node* sensor mati, maka rantai akan dibangun kembali dan *treshold* dapat diubah untuk menentukan LN. Sebuah LN baru dipilih di setiap *round* untuk menyeimbangkan konsumsi energi di jaringan.

2. Pembentukan Rantai (*Chain Esablishment*)

Ilustrasi WSN dengan PEGASIS tanpa rantai bercabang, dimana proses pembentukan rantai di PEGASIS adalah dengan menggunakan algoritma *Greedy* dan dimulai dari *node* sensor yang paling jauh dari BS yang ditunjukkan pada Gambar 2.3.



Gambar 2.3 Ilustrasi WSN Dengan PEGASIS Tanpa Rantai Bercabang[27]

Gambar 2.3 menunjukkan ilustrasi WSN pada algoritma *routing* protokol PEGASIS tanpa adanya rantai yang bercabang. Dalam pembentukan rantai dimulai dari *node* sensor yang memiliki jarak yang palih jauh dari BS. Selain itu, dapat dipastikan bahwa *node* sensor yang memiliki jarak yang paling jauh dari BS memiliki *node* sensor tetangga terdekat untuk meneruskan pengiriman paket data. Untuk menentukan *node* sensor tetangga terdekat, digunakanlah algoritma *Greedy*. Dalam algoritma *Greedy* memberikan solusi langkah per langkah (*step by step*). Oleh karena itu, pada setiap langkah harus dibuat keputusan yang terbaik dalam menentukan pilihan. Keputusan yang telah diambil pada suatu langkah tidak dapat diubah lagi pada langkah selanjutnya.

Dalam pembuatan rantai yang terdapat pada Gambar 2.3 terdapat 2 *node* sensor yang paling jauh dari BS, yaitu N1 dan N12. Pembentukan rantai dimulai

sesuai dengan urutan, yaitu N1. Kemudian N1 melakukan pencarian *node* sensor tetangga terdekat dengan mendeteksi sesuai dengan kekuatan sinyal. Setelah menemukan rute pembentukan *chain* yang dimulai dari N1, selanjutnya *node* sensor N12 menentukan rute pembentukan *chain* yang caranya sama dengan *node* sensor N1. Setelah pembentukan *chain* sudah terbentuk, maka proses agregasi paket data (pengumpulan paket data) bisa dilakukan.

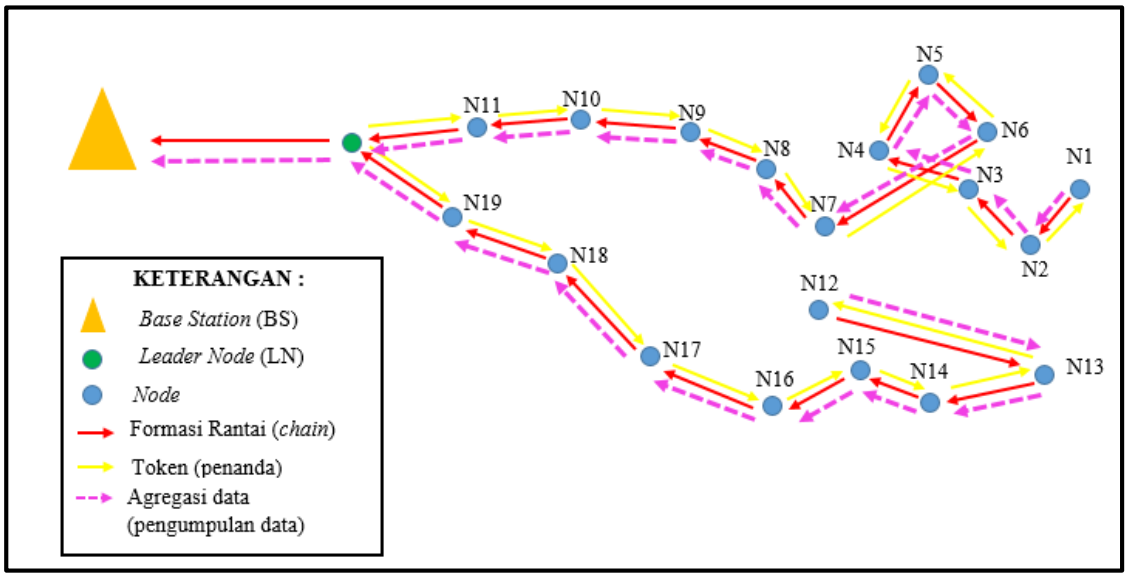
3. Agregasi Paket data (Pengumpulan Paket data)

Setelah pemilihan LN dan pembentukan *chain* selesai, maka agregasi paket data (pengumpulan paket data) dimulai. Pada agregasi paket data menggunakan pendekatan berdasarkan token (penanda) digunakan untuk mengumpulkan paket data. LN melewatkan sebuah token disepanjang *chain* hingga ke *node* sensor terakhir. Setelah menerima token, *node* sensor terakhir mengirimkan paket data dan token ke *node* sensor sensor berikutnya sepanjang rantai. Proses ini berlanjut sampai paket data sampai ke LN. Untuk *node* sensor sensor yang berada diposisi tengah, akan melakukan penggabungan paket data dengan menggabungkan paket data dari *node* sensor sensor tetangganya dengan paket data dari *node* sensor sensor itu sendiri dan membuat menjadi satu paket dengan panjang yang sama. Pada saat terakhir, LN akan mentransmisikan paket paket data ke BS. Dalam pembuatan *chain*, LN berada pada awal *chain*. Adapun ilustrasi agregrasi paket data ditunjukkan pada Gambar 2.4.

Gambar 2.4 menunjukkan ilustrasi agregasi paket data pada algoritma *routing* protokol PEGASIS. Sebelum melakukan agregasi paket data, LN mengirimkan token ke *node* sensor yang paling jauh dari BS. Terdapat 2 *node* sensor yang paling jauh dari BS, yaitu N1 dan N12. Token akan dirikimkan terlebih dahulu ke *node* sensor N1. Karena untuk melakukan agregasi paket data dilakukan sesuai urutan dari *node* sensor yang paling ujung. Setelah *node* sensor N1 menerima token, *node* sensor N1 mengirimkan paket data dan token menuju *node* sensor N2. Kemudian, pada *node* sensor N2 terdapat 2 paket data, yaitu dari dari *node* sensor N1 dan N2. Selanjutnya, paket data dari N1, N2 dan token diteruskan menuju *node* sensor N3. Kemudian *node* sensor N3 mengirimkan paket data dan token menuju selanjutnya sampai paket data serta token diterima oleh leader *node* sensor . Kemudian proses agregasi paket data ilanjutkan oleh *node* sensor N12

dengan mengirimkan token ke *node* sensor N12. Dimana proses agregasi paket data pada *node* sensor N12 memiliki cara yang sama ketika dilakukan *node* sensor N1 hingga paket data dari semua *node* sensor dan token diterima oleh LN. Selanjutnya LN mengirimkan paket data dari semua *node* sensor dan paket data LN sendiri menuju BS. Pada BS, paket data yang diterima dalah sebanyak 20 paket data, 19 paket data dihasilkan oleh *node* sensor sedangkan 1 paket data dihasilkan oleh LN [27].

Dengan menggunakan algoritma *Greedy*, tidak membutuhkan banyak konsumsi energi untuk melakukan pengiriman paket data. Hal ini dikarenakan pada algoritma *Greedy* memanfaatkan jarak terdekat untuk meneruskan pengiriman paket data. Namun, algoritma *Greedy* yang digunakan pada *routing* protokol PEGASIS mempunyai kelemahan, diantaranya adalah tidak selamanya memberikan solusi yang optimal, dikarenakan pencarian *local maximum* pada setiap langkahnya, tanpa memperhatikan solusi secara keseluruhan [17]. Selain itu, di dalam PEGASIS ada kemungkinan membentuk struktur rantai yang panjang dan dapat menghasilkan *delay* transmisi yang besar [18].



Gambar 2.4 Ilustrasi Agregasi Paket data (Pengumpulan Paket data)[27]

2.6 *Software* Simulasi NS3

Menurut Carneiro, Gustavo (2011), NS-3 merupakan simulasi jaringan yang memungkinkan *network* programmer untuk membuat simulasi dari jaringan yang akan dibangun dan saling berbagi *code* dari simulasi dan implementasi dari protokol yang dijalankan. NS-3 bahkan mampu memproses paket dalam ukuran besar.

NS-3 merupakan simulator jaringan acara diskrit untuk sistem internet, yang digunakan untuk penelitian dan pendidikan. NS-3 merupakan *software* gratis yang dilisensikan di bawah lisensi GNU GPLv2, dan tersedia untuk umum yang dapat digunakan untuk penelitian dan pengembangan. NS-3 merupakan *tool* yang selaras dengan kebutuhan simulasi penelitian tentang jaringan modern yang memungkinkan para peneliti untuk mempelajari protokol internet dan sistem berskala besar [28].

Adapun kelebihan dari *software* simulasi NS-3 adalah sebagai berikut :

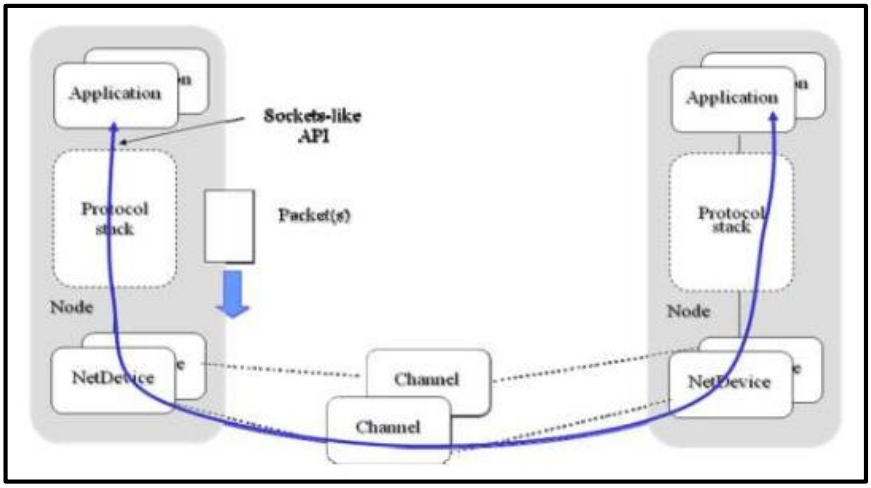
1. Sebagai perangkat lunak simulasi pembantu analisis dalam riset atau penelitian.
2. Memiliki *tool* validasi yang berfungsi untuk menguji validitas pemodelan yang ada pada NS-3.
3. Pembuatan simulasi lebih mudah dibandingkan dengan *software* developer yang lain.
4. Bersifat *open source*.
5. Pada NS-3 *user* tinggal membuat topologi dan skenario simulasi yang sesuai dengan kebutuhan.

Contoh simulasi yang menggunakan *software* simulasi NS-3 adalah sebagai berikut :

1. TCP/UDP/RTP.
2. *Traffic behaviour* (FTP, Telnet, CBR, dll).
3. *Queue management* (RED, FIFO, CBQ).
4. Algoritma *routing Unicast (Distance Vector, Link State)* dan *multicast*.
5. PIM SM, PIM DM, DVMRP, *Shared Tree* dan *Bi directional Shared Tree*.
6. Aplikasi multimedia yang berupa *layered* video.
7. QOS audio-video dan *transcoding*.

2.6.1 Dasar Model *Software* Simulasi NS-3

Berikut ini adalah dasar model simulasi menggunakan *software* NS-3 yang ditunjukkan pada Gambar 2.5. Gambar 2.5 menunjukan dasar dari model simulasi dengan menggunakan *software* NS-3. Semua simulasi jaringan yang dibuat dengan menggunakan *software* NS-3 mengikuti alur model tersebut. Dimana paket yang dihasilkan oleh *application* akan melewati berbagai susunan protokol sebelum dikirimkan melalui *channel* yang merupakan sebuah media yang menjadi tempat mengalirnya paket data dalam suatu jaringan. Dalam *Software* NS-3 Dev abstraksi komunikasi dasar *subnetwork* disebut *channel* dan diwakili di C++ oleh kelas *channel*, oleh *NetDevice*. *Application, protocol stack*, dan *NetDevice* tersebut terdapat dalam sebuah *node* sensor yang telah dibuat. Lalu paket tersebut akan dikirimkan ke *node* sensor yang dituju, yang pertama-tama akan diterima oleh *NetDevice node* sensor yang dituju, kemudian melewati lapisan protokol, dan akan dibaca dan ditampilkan isi dari paket tersebut oleh *application* juga. *Channel* berfungsi sebagai media perantara yang menjembatani antara *node* sensor sensor yang satu dengan yang lain.



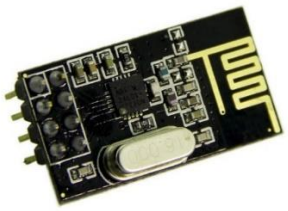
Gambar 2.5 Dasar Model Simulasi NS-3[29]

2.7 Modul *Wireless* NRF24L01

Modul *wireless* NRF24L01 merupakan *transceiver* 2.4GHz chip tunggal yang memiliki *embadded baseband protocol engine* (*Enhanced ShockBurst* ™),

dirancang khusus untuk memberikan dukungan pada aplikasi nirkabel *ultra low power*. *Enhanced ShockBurst*™ merupakan format protokol yang digunakan oleh modul *wireless* NRF24L01. Dimana *Enhanced ShockBurst*™ menggunakan *shockburst* ™ untuk penanganan dan pengaturan paket secara otomatis. Selama transmisi, *ShockBurst*™ merakit paket dan memasukkan bit-bit dalam paket paket data ke pemancar untuk transmisi. Selama menerima, *ShockBurst*™ secara konstan mencari alamat yang valid dalam sinyal yang didemodulasi. Ketika *ShockBurst*™ menemukan alamat yang valid, ia memproses sisa paket dan memvalidasinya dengan CRC. Jika paket valid, *payload* dipindahkan ke RX FIFO. Penanganan bit berkecepatan tinggi dan pengaturan waktu dikendalikan oleh *ShockBurst*™.

Modul *wireless* NRF24L01 dirancang untuk beroperasi pada pita frekuensi ISM di 2,4 – 2,5GHz. Dibutuhkan *Microcontroller Unit* (MCU) dan beberapa komponen pasif eksternal yang diperlukan untuk merancang sebuah sistem komunikasi radio menggunakan nRF24L01[30]. Modul ini memiliki *register map* tersedia melalui antarmuka SPI. *Register map* sendiri berisikan semua konfigurasi *registe*r pada NRF24L01 dan dapat diakses pada semua mode operasi dari *chip* [31]. Bentuk dari modul *wireless* NRF24L01 ditunjukkan dalam Gambar 2.6.



Gambar 2.6 Modul *Wireless* NRF24L01[32]

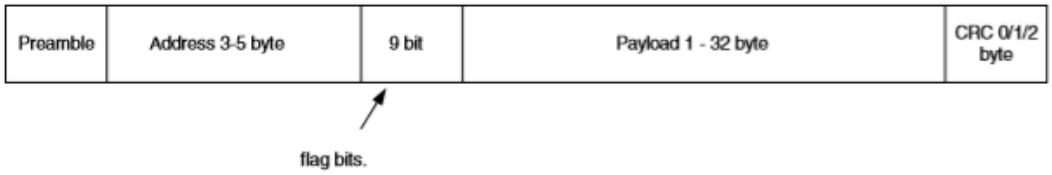
Pada modul *wireless* NRF24L01 menggunakan modulasi *Gaussian Frequency Shift Keying* (GFSK). Selain itu, modul ini juga mendukung *user configurable* dimana dapat melakukan konfigurasi meliputi parameter paket data rate dan channel frekuensi. Untuk paket data rate maksimum modul ini adalah sebesar 2Mbps. Adapun fitur yang disediakan modul *wireless* NRF24L01 sebagai berikut :

1. Modul transceiver 2.4GHz RF.
2. Tegangan kerja : 3.3V.

3. Arus: 50 – 250 mA.
4. Protokol komunikasi : SPI.
5. Baud Rate: 250 kbps - 2 Mbps.
6. Channel *ramge*: 125.
7. Solusi komunikasi *wireless* berdaya rendah [33].

Adapun format protokol yang digunakan modul *wireless* NRF24L01 ditunjukkan dalam Gambar 2.7. Gambar 2.7 menunjukkan format protokol yang digunakan pada modul *wireless* NRF24L01. Adapun fungsi dari masing-masing bagain protokol adalah sebagai berikut :

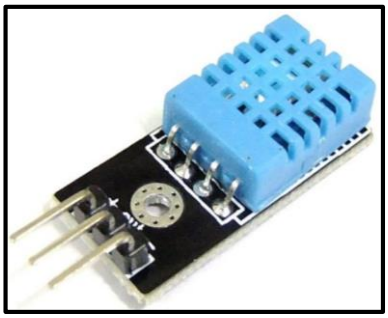
1. *Preamble* : Untuk menentukan apakah modul *wireless* berfungsi sebagai pengirim (RX) dan penerima (TX) dengan memberikan level 0 ataupun 1.
2. *Address* : Berfungsi untuk memberikan informasi berupa alamat penerima dan pengirim. Pada bagian ini memiliki panjang maksimum 5 byte atau 40 bit.
3. *Flags* : Pada bagian ini terdapat *Pakcet Identification* (PID) yang berfungsi untuk mendeteksi apakah paket yang diterima adalah baru atau dikirim ulang. PID ini ditambahkan di sisi TX untuk setiap paket baru yang diterima melalui *interface* SPI
4. *Payload* : Pada bagian ini berisikan paket data yang akan dikirim dengan panjang maksimum paket data 32 byte atau 256 bit.
5. CRC : Pada bagaian ini berfungsi untuk mengirimkan data ulang ketika data ada yang rusak atau hilang. Akan tetapi panjang data yang dikirimkan ulang adalah sebesar 0-2 byte.



Gambar 2.7 Format Protokol Modul *Wireless* NRF24L01[32]

2.8 Sensor Suhu dan Kelembaban Udara DHT11

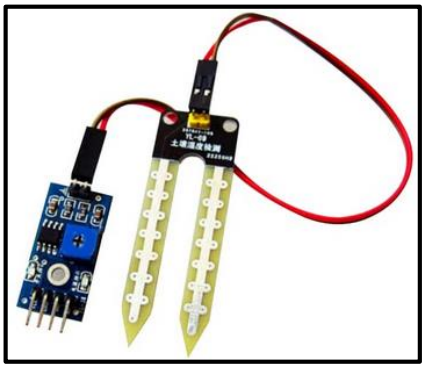
Sensor suhu adalah sensor yang melakukan pengukuran suhu dan kelembaban udara disekitar tanaman yang dapat mempengaruhi laju proses biokimia. Sensor DHT11 adalah sensor yang tersusun dari elemen polimer kapasitif (yang digunakan untuk mengukur kelembaban), dan sensor suhu. Didalam sensor ini terdapat memori kalibrasi yang berfungsi untuk menyimpan koefisien kalibrasi hasil pengukuran. Paket data yang dihasilkan dari sensor ini berupa digital *logic* yang dapat diakses secara serial. DHT11 adalah sensor suhu dan kelembapan yang memiliki kisaran pengukuran 20-90% *Relatife Humidity* (RH) dan ℃. Sensor bekerja pada dua kabel yaitu paket data dan Serial *Clock* (SCK). Paket data yang didapatkan berupa paket data pengukuran suhu lingkungan. Jika sensor membaca suhu rendah maka tegangan *pull down* yang dialirkan menjadi lebih besar dan tegangannya semakin besar[34]. DHT11 merupakan sensor digital yang digunakan untuk mengukur kelembaban dan suhu udara. Sensor ini terbuat dari komposit dengan sinyal keluaran yang telah dikalibrasi. Dengan menggunakan akuisisi signal digital dan teknologi sensing suhu dan kelembaban, membuat DHT11 memiliki keandalan dan stabilitas jangka. Sensor ini bekerja berdasarkan sensor kapasitif untuk mengukur kelembaban relatif udara. Uap didalam atmosfer akan mengubah permitivitas listrik diudara. Jarak atau ruang antar plat kapasitor dapat diisi dengan isolator dengan konstanta dielektrik yang berubah secara signifikan tergantung kelembaban. Sedangkan untuk mengukur suhu, sensor ini bekerja berdasarkan sensor *Negative Temperature Coefficient* (NTC). Resistansi dari sensor akan naik, jika suhu turun dan sebaliknya. Adapun bentuk sensor suhu dan kelembaban udara DHT11 ditunjukkan dalam Gambar 2.8.



Gambar 2.8 Sensor DHT11[35]

2.9 Sensor Kelembaban Tanah (*Soil moisture*)

Sensor *soil moisture* adalah sensor yang digunakan untuk mengukur kelembaban pada tanah. Sensor ini bekerja pada rentang pengukuran 0-100% dengan akurasi sebesar +-5% RH. Sensor ini beker berdasarkan prinsip sensor kapasitif. Terdapat dua buah plat kapasitor dengan yang dipisah dengan dielektrik. Kelembaban tanah akan mengubah permitivitas dielektrik antar plat kapasitor yang sebanding dengan tegangan yang dihasilkan. Adapun bentuk sensor kelembaban tanah (*soil moisture*) ditunjukkan dalam Gambar 2.9.



Gambar 2.9 Sensor *Soil Moisture*[36]

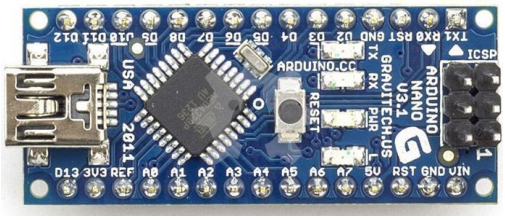
2.10 Arduino Nano

Arduino Nano merupakan sebuah *platform* dari *physical computing* yang bersifat *open source*. Arduino Nano tidak hanya sekedar sebuah alat pengembangan, tetap merupakan kombinasi dari *hardware*, bahasa pemrograman dan *Integrated Development Environment* (IDE) yang canggih. IDE adalah sebuah *software* yang sangat berperan untuk menulis program,

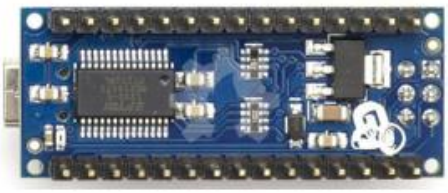
meng-*compile* menjadi kode biner dan meng-*upload* ke dalam memori mikrokontroller.

Arduino Nano merupakan salah satu papan pengembangan mikrokontroler yang berukuran kecil, lengkap dan mendukung penggunaan *breadboard*. Arduino Nano diciptakan dengan basis mikrokontroler atmega328 (untuk arduino nano versi 3.x) atau atmega 168 (untuk arduino versi 2.x). Arduino Nano kurang lebih memiliki fungsi yang sama dengan Arduino Duemilanove, tetapi dalam paket yang berbeda.

Arduino Nano tidak menyertakan colokan DC berjenis Barrel Jack, dan dihubungkan ke komputer menggunakan *port* USB Mini-B. Arduino Nano dirancang dan diproduksi oleh perusahaan Gravitech. Bentuk fisik Arduino Nano ditunjukkan dalam Gambar 2.10 dan 2.11.



Gambar 2.10 Bagian Depan Arduino Nano[37]



Gambar 2.11 Bagian Belakang Arduino Nano[37]

Arduino Nano menggunakan atmega328 (untuk Arduino Nano versi 3.x) atau Atmega 168 (untuk Arduino versi 2.x). Komunikasi serial ke komputer melalui *port* USB Mini-B. Adapun deskripsi dari Arduino nano pada Tabel 2.3.

Tabel 2.3 Deskripsi Arduino Nano[37]

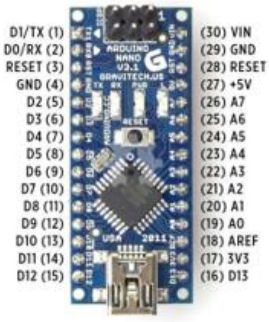
Nomor Pin Arduino Nano	Nama Pin Arduino Nano
1	Digital Pin 1 (TX)
2	Digital Pin 0 (TX)
3 dan 28	Reset
4 dan 29	GND
5	Digital Pin 2
6	Digital Pin 3 (PWM)
7	Digital Pin 4
8	Digital Pin 5 (PWM)
9	Digital Pin 6 (PWM)
10	Digital Pin 7
11	Digital Pin 8
12	Digital Pin 9 (PWM)
13	Digital Pin 10 (PWM-SS)
13	Digital Pin 10 (PWM-SS)
14	Digital Pin 11 (PWM-MOSI)
15	Digital Pin 12 (MISO)
16	Digital Pin 13 (SCK)
18	AREF
19	Analog *Input* 0
20	Analog *Input* 1
21	Analog *Input* 2
22	Analog *Input* 3
23	Analog *Input* 4
24	Analog *Input* 5
25	Analog *Input* 6
26	Analog *Input* 7
26	Analog *Input* 7
27	VCC
28	RESET
29	GND
30	V.in

Konfigurasi pin arduino nano. Arduino nano memiliki 30 pin. Berikut konfigurasi pin arduino nano :

1. VCC merupakan pin yang berfungsi sebagai pin masukan catu daya digital.
2. GND merupakan pin *ground* untuk catu daya digital.
3. AREF merupakan Referensi tegangan untuk *input* analog. Digunakan dengan fungsi analog*Reference*().

4. RESET merupakan rute *low* ini digunakan untuk me-reset (menghidupkan ulang) mikrokontroller. Biasanya digunakan untuk menambahkan tombol reset pada *shield* yang menghalangi papan utama arduino.
5. Serial RX (0) merupakan pin yang berfungsi sebagai penerima TTL paket data serial.
6. Serial TX (1) merupakan pin yang berfungsi sebagai pengirim TTL paket data serial.
7. *External Interrupt* (Interupsi Eksternal) merupakan pin yang dapat dikonfigurasi untuk memicu sebuah interupsi pada nilai yang rendah, meningkat atau menurun, atau perubahan nilai.
8. *Output* PWM 8-bit merupakan pin yang berfungsi untuk analog*Write*( ).
9. SPI merupakan pin yang berfungsi sebagai pendukung komunikasi.
10. LED merupakan pin yang berfungsi sebagai pin yag diset bernilai *high*, maka LED akan menyala, ketika pin diset bernilai *low* maka LED padam. LED Tersedia secara built-in pada papan arduino nano.
11. *Input* analog (A0-A7) merupakan pin yang berfungsi sebagi pin yang dapat diukur/diatur dari mulai g*round* sampai dengan 5 Volt (V), juga memungkinkan untuk mengubah titik jangkauan tertinggi atau terendah mereka menggunakan fungsi analog*Reference*().

Adapun konfigurasi pin *layout* arduino nano ditunjukkan seperti dalam Gambar 2.12.



Gambar 2.12 Konfigurasi Pin *Layout* Arduino Nano[37]

Adapun spesifikasi dari *board* arduino nano sebagai berikut :

1. Mikrokontroler atmel atmega168 atau atmega328.

2. 5V tegangan operasi.
3. 7-12V *input* Voltage (disarankan).
4. 6-20V *input* Voltage (limit).
5. Pin digital I/O14 (6 pin digunakan sebagai *output* PWM).
6. 8 Pin *input* analog.
7. 40 mA arus DC per pin I/O.
8. *Flash* memori 16kb (atmega168) atau 32kb (atmega328) 2kb digunakan oleh b*ootloader*.
9. 1 *Kbyte* SRAM (atmega168) atau 2 k*byte*(atmega328).
10. 512 *Byte* EEPROM (atmega168) atau 1 *Kbyte* (atmega328).
11. 16 MHz *Clock Speed*.
12. Ukuran1.85cm x 4.3cm.

Untuk sumber daya Arduino Nano dapat diaktifkan melalui koneksi USB Mini-B, atau melalui catu daya eksternal dengan tegangan belum teregulasi antara 6-20 Volt yang dihubungkan melalui pin 30 atau pin V.in, atau melalui catu daya eksternal dengan tegangan teregulasi 5V melalui pin 27 atau pin 5V. Sumber daya akan secara otomatis dipilih dari sumber tegangan yang lebih tinggi. *Chip* FTDI FT232L pada Arduino Nano akan aktif apabila memperoleh daya melalui USB, ketika arduino nano diberikan daya dari luar (non-USB) maka *chip* FTDI tidak aktif dan pin 3.3 Volt pun tidak tersedia (tidak mengeluarkan tegangan), sedangkan LED TX dan RX pun berkedip apabila pin digital 0 dan 1 berada posisi *high*.

2.11 Arduino Uno

Arduino Uno merupakan sebuah *platform* dari *physical computing* yang bersifat *open source*. Pada Arduino Uno menggunakan bahasa pemrograman dan *Integrated Development Environment* (IDE) yang canggih. Arduino Uno merupakan salah satu papan pengembangan mikrokontroler yang berukuran kecil, lengkap dan mendukung penggunaan *breadboard*. Arduino Uno merupakan sebuah *board* mikrokontroler yang dikontrol penuh oleh ATmega328. Seperti yang ditunjukan pada Gambar 2.13. Arduino Uno mempunyai 14 pin digital *input*/*output* (6 di antaranya dapat digunakan sebagai *output* PWM), 6 *input* analog, sebuah osilator kristal 16 MHz, sebuah koneksi

USB, sebuah power jack, sebuah ICSP *header*, dan sebuah tombol reset. Arduino Uno memuat semua yang dibutuhkan untuk menunjang mikrokontroller, mudah menghubungkannya ke sebuah komputer dengan sebuah kabel USB atau mensuplainya dengan sebuah adaptor AC ke DC.

Skematik Arduino *board* yang telah disederhanakan seperti pada Gambar 2.13 merupakan sebuah papan yang dapat dipasang diatas Arduino *board* untuk menambah kemampuan dari arduino *board*. Bahasa pemograman yang dipakai dalam Arduino bukan bahasa *assembler* yang relatif sulit, melainkan bahasa pemograman mirip dengan bahasa pemrograman C++ yang disederhanakan dengan bantuan pustaka-pustaka (*libraries*) Arduino [38]. Adapun spesifikasi paket data teknis yang terdapat pada *board* Arduino UNO R3 ditampilkan dalam Tabel 2.4

Gambar 2.13 Konfigurasi pin ATMega 328 Arduino uno R3[38]

Tabel 2.4 Spesifikasi Arduino Uno[38]

No.	Operasi Kerja	Keterangan
1.	Operating Voltage	5V
2.	*Input* Voltage (recommended)	7-12V
3.	*Input* Voltage (limit)	6-20V
4.	Digital I/O Pins	14 (of which 6 provide PWM *output*)
5.	PWM Digital I/O Pins	6
6.	Analog *Input* Pins	6
7.	DC Current per I/O Pin	20 mA
8.	DC Current for 3.3V Pin	50 mA
9.	Flash Memory	32 KB (ATmega328P) of which 0.5 KB used by bootloader
10.	SRAM	2 KB (ATmega328P)

Tabel 2.4 Spesifikasi Arduino Uno (Lanjutan)[38]

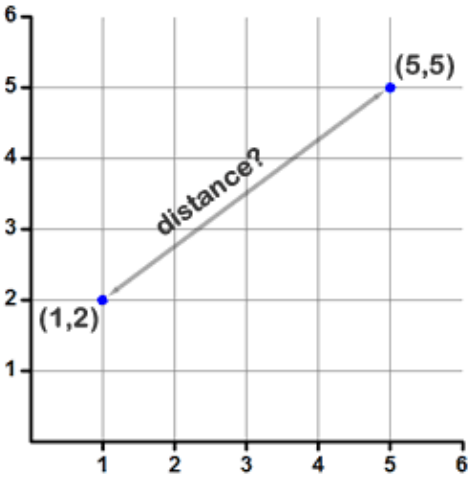
No.	Operasi Kerja	Keterangan
11.	EEPROM	1 KB (ATmega328P)
12.	Clock *Speed*	16 MHz

2.12 Euclidian distance

Merupakan sebuah metode yang digunakan untuk melakukan perhitungan jarak dari 2 buah titik dalam *Euclidean space. Euclidean space* diperkenalkan oleh Euclid, seorang matematikawan dari Yunani sekitar tahun 300 B.C.E. untuk mempelajari hubungan antara sudut dan jarak. *Euclidian* ini berkaitan dengan Teorema Phytagoras dan biasanya diterapkan pada 1, 2 dan 3 dimensi. Untuk perhitungan *euclidian distance* pada 2 dimensi, dapat dijelaskan sebagai berikut :

Misalkan terdapat 2 titik seperti yang ditunjukkan dalam Gambar 2.14. Pada titik pertama mempunyai kordinat (1,2). Titik kedua ada di kordinat (5,5). Untuk melakukan perhitungan jarak diantara 2 titik, dapat menggunakan persamaan 2.1. Caranya adalah kurangkan setiap kordinat titik kedua dengan titik yang pertama, yaitu, (5-1,5-2) sehingga menjadi (4,3). Kemudian pangkatkan masing-masing sehingga memperoleh (16,9). Kemudian tambahkan semuanya sehingga memperoleh nilai 16+9 = 25. Hasil ini kemudian diakarkan menjadi 5. Sehingga didapatkan jarak *euclidian* adalah 5[38].

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \quad (2.1)$$



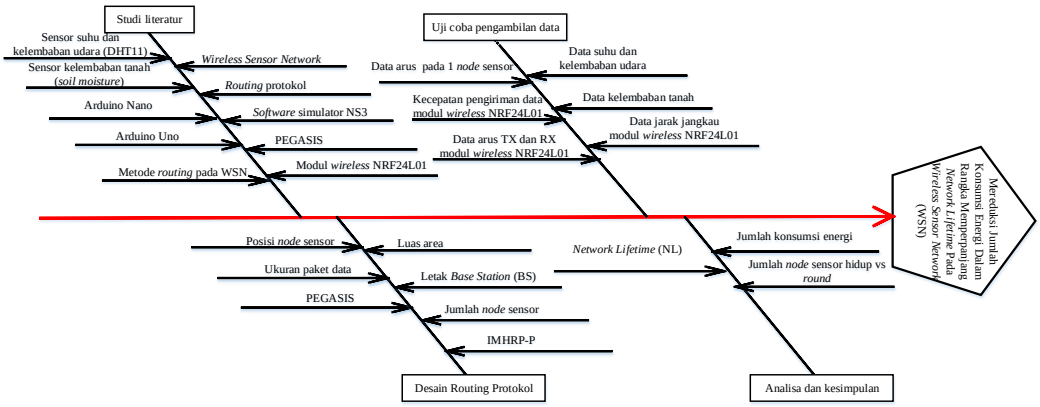
Gambar 2.14 *Euclidian Distace* pada 2 Dimensi[39]

Halaman ini sengaja dikosongkan

BAB 3
METODE PENELITIAN

3.1 Skema Penelitian

Dalam skema penelitian ini ditulis dan diringkas dalam blok diagram *fishbone*. Dalam diagram *fishbone* dijelaskan tahapan-tahapan secara detail dimulai dari tahap studi literatur, desain *routing* protokol, uji coba pengambilan paket data dan analisis dan kesimpulan. Untuk blok diagram *fishbone* ditunjukkan pada Gambar 3.1 sebagai berikut.



Gambar 3.1 Diagram Blok *Fishbone*

Gambar 3.1 menunjukkan diagram blok *fishbone*. Pada skema penelitian ini dilakukan beberapa tahapan. Pada tahap awal adalah melakukan literatur yang meluputi beberapa dasar teori yang mendukung dalam proses penelitian ini, yang meliputi : *Wireless Sensor Network* (WSN), *routing* protokol, *software* simulasi, PEGASIS, dan perangkat yang digunakan dalam pembuatan simulasi, diantaranya : sensor suhu dan kelembaban udara (DHT11), sensor kelembaban tanah (*soil moisture*), mikrokontroller Arduino Nano, Arduino dan modul *wireless* NRF24L01.

Pada tahap kedua yaitu melakukan perancangan desain *routing* protokol dari PEGASIS dan IMHRP-P yang meliputi parameter seperti : luas area, lokasi *Base Station* (BS), jumlah *node* sensor, posisi *node* sensor dan ukuran paket data.

Pada tahap ketiga adalah melakukan uji coba pengambilan data berupa data arus dan tegangan yang dibutuhkan sensor DHT11, *soil moisture* dan modul *wireless* NRF24L01. Selain itu, juga dibutuhkan data hasil pengujian suhu dan kelembaban udara, kelembaban tanah, jarak jangkau modul *wireless* NRF24L01, arus yang digunakan pada 1 *node* sensor, arus pengiriman yang digunakan modul *wireless* NRF24L01 dan kecepatan modul yang digunakan.

Sedangkan pada tahap keempat adalah melakukan analisa dan kesimpulan dari parameter pengujian PEGASIS dan IMHRP-P yang meliputi : jumlah konsumsi energi, NL dan jumlah *node* sensor hidup vs *round*.

3.2 Tahapan Penelitian

Pada penelitian ini, untuk menyelesaikan dengan hasil yang baik langkah-langkah proses penelitian sangat penting dalam mengidentifikasi bidang penelitian, memilih masalah utama, mengusulkan dan mengimplementasikan desain serta mengevaluasi dan memvalidasi hasil. Langkah-langkah utama dari proses penelitian adalah seperti yang ditunjukkan pada Gambar 3.2. Gambar 3.2 memperlihatkan diagram alir penelitian yang akan dilakukan dapat dijelaskan sebagai berikut:

Tahap pertama adalah mengidentifikasi masalah merumuskan masalah yang akan diteliti. Tahap ini merupakan tahap yang paling penting dalam penelitian, karena semua jalannya penelitian akan ditentukan oleh perumusan masalah.

Tahap kedua adalah studi literatur tentang *Wireless Sensor Network* (WSN), *routing* protokol, *software* simulasi, PEGASIS dan perangkat yang digunakan dalam pembuatan simulasi, diantaranya : sensor suhu dan kelembaban udara (DHT11), sensor kelembaban tanah (*soil moisture*), Arduino Nano, Arduino Uno dan modul *wireless* NRF24L01.

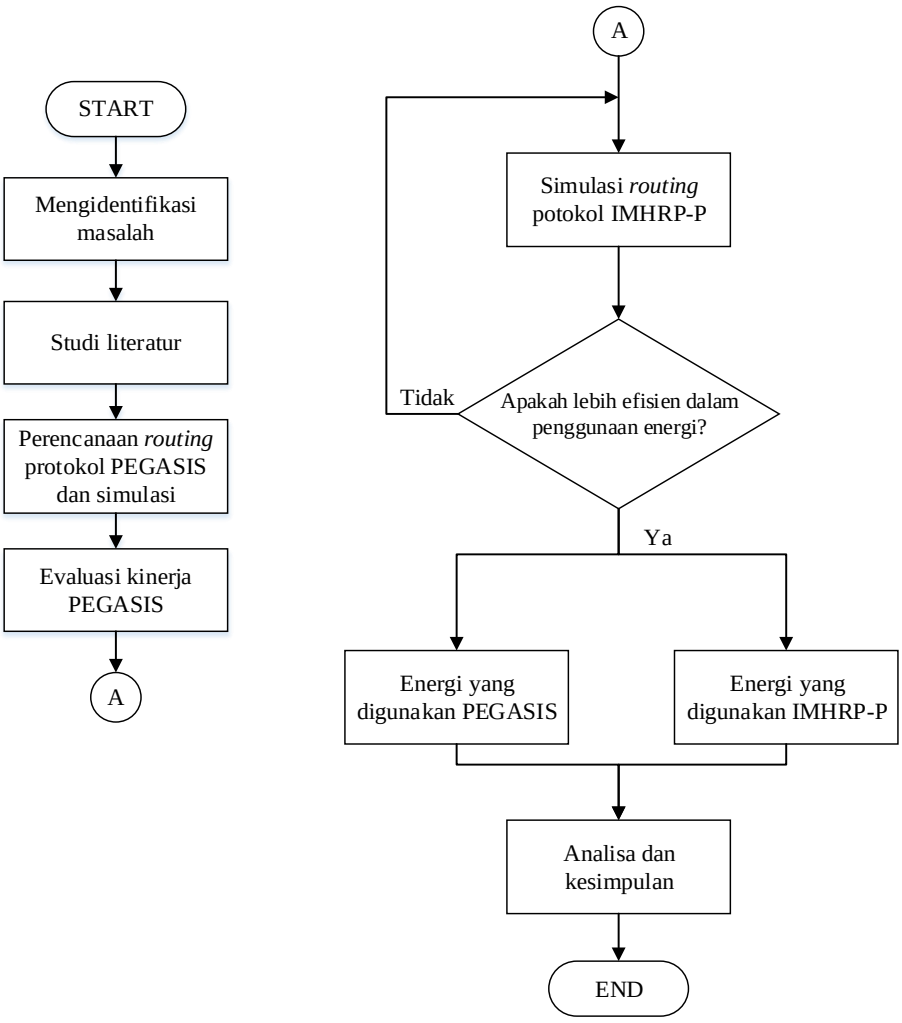
Tahap ketiga adalah menentukan perencanaan *routing* protokol PEGASIS dan simulasi. Dimana perencaan *routing* protokol ini meliputi parameter yang digunakan, diantaranya : luas area, lokasi BS, jumlah *node* sensor, posisi *node* sensor, ukuran paket paket data. Kemudian dari parameter yang digunakan dilakukan pembuatan simulasi pada *software* simulator NS3.

Tahap keempat adalah melakukan evaluasi kinerja PEGASIS. Pada tahap ini bertujuan untuk mengetahui kelemahan dari *routing* protokol tersebut. Selain itu, hal ini bertujuan untuk menemukan solusi untuk mengatasi kelemahan yang dimiliki oleh PEGASIS.

Tahap kelima adalah melakukan simulasi dari *routing* protokol Improved Multi*hop Routing* Protocol based on PEGASIS (IMHRP-P) yang bertujuan untuk menutupi kekurangan yang dimiliki oleh PEGASIS. Selain itu, dengan adanya usulan *routing* protokol bertujuan untuk memperpanjang *Network Lifetime* (NL) dan menghemat konsumsi energi pada masing-masing *node* sensor. Dengan melakukan simulasi *routing* protokol IMHRP-P bertujuan untuk mengetahui kelebihan dari *routing* protokol yang diusulkan.

Tahap keenam adalah evaluasi sistem kerja dari *routing* protokol IMHRP-P yang meliputi apakah lebih efisien dalam menggunakan energi. Untuk mengukur apakah lebih efisien atau tidak dilakukan pengecekan seberapa lama *node* sensor aktif atau memiliki energi yang digunakan mengirimkan paket data menuju BS. Jika *routing* protokol IMHRP-P belum efisien dalam penggunaan energi maka dilakukan pengecekan terhadap simulasi *routing* protokol IMHRP-P. Jika *routing* protokol IMHRP-P sudah efisien dalam penggunaan energi maka akan dilakukan perbandingan dengan PEGASIS.

Tahap ketujuh adalah analisa dan kesimpulan dari *routing* protokol PEGASIS dan IMHRP-P. Untuk melakukan analisa meliputi parameter jumlah *node* sensor hidup vs *round*, analisa *euclidian distance*, penggunaan daya dan *Network Lifetime* (NL).



Gambar 3.2 Diagram Alir Penelitian

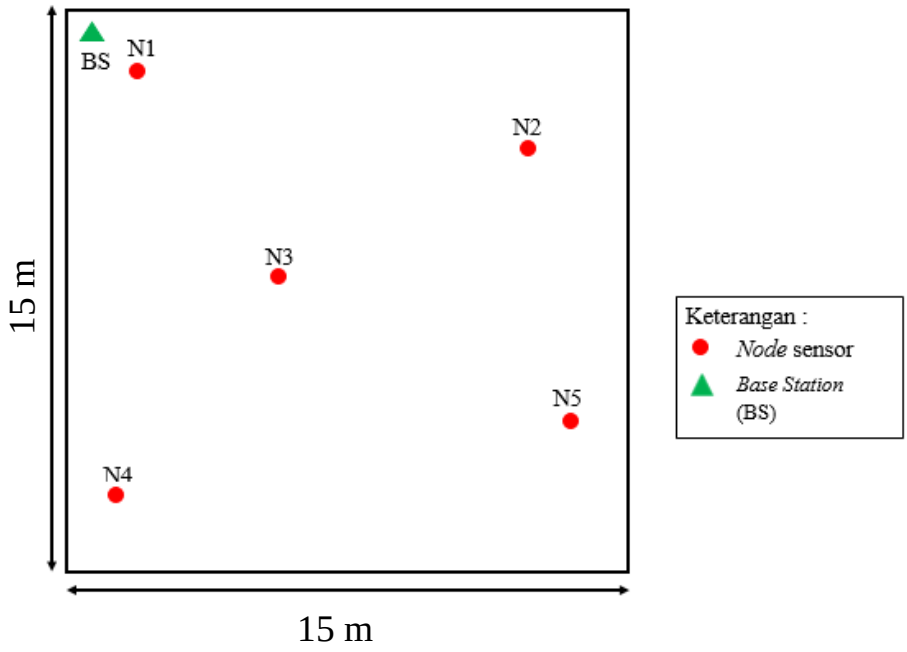
3.3 Desain Perencanaan *Routing* Protokol

Sub bab ini akan menjelaskan tentang desain alokasi *node* sensor pada sistem monitoring. Dimana untuk alokasi dari *node* sensor sensor disesuaikan dengan peletakan dari perangkat sistem monitoring didalam *greenhouse* dengan posisi BS terletak di tengah dengan luas area yang digunakan sebesar 225 m^2 memiliki panjang 15 meter dan lebar 15 meter. Sedangkan untuk jumlah *node* sensor sebanyak 5 dan 1 BS. Dimana 5 *node* sensor diletakkan secara *random* dan tidak bergerak. Selain itu dilakukan desain perancangan untuk luas area berukuran 100 m x 100 m dengan jumlah *node* sensor bervariasi dari 20 dan 35. Untuk mengetahui seberapa efisien energi yang digunakan dalam pemodelan *routing* protokol PEGASIS dan IMHRP-P digunakanlah 2 skenario. Pada skenario pertama BS

terletak di sudut dan skenario kedua BS terletak di tengah luas area seperti yang sudah digunakan dalam sistem monitoring sebelumnya. Untuk dapat mengetahui jarak dari *node* sensor menuju ke BS maupun jarak antar *node* sensor dengan cara melakukan perhitungan menggunakan rumus *euclidian distance* yang mengacu pada persamaan (2.1). Selain itu, juga dilakukan pembuatan skema untuk *node* sensor yang aktif dan tidak aktif. Dimana untuk *node* sensor yang aktif memiliki rentang waktu 1-10 detik. *Node* sensor dalam keadaan aktif ketika melakukan pembuatan koneksi dan pengiriman paket data ke *node* sensor yang lain. Setelah lebih dari 10 detik *node* sensor dalam keadaan *standby*. *Standby* merupakan kondisi dimana *node* sensor sedang tidak melakukan proses apapun akan tetapi *node* sensor siap untuk digunakan.

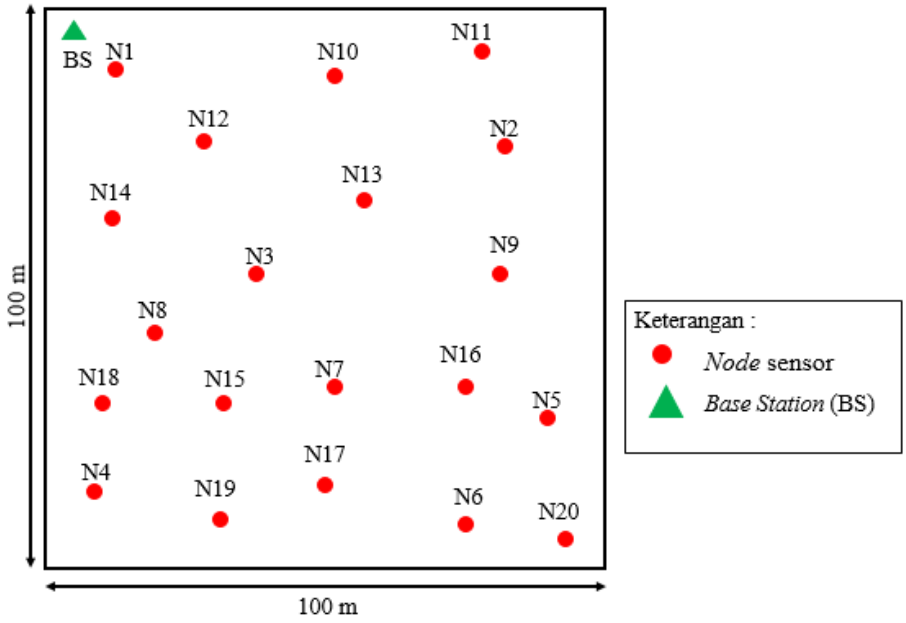
3.3.1 Skenario Pertama

Adapun desain perancangan untuk luas area 15 m x 15 m dengan 5 *node* sensor dan 1 BS menggunakan skenario pertama seperti yang ditunjukkan pada Gambar 3.3.



Gambar 3.3 Skenario Pertama dengan 5 *Node* Sensor

Gambar 3.3 menunjukkan desain perencanaan peletakan BS skenario pertama dengan 5 *node* sensor. Dimana skenario pertama dengan posisi BS berada di sudut. Sedangkan untuk desain perancangan untuk luas area 100 m x 100 m dengan jumlah *node* sensor sebanyak 20 dan 35 pada skenario pertama ditunjukkan pada Gambar 3.4 dan Gambar 3.5.

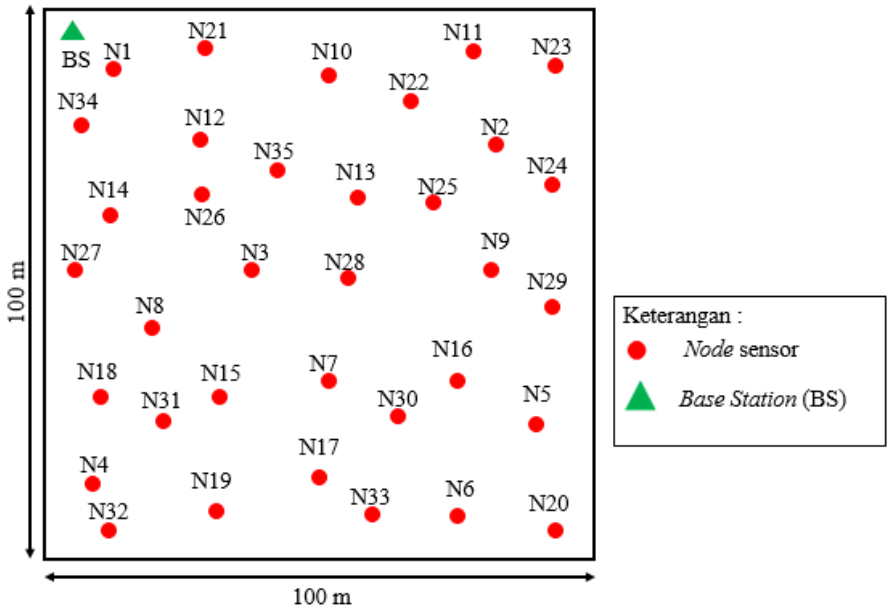


Gambar 3.4 Skenario Pertama dengan 20 *Node* Sensor

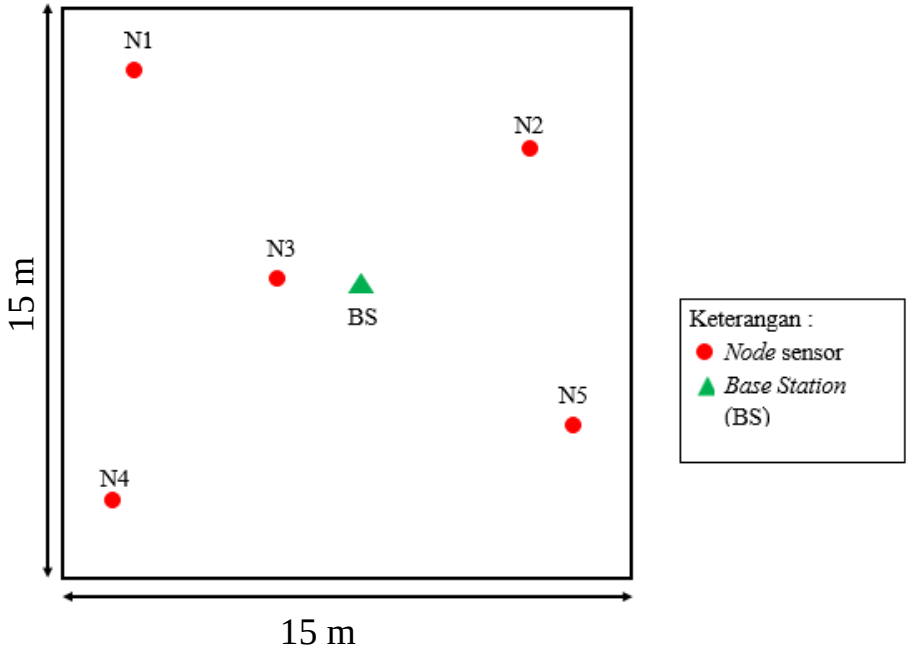
3.3.2 Skenario Kedua

Sedangkan pada skenario kedua dimana BS terletak di tengah luas area seperti yang digunakan dalam sistem monitoring dengan luas area 15 m x 15 m dan jumlah *node* sensor sebanya 5 serta 1 BS seperti yang ditunjukkan pada Gambar 3.6.

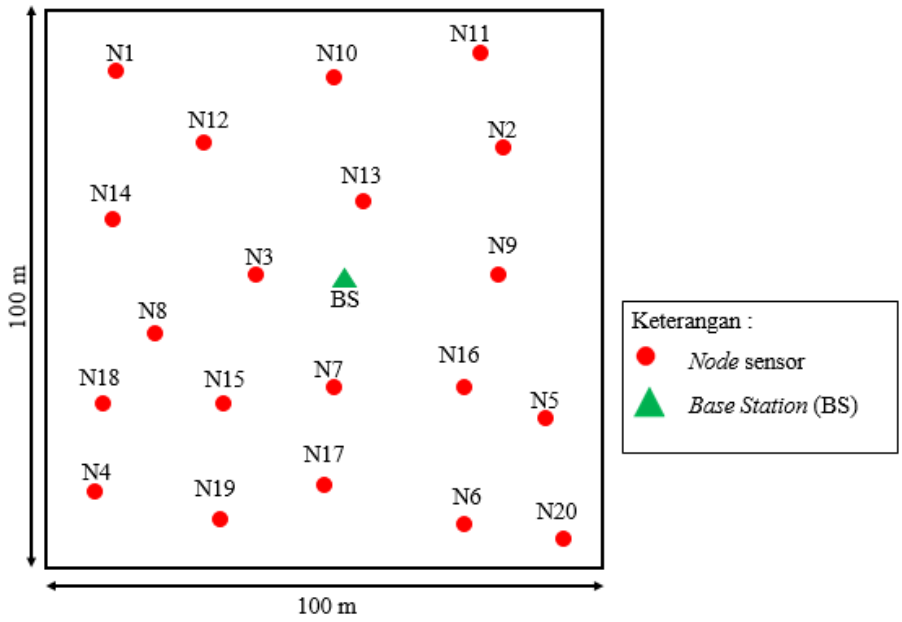
Gambar 3.6 menunjukkan desain perencanaan peletakan BS skenario kedua dengan 5 *node* sensor sensor. Dimana skenario kedua dengan posisi BS berada di tengah. Sedangkan untuk desain perancangan untuk luas area 100 m x 100 m dengan jumlah *node* sensor sebanyak 20 dan 35 pada skenario kedua ditunjukkan pada Gambar 3.7 dan Gambar 3.8.



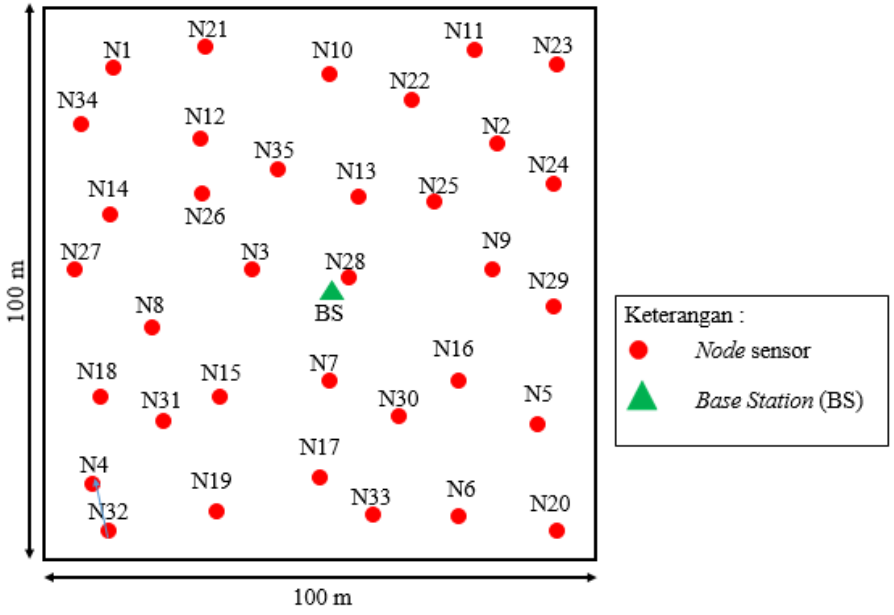
Gambar 3.5 Skenario Pertama dengan 35 *Node* Sensor



Gambar 3.6 Skenario Kedua dengan 5 *Node* Sensor



Gambar 3.7 Skenario Kedua dengan 20 *Node* Sensor



Gambar 3.8 Skenario Kedua dengan 35 *Node* Sensor

Dalam penelitian ini, untuk dapat merealisasikan desain perancangan *routing* protokol pada *hardware*, perangkat *node* sensor dan BS harus diletakkan dalam keadaan bebas hambatan atau *Line of Sight* (LOS). Sehingga antara simulasi dan

impelementasi *hardware* terdapat kesamaan hasil baik untuk hasil perhitungan jarak maupun energi yang digunakan masing-masing *node* sensor.

3.4 PEGASIS

Pada proses pembentukan *chain routing* protokol PEGASIS menggunakan algoritma *Greedy*. Algoritma *Greedy* merupakan sebuah algoritma yang dapat menentukan dan membuat sebuah rute berdasarkan *node* sensor terdekat. Adapun *flowchart* PEGASIS ditunjukkan pada Gambar 3.9.

Gambar 3.9 memperlihatkan *flowchart* PEGASIS. *Flowchart* ini menjelaskan cara kerja dari PEGASIS mulai dari menentukan *node* sensor terdekat, pembuatan *chain*, proses pengiriman paket data dan pembuatan ulang *chain* ketika ada LN atau *node* sensor yang mati. Adapun penjelasan *flowchart* PEGASIS, dapat dijelaskan sebagai berikut:

Tahap pertama adalah melakukan inisialisasi terhadap jumlah BS dan *node* sensor yang akan digunakan. Inisialisasi yang digunakan meliputi alamat yang digunakan oleh BS dan masing-masing dari *node* sensor.

Tahap kedua adalah melakukan inisialisasi parameter energi. Pada inisialisasi ini adalah memasukkan kapasitas daya yang digunakan pada masing-masing *node* sensor. Untuk kapasitas daya yang digunakan adalah sebesa 2,2 Ah.

Tahap ketiga adalah melakukan inisialisasi *poin to point*. Untuk melakukan insialisasi *point to point*, terlebih dahulu melakukan perhitungan jarak dari *node* sensor ke BS dan jarak dari *node* sensor satu dengan yang lain. Hal ini bertujuan untuk mempermudah dalam pembuatan inisialisasi *routing* pada tahap selanjutnya. Setelah diketahui jarak dari *node* sensor menuju ke BS maupun dari *node* sensor satu ke *node* sensor yang lain, dapat menentukan *node* sensor yang menjadi LN. Dimana yang menjadi LN adalah *node* sensor yang memiliki jarak yang paling dekat dengan BS. Kemudian akan dilakukan inisialisasi *point to point* yang dimulai dari BS ke LN, LN ke *node* sensor selanjutnya, *node* sensor selanjutnya ke *node* sensor yang lain. Proses pembentukan inisialisasi *point to point* berlanjut sampai *node* sensor terakhir.

Tahap keempat adalah melakukan inisialisasi *routing*. Pada inisialisasi ini bertujuan untuk pembuatan rute yang telah diinisialisasi pada pembuatan koneksi *point to point*. Sehingga pada tahap ini tinggal dilakukan pembuatan rute berdasarkan hasil inisialisasi koneksi *point to point*.

Tahap kelima adalah memulai pembentukan koneksi yang dimulai dari *node* sensor sensor yang paling jauh dari BS, yaitu *node* sensor terjauh ke 1. Dari *node* sensor terjauh ke 1 melakukan pembentukan koneksi pada *node* sensor sensor terjauh ke 2. Jika koneksi berhasil terbentuk, maka *node* sensor terjauh ke 1 akan memilih *node* sensor terjauh ke 2 sebagai rute terdekatnya. Sedangkan jika koneksi gagal terbentuk maka *node* sensor terjauh ke 1 akan memilih *node* sensor terjauh ke 2 sebagai rute terdekat selanjutnya.

Tahap keenam adalah melakukan penyimpanan alamat dari masing-masing *node* sensor yang sudah berhasil membuat koneksi.

Tahap ketujuh adalah memastikan apakah *node* sensor sekarang berada dalam posisi *node* sensor yang terakhir. Jika *node* sensor menjadi *node* sensor yang terakhir maka akan dibuatka rute selanjutnya. Jika *node* sensor bukan menjadi *node* sensor yang terakhir, makan rute tersebut akan dilewati.

Tahap kedelapan adalah memastikan bahwa *node* sensor sensor selanjutnya adalah BS atau bukan. Jika *node* sensor selanjutnya adalah BS maka akan dibuatkan rute menuju BS. Jika rute selanjutnya bukan BS maka akan dilakukan pengulangan pada menyimpan alamat *node* sensor.

Tahap kesembilan adalah memastikan bahwa proses pada *routing* protokol PEGASIS sudah selesai atau belum. Jika tahapan-tahapan pada *routing* protokol PEGASIS belum selesai maka akan dilakukan pengulangan pada tahap inisialisasi *routing*. Jika tahapan-tahapan sudah selesai maka proses akan berhenti atau berakhir.

3.5 Improved Multi-*Hop Routing* Protocol PEGASIS (IMHRP-P)

Adapun tahapan-tahapan sistem kerja dari IMHRP-P ditunjukkan dalam Gambar 3.10. Gambar 3.10 memperlihatkan *flowchart* IMHRP-P. *Flowchart* ini menjelaskan cara kerja dari IMHRP-P mulai dari melakukan *request* pada masing-

masing *node* sensor sensor sampai proses pengiriman paket data selesai dilakukan. Adapun penjelasan *flowchart* IMHRP-P, dapat dijelaskan sebagai berikut:

Tahap pertama adalah melakukan inisialisasi terhadap jumlah *node* sensor yang akan digunakan. Inisialisasi yang digunakan meliputi alamat yang digunakan oleh masing-masing dari *node* sensor.

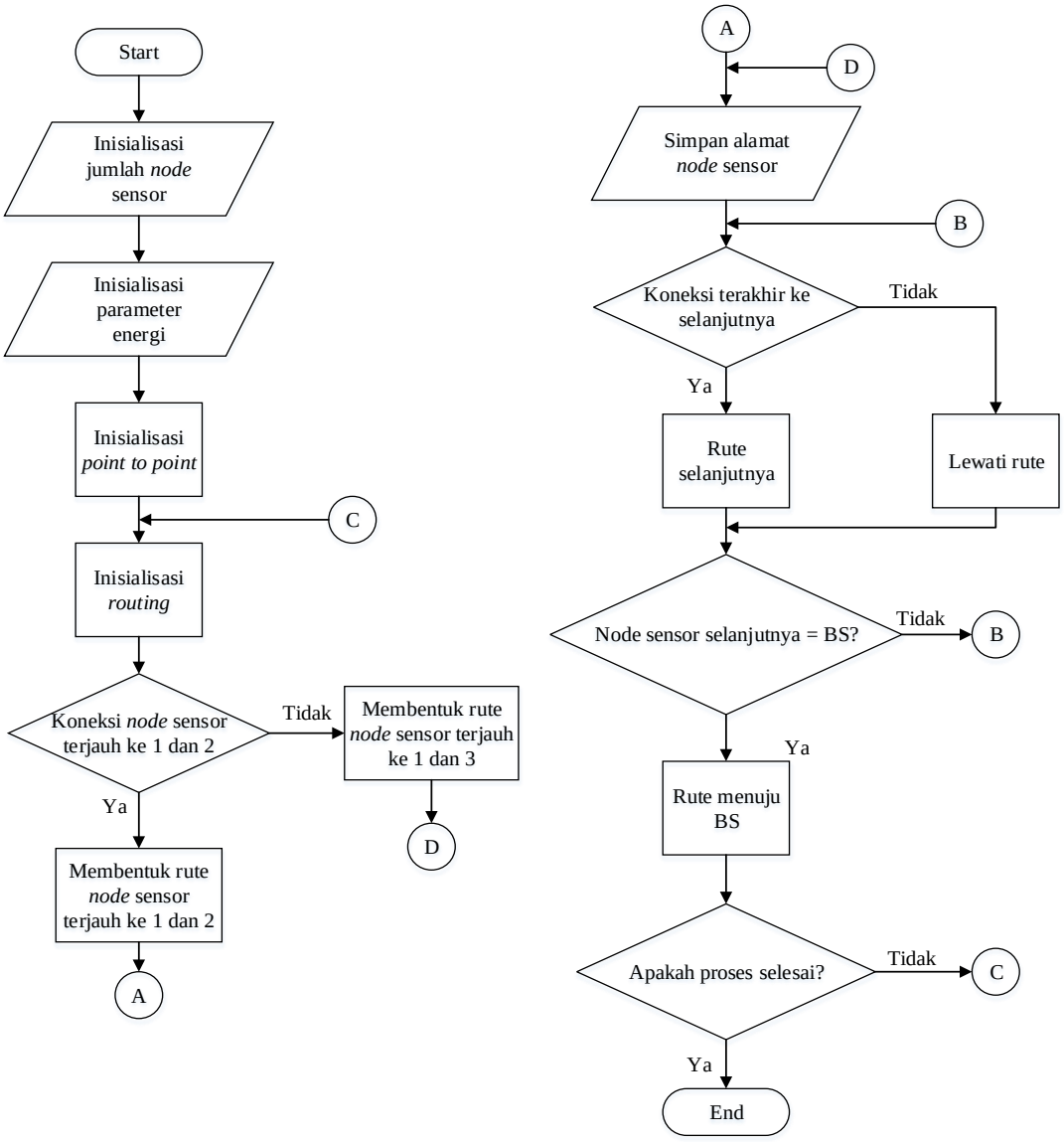
Tahap kedua adalah melakukan inisialisasi parameter energi. Pada inisialisasi ini adalah memasukkan daya yang digunakan pada masing-masing *node* sensor. Untuk kapasitas baterai sebesar 2,2 Ah dan tegangan sebesar 3,67 Volt pada saat baterai dalam kondisi penuh.

Tahap ketiga adalah melakukan inisialisasi *poin to point*. Pada inisialisasi ini bertujuan untuk melakukan koneksi antar *node* sensor dan memastikan pula bahwa *node* sensor masih memiliki energi untuk dapat melakukan komunikasi.

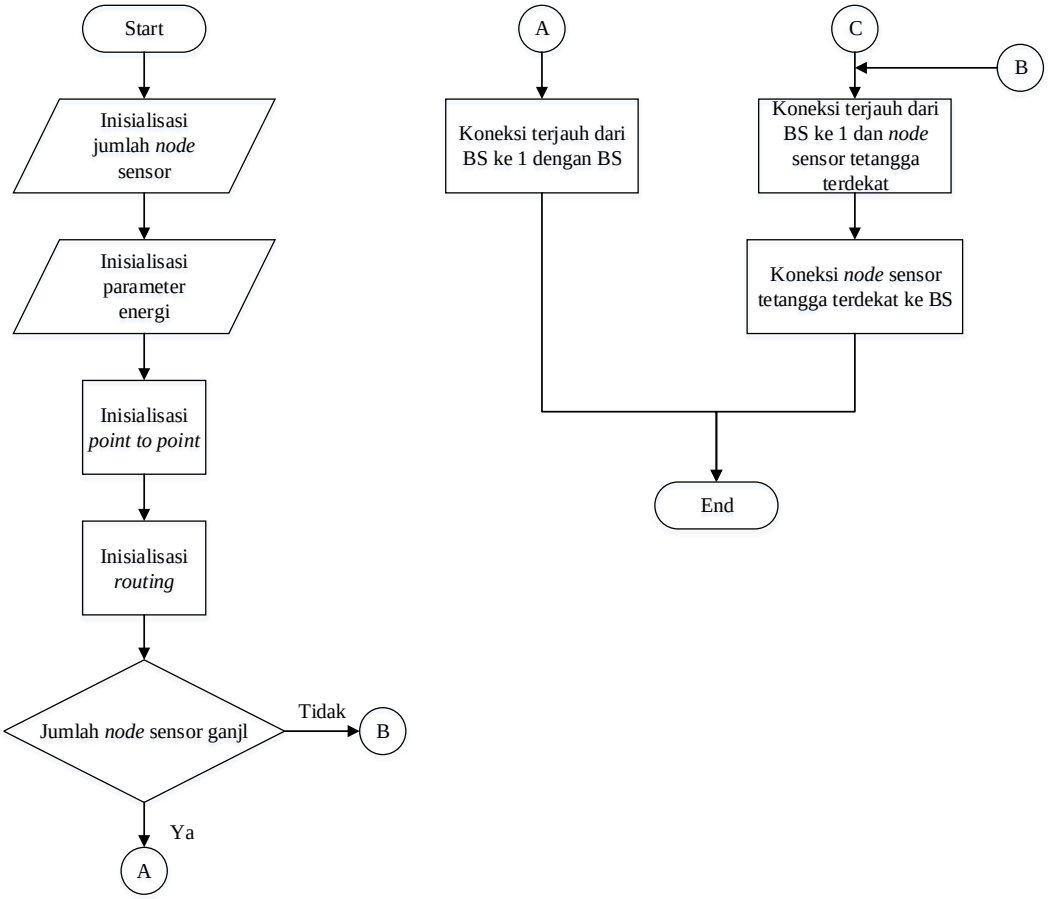
Tahap keempat adalah melakukan inisialisasi *routing*. Pada inisialisasi ini bertujuan untuk menentukan rute terdekat dari masing-masing *node* sensor. Dimana masing-masing *node* sensor memiliki tabel *routing* yang dapat digunakan ketika terdapat *node* sensor terdekat kehabisa energi. Sehingga dapat mencari *node* sensor terdekat kedua untuk dapat menjadi rute alternatif inisialisasi *routing*.

Tahap kelima adalah membagi *node* sensor menjadi 2 jenis klasifikasi, yaitu *node* sensor genap dan ganjil. Untuk *node* sensor genap, dapat membuat koneksi dengan *node* sensor terjauh ke 1 dan *node* sensor tetangga terdekat. Kemudian dari *node* sensor tetangga terdekat dapat membuat koneksi dengan BS. Kemudian proses akan berhenti.

Seperti yang terlihat pada Gambar 3.9 dan Gambar 3.10 yang menunjukan *flowchart* dari pemodelan *routing* protokol PEGASIS dan IMHRP-P, dapat dilihat dari dua *routing* protokol ini bahwa pada IMHRP-P memiliki *flowchart* yang lebih sederhana dalam pembentukan rute untuk mengirimkan paket dari *node* sensor menuju ke BS. Selain itu, pada IMHRP-P tidak dilakukan pemilihan LN seperti pada PEGASIS. Hal ini dapat menghemat waktu dalam pembentukan rute untuk pengiriman paket data.



Gambar 3.9 *Flowchart* PEGASIS



Gambar 3.10 *Flowchart* IMHRP-P

3.6 Protokol Modul *Wireless*

Pada proses pengiriman paket data dilakukan sesuai dengan format protokol yang digunakan oleh modul *wireless*. Dalam setiap satu kali pengiriman jumlah paket yang dikirimkan sebanyak 329 bit. Dimana jumlah ini didapatkan dari penjumlahan total dari masing-masing bagian format protokol yang akan digunakan. Adapun format protokol modul *wireless* ditunjukkan pada Gambar 3.11 adalah sebagai berikut :

Preamble 8 bit	Address 40 bit	Flag 9 bit	Payload 256 bit	CRC 16 bit

Gambar 3.11 Format Protokol Modul *Wireless* NRF24L01

Keterangan :

1. *Preamble* : pada bagian ini berfungsi untuk menentukan level 0 atau 1. Dimana level tersebut berfungsi menjadikan modul *wireless* NRF24L01 sebagai pengirim (TX) atau penerima (RX).
2. *Address* : pada bagian ini berisikan alamat pengirim dan penerima yang memiliki ukuran alamat maksimum sebanyak 40 bit.
3. *Flag* : pada bagian ini terdapat PID berfungsi untuk mendeteksi apakah paket yang diterima adalah baru atau dikirim ulang. PID ini ditambahkan di sisi TX untuk setiap paket baru yang diterima melalui *interface* SPI.
4. *Payload* : pada bagian ini berisikan paket data yang akan dikirim dengan ukuran sebesar 256 bit.
5. CRC : pada bagaian ini berfungsi untuk mengirimkan paket data ulang ketika ada paket data yang rusak atau hilang. Akan tetapi panjang paket data yang dikirimkan ulang adalah sebesar 16 bit.

3.7 Perhitungan Konsumsi Energi

Untuk melakukan perhitungan konsumsi energi yang digunakan pada pemodelan *routing* protokol PEGASIS dan IMHRP-P terdapat lima tahap, diantaranya :

1. Perhitungan Jarak

Pada tahap ini terlebih dahulu melakukan perhitungan jarak masing-masing *node* sensor sensor terhadap BS dan *node* sensor yang lain pada luas area 15m x 15m dimana *node* sensor diletakkan secara random sesuai dengan peletakan *node* sensor yang digunakan pada sistem monitoring seperti yang ditunjukkan Gambar 3.3. Dimana pada perhitungan ini menghitung titik koordinat X dan Y dari posisi BS dan masing-masing *node* sensor. Untuk melakukan perhitungan jarak ini, dapat menggunakan rumus *euclidian distance* untuk menentukan jarak antara dua titik

koordinat yang mengacu pada persamaan (2.1) baik untuk skenario pertama maupun kedua seperti yang ditunjukkan pada Gambar 3.3 dan 3.6. Setelah diketahui jarak masing-masing *node* sensor terhadap BS dan *node* sensor yang lain, kemudian dilakukan pengujian jarak jangkau maksimum dari modul *wireless* yang digunakan. Berikut ini contoh perhitungan jarak dari BS ke *node* sensor N1 dengan skenario BS yang pertama dimana sebagai berikut.

Diketahui :

BS : x_1 = 0,46 m

y_1 = 0,69 m

Node sensor sensor N1 : x_2 = 1,5 m

y_2 = 1,4 m

$$euclidian\ distance = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

$$euclidian\ distance = \sqrt{(0{,}46 - 1{,}5)^2 + (0{,}69 - 1{,}4)^2}$$

$$euclidian\ distance = \sqrt{1{,}08 + 0{,}5}$$

$$euclidian\ distance = 1{,}25 \text{ m}$$

Dari hasil perhitungan jarak dari BS menuju ke *node* sensor N1 adalah sejauh 1,25 m. Jarak inilah yang digunakan untuk menentukan jenis daya pancar yang akan digunakan perangkat *node* sensor berdasarkan dari hasil pengujian jarak jangkau modul *wireless* NRF24L01. Dimana jenis penggunaan daya pancar yang digunakan akan berpengaruh terhadap daya (Watt) dari masing-masing perangkat *node* sensor.

2. Pengujian Jarak Jangkau Maksimum Modul *Wireless* NRF24L01

Pada tahap ini, dilakukan pengujian jarak jangkau maksimum dari modul *wireless* yang akan digunakan. Hal ini bertujuan untuk mengetahui berapa jenis daya pancar yang akan digunakan untuk melakukan pemodelan *routing* protokol PEGASIS dan IMHRP-P berdasarkan hasil perhitungan jarak antara *node* sensor menuju ke BS maupun jarak *node* sensor satu dengan yang lainnya seperti perhitungan yang telah dilakukan pada tahap sebelumnya. Semakin besar daya pancar yang digunakan maka semakin besar pula daya yang digunakan. Sehingga berpengaruh terhadap konsumsi energi pada masing-masing pemodelan *routing* protokol.

Untuk melakukan pengujian jarak jangkau maksimum dari modul *wireless*, dilakukan pembuatan perangkat *node* sensor dan BS seperti yang ditunjukkan pada Gambar 3.13 dan 3.15. Dimana pada pengujian ini, *node* sensor difungsikan sebagai *client* dan BS difungsikan sebagai server. Modul *wireless* NRF24L01 memiliki 4 jenis daya pancar, yaitu : -18 dBm, -12 dBm, -6 dBm dan 0 dBm. Pengujian jarak jangkau modul *wireless* diukur dengan menggunakan *baud rate* diatur sebesar 9600 bit per *second* (bps) dan pengujian dilakukan pada masing-masing perubahan daya dalam kondisi LOS (*Line of Sight*) antar perangkat dengan keadaan antenna mikrostrip modul *wireless* lurus berhadapan 180° dengan menggunakan alat ukur meter. Saat *client* mengirim informasi "siap" dan server tidak dapat menerima, maka server digeser maju menuju *client* hingga server mendapat informasi "siap" yang ditunjukkan di serial monitor *software* Arduino IDE, sehingga dapat ditemukan jarak jangkauan maksimal dan saat itu juga pengukuran jarak jangkau maksimum mulai dilakukan.

3. Pengujian Perangkat Monitoring

Pada tahap pengujian perangkat monitoring ini bertujuan untuk menguji kapasitas daya baterai dari perangkat *node* sensor dengan beberapa jenis penggunaan daya pancar dari modul *wireless*. Untuk penggunaan daya pancar disesuaikan dengan hasil perhitungan jarak dari *node* sensor sensor menuju ke BS ataupun jarak dari masing-masing *node* sensor sensor ke *node* sensor lainnya. Dimana jarak akan berpengaruh pada penggunaan daya dari masing-masing *node* sensor. Pada pengujian ini dilakukan dengan cara perangkat *node* sensor digunakan secara terus menerus untuk melakukan pengambilan data berupa suhu, kelembaban udara dan kelembaban tanah melalui sensor yang digunakan dan mengirim data tersebut menuju ke BS pada saat kondisi baterai penuh hingga baterai sudah tidak mampu memberikan daya ke *node* sensor untuk beroperasi. Untuk mengetahui daya baterai dalam kondisi penuh, dilakukukan pengecekan nilai tegangan dan arus baterai menggunakan AVO meter. Dari hasil pengecekan, dapat diketahui energi baterai dalam kondisi penuh menggunakan persamaan (3.1). Dimana 1 Watt = 1 Joule/detik. Adapun contoh perhitungan adalah sebagai berikut jika diketahui tegangan sebesar 3,7 Volt dan arus 2 Ah.

$$P = V\ X\ I \qquad (3.1)$$

Dimana :

P = daya listrik dengan satuan Watt (W)

V = tegangan listrik dengan satuan Volt (V)

I = arus listrik dengan satuan Ampere (A)

Maka :

$P_{penuh} = V\ X\ I$

$P_{penuh} = 3{,}7\ Volt\ X\ 2\ A$

$P_{penuh} = 7{,}4\ Watt$

Setelah diketahui nilai energi pada saat baterai dalam kondisi penuh, kemudian dilakukan perhitungan energi yang digunakan dalam satu jenis daya pancar. Contoh nilai arus yang digunakan daya pancar -18 dBm adalah 0,05 A dan tegangan baterai sebesar 3,7 Volt, maka :

P_{-18dBm} = $V\ X\ I$

P_{-18dBm} = 3,7 $Volt\ X$ 0,05 A

P_{-18dBm} = 0,185 $Watt$

Dari hasil perhitungan daya -18 dBm kemudian dilakukan perhitungan energi pemakaian 0,185 Watt dalam waktu 13 jam. Dimana kurun waktu tersebut yang digunakan baterai untuk memberikan energi pada perangkat *node* sensor dimulai dalam kondisi penuh hingga tidak dapat memberikan energi pada perangkat *node* sensor. Adapun contoh perhitungannya sebagai berikut menggunakan persamaan (3.2).

$Wh = P_{-18dBm}\ x\ Jam$ (3.2)

$Wh = 0{,}185\ Watt\ x\ 13\ Jam$

$Wh = 2{,}4$

Dari hasil perhitungan ini dapat diketahui bahwa dengan menggunakan daya pancar -18 dBm, perangkat *node* sensor dapat bertahan hingga 13 jam dan jumlah energi yang digunakan dalam 13 jam adalah 2,4 Wh.

4. Perhitungan Kapasitas Baterai

Setelah diketahui energi yang digunakan pada setiap daya pancar modul *wireless* yang digunakan, selanjutnya dilakukan pengujian terhadap kapasitas baterai yang dapat digunakan untuk memberikan energi pada *node* sensor sensor. Pengujian ini dilakukan karena tegangan yang dimiliki oleh baterai tidak sepenuhnya dapat digunakan untuk memberikan daya pada *node* sensor seperti yang sudah dijelaskan pada tahap sebelumnya. Misalnya baterai memiliki tegangan pada saat penuh sebesar 3,7 Volt dan baterai sudah tidak bisa memberikan energi perangkat *node* sensor sebesar 2,5 Volt. Dimana tegangan baterai 2,5 Volt merupakan *Depth of Discharge* (DOD) yang dapat digunakan untuk memberikan daya perangkat *node* sensor. Adapun contoh perhitungannya adalah sebagai berikut yang menggunakan persamaan (3.3)

$$Tegangan_{DOD} = Tegangan_{total} - Tegangan_{drop} \quad (3.3)$$

$Tegangan_{DOD}$ = 3,7 Volt – 2,5 Volt

$Tegangan_{DOD}$ = 1,2 Volt

Setelah diketahui tegangan DOD, selanjutnya dilakukan perhitungan mencari persentase tegangan DOD yang didapatkan dari persentase tegangan total baterai sebesar 3,7 Volt menggunakan persamaan (3.4) sebagai berikut.

$$Persentase_{DOD} = \frac{Tegangan_{DOD}}{Tegangan_{total}} \text{ x } 100\% \quad (3.4)$$

$Persentase_{DOD} = \frac{1{,}2\ Volt}{3{,}7\ Volt}$ x 100%

$Persentase_{DOD}$ = 32,4%

Setelah diketahui persentase DOD, selanjutnya dapat menghitung kapasitas baterai Jam Ampere pada saat terjadi DOD menggunakan persamaan (3.5) sebagai berikut.

$$C_{DOD} = Persentase_{DOD}\ x\ kapasitas\ baterai \quad (3.5)$$

$C_{DOD} = \frac{32{,}4}{100}\ x\ 2\ Ah$

C_{DOD} = 0,64 Ah

5. Perhitungan Daya Baterai Perangkat

Pada tahap ini dilakukan perhitungan daya baterai yang digunakan dalam pemodelan *routing* protokol PEGASIS maupun IMHRP-P baik untuk skenario pertama maupun kedua. Setelah diketahui kapasitas baterai DOD, kemudian dilakukan perhitungan daya pada saat baterai dalam kondisi DOD. Dimana sudah diketahui persentase baterai pada saat kondisi DOD, yaitu sebesar 32,4%. Maka dapat dilakukan perhitungan daya DOD menggunakan persamaan (3.6) sebagai berikut.

$$P_{DOD} = Persentase_{DOD}\ x\ P_{penuh} \quad (3.6)$$

$$P_{DOD} = \frac{32,4}{100}\ x\ 7,4\ Watt$$

$$P_{DOD} = 2,3\ Wh$$

Setelah diketahui daya DOD dari baterai, kemudian dilakukan perhitungan daya yang digunakan perangkat *node* sensor berdasarkan jenis dari daya pancar. Contoh dalam luas area 15m x 15m dengan *node* sensor sebanyak 5 dan 1 BS, dimana semua *node* sensor sensor menggunakan daya pancar -18 dBm, maka daya yang digunakan adalah 0,185 Watt. Dimana nilai 0,185 Watt didapatkan dari hasil perhitungan menggunakan persamaan (3.1). Setelah diketahui daya yang digunakan pada masing-masing *node* sensor, kemudian dilakukan perhitungan rata-rata daya yang digunakan masing-masing *node* sensor tersebut. Seperti yang dicontohkan pada persamaan (3.7) dengan pemodelan *routing* protokol PEGASIS pada skenario pertama.

$$P_{PEGASIS1} = \frac{Daya\ perangkat\ masing{-}masing\ node\ sensor}{Jumlah\ node\ sensor} \quad (3.7)$$

$$P_{PEGASIS1} = \frac{P_{-18\ dBm} + P_{-18\ dBm} + P_{-18\ dBm} + P_{-18\ dBm} + P_{-18\ dBm}}{5}$$

$$P_{PEGASIS1} = \frac{0,185\ Watt + 0,185\ Watt + 0,185\ Watt + 0,185\ Watt + 0,185\ Watt}{5}$$

$$P_{PEGASIS1} = 0,185\ Watt$$

Dari hasil perhitungan yang telah dilakukan, didapatkan daya yang digunakan PEGASIS sebesar 0,185 Watt. Selanjutnya dilakukan perhitungan daya DOD yang dibagi dengan daya $P_{PEGASIS1}$. Hal ini bertujuan untuk mengetahui *Network Lifetime* (NL) dari semua *node* sensor pada PEGASIS dengan skenario pertama. Adapun contoh perhitungannya menggunakan persamaan (3.8) sebagai berikut.

$$NL_{PEGASIS1} = \frac{P_{DOD}}{P_{PEGASIS1}} \quad (3.8)$$

$$NL_{PEGASIS1} = \frac{2{,}3\ Wh}{0{,}185\ Wh}$$

$NL_{PEGASIS1}$ = 12,4 jam

Dari hasil contoh perhitungan menunjukkan bahwa daya DOD sebesar 2,19 Wh dapat bertahan hingga 12,4 jam dengan daya rata-rata yang digunakan adalah 0,185 Watt. Setelah diketahui daya rata-rata perangkat yang digunakan pada persamaan (3.7) untuk satu kali pengiriman paket data menuju ke BS dimana setiap satu kali pengiriman dilakukan setiap 30 menit sekali. Dimana dalam 1 jam melakukan pengiriman sebanyak 2 kali. Kemudian dapat dilakukan perhitungan daya (Watt) yang akan digunakan dalam kurun waktu 1, 6 dan 12 bulan dengan cara mengalikan berapa banyak pengiriman dalam 1 jam dengan jumlah jam dalam 1,2 dan 12 bulan dan daya yang digunakan perangkat menggunakan persamaan (3.9) sebagai berikut. Dimana 1 Watt = 1 Joule/detik.

Diketahui :

1 bulan = 720 jam

Maka :

1 bulan = 2 x 720 x $P_{PEGASIS1}$

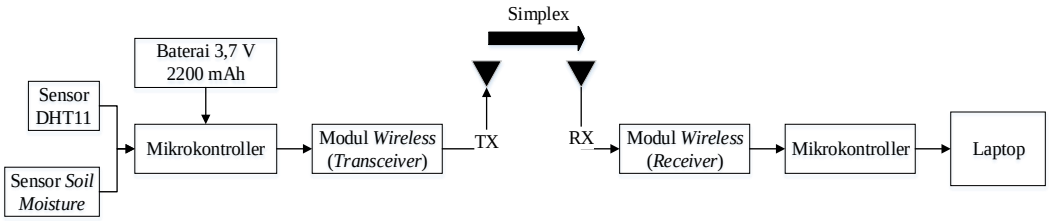
1 bulan = 2 x 720 x 0,185 Watt

1 bulan = 266,4 Watt

Dari hasil contoh perhitungan, maka daya (Watt) yang digunakan dalam kurun waktu 1 bulan sebanyak 266,4 Watt.

3.8 Diagram Blok Sistem Komunikasi

Pada sub bab ini menjelaskan tentang diagram blok sistem komunikasi antara *node* sensor dengan BS. Dalam satu perangkat *node* sensor terdiri dari sensor DHT11, *soil moisture*, mikrokontroller, baterai dan modul *wireless*. Sedangkan dalam satu perangkat BS terdiri dari modul *wireless*, mikrokontroller dan laptop. Seperti yang ditunjukkan pada Gambar 3.12.

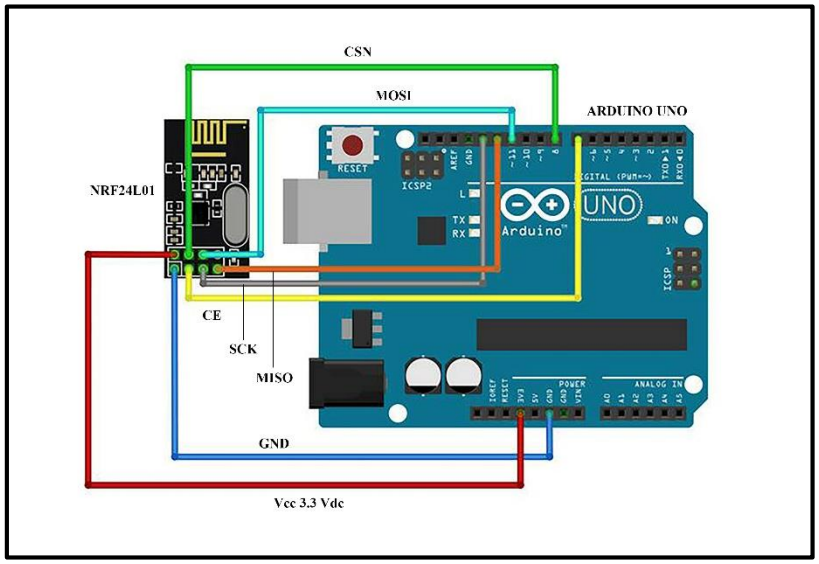


Gambar 3.12 Diagram Blok Komunikasi Antara *Node* Sensor dan BS

Gambar 3.12 menunjukkan diagram blok komunikasi antara *node* sensor dan BS. Dimana dalam satu perangkat *node* sensor dicatu oleh baterai dengan tegangan 3,7 Volt dan arus sebesar 2200 mAh. *Node* sensor memperoleh input data dari sensor DHT11 dan *soil moisture*. Kemudian data tersebut akan diproses oleh mikrokontroller Arduino Nano. Dimana pada mikrokontroller akan diberikan *header* sesuai dengan format pengiriman data dari modul *wireless* NRF24L01 seperti yang ditunjukkan dalam Gambar 3.11. Kemudian data yang sudah diberubah menjadi paket data yang sudah dilengkapi dengan *header* yang digunakan, selanjutnya paket data tersebut dikirimkan menuju ke BS. Dimana pada proses pengiriman dari *node* sensor ke BS menggunakan metode transmisi *simplex*. *Simplex* merupakan salah satu metode komunikasi antara pemancar (TX) dan penerima (RX) yang dikirim secara satu arah. Kemudian paket data diterima oleh BS melalui modul *wireless* NRF24L01, setelah itu paket data diproses oleh mikrokontroller Arduino Uno. Dimana pada proses ini dilakukan pengecekan alamat pengirim dan penerima serta dilakukan pengambilan data pada bagian *payload* dari *header* yang digunakan. Kemudian data ini ditampilkan pada laptop menggunakan *software* Arduino IDE.

3.9 Desain BS

Perencanaan desain *hardware* BS yang akan digunakan untuk melakukan pengujian sistem monitoring ditunjukkan dalam Gambar 3.13. Gambar 3.13 memperlihatkan desain skematik *hardware* yang akan digunakan untuk melakukan sistem monitoring yang terdiri dari arduino uno sebagai mikrokontroller dimana berfungsi sebagai pengolah paket data *input* yang masuk dan modul *wireless* NRF24L01 sebagai modul yang mengirimkan paket data hasil pendeteksian sensor ke *node* sensor selanjutnya. Adapun sistem kerja dari BS dijelaskan dalam bentuk *flowchart* seperti yang ditunjukkan pada Gambar 3.14 dan *source coding* dari BS ditampilkan dalam lampiran 1.



Gambar 3.13 Desain BS

Gambar 3.14 menunjukkan *flowchart* sistem kerja dari BS. Adapun penjelasan dari sistem kerja BS adalah sebagai berikut :

Tahap pertama BS akan mengirimkan *request* yang berisikan perintah ke *node* sensor untuk melakukan pengiriman data sensor menuju BS. Dimana untuk melakukan pengiriman data dari *node* sensor menuju ke BS menggunakan modul *wireless* NRF24L01.

Tahap kedua BS akan berubah menjadi mode penerima. Hal ini bertujuan supaya BS bisa menerima paket data yang dikirimkan oleh *node* sensor. Kemudian

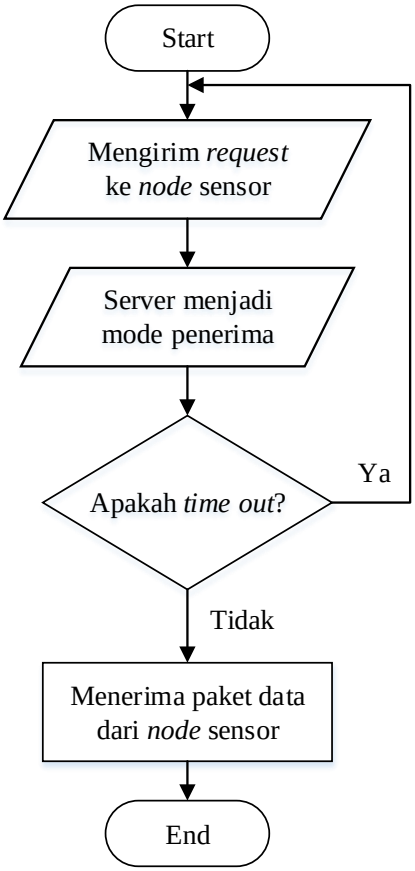
dari *request* yang dikirimkan oleh BS pada *node* sensor apakah terjadi *time out* atau tidak. Dimana *time out* sudah diatur yang memiliki durasi > 4 detik. Jika terjadi *time out* maka BS akan melakukan pengiriman request ulang menuju *node* sensor sampai BS menerima paket data yang dikirimkan oleh *node* sensor. Sedangkan jika tidak terjadi *time out* maka BS akan menerima paket data dari *node* sensor.

3.10 Desain *Node* Sensor

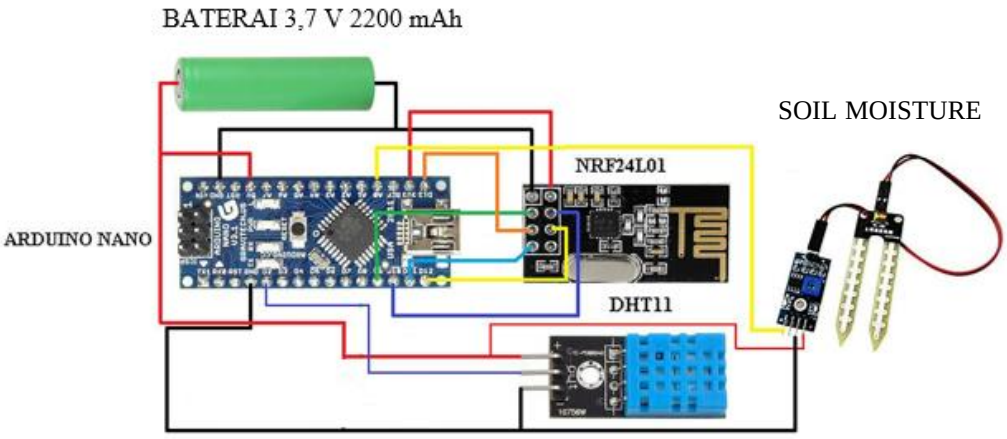
Perencanaan desain *hardware* BS yang akan digunakan untuk melakukan pengujian sistem monitoring ditunjukkan dalam Gambar 3.15. Gambar 3.15 memperlihatkan skematik *node* sensor yang terdiri dari Arduino Nano sebagai mikrokontroller dimana berfungsi sebagai pengolah paket data *input* yang masuk, sensor DHT11 sebagai alat untuk mendeteksi suhu dan kelembaban udara, sensor *soil moisture* sebagai alat untuk mendeteksi kadar kelembaban tanah dan modul *wireless* NRF24L01 sebagai modul yang mengirimkan paket data hasil pendeteksian sensor ke BS. Adapun sistem kerja dari *node* sensor dijelaskan dalam bentuk *flowchart* seperti yang ditunjukkan pada Gambar 3.16 dan *source coding* dari *node* sensor ditampilkan dalam lampiran 2.

Gambar 3.16 menunjukkan *flowchart* sistem kerja dari *node* sensor. Adapun penjelasan dari sistem kerja *node* sensor adalah sebagai berikut :

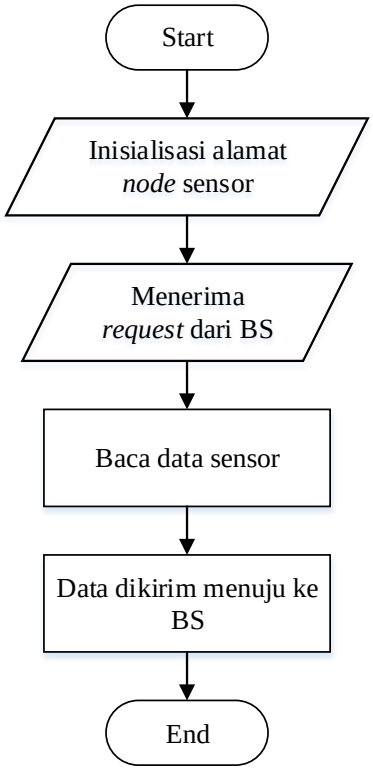
Tahap pertama adalah melakukan inisialisasi alamat dari masing-masing *node* sensor sensor. Kemudian pada tahap kedua adalah *node* sensor menerima *request* yang dikirimkan oleh BS. Dimana *request* tersebut berisikan perintah untuk mengirimkan paket data menuju BS. Setelah *request* diterima oleh *node* sensor, kemudian *node* sensor melakukan pembacaan data sensor, berupa data suhu, kelembaban udara dan kelembaban tanah. Setelah dilakukan pembacaan data sensor, kemudian data tersebut dirubah menjadi paket data yang berisikan *header* alamat pengirim dan penerima sesuai dengan format protokol modul *wireless* yang digunakan seperti yang ditunjukkan pada Gambar 3.11. Setelah *header* sudah lengkap, kemudian paket data dikirim menuju ke BS menggunakan modul *wireless* NRF24L01.



Gambar 3.14 *Flowchart* Sistem Kerja BS



Gambar 3.15 Desain *Node* Sensor



Gambar 3.16 *Flowchart* Sistem Kerja *Node* Sensor

3.11 Parameter Simulasi

Parameter simulasi yang digunakan untuk *routing* protokol PEGASIS dan IMHRP-P ditunjukkan pada Tabel 3.1.

Tabel 3.1 Parameter Simulasi

No.	Parameter yang Digunakan	Nilai
1.	Lokasi *Base Station* (BS)	Tengah dan sudut
2.	Posisi *node* sensor	Fix dan *random*
4.	Luas area	15m x 15 m dan 100 m x 100 m
5.	Jumlah *node* sensor	5, 20 dan 35
6.	Ukuran paket paket data	329 bit
7.	Obstacle	Tidak ada

Tabel 3.1 Memperlihatkan parameter yang digunakan untuk simulasi *routing* protokol PEGASIS dan IMHRP-P. Adapun keterangan dari Tabel 3.1 adalah sebagai berikut :

1. Jumlah *node* sensor : menunjukkan jumlah *node* sensor yang akan digunakan.
2. Jumlah paket data : menunjukkan jumlah paket data yang akan dikirimkan oleh masing- masing *node* sensor pada setiap *round* (putaran).
3. Luas area : menunjukkan luas area yang akan digunakan untuk implementasi sistem monitoring.

3.12 Indikator Kinerja

Untuk menghasilkan penelitian yang baik pada *routing* protokol WSN, diperlukan adanya kinerja sistem. Kinerja yang diukur pada *routing* protokol yaitu melalui perbandingan antara *routing* protokol PEGASIS dan IMHRP-P. Kinerja sistem ini terdiri dari 3 indikator, sebagai berikut :

3.12.1 Jumlah Konsumsi Energi

Pada indikator kinerja ini berfungsi untuk melakukan pengujian terhadap jumlah energi yang digunakan dalam pemodelan *routing* protokol PEGASIS dan IMHRP-P berdasarkan jumlah *node* sensor dan luas area yang digunakan.

3.12.2 *Network Lifetime* (NL) *Routing* Protokol

Pada indikator kinerja ini berfungsi untuk mengetahui berapa jam masing-masing *node* sensor dapat beroperasi. Dimana NL bergantung pada jumlah penggunaan energi dari masing-masing *node* sensor.

3.12.3 Jumlah *Node* Sensor Hidup vs *Round*

Pada indikator kinerja ini menandakan banyaknya *node* sensor yang masih memiliki energi untuk mengirimkan paket data setiap *round*. Dimana perhitungan energi dilakukan oleh masing-masing *node* sensor. Sehingga dapat mengetahui berapa banyak jumlah *node* sensor yang masih hidup atau memiliki energi untuk mengirimkan paket data di setiap *round*. Dimana setiap *round* adalah berapa banyak pengulangan pengiriman paket data dilakukan pada saat pembuatan rute oleh *node*

sensor hingga paket data diterima oleh BS. Pada setiap *round* atau satu kali pengiriman menuju BS memiliki jadwal 30 menit sekali.

Halaman ini sengaja dikosongkan

BAB 4
HASIL DAN PEMBAHASAN

Bab IV ini berisi tentang pembahasan secara keseluruhan dari hasil desain dan perancangan *hardware, software* dan membahas tentang hasil pengujian kinerja sistem serta pembahasannya. Adapun tujuan pengujian sistem adalah untuk mengetahui sistem yang dirancang dapat berfungsi dan bekerja sesuai dengan perencanaan. Pada perencanaan ini akan dijelaskan mengenai gambaran sistem komunikasi pada pembacaan sensor DHT11, sensor *soil moisture,* konsumsi energi dan perbandingan pemodelan *routing* protokol.

4.1 Hasil Implementasi *Hardware*

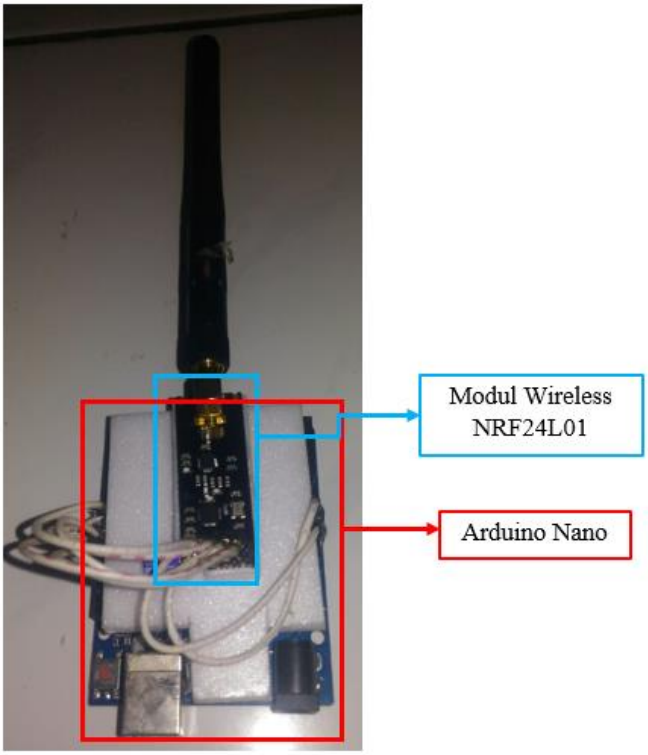
Pada hasil implementasi *hardware* akan menjelaskan mengenai rangkaian *hardware* yang akan digunakan sebagai sistem monitoring kondisi di dalam *greenhouse,* diantaranya : arduino uno, arduino nano, sensor DHT11, sensor *soil moisture* dan modul *wireless* NRF24L01.

4.1.1 Implementasi BS

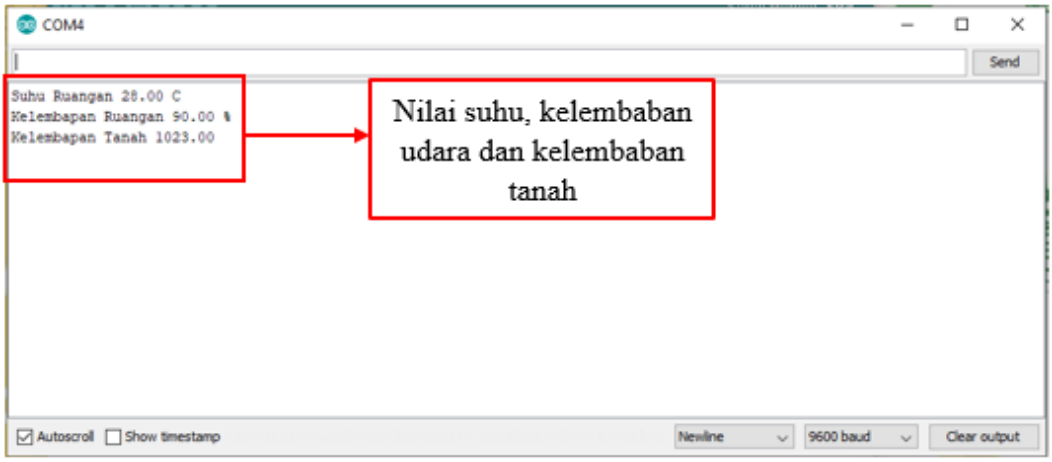
Dalam implementasi BS, terdapat beberapa *hardware* yang digunakan diantaranya : arduino uno dan modul *wireless* NRF24L01. Arduino uno berfungsi untuk menyimpan perintah untuk menerima paket data dari *node* sensor melalui modul *wireless* NRF24L01. Sedangkan modul *wireless* NRF24L01 berfungsi untuk menerima paket data dari *node* sensor. Selain itu, modul *wireless* ini dilengkapi dengan antena eksternal yang berfungsi untuk menerima pesan dari masing-masing *node* sensor dengan jarak jangkau yang lebih luas. Adapun komponen yang digunakan untuk implementasi *hardware* dari BS, ditunjukkan dalam Gambar 4.1.

Gambar 4.1 menunjukkan implementasi *hardware* BS. BS berfungsi untuk menyimpan semua paket data berupa suhu, kelembaban udara dan kelembaban tanah yang dikirim oleh *node* sensor. Paket data yang disimpan pada BS akan ditampilkan dalam *software* Arduino IDE sehingga dapat mengetahui kondisi didalam *greenhouse.* Adapun tampilan paket data yang diterima oleh BS

ditunjukkan dalam Gambar 4.2. Dari Gambar 4.2 menunjukkan tampilan paket data yang diterima oleh BS dari masing-masing *node* sensor. Dimana tampilan tersebut berisikan paket data berupa suhu, kelembaban udara dan kelembaban tanah.



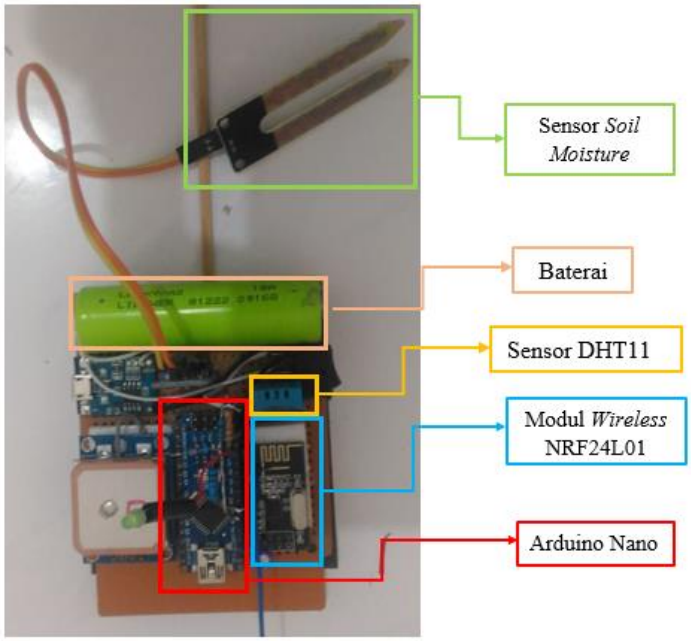
Gambar 4.1 Implementasi *Hardware* BS



Gambar 4.2 Tampilan *Software* Arduino IDE

4.1.2 Implementasi *Node* Sensor

Node sensor berfungsi untuk menghasilkan paket data suhu, kelembaban udara dan kelembaban tanah. Untuk paket data suhu dan kelembaban udara diperoleh dari hasil pendeteksian sensor DHT11 sedangkan untuk paket data kelembaban tanah diperoleh dari hasil pendeteksian sensor *soil moisture*. Adapun komponen yang digunakan untuk implementasi *node* sensor ditujukkan dalam Gambar 4.3.



Gambar 4.3 Implementasi *Node* Sensor

Gambar 4.3 menunjukkan komponen yang digunakan untuk menyusun *node* sensor. Selain terdapat 2 sensor, dalam satu perangkat *node* sensor terdapat Arduino Nano dan modul *wireless* NRF24L01. Arduino Nano berfungsi untuk menyimpan perintah untuk melakukan pembacaan suhu, kelembaban udara, kelembaban tanah serta mengirimkan paket data pembacaan tersebut melalui modul *wireless* NRF24L01. Sedangkan modul *wireless* NRF24L01 berfungsi sebagai mengirimkan paket data ke *node* sensor selanjutnya. Selain itu, untuk mengoperasikan *node* sensor digunakanlah catu daya berupa baterai yang memiliki kapasitas daya sebesar 2,2 Ah.

4.2 Hasil Perhitungan Jarak

Pada tahap ini akan dijelaskan contoh perhitungan jarak dari *node* sensor menuju ke BS maupun antar *node* sensor satu dengan yang lain baik untuk skenario pertama dan kedua untuk pemodelan *routing* protokol PEGASIS dan IMHRP-P. Untuk contoh perhitungan jarak digunakan luas area 15 m x 15 m dengan menghitung titik koordinat X dan Y dari posisi BS dan masing-masing *node* sensor. Secara implementasi *hardware* dapat menggunakan kekuatan sinyal RSSI sebagai *threshold* untuk menentukan rute terdekat. Akan tetapi pada simulasi ini menggunakan perhitungan rute terdekat menggunkan rumus *euclidian distance*. Dimana perhitungan jarak yang dilakukan menggunakan rumus *euclidian distance* seperti pada persamaan (2.1) baik untuk skenario pertama maupun kedua seperti yang ditunjukkan pada Gambar 3.3 dan 3.6.

4.2.1 Skenario Pertama

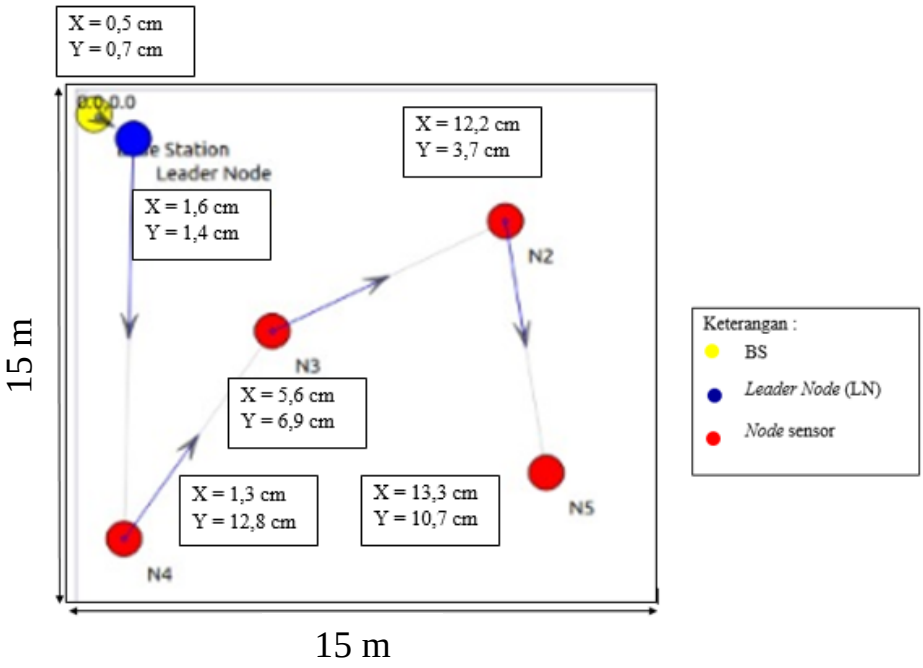
Adapun hasil perhitungan yang telah dilakukan untuk pemodelan *routing* protokol PEGASIS dan IMHRP-P untuk skenario pertama ditunjukkan dalam Tabel 4.1. Dari hasil perhitungan jarak yang ditunjukkan pada Tabel 4.1 kemudian dilakukan pemodelan *routing* protokol PEGASIS pada *software* simulasi NS3. Dimana pada simulasi NS3 diasumsikan tidak halangan sama sekali. Sehingga tidak diperlukan pengaturan kanal supaya tidak terjadi interferensi antar frekuensi. Adapun tampilan pemodelan *routing* protokol PEGASIS pada *software* NS3 dengan skenario pertama dimana jumlah *node* sensor sebanyak 5 dan posisi BS berada di sudut ditunjukkan pada Gambar 4.4.

Tabel 4.1 Hasil Perhitungan Jarak BS dan *Node* Sensor dengan Skenario Pertama

No.	Keterangan	*Node* Sensor 1	*Node* Sensor 2	*Node* Sensor 3	*Node* Sensor 4	*Node* Sensor 5
1.	BS	1,35 m	12,2 m	8 m	12,1 m	16,4 m
2.	*Node* Sensor 1	-	10,9 m	6,7 m	11,4 m	15,1 m
3.	*Node* Sensor 2	10,9 m	-	7,3 m	14,1 m	7,2 m

Tabel 4.1 Hasil Perhitungan Jarak BS dan *Node* Sensor dengan Skenario Pertama (Lanjutan)

No.	Keterangan	*Node* Sensor 1	*Node* Sensor 2	*Node* Sensor 3	*Node* Sensor 4	*Node* Sensor 5
4.	*Node* Sensor 3	6,7 m	7,3 m	-	7,3 m	8,8 m
5.	*Node* Sensor 4	11,4 m	14,1 m	7,3 m	-	12,1 m
6.	*Node* Sensor 5	15,1	7,2 m	8,8 m	12,1 m	-



Gambar 4.4 Pemodelan *Routing* Protokol PEGASIS Skenario Pertama

Dari Gambar 4.4 menunjukkan tampilan pemodelan *routing* protokol PEGASIS dengan jumlah *node* sensor sebanyak 5 pada *software* simulasi NS3 dengan skenario pertama. Adapun alur kerja *routing* protokol PEGASIS dari hasil simulasi pada *software* NS3 adalah sebagai berikut :

1. Pemilihan LN

Pada tahap pertama, dilakukan perhitungan jarak antar *node* sensor dengan BS. Seperti yang tunjukkan pada Tabel 4.1. Dari Tabel 4.1 dapat diketahui jarak terdekat dan terjauh *node* sensor menuju BS. Berdasarkan hasil perhitungan jarak

pada Tabel 4.1 yang menjadi LN adalah *node* sensor N1 karena *node* sensor N1 memiliki jarak yang paling dekat dengan BS, yaitu 1,35 m. Untuk LN akan berubah-ubah sesuai dengan kapasitas daya baterai yang dimiliki oleh masing-masing *node* sensor disetiap *round*.

2. Pembentukan Rantai (*Chain Establishment*)

Setelah diketahui jarak antar *node* sensor dengan BS seperti pada Tabel 4.1, dapat diketahui jarak terjauh dari *node* sensor menuju BS adalah *node* sensor N5. Kemudian melakukan perhitungan jarak antar *node* sensor. Hal ini bertujuan supaya masing-masing *node* sensor dapat menentukan *node* sensor tetangga terdekatnya. Berdasarkan hasil perhitungan jarak antar *node* sensor seperti yang ditunjukkan pada Tabel 4.1, maka pembentukan rantai akan ditunjukkan dalam Tabel 4.2.

Tabel 4.2 Pembentukan Rantai PEGASIS dengan Skenario Pertama

No.	Rute	Jarak (m)
1.	*Node* sensor N5 - *node* sensor N2	7,2
2.	*Node* sensor N2 - *node* sensor N3	7,3
3.	*Node* sensor N3 - *node* sensor N4	7,3
4.	*Node* sensor N4 - *node* sensor N1	11,4
5.	*Node* sensor N1 - BS	1,35

Tabel 4.2 menunjukkan pembentukan rantai PEGASIS dengan skenario BS di sudut. Pembentukan rantai dimulai dari *node* sensor terjauh dari BS, yaitu *node* sensor sensor N5. Dari *node* sensor N5 akan mencari *node* sensor tetangga terdekat dari *node* sensor sensor N5 adalah *node* sensor N2. Dari *node* sensor N2 memiliki *node* sensor tetangga terdekat N3. Kemudian dari *node* sensor N3 memiliki *node* sensor tetangga terdekat N4. Dari *node* sensor N4 memiliki *node* sensor tetangga terdekat N1. Dari *node* sensor N1 akan membuat rantai menuju BS.

3. Agregasi Paket data (Pengumpulan Paket data)

Setelah pembentukan *chain* dan pemilihan LN, maka agregasi paket data (pengumpulan paket data) dapat dimulai. Dimana paket data yang dikirimkan merupakan data hasil pengujian suhu, kelembaban udara dan kelembaban tanah dengan ukuran paket data secara keseluruhan adalah 6 byte. Paket data yang dikirimkan berukuran sama baik untuk skenario pertama dengan posisi BS berada di sudut maupun untuk skenario kedua dengan posisi BS berada di tengah luas area.

Pada agregasi paket data menggunakan pendekatan berdasarkan token (penanda) digunakan untuk mengumpulkan paket data. BS mengirimkan penanda ke LN kemudian dari LN akan melewatkan penanda disepanjang *chain* hingga ke *node* sensor terakhir seperti yang ditunjukkan pada Gambar 4.4 dengan skenario BS berada di sudut. Setelah menerima *node* sensor terakhir menerima penanda, *node* sensor terakhir mengirimkan paket data dan penanda ke *node* sensor berikutnya sepanjang rantai. Proses ini berlanjut sampai paket data sampai ke LN . Untuk *node* sensor yang berada diposisi tengah, akan melakukan penggabungan paket data dengan menggabungkan paket data dari *node* sensor tetangganya dengan paket data dari *node* sensor itu sendiri dan membuat menjadi satu paket dengan panjang yang sama. Pada saat terakhir, LN akan mengirimkan paket paket data ke BS.

Sedangkan untuk tampilan pemodelan *routing* protokol IMHRP-P pada *software* simulasi NS3 ditunjukkan pada Gambar 4.5. Dari Gambar 4.5 menunjukkan tampilan pemodelan *routing* protokol IMHRP-P pada *software* simulasi NS3 dengan skenario pertama dan jumlah *node* sensor sebanyak 5. Adapun alur kerja *routing* protokol IMHRP-P dari hasil simulasi pada *software* NS3 adalah sebagai berikut :

1. Menentukan Rute

Untuk mulai menentukan rute, terlebih dahulu dilakukan perhitungan jarak antar *node* sensor dengan BS. Seperti yang tunjukkan pada Tabel 4.1. Dari Tabel 4.1 dapat diketahui jarak terdekat dan terjauh *node* sensor menuju BS. Semakin jauh jarak antara *node* sensor dengan BS maka semakin besar pula daya yang digunakan untuk mengirimkan paket data. Semakin besar daya yang digunakan maka kapasitas baterai akan cepat berkurang. Maka untuk mengurangi penggunaan daya yang cukup besar, *node* sensor yang memiliki jarak cukup jauh dari BS, dapat mengirimkan paket data menuju *node* sensor tetangga terdekat. Kemudian dari *node* sensor tetangga terdekat akan mengirimkan data menuju BS.

2. Pembuatan Rute

Pada pembuatan rute untuk skenario BS berada di sudut, *node* sensor terjauh ke 1 dari BS akan mencari *node* sensor tetangga terdekat. Dari *node* sensor tetangga terdekat, akan membuat koneksi langsung menuju BS. Berdasarkan hasil perhitungan jarak antar *node* sensor dengan skenario BS berada di sudut

ditunjukkan pada Tabel 4.1, maka pembuatan rute akan ditunjukkan dalam Tabel 4.3.

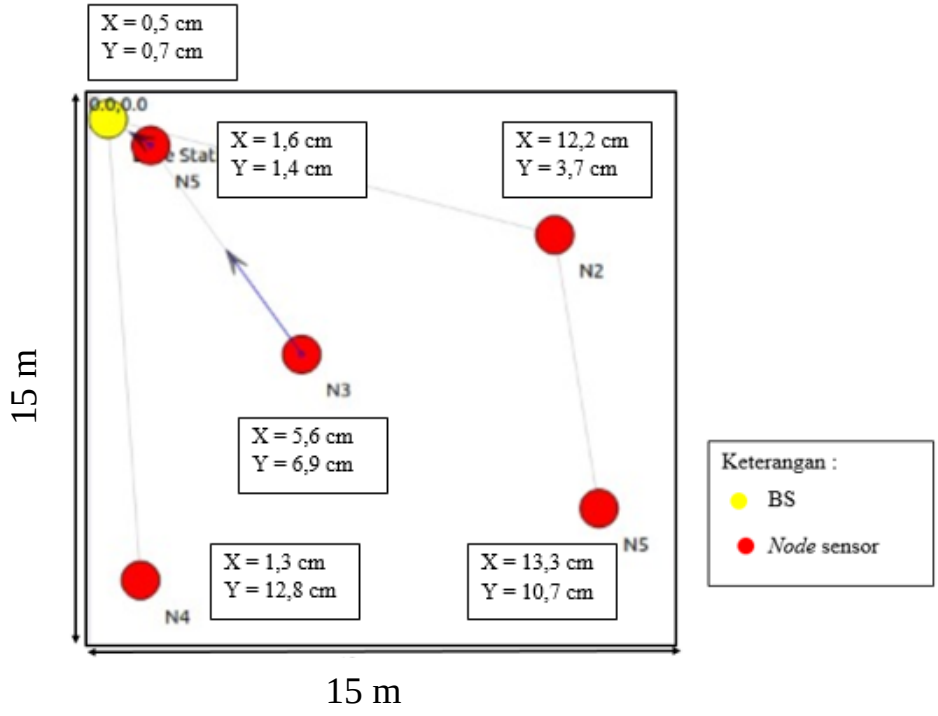
Tabel 4.3 Pembuatan Rute IMHRP-P dengan Skenario Pertama

No.	Rute	Jarak (m)
1.	*Node* sensor N5 - *node* sensor N2	7,2
2.	BS - *node* sensor N2	12,2
3.	*Node* sensor N4 - BS	12,1
4.	*Node* sensor N3 - *node* sensor N1	6,7
5.	*Node* sensor N1 - BS	1,35

Tabel 4.3 menunjukkan pembuatan rute pada IMHRP-P. Untuk pembuatan rute, dimulai dari *node* sensor yang memiliki jarak paling jauh dengan BS, yaitu *node* sensor N5 dan N4. Pembuatan rute dimulasi dari *node* sensor N5, dari *node* sensor N5 akan mencari *node* sensor tetangga terdekatnya yaitu *node* sensor N2. Dari *node* sensor N2 akan membuat koneksi dengan BS. Kemudian *node* sensor N4 akan membuat koneksi langsung menuju BS. Selanjutnya untuk *node* sensor N3 akan mencari *node* sensor tetangga terdekat, yaitu *node* sensor N1. Dari *node* sensor N1 akan membuat koneksi menuju BS.

3. Agregasi Paket data (Pengumpulan Paket data)

Setelah rute terbentuk, selanjutnya paket data akan dirikim menuju BS. Dimana paket data yang dikirimkan merupakan data hasil pengujian suhu, kelembaban udara dan kelembaban tanah dengan ukuran paket data secara keseluruhan adalah 6 byte. Paket data yang dikirimkan berukuran sama baik untuk skenario pertama dengan posisi BS berada di sudut maupun untuk skenario kedua dengan posisi BS berada di tengah luas area. Untuk pengiriman paket data, dilakukan pembuatan *scheduling* (penjadwalan). Hal ini bertujuan supaya tidak terjadi tabrakan data. Pada 30 menit ke 1, pengiriman data menuju BS akan dilakukan oleh *node* sensor N1. Kemudian dari *node* sensor N1 akan mencari *node* sensor tetangga terdekat, yaitu N3. Kemudian paket data dari *node* sensor N1 akan dikirimkan menuju *node* sensor N3. Dari *node* sensor sensor N3 akan mengirimkan paket data *node* sensor N1 dan N3 menuju BS.



Gambar 4.5 Pemodelan *Routing* Protokol IMHRP-P Skenario Pertama

4.2.2 Skenario Kedua

Untuk perbedaan antara skenario pertama dan kedua adalah peletakan posisi dari BS. Dimana pada skenario pertama posisi BS berada di sudut sedangkan skenario kedua posisi BS berada di tengah dengan posisi dari masing-masing *node* sensor tidak mengalami perubahan. Adapun hasil perhitungan yang telah dilakukan untuk pemodelan *routing* protokol PEGASIS dan IMHRP-P untuk skenario kedua ditunjukkan dalam Tabel 4.4.

Tabel 4.4 Hasil Perhitungan Jarak BS dan *Node* Sensor Skenario Kedua

No.	Keterangan	*Node* Sensor 1	*Node* Sensor 2	*Node* Sensor 3	*Node* Sensor 4	*Node* Sensor 5
1.	BS	8,3 m	6 m	2 m	8,2 m	7 m
2.	*Node* Sensor 1	-	10,9 m	6,7 m	11,4 m	15,1 m
3.	*Node* Sensor 2	10,9 m	-	7,3 m	14,1 m	7,2m
4.	*Node* Sensor 3	6,7 m	7,3 m	-	7,3 m	8,8 m

Tabel 4.4 Hasil Perhitungan Jarak BS dan *Node* Sensor Skenario Kedua (Lanjutan)

No.	Keterangan	*Node* Sensor 1	*Node* Sensor 2	*Node* Sensor 3	*Node* Sensor 4	*Node* Sensor 5
5.	*Node* Sensor 4	11,4 m	14,1 m	7,3 m	-	12,1 m
6.	*Node* Sensor 5	15,1	7,2 m	8,8 m	12,1 m	-

Dari hasil perhitungan jarak yang ditunjukkan pada Tabel 4.4 kemudian dilakukan pemodelan *routing* protokol PEGASIS pada *software* simulasi NS3. Dimana pada simulasi NS3 diasumsikan tidak halangan sama sekali. Sehingga tidak diperlukan pengaturan kanal supaya tidak terjadi interferensi antar frekuensi. Tampilan pemodelan *routing* protokol PEGASIS pada *software* NS3 dengan skenario kedua dimana posisi BS berada di sudut ditunjukkan pada Gambar 4.6.

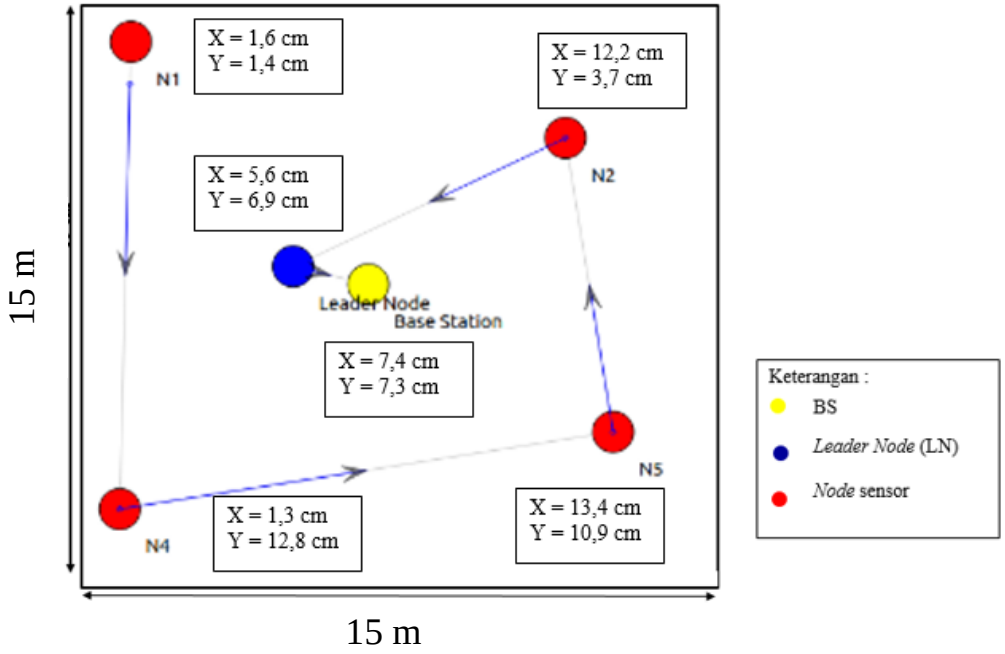
Pada Gambar 4.6 menunjukkan tampilan skenario BS berada di tengah luas area yang digunakan. Adapun alur kerja *routing* protokol PEGASIS dari hasil simulasi pada *software* NS3 sama seperti yang sudah dijelaskan pada skenario pertama. Untuk pemodelan *routing* PEGASIS terdapat 3 fase, yaitu menentukan LN, pembentukan rantai (*chain establishment*) dan agregasi paket data (pengumpulan paket data). Adapun pembentukan rantai untuk pemodelan *routing* PEGASIS pada skenario kedua ditunjukkan dalam Tabel 4.5.

Tabel 4.5 Pembentukan Rantai PEGASIS dengan Skenario Kedua

No.	Rute	Jarak (m)
1.	*Node* sensor N1 - *node* sensor N4	11,4
2.	*Node* sensor N4 - *node* sensor N5	12,1
3.	*Node* sensor N5 - *node* sensor N2	7,2
4.	*Node* sensor N2 – *node* sensor N3	7,3
5.	*Node* sensor N3 - BS	8

Tabel 4.5 menunjukkan pembentukan rantai PEGASIS dengan skenario kedua. Pembentukan rantai dimulai dari *node* sensor sensor terjauh dari BS, yaitu *node* sensor sensor N1. Dari *node* sensor N1 akan mencari *node* sensor tetangga terdekat dari *node* sensor N1 adalah *node* sensor N4. Dari *node* sensor N4 memiliki *node* sensor tetangga terdekat N5. Kemudian dari *node* sensor N5 memiliki *node*

sensor tetangga terdekat N2. Dari *node* sensor N2 memiliki *node* sensor tetangga terdekat N3. Dari *node* sensor N3 akan membuat rantai menuju BS.



Gambar 4.6 Pemodelan *Routing* Protokol PEGASIS Skenario Kedua

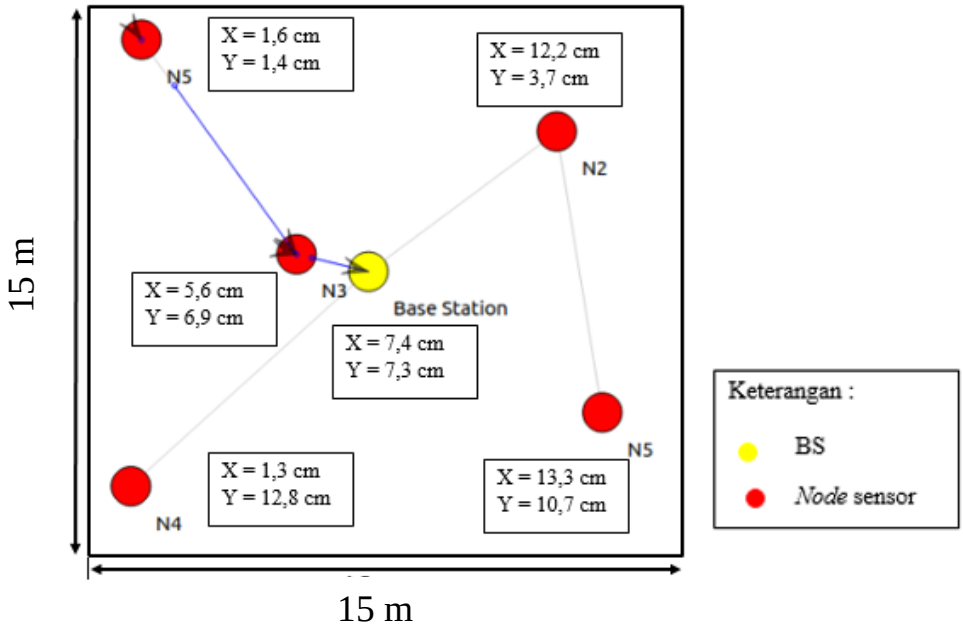
Sedangkan untuk pemodelan *routing* protokol IMHRP-P pada *software* simulasi NS3 ditunjukkan pada Gambar 4.7. Dari Gambar 4.7 menunjukkan tampilan pemodelan *routing* protokol IMHRP-P pada *software* simulasi NS3 dengan skenario kedua. Adapun alur kerja *routing* protokol IMHRP-P sama dengan skenario pertama dimana terdapat 3 tahap, diantaranya : menentukan rute, pembuatan rute dan agregasi paket data (pengumpulan paket data). Adapun pembentukan rantai untuk pemodelan *routing* IMHRP-P pada skenario kedua ditunjukkan dalam Tabel 4.6.

Tabel 4.6 menunjukkan pembuatan rute pada IMHRP-P dengan skenario BS berada di tengah. Untuk pembuatan rute, dimulai dari *node* sensor yang memiliki jarak paling jauh dengan BS, yaitu *node* sensor N1, N4 dan N5. Pembuatan rute dimulai dari *node* sensor N1 dimana *node* sensor N1 langsung membuat koneksi dengan *node* sensor N3. Dari *node* sensor N3 langsung membuat koneksi dengan BS. Kemudian pembuatan rute selanjutnya adalah *node* sensor N4, dari *node* sensor

N4 akan membuat koneksi langsung dengan BS. Kemudian untuk *node* sensor terjauh selanjutnya adalah *node* sensor N5. Dimana *node* sensor N5 akan mencari *node* sensor tetangga terdekatnya, yaitu *node* sensor N2. Dari *node* sensor N2 akan membuat koneksi langsung dengan BS.

Tabel 4.6 Pembuatan Rute IMRP-P dengan Skenario Kedua

No.	Rute	Jarak (m)
1.	*Node* sensor N1 - *node* sensor N3	6,7
2.	*Node* sensor N3 - BS	2
3.	*Node* sensor N5 - *node* sensor N2	7,2
4.	*Node* sensor N2 - BS	6
5.	*Node* sensor N4 - BS	8,2

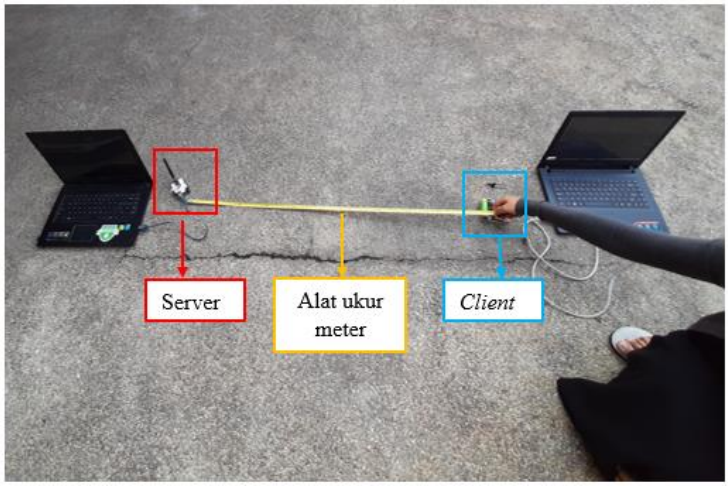


Gambar 4.7 Pemodelan *Routing* Protokol IMHRP-P Skenario Kedua

4.3 Hasil Pengujian Jarak Jangkau Modul *Wireless* NRF24L01

Pada pengujian jarak jangkau modul *wireless* NRF24L01 dilakukan bertujuan untuk mengetahui jarak maksimal pengiriman paket data antar modul *wireless* NRF24L01. Jarak yang diukur adalah jarak dari setiap daya pancar modul *wireless* NRF24L01. Pada pengujian ini menggunakan dua modul *wireless* NRF2401. Satu

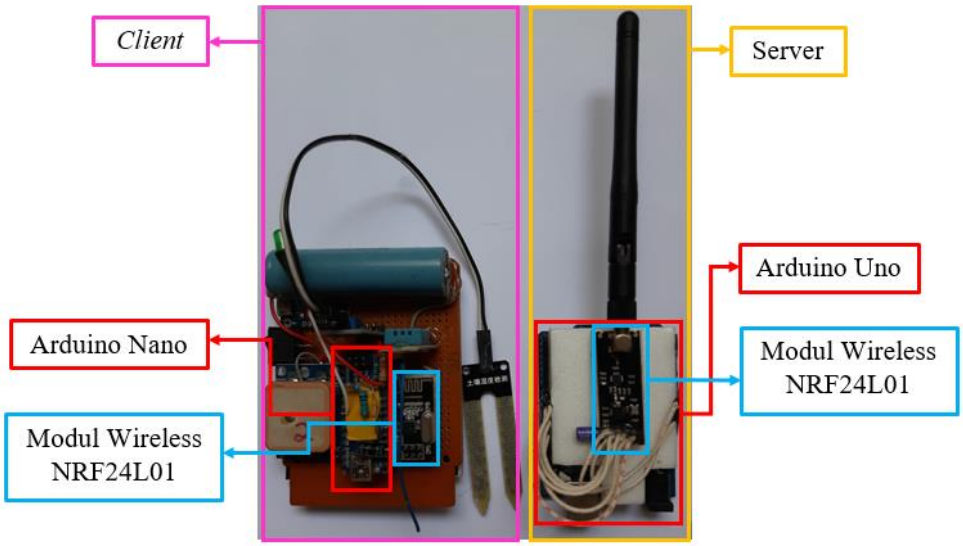
modul berfungsi sebagai server dan satu modul berfungsi sebagai *client*. Modul *wireless* NRF24L01 memiliki 4 jenis daya pancar, yaitu : -18 dBm, -12 dBm, -6 dBm dan 0 dBm. Pengukuran jarak dilakukan pada masing-masing perubahan daya dalam kondisi LOS (*Line of Sight*) antar perangkat dengan keadaan antenna mikrostrip modul *wireless* NRF24L01 seperti yang ditunjukkan pada Gambar 4.8.



Gambar 4.8 Cara Pengukuran Jarak Jangkau Modul *Wireless* NRF24L01

Gambar 4.8 memperlihatkan cara pengukuran jarak jangkau modul *wireless* NRF24L01. Pada pengujian ini menggunakan alat ukur meter yang berfungsi untuk mengetahui jarak maksimum antara server dan *client* untuk bisa saling berkomunikasi. Adapun komponen yang digunakan pada perangkat server dan *client* ditunjunkkan pada Gambar 4.9. Gambar 4.9 memperlihatkan perangkat yang digunakan untuk melakukan pengujian jarak jangkau modul *wireless* NRF24L01. Pada pengujian ini, server mengirimkan informasi berupa kata “siap” kepada *client* dan *client* mendapat informasi “siap” yang ditunjukkan di serial monitor *software* Arduino IDE, sehingga dapat ditemukan jarak jangkauan maksimal dan saat itu juga pengukuran mulai dilakukan. Adapun tampilan serial monitor dari *software* Arduino IDE ditunjukkan dalam Gambar 4.10. Dari pengujian yang telah dilakukan, didapatkan jarak jangkau modul *wireless* NRF4L01 yang ditunjukkan pada Tabel 4.7.

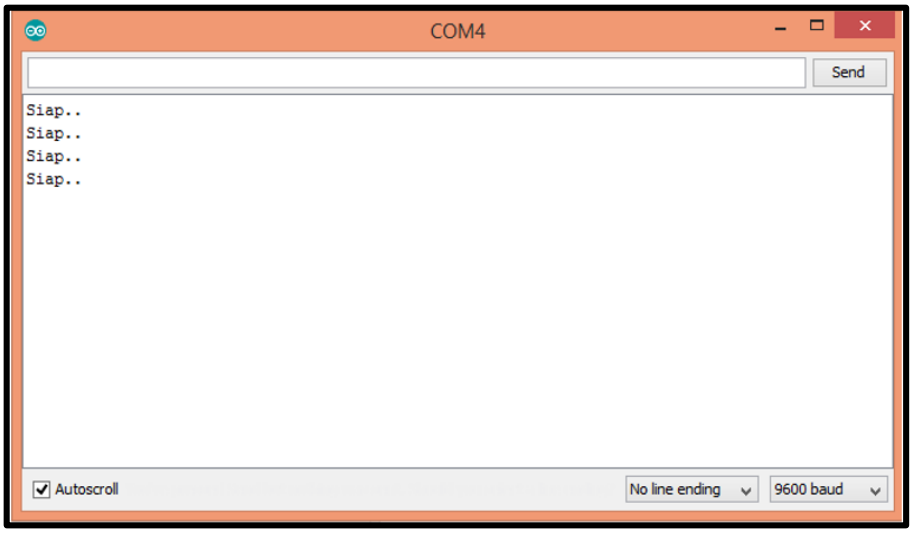
Tabel 4.7 menunjukkan hasil pengukuran jarak jngkau modul *wireless* NRF24L01. Dari hasil pengujian, menunjukkan bahwa jarak jangkau terpendek adalah 10,5 meter dengan daya pancar sebesar -18 dBm. Sedangkan untuk jarak jangkau terjauh adalah 37,8 meter dengan daya pancar sebesar 0 dBm.



Gambar 4.9 Perangkat Pengujian Jarak Modul *Wireless* NRF24L01

Tabel 4.7 Hasil Pengukuran Jarak Jangkau Modul *Wireless* NRF24L01

No.	Daya Pancar (dBm)	Jarak (meter)
1.	-18	10,5
2.	-12	18
3.	-6	26,7
4.	0	37,8



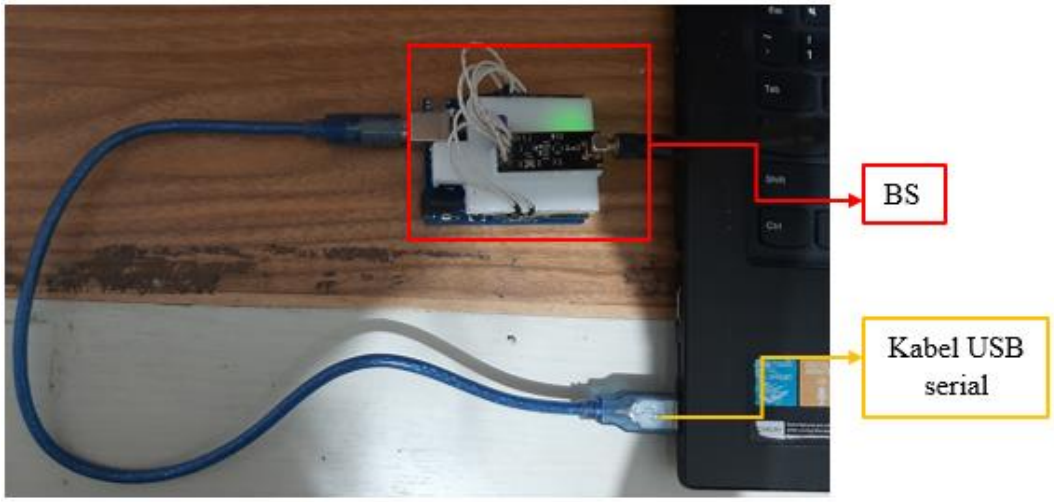
Gambar 4.10 Tampilan Serial Monitor *Software* Arduino IDE

4.4 Hasil Pengujian Perangkat Monitoring

Pada pengujian ini bertujuan untuk mendapatkan data berupa suhu, kelembaban udara, kelembaban tanah, arus dan tegangan yang dibutuhkan sebuah *node* sensor untuk melakukan monitoring dan mengirimkan paket data menuju BS. Untuk pengujian perangkat monitoring dilakukan di Balai Pengkajian Teknologi Pertanian Jawa Timur (BPTP Jatim) yang beralamatkan di Jl. Raya Karangploso Km. 4, Kepuharjo, Kecamatan Karang Ploso, Kabupaten Malang. Pengujian perangkat monitoring dilakukan didalam *greenhouse* seperti yang ditunjukkan pada Gambar 4.11.

Adapun perangkat yang digunakan untuk melakukan pengujian monitoring seperti yang ditunjukkan pada Gambar 4.9. Perangkat *client* atau *node* sensor berfungsi untuk mengumpulkan data berupa suhu, kelembaban udara dan kelembaban tanah. Sedangkan untuk perangkat server berfungsi untuk menampilkan data sensor suhu, kelembaban udara dan kelembaban tanah yang dihasilkan oleh *node* sensor. Untuk menampilkan paket data tersebut, BS terhubung dengan laptop menggunakan kabel USB (*Universal Serial Bus*) serial. Seperti yang ditunjukkan dalam Gambar 4.12.

Gambar 4.11 Lokasi Pengujian Perangkat Monitoring

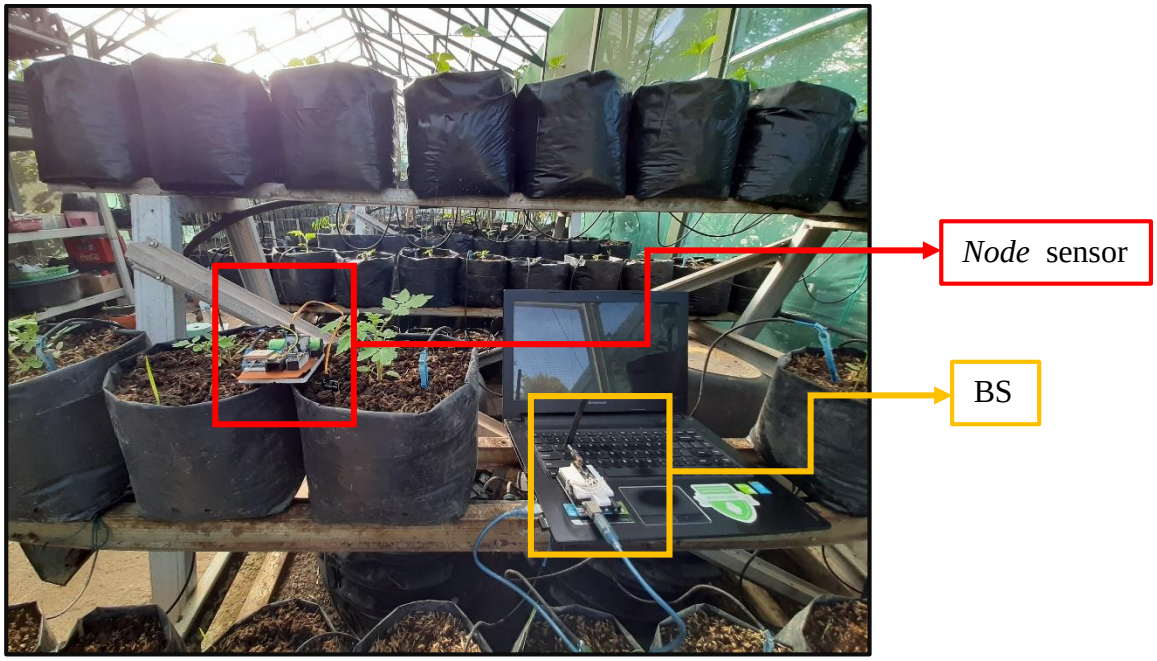


Gambar 4.12 Kabel USB Penghubung Arduino Uno dengan Laptop

Gambar 4.12 menunjukkan kabel USB yang digunakan untuk menghubungkan perangkat BS dengan laptop. Kemudian data hasil pengujian suhu, kelembaban udara dan kelembaban tanah akan tampil pada serial monitor *software* Arduino IDE seperti yang ditunjukkan pada Gambar 4.2. Adapun cara untuk melakukan cara pengujian suhu dan kelembaban udara di dalam *greenhouse* ditunjukkan dalam Gambar 4.13.

Gambar 4.13 menunjukkan pengujian suhu dan kelembaban udara. Dimana pada pengujian suhu, *node* sensor menggunakan sensor DHT11 yang sudah

terpasang pada *node* sensor. Kemudian *node* sensor diletakkan didalam *greenhouse* yang akan melakukan pengambilan data suhu dan kelembaban udara. Selanjutnya dilakukan pengecekan suhu dan kelembaban udara setiap 30 menit sekali. Sedangkan cara untuk melakukan pengujian kelembaban tanah sama dengan cara pengujian suhu dan kelembaban udara yang ditunjukkan pada Gambar 4.13.

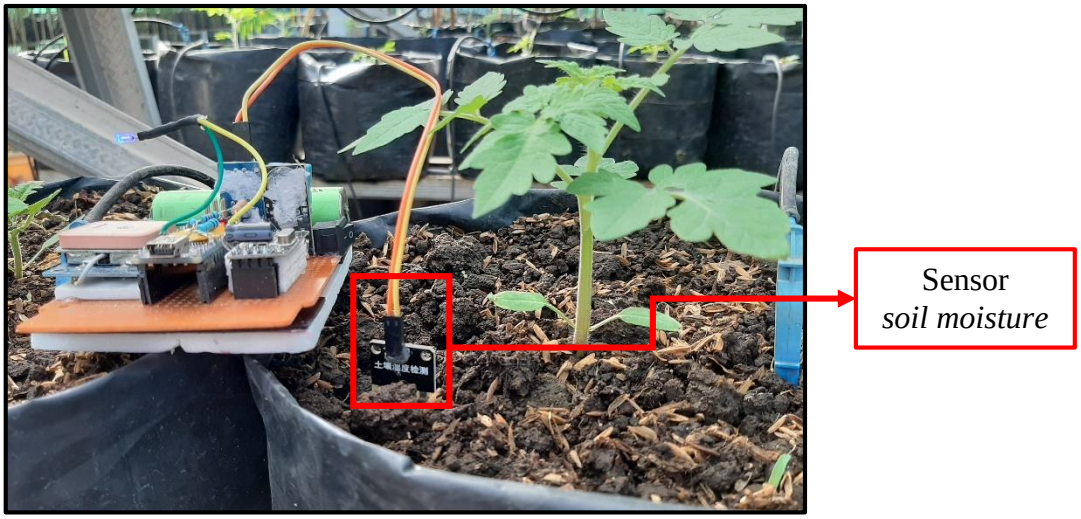


Gambar 4.13 Pengujian Suhu dan Kelembaban Udara *Greenhouse*

Gambar 4.14 menunjukkan pengujian kelembaban tanah di dalam *greenhouse.* Pada pengujian kelembaban tanah dilakukan menggunakan sensor *soil moisture* yang sudah terpasang di *node* sensor. Dimana sensor ini ditancapkan kedalam tanah. Untuk menancapkan sensor tersebut perlu diperhatikan kedalaman dari sensor masuk kedalam tanah. Semakin dalam sensor ditancapkan maka hasilnya lebih baik daripada sensor ditancapkan tidak terlalu dalam. Setelah sensor sudah tertancapkan ke dalam tanah, kemudian dilakukan monitoring kelembaban tanah setiap 30 menit sekali.

Setelah sensor selesai melakukan pembacaan suhu, kelembaban udara dan kelembaban tanah kemudian sekumpulan data tersebut akan dikirim menuju ke BS. Pada proses pengiriman data dilakukan melalui modul *wireless* NRF24L01.

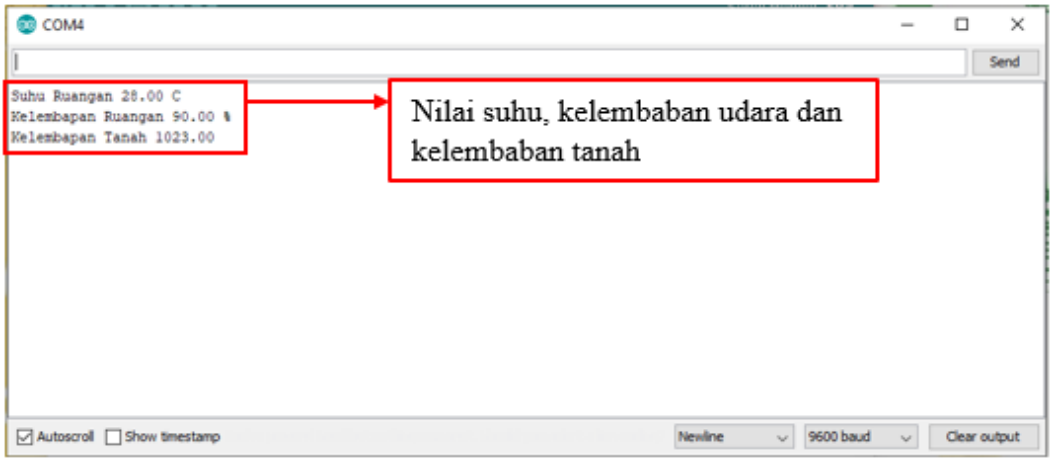
Dimana pada proses pengiriman data ini, data diberikan *header* yang sesuai dengan format protokol yang digunakan modul *wireless*. Adapun format protokol yang digunakan pada modul *wireless* ini ditunjukkan pada Gambar 3.8. Pada pengujian pengiriman data dari *node* sensor menuju ke BS dilakukan pengaturan kanal. Hal ini bertujuan supaya meminimalisir interferensi dengan perangkat lain yang sama-sama menggunakan frekuensi 2,4 GHz. Dalam modul *wireless* NRF24L01 ini terdapat *range* alokasi frekuensi yang bisa digunakan dari 2,4 GHz sampai 2,5 GHz dengan 125 alokasi kanal. Dimana dalam 1 kanal memiliki selisih frekuensi sebesar 8 MHz. Pada pengujian dilakukan pengaturan dalam penggunaan kanal untuk komunikasi antara *node* sensor dan BS, yaitu kanal ke 5 yang memiliki frekuensi sebesar 2,440 GHz. Setelah paket data diterima oleh BS, selanjutnya data ditampilkan dalam *software* Arduino IDE. Adapun bentuk tampilan *software* Arduino IDE dalam menampilkan data suhu dan kelembaban udara ditunjukkan dalam Gambar 4.15.



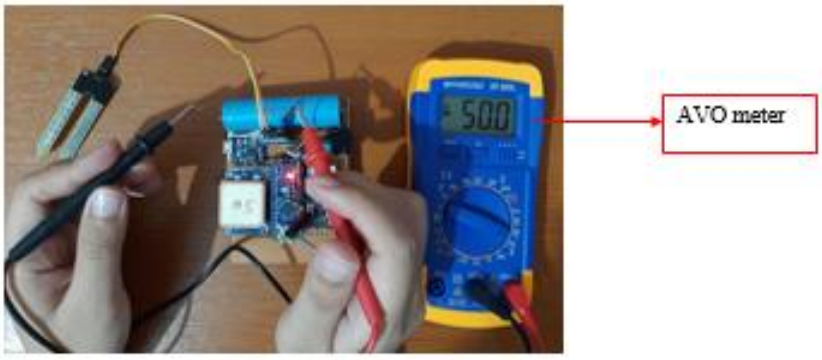
Gambar 4.14 Pengujian Kelembaban Tanah *Greenhouse*

Pada pengujian ini, juga dilakukan pengujian arus dan tegangan yang digunakan untuk melakukan monitoring suhu, kelembaban udara, kelembaban tanah serta mengirimkan paket data menuju BS. Untuk melakukan pengujian arus dan tegangan, digunakanlah alat ukur berupa AVO meter digital. Adapun cara pengujian arus ditunjukkan dalam Gambar 4.16.

Gambar 4.16 menunjukkan cara pengujian arus menggunakan AVO meter digital. Dalam pengujian arus ini, *probe* merah (+) ditempelkan pada ujung kabel positif dari baterai sedangkan untuk *probe* hitam (-) ditempelkan pada ujung kabel beban (hambatan) positif dari perangkat yang digunakan. Kemudian nilai arus akan muncul pada AVO meter. Sedangkan cara untuk melakukan pengujian tegangan pada perangkat ditunjukkan pada Gambar 4.17.



Gambar 4.15 Tampilan *Serial Monitor* Arduino IDE



Gambar 4.16 Pengujian Arus Menggunakan AVO Meter Digital

Gambar 4.17 menunjukkan pengujian tegangan pada perangkat menggunakan alat ukur AVO meter. Pada pengujian tegangan dilakukan dengan menempelkan *probe* merah (+) dengan ujung baterai positif dan menempelkan *probe* hitam (-) dengan ujung betarai negatif. Setelah kedua probe tertempel pada masing-masing ujung baterai, maka nilai tegangan akan muncul pada AVO meter.

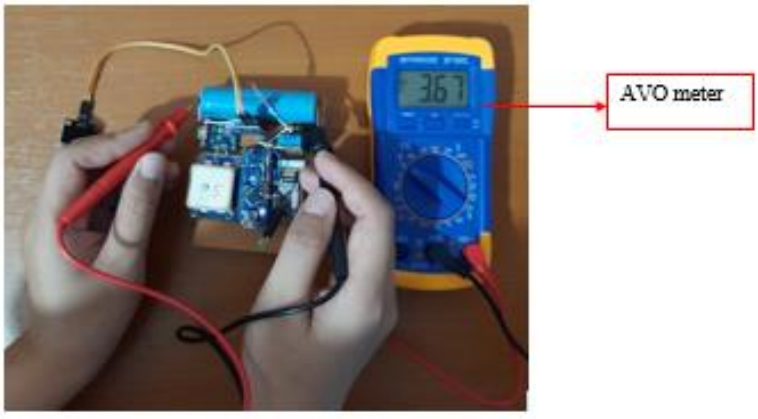
Sebelum melakukan pengujian perangkat untuk monitoring, dilakukan perhitungan daya ketika baterai dalam kondisi penuh. Dimana kapasitas baterai adalah 2,2 Ah dan tegangan 3,67 Volt sehingga daya. Dimana daya listrik didefinisikan sebagai laju hantaran energi listrik dalam rangkaian listrik. Daya listrik memiliki satuan watt yang menyatakan banyaknya tenaga listrik yang mengalir per satuan waktu (joule/detik). Perhitungan daya dapat dihitung menggunakan persamaan (3.1) sebagai berikut.

$P_{penuh} = V\ X\ I$

$P_{penuh} = 3{,}67\ Volt\ X\ 2{,}2\ Ah$

$P_{penuh} = 8{,}07\ W$att

Dari hasil perhitungan daya yang telah dilakukan, didapatkan daya pada kondisi baterai penuh adalah 8,07 Watt. Dimana dalam satu jam daya yang digunakan adalah sebesar 8,07 Watt. Dikarenakan dari hasil perhitungan jarak antar *node* sensor dengan BS maupun jarak antar *node* sensor, terdapat beberapa *node* sensor menggunakan daya pancar -18, -12, -6 dan 0 dBm. Maka dilakukan pengujian perangkat monitoring menggunakan 4 jenis daya pancar tersebut. Hal ini bertujuan untuk mengetahui arus rata-rata dan tegangan minimum yang dapat digunakan pada masing-masing daya pancar. Untuk hasil pengujian perangkat yang menggunakan daya pancar -18 dBm ditunjukkan dalam Tabel 4.8.



Gambar 4.17 Pengujian Tegangan Menggunakan AVO Meter

Tabel 4.8 Hasil Pengujian Perangkat Monitoring dengan Daya Pancar -18 dBm

No.	Pukul	Nilai Suhu (°C)	Nilai Kelembaban Udara (%)	Nilai Kelembaban Tanah (%)	Arus (A)	Tegangan (V)
1.	08.00	23	80	10,3	0,05	3,67
2.	08.30	23	80	82,4	0,05	3,67
3.	09.00	24	78	82,4	0,05	3,67
4.	09.30	24	78	77,6	0,048	3,5
5.	10.00	24,5	74,5	77,6	0,048	3,5
6.	10.30	24,5	70	65,3	0,048	3,5
7.	11.00	25	70	61	0,048	3,3
8.	11.30	25	68	61	0,046	3,3
9.	12.00	25	68	54,2	0,046	3,2
10.	12.30	26	68	54,2	0,046	3,2
11.	13.00	26	65	54,2	0,044	3,2
12.	13.30	27	65	48,6	0,044	3
13.	14.00	27	63,3	48,6	0,044	3
14.	14.30	25	62	43,5	0,044	3
15.	15.00	26	66	43,5	0,043	3
16.	15.30	26	69	54,2	0,043	2,9
17.	16.00	25	69	54,2	0,043	2,9
18.	16.30	25	70	54,2	0,043	2,9
19.	17.00	25	72	62	0,042	2,9
20.	17.30	25	72	62	0,042	2,9
21.	18.00	24,5	72	62	0,04	2,9
22.	18.30	23	74	66	0,04	2,8
23.	19.00	23	74	66	0,04	2,8
24.	19.30	23	74	66	0,04	2,8
25.	20.00	22	74	70	0,04	2,8
26.	20.30	22	74	70	0,04	2,8
27.	21.00	22	73	70	0,04	2,7
28.	21.30	21	73	70	0,04	2,7
Rata-Rata Arus (A)						**0,044**

Tabel 4.8 menunjukkan hasil pengujian perangkat monitoring menggunakan daya pancar -18 dBm. Dimana dari hasil pengujian menunjukkan bahwa arus rata-rata yang digunakan adalah 0,044 A dengan kapasitas baterai yang digunakan 2,2 Ah dan tegangan sebesar 3,67 Volt dapat bertahan selama 13,5 jam dari pukul 08.00 WIB-21.30 WIB. Hal ini dikarenakan tidak seluruh daya baterai dapat digunakan untuk mengoperasikan perangkat. Berdasarkan hasil pengujian yang telah dilakukan, tegangan minimum yang dapat mengoperasikan perangkat adalah 2,7

Volt. Dari hasil pengujian perangkat monitoring -18 dBm dapat dihitung energi yang digunakan dengan persamaan (3.1) sebagai berikut.

$P_{-18dBm} = V\ X\ I$

P_{-18dBm} = 3,67 $Volt\ X$ 0,044 A

P_{-18dBm} = 0,161 Watt

Dari hasil perhitungan daya -18 dBm didapatkan hasil 0,161 Watt. Kemudian dilakukan perhitungan daya pemakaian 0,161 Watt dalam waktu 13,5 jam menggunakan persamaan (3.2) sebagai berikut.

$Wh = P_{-18dBm}\ x\ Jam$ (3.2)

$Wh = 0{,}161\ Watt\ x\ 13{,}5\ Jam$

$Wh = 2{,}17$

Dari hasil perhitungan menunjukkan bahwa jumlah pemakaian daya dalam 13,5 jam adalah sebesar 2,17 Wh. Selanjutnya dilakukan pengujian perangkat monitoring yang menggunakan daya pancar -12 dBm yang ditunjukkan dalam Tabel 4.9.

Tabel 4.9 Hasil Pengujian Perangkat Monitoring dengan Daya Pancar -12 dBm

No.	Pukul	Nilai Suhu (°C)	Nilai Kelembaban Udara (%)	Nilai Kelembaban Tanah (%)	Arus (A)	Tegangan (V)
1.	08.00	24	89	10,3	0,058	3,67
2.	08.30	24	84	73	0,058	3,67
3.	09.00	26	82	75,2	0,058	3,67
4.	09.30	26	82,2	75,2	0,057	3,5
5.	10.00	27	79	73	0,057	3,5
6.	10.30	26	76,6	68,7	0,057	3,5
7.	11.00	27	73	65,3	0,055	3,4
8.	11.30	27	70	65,3	0,055	3,4
9.	12.00	29	68,2	57,5	0,054	3,3
10.	12.30	29,4	65	54,2	0,054	3,3
11.	13.00	30	63	49	0,052	3,3
12.	13.30	29	60	43,6	0,052	3,2
13.	14.00	26	61,3	36,5	0,052	3,2

Tabel 4.9 Hasil Pengujian Perangkat Monitoring dengan Daya Pancar -12 dBm (Lanjutan)

No.	Pukul	Nilai Suhu (°C)	Nilai Kelembaban Udara (%)	Nilai Kelembaban Tanah (%)	Arus (A)	Tegangan (V)
14.	14.30	25	62	32,6	0,049	3,2
15.	15.00	26	66	30	0,049	3,1
16.	15.30	26	68,8	38,4	0,048	3,1
17.	16.00	25	68,7	78	0,047	3,1
18.	16.30	25	70	78	0,047	3
19.	17.00	24,7	76,3	73,1	0,046	3
20.	17.30	25	78	72,3	0,046	3
21.	18.00	24,5	80	72,3	0,045	2,9
22.	18.30	23	80,4	65,3	0,045	2,9
23.	19.00	22	82	65,3	0,044	2,8
24.	19.30	23,4	82	65,3	0,043	2,8
Rata-Rata Arus (A)					**0,047**	**-**

Tabel 4.9 menunjukkan hasil pengujian perangkat monitoring menggunakan daya pancar -12 dBm. Dimana dari hasil pengujian menunjukkan bahwa arus rata-rata yang digunakan adalah 0,047 A dengan kapasitas baterai yang digunakan 2,2 Ah dan tegangan sebesar 3,67 Volt dapat bertahan bertahan 12 jam yang dimulai dari pukul 08.00 WIB-19.30 WIB. Hal ini dikarenakan tidak seluruh daya baterai dapat digunakan untuk mengoperasikan perangkat. Berdasarkan hasil pengujian yang telah dilakukan, tegangan minimum yang dapat mengoperasikan perangkat adalah 2,7 Volt. Dari hasil pengujian perangkat monitoring -12 dBm dapat dihitung daya yang digunakan dengan persamaan (3.1) sebagai berikut.

P_{-12dBm} = $V\ X\ I$

P_{-12dBm} = 3,67 $Volt\ X$ 0,047 A

P_{-12dBm} = 0,172 Watt

Dari hasil perhitungan daya -12 dBm didapatkan hasil 0,172 Watt. Kemudian dilakukan perhitungan daya pemakaian 0,172 Watt dalam waktu 12 jam menggunakan persamaan (3.2) sebagai berikut.

$Wh = P_{-12dBm} x\ Jam$

$Wh = 0{,}172\ Wh\ x\ 12\ Jam$

$Wh = 2{,}06$

Dari hasil perhitungan menunjukkan bahwa jumlah pemakaian daya dalam 12 jam adalah sebesar 2,06 Wh. Selanjutnya dilakukan pengujian perangkat monitoring yang menggunakan daya pancar -6 dBm yang ditunjukkan dalam Tabel 4.10.

Tabel 4.10 Hasil Pengujian Perangkat Monitoring dengan Daya Pancar -6 dBm

No.	Pukul	Nilai Suhu (°C)	Nilai Kelembaban Udara (%)	Nilai Kelembaban Tanah (%)	Arus (A)	Tegangan (V)
1.	08.00	24	91	30	0,056	3,67
2.	08.30	24	88,1	63	0,056	3,67
3.	09.00	24	88,1	70	0,055	3,55
4.	09.30	24	85	75,2	0,055	3,55
5.	10.00	26	85	73	0,053	3,3
6.	10.30	26	83	70	0,053	3,3
7.	11.00	26	83,2	68,4	0,053	3,3
8.	11.30	27	81,1	65,3	0,051	3,26
9.	12.00	27	81,1	57,5	0,05	3,24
10.	12.30	26	77	52	0,05	3,3
11.	13.00	26	77	55,5	0,049	3,3
12.	13.30	27.5	75	53,6	0,049	3,2
13.	14.00	27	75,3	50,3	0,048	3,2
14.	14.30	27	72,1	48	0,048	3,05
15.	15.00	27	72,1	42,6	0,048	3
16.	15.30	25	69	38,4	0,047	2,9
17.	16.00	25	69	80,6	0,047	2,9
18.	16.30	25	63,3	79	0,046	2,9
19.	17.00	24.7	63,1	75	0,046	2,82
20.	17.30	24.7	62,1	72,3	0,046	2,82
Rata-Rata Arus (A)					**0,05**	**-**

Tabel 4.10 menunjukkan hasil pengujian perangkat monitoring menggunakan daya pancar -6 dBm. Dimana dari hasil pengujian menunjukkan bahwa arus rata-rata yang digunakan adalah 0,05 A dengan kapasitas baterai yang digunakan 2,2 Ah dan tegangan sebesar 3,67 Volt dapat bertahan bertahan 10,5 jam yang dimulai dari pukul 08.00 WIB-17.30 WIB. Hal ini dikarenakan tidak seluruh daya baterai dapat digunakan untuk mengoperasikan perangkat. Berdasarkan hasil pengujian yang

telah dilakukan, tegangan minimum yang dapat mengoperasikan perangkat adalah 2,7 Volt. Dari hasil pengujian perangkat monitoring -6 dBm dapat dihitung daya yang digunakan dengan persamaan (3.1) sebagai berikut.

$P_{-6dBm} = V\ X\ I$

$P_{-6dBm} = 3{,}67\ Volt\ X\ 0{,}05\ A$

$P_{-6dBm} = 0{,}183$ Watt

Dari hasil perhitungan daya -6 dBm didapatkan hasil 0,183 Watt. Kemudian dilakukan perhitungan daya pemakaian 0,183 Watt dalam waktu 10,5 jam menggunakan persamaan (3.2) sebagai berikut.

$Wh = P_{-6dBm} x\ Jam$

$Wh = 0{,}183\ Watt\ x\ 10{,}5\ Jam$

$Wh = 1{,}92$

Dari hasil perhitungan menunjukkan bahwa jumlah pemakaian daya dalam 10,5 jam adalah sebesar 1,92 Wh. Selanjutnya dilakukan pengujian perangkat monitoring yang menggunakan daya pancar 0 dBm yang ditunjukkan dalam Tabel 4.11.

Tabel 4.11 Hasil Pengujian Perangkat Monitoring dengan Daya Pancar 0 dBm

No.	Pukul	Nilai Suhu (°C)	Nilai Kelembaban Udara (%)	Nilai Kelembaban Tanah (%)	Arus (A)	Tegangan (V)
1.	08.00	25	90,1	26,3	0,059	3,67
2.	08.30	26,4	90,1	70	0,059	3,67
3.	09.00	26,4	86,5	76	0,058	3,58
4.	09.30	27	86,5	76	0,058	3,58
5.	10.00	27	82,3	67	0,056	3,5
6.	10.30	27	82,3	62,4	0,056	3,47
7.	11.00	27,8	79	62,6	0,053	3,4
8.	11.30	27,8	79	58	0,053	3,4
9.	12.00	27,8	77	57,5	0,053	3,23
10.	12.30	28,4	77	53,2	0,05	3,23
11.	13.00	28,4	77	52	0,05	3,1
12.	13.30	28,9	74,3	52	0,049	2,9

Tabel 4.11 Hasil Pengujian Perangkat Monitoring dengan Daya Pancar 0 dBm (Lanjutan)

No.	Pukul	Nilai Suhu (°C)	Nilai Kelembaban Udara (%)	Nilai Kelembaban Tanah (%)	Arus (A)	Tegangan (V)
13.	14.00	26	74,3	49	0,049	2,9
14.	14.30	24,6	76	49	0,047	2,84
15.	15.00	24,6	78,5	54,3	0,047	2,84
16.	15.30	23	78,5	54,3	0,045	2,76
17.	16.00	23	80	82	0,045	2,76
Rata-Rata Arus (A)					**0,052**	**-**

Tabel 4.11 menunjukkan hasil pengujian perangkat monitoring menggunakan daya pancar 0 dBm. Dimana dari hasil pengujian menunjukkan bahwa arus rata-rata yang digunakan adalah 0,052 A dengan kapasitas baterai yang digunakan 2,2 Ah dan tegangan sebesar 3,67 Volt dapat bertahan bertahan 9 jam yang dimulai dari pukul 08.00 WIB-16.00 WIB. Hal ini dikarenakan tidak seluruh daya baterai dapat digunakan untuk mengoperasikan perangkat. Berdasarkan hasil pengujian yang telah dilakukan, tegangan minimum yang dapat mengoperasikan perangkat adalah 2,7 Volt. Dari hasil pengujian perangkat monitoring 0 dBm dapat dihitung daya yang digunakan dengan persamaan (3.1) sebagai berikut.

$P_{0dBm} = V\ X\ I$

$P_{0dBm} = 3{,}67\ Volt\ X\ 0{,}052\ A$

$P_{0dBm} = 0{,}191$ Watt

Dari hasil perhitungan daya -6 dBm didapatkan hasil 0,191 Watt. Kemudian dilakukan perhitungan daya pemakaian 0,191 Watt dalam waktu 9 jam menggunakan persamaan (3.2) sebagai berikut.

$Wh = P_{0dBm} x\ Jam$

$Wh = 0{,}191\ Watt\ x\ 9\ Jam$

$Wh = 1{,}72$

Dari hasil perhitungan menunjukkan bahwa jumlah pemakaian daya dalam 9 jam adalah sebesar 1,72 Wh.

4.5 Hasil Perhitungan Kapasitas Baterai

Kapasitas baterai merupakan banyaknya muatan listrik (*charge*) yang dapat disalurkan pada tegangan nominal baterai, dinyatakan dalam Ampere Hour (Ah) atau miliAmpere Hour (mAh). Dalam suatu kapasitas baterai terdapat *State of Charge* (SOC) dan *Depth of Discharge* (DOD). SOC merupakan persentase kapasitas baterai yang masih tersedia (tersisa), terhadap kapasitas maksimumnya. Sedangkan untuk DOD adalah persentase kapasitas baterai yang telah dikosongkan (terpakai), terhadap kapasitas maksimum baterai. Pada perhitungan kapasitas baterai ini, menggunakan kapasitas sebesar 2,2 Ah dengan tegangan total sebesar 3,67 Volt dan tegangan drop sebesar 2,7 Volt. Dimana tegangan drop ini merupakan tegangan minimun yang dapat digunakan untuk memberi daya pada perangkat yang digunakan. Untuk mengetahui persentase dari DOD, pertama dilakukan perhitungan untuk mengetahui tegangan DOD menggunakan persamaan (3.3) sebagai berikut.

$$Tegangan_{DOD} = Tegangan_{total} - Tegangan_{drop}$$

$$Tegangan_{DOD} = 3{,}67 \text{ Volt} - 2{,}7 \text{ Volt}$$

$$Tegangan_{DOD} = 0{,}97 \text{ Volt}$$

Setelah diketahui tegangan DOD, selanjutnya dilakukan perhitungan untuk mencari persentase tegangan DOD yang didapatkan dari persentase tegangan total baterai sebesar 3,67 Volt menggunakan persamaan (3.4) sebagai berikut.

$$Persentase_{DOD} = \frac{Tegangan_{DOD}}{Tegangan_{total}} \text{ x } 100\%$$

$$Persentase_{DOD} = \frac{0{,}97\ Volt}{3{,}67\ Volt} \text{ x } 100\%$$

$$Persentase_{DOD} = 27{,}24\%$$

Dari hasil perhitungan persentase DOD, didapatkan hasil sebesar 27,24% dari tegangan total baterai berada dalam kondisi DOD atau baterai membutuhkan isi ulang untuk dapat mengoperasikan perangkat. Setelah diketahui persentase DOD, selanjutnya dapat menghitung kapasitas baterai Jam Ampere pada saat terjadi DOD menggunakan persamaan (3.5) sebagai berikut.

C_{DOD} = $Persentase_{DOD}$ x $kapasitas\ baterai$

C_{DOD} = $\frac{27,24}{100}$ x 2,2 Ah

C_{DOD} = 0,599 Ah

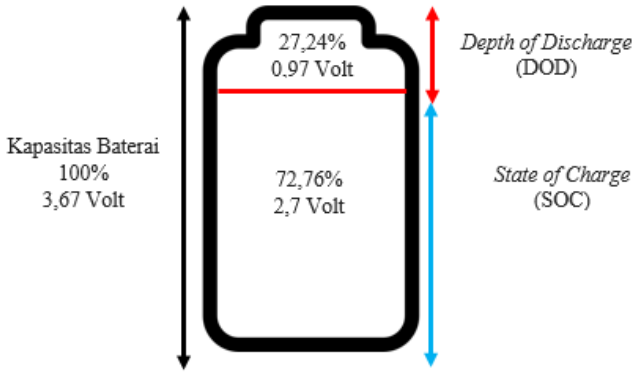
Dari hasil perhitungan kapasitas DOD, sehingga didapatkan kapasitas sebesar 0,599 Ah yang dapat digunakan untuk menyalakan perangkat. Dari hasil perhitungan daya pada saat kapasitas baterai dalam kondisi 100% yang telah dilakukan menggunakan persamaan (3.1) didapatkan daya sebesar 8,07 Watt. Kemudian dilakukan perhitungan persentase daya pada saat baterai dalam kondisi DOD. Sehingga didapatkan persentase baterai pada saat kondisi DOD, yaitu sebesar 27,24% yang didapatkan dari perhitungan menggunakan persamaan (3.6) sebagai berikut.

P_{DOD} = $Persentase_{DOD}$ x P_{penuh}

P_{DOD} = $\frac{27,24}{100}$ x 8,07 $Watt$

P_{DOD} = 2,19 $Watt$

Dari hasil perhitungan daya pada saat baterai dalam kondisi DOD didapatkan hasil sebesar 2,19 Watt. Adapun ilustrasi pembagian dari persentase SOC dan DOD baterai ditunjukkan pada Gambar 4.18.



Gambar 4.18 Pembagian Persentase SOC dan DOD

Gambar 4.18 menunjukkan pembagian persentasi SOC dan DOD. Dimana pada persentase DOD didapatkan sebesar 27,24% dengan tegangan sebesar 0,97 Volt. Sedangkan untuk persentasi SOC didapatkan sebesar 72,76% dengan tegangan sebesar 2,7 Volt. Dimana tegangan SOC sebesar 2,7 Volt merupakan tegangan minimum untuk dapat mengoperasikan perangkat.

4.6 Hasil Eksperimen dan Analisa

Pada bagian ini akan dijelaskan mengenai analisa dari *routing* protokol, PEGASIS dan IMHRP-P. Beberapa parameter yang digunakan untuk melakukan pengujian pemodelan *routing* protokol PEGASIS dan IMHRP-P yang meliputi : jumlah *node* sensor hidup vs *round,* jumlah energi yang digunakan pengiriman, jumlah energi yang digunakan dalam 1 bulan dan *Network Lifetime* (NL). Dimana setiap *round* adalah berapa banyak pengulangan pengiriman data yang dapat dilakukan oleh *node* sensor menuju BS.

4.6.1 Jumlah Konsumsi Energi

Selanjutnya dilakukan perhitungan jumlah konsumsi energi pada masing-masing pemodelan *routing* baik untuk skenario pertama maupun kedua yang menggunakan P_{DOD}. Hal ini dikarenakan P_{DOD} merupakan energi yang bisa digunakan oleh perangkat *node* sensor. Dari hasil perhitungan jarak yang telah dibahas pada sub bab perhitungan jarak, dapat menentukan jenis daya pancar yang akan digunakan berdasarkan hasil pengujian modul *wireless* NRF24L01 seperti yang ditunjukkan dalam Tabel 4.7. Maka perhitungan penggunaan daya masing-masing *node* sensor dalam pemodelan *routing* protokol PEGASIS pada skenario pertama menggunakan persamaan (3.7) sebagai berikut.

$$P_{PEGASIS1} = \frac{Daya\ perangkat\ masing{-}masing\ node\ sensor}{Jumlah\ node\ sensor}$$

$$P_{PEGASIS1} = \frac{P_{-18\ dBm} + P_{-18dBm} + P_{-18\ dBm} + P_{-18dBm} + P_{-12dBm}}{5}$$

$$P_{PEGASIS1} = \frac{0{,}161\ Watt + 0{,}161\ Watt + 0{,}161\ Watt + 0{,}161\ Watt + 0{,}172\ Watt}{5}$$

$P_{PEGASIS1}$ = 0,163 $Watt$

Dari hasil perhitungan yang telah dilakukan, didapatkan daya rata-rata masing-masing *node* sensor yang digunakan pada *routing* protokol PEGASIS skenario pertama sebesar 0,163 Watt untuk satu kali pengiriman paket data menuju ke *node* sensor selanjutnya atau ke BS. Selanjutnya dilakukan perhitungan daya DOD yang dibagi dengan daya $P_{PEGASIS1}$. Hal ini bertujuan untuk mengetahui *Network Lifetime* (NL) dari semua *node* sensor dengan daya DOD pada PEGASIS dengan skenario pertama. Adapun perhitungannya menggunakan persamaan (3.8) sebagai berikut.

$$NL_{PEGASIS1} = \frac{P_{DOD}}{P_{PEGASIS1}}$$

$$NL_{PEGASIS1} = \frac{2{,}19\ Wh}{0{,}163\ Watt}$$

$NL_{PEGASIS1}$ = 13,4 jam

Dari hasil perhitungan menunjukkan bahwa daya DOD sebesar 2,19 Watt dapat bertahan hingga 13,4 jam dengan daya rata-rata perangkat yang digunakan adalah 0,163 Watt untuk satu kali pengiriman paket data menuju ke BS. Selain itu dilakukan pengujian penggunaan daya rata-rata perangkat *node* sensor pada pemodelan *routing* protokol IMHRP-P dengan jumlah *node* sensor 5, 20 dan 35. Dimana untuk jumlah *node* sensor 5 menggunakan luas area 15 m x 15 m sedangkan untuk jumlah *node* sensor 20 dan 35 menggunakan luas area 100 m x 100 m baik untuk skenario pertama maupun skenario kedua. Dari hasil pengujian daya rata-rata perangkat *node* sensor yang digunakan dalam satu kali pengiriman paket data ke *node* sensor selanjutnya atau ke BS pada skenario pertama dan kedua yang ditunjukkan dalam Tabel 4.12.

Tabel 4.12 menunjukkan perhitungan konsumsi daya rata-rata perangkat *node* sensor baik untuk skenario pertama maupun kedua. Dimana perhitungan ini dilakukan pada *software* simulasi NS3. Dari hasil perhitungan menunjukkan bahwa pada skenario pertama terjadi kenaikan terhadap daya rata-rata perangkat *node* sensor yang digunakan dalam pemodelan *routing* protokol PEGASIS dan IMHRP-P dengan bertambahnya jumlah *node* sensor dari 5 menjadi

20 dan luas area dari 15 m x 15 m menjadi 100 m x 100 m. Dimana pada PEGASIS memiliki daya rata-rata perangkat *node* sensor 0,163 Watt dengan jumlah *node* sensor sebanyak 5 dan luas area sebesar 15 m x 15 m kemudian terjadi kenaikan daya menjadi 0,183 Watt dengan luas area yang semakin luas, yaitu 100 m x 100 m. Terjadinya kenaikan daya rata-rata perangkat *node* sensor sensor disebabkan adanya beberapa *node* sensor yang menggunakan daya cukup besar untuk dapat mengirimkan paket data ke *node* sensor selanjutnya atau ke BS dengan semakin bertambahnya luas area yang digunakan. Begitu pula dengan *routing* protokol IMHRP-P, terjadi kenaikan penggunaan daya rata-rata perangkat *node* sensor dari 0,165 Watt dengan luas area 15 m x 15 m menjadi 0,176 Watt dengan luas area 100 m x 100 m. Terjadinya kenaikan penggunaan daya disebabkan terdapat beberapa *node* sensor yang menggunakan daya cukup besar untuk melakukan pengiriman paket data ke *node* sensor selanjutnya atau ke BS. Akan tetapi terjadi penurunan penggunaan daya rata-rata perangkat *node* sensor pada pemodelan *routing* PEGASIS dan IMHRP-P dengan jumlah *node* sensor 35 dan luas area sebesar 100 m x 100 m. Penurunan penggunaan daya ini disebabkan jumlah *node* sensor yang bertambah sehingga jarak pengiriman paket data ke *node* sensor selanjutnya atau ke BS cukup pendek. Dengan jarak yang cukup pendek maka tidak memerlukan penggunaan daya yang cukup besar untuk mengirimkan paket data.

Tabel 4.12 Konsumsi Daya *Node* Sensor

Routing	**Jumlah *Node* Sensor**	**Keterangan**	
		Daya (Watt) Skenario Pertama	**Daya (Watt) Skenario Kedua**
PEGASIS	5	0,163	0,165
IMHRP-P	5	0,165	0,161
PEGASIS	20	0,183	0,178
IMHRP-P	20	0,176	0,177
PEGASIS	35	0,176	0,174
IMHRP-P	35	0,172	0,171

Sedangkan dari hasil perhitungan konsumsi daya rata-rata perangkat *node* sensor pada skenario kedua sama halnya dengan skenario pertama terjadi kenaikan terhadap daya yang digunakan dalam PEGASIS dan IMHRP-P dengan bertambahnya jumlah *node* sensor dari 5 menjadi 20 dan luas area dari 15 m x 15

m menjadi 100 m x 100 m. Dimana pada PEGASIS memiliki daya rata-rata perangkat *node* sensor 0,165 Watt dengan jumlah *node* sensor sebanyak 5 dan luas area sebesar 15 m x 15 m kemudian terjadi kenaikan daya menjadi 0,178 Watt dengan luas area yang semakin luas, yaitu 100 m x 100 m. Terjadinya kenaikan daya disebabkan terdapat beberapa *node* sensor yang menggunakan daya cukup besar untuk dapat mengirimkan paket data ke *node* sensor selanjutnya atau ke BS dengan semakin bertambahnya luas area yang digunakan maka jarak jangkau *node* sensor akan semakin jauh. Begitu pula dengan *routing* protokol IMHRP-P, terjadi kenaikan penggunaan daya dari 0,161 Watt dengan luas area 15 m x 15 m menjadi 0,177 Watt dengan luas area 100 m x 100 m. Terjadinya kenaikan penggunaan daya disebabkan terdapat beberapa *node* sensor yang menggunakan daya cukup besar untuk melakukan pengiriman paket data ke *node* sensor selanjutnya atau ke BS. Hal ini dikarenakan semakin luas area yang digunakan maka akan semakin jauh pula jarak pengiriman paket data menuju *node* sensor selanjutnya atau BS. Akan tetapi terjadi penurunan penggunaan daya pada pemodelan *routing* PEGASIS dan IMHRP-P dengan jumlah *node* sensor 35 dan luas area sebesar 100 m x 100 m. Penurunan penggunaan daya ini disebabkan jumlah *node* sensor yang bertambah pada luas area 100 m x 100 m sehingga jarak pengiriman paket data ke *node* sensor selanjutnya atau ke BS cukup pendek. Dengan jarak yang cukup pendek maka tidak memerlukan penggunaan daya yang cukup besar untuk mengirimkan paket data.

Selanjutnya dilakukan perhitungan daya yang digunakan masing-masing *node* sensor untuk jangka waktu selama 1 bulan. Dimana dalam 1 bulan terdapat 720 jam dan setiap jam melakukan 2 kali pengiriman paket data. Hasil perhitungan daya dalam kurun waktu 1 bulan baik untuk skenario pertama dan kedua ditunjukkan dalam Tabel 4.13.

Tabel 4.13 Konsumsi Daya *Node* Sensor dalam 1 Bulan

Routing	**Jumlah *Node* Sensor**	**Keterangan**	
		Daya (Watt) Skenario Pertama	**Daya (Watt) Skenario Kedua**
PEGASIS	5	234,7	237,6
IMHRP-P	5	237,6	231,8
PEGASIS	20	263,5	256,3

Tabel 4.13 Konsumsi Daya *Node* Sensor dalam 1 Bulan (Lanjutan)

Routing	**Jumlah *Node* Sensor**	**Keterangan**	
		Daya (Watt) Skenario Pertama	**Daya (Watt) Skenario Kedua**
IMHRP-P	20	253,4	254,8
PEGASIS	35	253,4	250,5
IMHRP-P	35	247,6	246,2

Tabel 4.13 menunjukkan penggunaan daya pada pemodelan *routing* protokol PEGASIS dan IMHRP-P baik untuk skenario pertama maupun kedua. Setelah didapatkan penggunaan daya rata-rata pemodelan *routing* protokol PEGASIS dan IMHRP-P, kemudian dilakukan perhitungan penggunaan daya rata-rata tersebut dalam kurun waktu satu bulan. Hal ini bertujuan untuk mengetahui berapa banyak daya rata-rata yang digunakan perangkat *node* sensor pada pemodelan *routing* protokol PEGASIS dan IMHRP-P dalam kurun waktu 1 bulan. Berdasarkan konsumsi daya rata-rata perangkat *node* sensor yang telah ditunjukkan dalam Tabel 4.12 terjadi peningkatan penggunaan daya pada pemodelan *routing* protokol PEGASIS dan IMHRP-P untuk skenario pertama dengan jumlah *node* sensor dari 5 menjadi 20 dan juga penggunaan luas area yang lebih besar. Dengan meningkatnya penggunaan daya rata-rata perangkat *node* sensor, maka konsumsi daya dalam kurun waktu 1 bulan akan terjadi peningkatan pula. Hal ini dibuktikan dengan hasil perhitungan dalam pemodelan *routing* protokol PEGASIS dan IMHRP-P baik untuk jumlah *node* sensor 5 dengan luas area sebesar 15 m x 15 m maupun dengan jumlah *node* sensor 20 dengan luas area sebesar 100 m x 100 m pada skenario pertama dan kedua. Dimana hasil menunjukkan konsumsi daya yang digunakan PEGASIS pada skenario pertama dengan jumlah *node* sensor sensor 5 sebanyak 234,7 Watt sedangkan untuk jumlah *node* sensor 20 mengkonsumsi daya sebesar 263,5 Watt. Untuk konsumsi daya yang digunakan IMHRP-P pada skenario pertama dengan jumlah *node* sensor 5 sebanyak 237,6Watt sedangkan untuk jumlah *node* sensor 20 mengkonsumsi daya sebesar 253,4 Watt. Akan tetapi terjadi penurunan konsumsi daya pada saat jumlah *node* sensor semakin banyak, yaitu 35. Dimana konsumsi daya PEGASIS turun menjadi 253,4 Watt dari 263,5 Watt. Hal ini dikarenakan jumlah *node* sensor yang digunakan semakin banya pada luas area

100 m x 100 m sehingga *node* sensor dapat meminimalisi penggunaan daya pancar yang menghabiskan daya (Watt) yang cukup banyak. Hal ini juga berlaku untuk pemodelan *routing* protokol IMHRP-P dimana terjadi penurunan konsumsi daya dari 253,4 Watt menjadi 247,6 Watt.

Sedangkan untuk penggunaan konsumsi daya untuk skenario kedua sama halnya dengan skenario pertama. Dimana terjadi peningkatan konsumsi daya pada saat jumlah *node* sensor sebanyak 5 dan 20 dengan luas area sebesar 15 m x 15 m serta 100 m x 100 m. Dimana konsumsi daya dalam 1 bulan yang digunakan PEGASIS dengan jumlah *node* sensor 5 adalah 237,6 Watt sedangkan untuk jumlah *node* sensor 20 mengkonsumsi daya sebesar 256,3 Watt. Untuk konsumsi daya yang digunakan IMHRP-P pada skenario kedua dengan jumlah *node* sensor 5 sebanyak 231,8 Watt sedangkan untuk jumlah *node* sensor 20 mengkonsumsi daya sebesar 254,8 Watt. Akan tetapi terjadi penurunan konsumsi daya pada saat jumlah *node* sensor semakin banyak, yaitu 35. Dimana konsumsi daya PEGASIS turun menjadi 253,4 Watt dari 263,5 Watt. Akan tetapi terjadi penurunan konsumsi daya pada saat jumlah *node* sensor semakin banyak, yaitu 35. Dimana konsumsi daya PEGASIS turun menjadi 250,5 Watt dari 256,3 Watt. Hal ini dikarenakan jumlah *node* sensor yang digunakan semakin banya pada luas area 100 m x 100 m sehingga *node* sensor dapat meminimalisi penggunaan daya pancar yang menghabiskan daya (Watt) yang cukup banyak. Hal ini juga berlaku untuk pemodelan *routing* protokol IMHRP-P dimana terjadi penurunan konsumsi daya dari 256,3 Watt menjadi 246,2 Watt.

4.6.2 *Network Lifetime* (NL) *Routing* Protokol

Berkaitan dengan penggunaan daya pada masing-masing pemodelan *routing* protokol maka ada keterkaitannya dengan NL yang dimiliki oleh masing-masing pemodelan *routing* protokol. Semakin kecil daya rata-rata perangkat yang digunakan dalam suatu pemodelan *routing* maka akan semakin panjang NL yang dimiliki oleh pemodelan *routing* tersebut. Hal ini dikarenakan penggunaan daya yang tidak terlalu besar sehingga perangkat *node* sensor dapat bertahan lebih lama untuk melakukan monitoring suhu, kelembaban udara dan kelembaban tanah serta melakukan pengiriman paket data menuju *node* sensor selanjutnya atau ke BS. Dari

hasil pengujian NL pada skenario pertama untuk pemodelan *routing* PEGASIS dan IMHRP-P ditunjukkan dalam Tabel 4.14.

Tabel 4.14 Pengujian NL Skenario Pertama

Routing	**Jumlah *Node* Sensor**	***Network Lifetime* (Jam)**
PEGASIS	5	13,4
IMHRP-P	5	13,2
PEGASIS	20	11,9
IMHRP-P	20	12,4
PEGASIS	35	12,4
IMHRP-P	35	12,7

Dari Tabel 4.14 menunjukkan bahwa semakin besar daya rata-rata yang digunakan perangkat maka akan semakin pendek NL yang dimiliki oleh pemodelan *routing* protokol tersebut. Hal ini dibuktikan pada *routing* PEGASIS dengan jumlah *node* sensor sebanyak 20 dan luas area sebesar 100 m x 100 m. Dimana daya rata-rata perangkat yang digunakan sebesar 0,183 Watt yang ditunjukkan dalam Tabel 4.12. Untuk mengetahui NL pada *routing* protokol ini, dilakukan pembagian antara daya baterai pada saat DOD, yaitu 2,19 Watt yang didapatkan dari hasil perhitungan menggunakan persamaan (3.6) dibagi dengan daya rata-rata perangkat PEGASIS sebesar 0,183 Watt. Sehingga didapatkan NL selama 11,9 jam.

Sedangkan untuk pengujian NL pada skenario kedua ditunjukkan dalam Tabel 4.15. Dari 4.15 menunjukkan bahwa NL yang terpendek terjadi pada pemodelan *routing* protokol PEGASIS dengan jumlah *node* sensor sebanyak 20 dan luas area yang digunakan adalah 100 m x 100 m. PEGASIS pada skenario kedua ini menggunakan daya rata-rata perangkat yang paling besar 0,178 Watt, yaitu seperti yang ditunjukkan dalam Tabel 4.12. Untuk mengetahui NL pada *routing* protokol ini, maka dilakukan pembagian antara daya baterai DOD seperti telah dihitung menggunakan persamaan (3.6) dengan daya rata-rata perangkat PEGASIS skenario kedua sebesar 0,178 Watt. Sehingga didapatkan NL selama 12,2 jam.

Tabel 4.15 Pengujian NL Skenario Kedua

Routing	**Jumlah *Node* Sensor**	***Network Lifetime* (Jam)**
PEGASIS	5	13,2
IMHRP-P	5	13,6
PEGASIS	20	12,2
IMHRP-P	20	12,3
PEGASIS	35	12,5
IMHRP-P	35	12,8

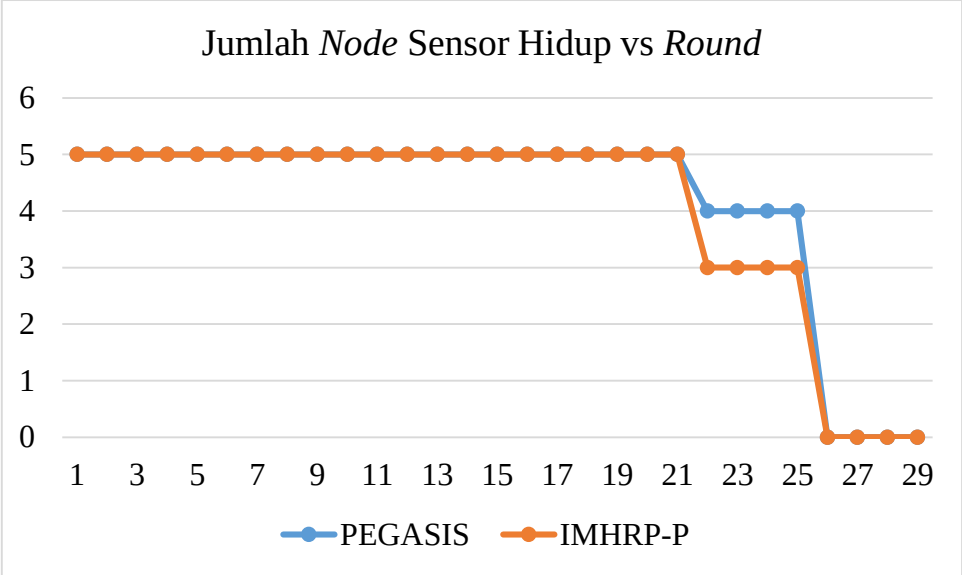
4.6.3 Jumlah *Node* Sensor Hidup vs *Round*

Pada pengujian ini dilakukan pengujian jumlah *node* sensor yang hidup vs *round*. Setelah diketahui daya rata-rata yang digunakan oleh perangkat *node* sensor, dapat melakukan perhitungan jumlah *node* sensor yang hidup dalam maksimum *round* masing-masing *node* sensor. Untuk mengetahui maksimum *round* dari masing-masing *node* sensor dilakukan pengujian perangkat pada masing-masing daya pancar. Dari hasil pengujian ini dapat diketahui masing-masing daya pancar dapat bertahan hingga berapa jam. Dimana setiap *round* yang digunakan dalam simulasi ini memiliki penjadwalan pengiriman paket data setiap 30 menit sekali baik pengiriman paket data menuju ke *node* sensor selanjutnya atau ke BS. Dari hasil pengujian jumlah *node* sensor yang hidup vs *round* yang telah dilakukan pada NS3 ditunjukkan dalam Tabel 4.16. Tabel 4.16 menunjukkan hasil pengujian jumlah *node* sensor hidup vs *round* untuk skenario pertama dengan jumlah *node* sensor yang bervariasi, yaitu 5, 20 dan 35. Selain itu juga menggunakan 2 luas area yang berbeda, yaitu 15 m x 15 m dan 100 m x 100. Untuk luas area 15 m x 15 m menggunakan *node* sensor sebanyak 5 sedangkan dilakukan pengujian dengan jumlah *node* sensor yang bervariasi, yaitu 20 dan 35. Dimana dari hasil pengujian menunjukkan bahwa pada pemodelan *routing* protokol IMHRP-P dengan jumlah *node* sensor sebanyak 35 jauh lebih unggul dalam hal jumlah *node* sensor yang hidup hingga *round* ke 27 daripada PEGASIS dan IMHRP-P baik untuk jumlah *node* sensor 35, 5 maupun 20. Hal ini dikarenakan dalam IMHRP-P terdapat 10 *node* sensor yang menggunakan daya pancar -18 dBm yang mengkonsumsi daya (Watt) yang cukup rendah, yaitu 0,161 Watt. Sehingga 10 *node* sensor ini dapat bertahan hingga 13,5 jam atau 27 *round*. Untuk mempermudah dalam melakukan

perbandingan jumlah *node* sensor yang hidup vs *round*, data ditampilkan dalam bentuk grafik terhadap setiap adanya penambahan *node* sensor. Adapun grafik perbandingan antara PEGASIS dan IMHRP-P untuk jumlah *node* sensor sebanyak 5 ditunjukkan dalam Gambar 4.19.

Tabel 4.16 Jumlah *Node* Sensor Hidup vs *Round* Skenario Pertama

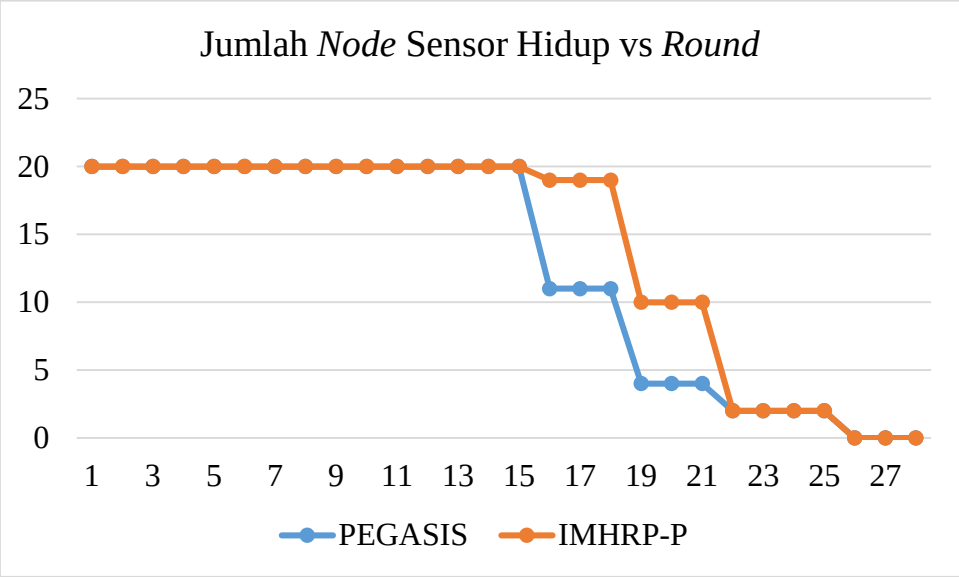
Routing	**Jumlah *Node* Sensor**	***Round***				
		18	**21**	**24**	**27**	**30**
PEGASIS	5	5	5	5	4	0
IMHRP-P	5	5	5	5	3	0
PEGASIS	20	11	4	2	2	0
IMHRP-P	20	19	10	2	2	0
PEGASIS	35	29	21	5	5	0
IMHRP-P	35	34	25	10	10	0



Gambar 4.19 Jumlah *Node* Sensor Hidup dengan 5 *Node* Sensor Skenario Pertama

Gambar 4.19 menunjukkan grafik perbandingan antara PEGASIS dan IMHRP-P dengan jumlah *node* sensor sebanyak 5. Hasil dari grafik tersebut menunjukkan bahwa PEGASIS lebih unggul dari IMHRP-P. Dimana jumlah *node* sensor yang hidup sebanyak 4 pada *round* ke 24 sedangkan IMHRP-P terdapat 3 *node* sensor yang hidup pada *round* yang sama. Hal ini dikarenakan pada PEGASIS membentuk struktur rantai sehingga jarak antar *node* sensor lebih dekat. Dengan

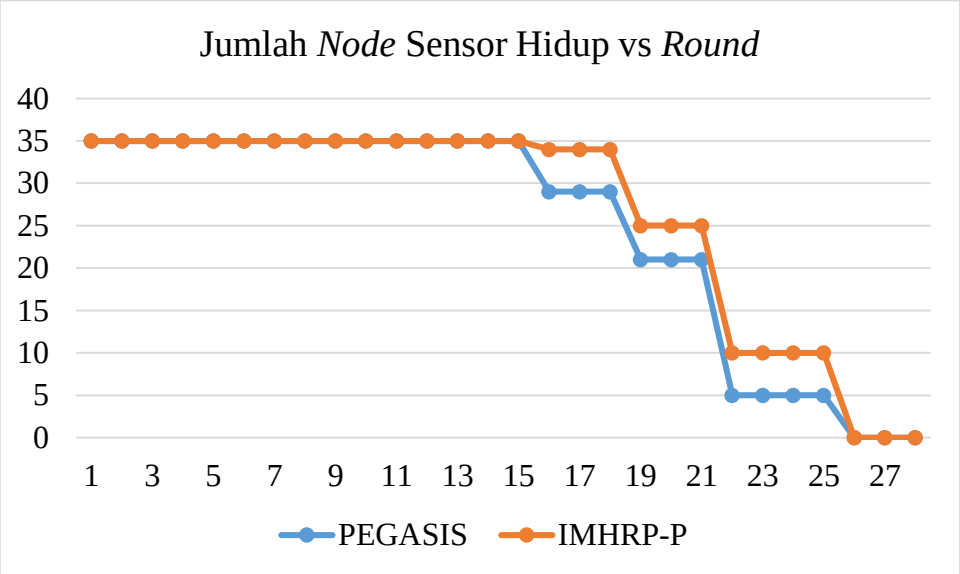
tidak terlalu jauh jarak antar *node* sensor maka konsumsi daya (Watt) tidak terlalu besar pula. Sedangkan untuk grafik perbandingan PEGASIS dan IMHRP-P dengan jumlah *node* sensor sensor sebanyak 20 ditunjukkan dalam Gambar 4.20.



Gambar 4.20 Jumlah *Node* Sensor Hidup dengan 20 *Node* Sensor Skenario Pertama

Gambar 4.20 menunjukkan grafik perbandingan antara PEGASIS dan IMHRP-P dengan jumlah *node* sensor sebanyak 20. Dari grafik tersebut menunjukkan bahwa IMHRP-P jauh lebih unggul dalam hal jumlah *node* sensor hidup vs *round* daripada PEGASIS. Dimana *round* ke 18-20 dan 21-23 jumlah *node* sensor IMHRP-P sebanyak 19 dan 10 sedangkan PEGASIS jumlah *node* sensor yang hidup pada *round* yang sama adalah sebanyak 11 dan 4. Hal ini dikarenakan pada IMHRP-P dilakukan pemecahan rute pengiriman paket data sehingga tidak membentuk struktur rantai yang panjang dan dalam pemecahan rute meminimalisir penggunaan daya pancar yang besar sehingga dapat meminimalisir penggunaan daya (Watt) pada masing-masing perangkat *node* sensor. Dengan meminimalisir penggunaan daya (Watt) dapat meningkatkan NL pada pemodelan *routing* protokol tersebut. Sedangkan untuk grafik perbandingan PEGASIS dan IMHRP-P dengan jumlah *node* sensor sebanyak 20 ditunjukkan dalam Gambar 4.21.

Gambar 4.21 menunjukkan grafik perbandingan jumlah *node* sensor hidup vs *round* pemodelan *routing* protokol PEGASIS dan IMHRP-P dengan jumlah *node* sensor sebanyak 35. Dari hasil pembuatan grafik menunjukkan IMHRP-P jauh lebih unggul dalam hal jumlah *node* sensor hidup daripada PEGASIS. Hal ini dikarenakan pada IMHRP-P dilakukan pemecahan rute sehingga tidak membentuk struktur rantai yang panjang seperti PEGASIS. Selain itu dengan bertambahnya jumlah *node* sensor maka jarak antar *node* sensor satu dengan yang lainnya atau jarak *node* sensor dengan BS. Sehingga tidak membutuhkan daya (Watt) yang cukup besar untuk melakukan pengiriman paket data. Dengan meminimalisir penggunaan daya (Watt) maka dapat meningkatkan NL pada pemodelan *routing* protokol tersebut. Dari hasil grafik menunjukkan bahwa pada *round* 18-20 IMHRP-P memiliki jumlah *node* sensor yang hidup sebanyak 34 sedangkan pada PEGASIS terdapat 29 *node* sensor yang hidup. Sedangkan pada *round* ke 21-23 pada IMHRP-P terdapat 25 *node* sensor yang hidup sedangkan pada PEGASIS terdapat 21 *node* sensor. Selain itu pada *round* ke 24-27 *routing* protokol IMHRP-P terdapat 10 *node* sensor yang masih hidup dan pada PEGASIS memiliki 5 *node* sensor yang hidup.



Gambar 4.21 Jumlah *Node* Sensor Hidup dengan 35 *Node* Sensor Skenario Pertama

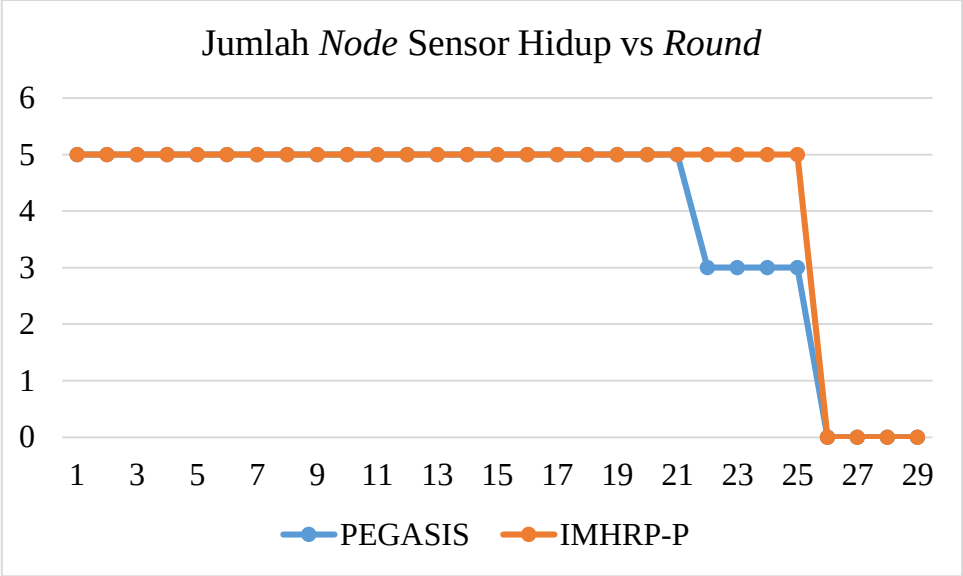
Sedangkan untuk hasil pengujian jumlah *node* sensor hidup vs *round* ditunjukkan dalam Tabel 4.17. Untuk pengujian pada skenario kedua menggunakan jumlah *node* sensor yang sama dengan skenario pertama, yaitu 2, 20 dan 35 *node* sensor. Selain itu juga menggunakan luas area yang sma, yaitu 15 m x 15 m dan 100 m x 100 m. Dimana dari hasil pengujian menunjukkan bahwa pada pemodelan *routing* protokol IMHRP-P dengan jumlah *node* sensor sebanyak 35 jauh lebih unggul dalam hal jumlah *node* sensor yang hidup hingga *round* ke 27 daripada PEGASIS dan IMHRP-P baik untuk jumlah *node* sensor 35, 5 maupun 20. Hal ini dikarenakan dalam IMHRP-P terdapat 12 *node* sensor yang menggunakan daya pancar -18 dBm yang mengkonsumsi daya (Watt) yang cukup rendah, yaitu 0,161 Watt. Sehingga 12 *node* sensor ini dapat bertahan hingga 13,5 jam atau 27 *round*. Untuk mempermudah dalam melakukan perbandingan jumlah *node* sensor yang hidup vs *round*, data ditampilkan dalam bentuk grafik terhadap setiap adanya penambahan *node* sensor. Adapun grafik perbandingan antara PEGASIS dan IMHRP-P untuk jumlah *node* sensor sebanyak 5 ditunjukkan dalam Gambar 4.22.

Tabel 4.17 Jumlah *Node* Sensor Hidup vs *Round* Skenario Kedua

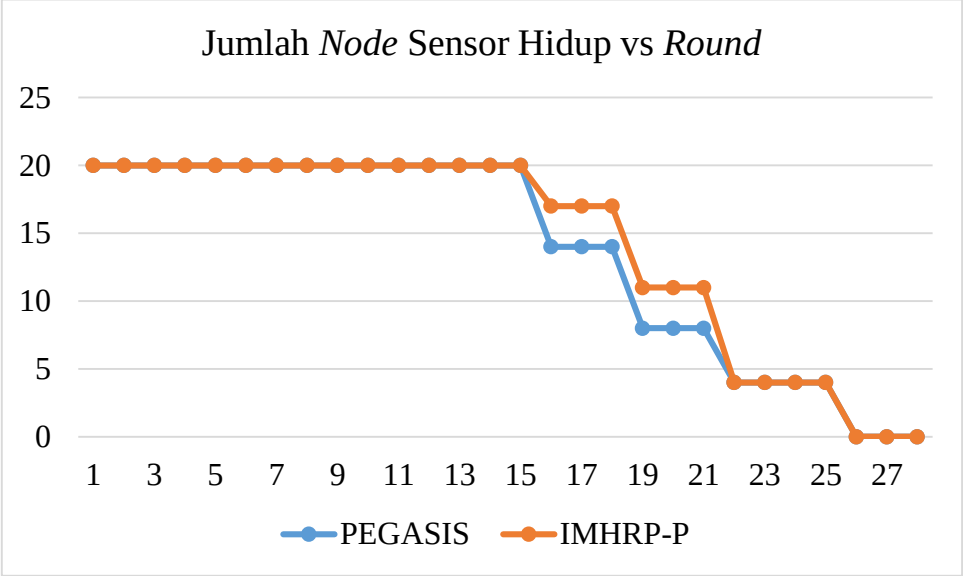
Routing	**Jumlah *Node* Sensor**	***Round***				
		18	**21**	**24**	**27**	**30**
PEGASIS	5	5	5	5	3	0
IMHRP-P	5	5	5	5	5	0
PEGASIS	20	14	8	4	4	0
IMHRP-P	20	16	9	3	3	0
PEGASIS	35	31	19	10	10	0
IMHRP-P	35	33	19	12	12	0

Gambar 4.22 menunjukkan grafik perbandingan antara PEGASIS dan IMHRP-P dengan jumlah *node* sensor sebanyak 5. Hasil dari grafik tersebut menunjukkan bahwa IMHRP-P lebih unggul dari PEGASIS. Dimana jumlah *node* sensor yang hidup sebanyak 5 sampai *round* ke 27 sedangkan PEGASIS terdapat penurunan jumlah *node* sensor yang hidup pada *round* ke 24-27 menjadi 3 *node* sensor yang hidup pada *round* tersebut. Hal ini dikarenakan pada IMHRP-P dilakukan pemecahan rute menjadi 3 bagian. Dimana masing-masing *node* sensor dapat menggunakan daya pancar yang sama, yaitu -18 dBm sehingga dapat

meminimalisir penggunaan daya (Watt) masing-masing *node* sensor. Sedangkan untuk grafik perbandingan PEGASIS dan IMHRP-P dengan jumlah *node* sensor sebanyak 20 ditunjukkan dalam Gambar 4.23.

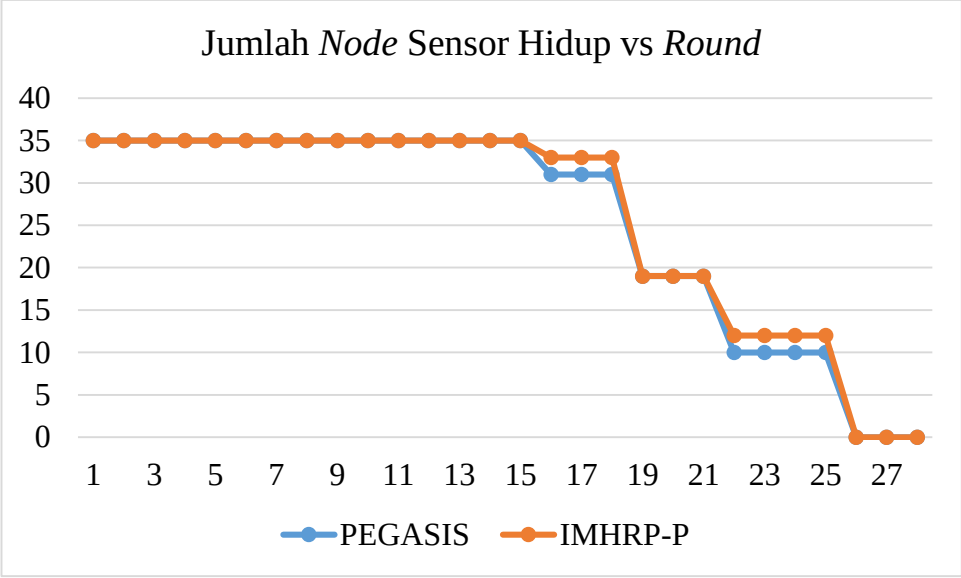


Gambar 4.22 Jumlah *Node* Sensor Hidup dengan 5 *Node* Sensor Skenario Kedua



Gambar 4.23 umlah *Node* Sensor Hidup dengan 20 *Node* Sensor Skenario Kedua

Gambar 4.23 menunjukkan grafik perbandingan antara PEGASIS dan IMHRP-P dengan jumlah *node* sensor sebanyak 20. Dari grafik tersebut menunjukkan bahwa IMHRP-P jauh lebih unggul dalam hal jumlah *node* sensor hidup vs *round* daripada PEGASIS pada *round* 1-23. Dimana *round* ke 18-20 jumlah *node* sensor hidup IMHRP-P jauh lebih unggul dari PEGASIS, yaitu 17 *node* sensor yang masih hidup sedangkan pada PEGASIS terdapat 14 *node* sensor yang masih hidup. Pada *round* ke 21-23 jumlah *node* sensor yang hidup pada IMHRP-P 11 sebanyak *node* sensor sedangkan pada PEGASIS terdapat sebanyak 8 *node* sensor yang hidup. Sedangkan pada *round* ke 24-27 jumlah *node* sensor yang masih hidup pada PEGASIS dan IMHRP-P sama. Hal ini dikarenakan pada kedua model *routing* menggunakan jenis daya pancar yang sama dengan jumlah *node* sensor yang sama. Sedangkan untuk grafik perbandingan PEGASIS dan IMHRP-P dengan jumlah *node* sensor sebanyak 35 ditunjukkan dalam Gambar 4.24.



Gambar 4.24 Jumlah *Node* Sensor Hidup dengan 35 *Node* Sensor Skenario Kedua

Gambar 4.24 menunjukkan grafik perbandingan jumlah *node* sensor hidup vs *round* pemodelan *routing* protokol PEGASIS dan IMHRP-P dengan jumlah *node* sensor sebanyak 35. Dari hasil pembuatan grafik menunjukkan IMHRP-P jauh lebih

unggul dalam hal jumlah *node* sensor hidup daripada PEGASIS. Hal ini dikarenakan pada IMHRP-P dilakukan pemecahan rute menjadi 3 bagian sehingga tidak membentuk struktur rantai yang panjang seperti PEGASIS. Selain itu dengan bertambahnya jumlah *node* sensor maka jarak antar *node* sensor satu dengan yang lainnya atau jarak *node* sensor dengan BS. Sehingga tidak membutuhkan daya (Watt) yang cukup besar untuk melakukan pengiriman paket data. Dengan meminimalisir penggunaan daya (Watt) maka dapat meningkatkan NL pada pemodelan *routing* protokol tersebut. Dari hasil grafik menunjukkan bahwa pada *round* 18-20 IMHRP-P memiliki jumlah *node* sensor yang hidup sebanyak 33 sedangkan pada PEGASIS terdapat 31 *node* sensor yang hidup. Sedangkan pada *round* ke 24-27 pada IMHRP-P terdapat 12 *node* sensor yang hidup sedangkan pada PEGASIS terdapat 10 *node* sensor.

Untuk pengujian jumlah *node* sensor hidup vs *round* baik untuk skenario pertama maupun kedua, waktu maksimum *node* sensor hanya dapat beroperasi hingga 13,5 jam dengan menggunakan daya pancar yang paling kecil, yaitu -18 dBm. Kemudian untuk daya pancar -12 dBm dapat beroperasi hingga 12 jam, daya pancar -6 dBm dapat beroperasi hingga 10,5 jam sedangkan untuk 0 dBm hanya dapat beroprasi hingga 9 jam seperti hasil yang telah ditunjukkan dalam pengujian perangkat monitoring. Hal ini dikarenakan adanya keterbatasan daya yang diberikan oleh baterai untuk memberikan daya pada *node* sensor dalam beroperasi.

Halaman ini sengaja dikosongkan

BAB 5
KESIMPULAN

5.1 Kesimpulan

Berdasarkan hasil pembahasan penelitian dalam tesis ini, maka dapat diambil kesimpulan sebagai berikut.

1. Untuk memperpendek rantai pada IMHRP-P, *node* sensor dibagi menjadi 2 klasifikasi, yaitu *node* sensor ganjil dan genap. Untuk *node* sensor ganjil dapat langsung mengirimkan paket data menuju BS menggunakan teknik *single hop*. Sedangkan untuk *node* sensor genap dapat mengirimkan data menuju *node* sensor selanjutnya terdekat kemudian dari *node* sensor selanjutnya terdekat akan mengirimkan paket data menuju BS menggunakan teknik *multi hop*.
2. Hasil perbandingan energi antara pemodelan *routing* protokol PEGASIS dan IMHRP-P dengan jumlah *node* sensor sebanyak 5 serta luas area yang disimulasikan adalah 15 m x 15 m menunjukkan bahwa pada skenario pertama PEGASIS jauh lebih unggul dari IMHRP-P dimana mengkonsumsi daya sebesar 0,163 Watt sedangkan IMHRP-P mengkonsumsi daya sebesar 0,165 Watt. Pada skenario kedua, IMHRP-P lebih efisien dalam penggunaan daya (Watt) daripada PEGASIS. Dimana IMHRP-P menggunakan daya sebesar 0,165 Watt dan PEGASIS menggunakan daya sebesar 0,165 Watt.
3. Dari hasil pengujian yang menambahkan *node* sensor dari 5 menjadi 20 dan 35 serta dengan penambahan luas area dari 15 m x 15 m menjadi 100 m x 100 m menunjukkan bahwa adanya peningkatan penggunaan daya (Watt) dari 5 *node* sensor sensor ke 20 *node* sensor baik untuk skenario pertama maupun kedua. Sedangkan untuk penambahan *node* sensor dari 20 ke 35 terjadi penurunan penggunaan daya (Watt).

5.2 Saran

1. Untuk mengetahui kapasitas baterai (Ah) secara spesifik dapat dilakukan pengujian implementasi *routing* protokol pada *hardware*. Sehingga dapat mengetahui kapasitas baterai (Ah) secara spesifik. Karena pada *software* simulasi, arus yang digunakan konstan.

DAFTAR PUSTAKA

[1] Subdirektorat Statistik Demografi, BPS Kementrian PPN/ Bappenas, 2018, Proyeksi Penduduk Indonesia 2015 – 2045, Jakarta.

[2] N. Wang, N. Zhang, and M. Wang, "*Wireless* sensors in agriculture and food industry—Recent development and future perspective," Comput. Electron. Agricult., vol. 50, no. 1, pp. 1–14, 2006.

[3] S. Ivanov, K. Bhargava, and W. Donnelly, "Precision farming: Sensor analytics," IEEE Intell. Syst., vol. 30, no. 4, pp. 76–80, Jul./Aug. 2015.

[4] G. H. E. L. de Lima, L. C. e Silva, and P. F. R. Neto, "WSN as a tool for sup*port*ing agriculture in the precision irrigation," in Proc. 6th Int. Conf. Netw. Services, Cancún, Mexico, Mar. 2010, pp. 137–142.

[5] W. Merrill, "Where is the return on investment in *wireless* sensor networks?" IEEE *Wireless* Commun., vol. 17, no. 1, pp. 4–6, Feb. 2010

[6] S. E. Díaz, J. C. Pérez, A. C. Mateos, M.-C. Marinescu, and B. B. Guerra, "A novel methodology for the monitoring of the agricultural production process based on *wireless* sensor networks," Comput. Electron. Agricult., vol. 76, no. 2, pp. 252–265, 2011.

[7] M. S. M. Amin, W. Aimrun, S. M. Eltaib, and C. S. Chan, "Spatial soil variability mapping using electrical conductivity sensor for precision farming of rice," Int. J. Eng. Technol., vol. 1, no. 1, pp. 47–57, 2004.

[8] Lee, W.S. and R. Ehsani, "Sensing systems for precision agriculture in Florida", Computers and Electronics in Agriculture, 2015. 112: pp. 29.

[9] N. Yazdi, A. Mason, K. Najafi, and K. D.Wise, "A generic interface chip for capacitive sensors in low-power multi-parameter microsystems," Sensors and Actuators A: Physical, vol. 84, no. 3, pp. 351–361, 2000.

[10] IEEE 1451.2 Standard, "A Smart Transducer Interface for Sensors and Actuators. Transducer to Microprocessor Communication Protocols and

Transducer Electronic Paket data Sheet (TEDS) Formats," Piscataway, NJ: IEEE Standards Department, 1998.

[11] Muhammad Adi Permana, Analisa Algoritma LEACH Pada Jaringan Sensor Nirkabel, Proceeding Seminar Tugas Akhir Jurusan Teknik Elektro FTI-ITS, 2008.

[12] Mr. D. Shinde dan Prof. N. Siddiqui, "IOT Based Environment change Monitoring & Controlling in *Greenhouse* using WSN," International Conference on Information, Communication, Engineering and Technology (ICICET) 2018, pp. 1-5.

[13] S. Li, "Research on Intelligent Monitoring System of*Greenhouse* Intensity and CO2 Concentration Basedon STM32," IEEE International Conference on Mechatronics and Automation (ICMA) 2018, pp. 666-670.

[14] Z. Xinrong, ChangBo dan Jiang Mingxin, "Design of *Wireless* Monitoring System for *Greenhouse* Environmental Parameters Based on Fuzzy Control," 2017 International Conference on Computer Technology, Electronics and Communication (ICCTEC), pp. 1204-1207.

[15] S. Lindsey, C. Raghavendra dan K. M. Sivalingam, "Paket data Gathering Algorithms in Sensor *Network*s Using Energy Metrics," *IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS*, vol. 13 No. 9, pp. 924-935, September 2002.

[16] M. R. Mundada, S. Kiran, S. Khobanna, R. N. Varsha. dan S. A. George, "A Study on Energy Efficient *Routing* Protocols in *Wireless* Sensor *Network*s," *International Journal of Distributed and Parallel Systems (IJDPS)*, vol. 3, May 2012.

[17] Munir, Rinaldi dan e. al, "Eksplorasi Algoritma Brute Force, *Greedy,* dan Dynamic Programming untuk Persoalan Integer Knapsack," www.informatika.org, 2018.

[18] E. Fasolo, M. Rossi, J. Widmer dan M. Zorzi, "In-*Network* Aggregation Techniques For *Wireless* Sensor *Network*s: A Survey," *IEEE Wireless Comun,* vol. 14 no.2, pp. 70-87, April 2007.

[19] M. R. Mufid, M. U. H. A. Rasyid dan I. Syarif, "Performance Evaluation of PEGASIS Protocol for Energy Efficiency," *International Electronics Symposium on Engineering Technology and Applications (IES-ETA,* pp. 241-246, 2018.

[20] V. Kumar dan Ajay Khunteta," Energy EfficientPEGASIS *Routing* Protocol for *Wireless* Sensor Networks," 2018 2nd International Conference on Micro-Electronics and Telecommunication Engineering, pp. 91-95.

[21] J. Kulshrestha dan M. K. Mishra, "DPEGASIS: Distributed PEGASIS for *chain* construction by the *node* sensor s in the network or in a zone without having global network topology information," International Conference on Multimedia, Signal Processing and Communication Technologies (IMPACT) 2017, pp. 13-17.

[22] R. K. Yadav dan A. Singh, "Comparative Study of PEGASIS based Protocols in *Wireless* Sensor Netwroks," 1st India International Conference on Information Processing (IICIP) 2016.

[23] Pratama, I. P., & Suakanto, S. (2015). *Wireless Sensor Network.* Bandung: Informatika.

[24] Rosuliya, I. A. (2015). *IMPLEMENTASI ROUTING PROTOKOL WIRELESS SENSOR NETWORK PADA MIKROKONTROLER BERBASIS EFISIENSI DAYA.* Surabaya: Skripsi.

[25] Ahad, Abdul dan e. al, "Comparison of Energy Efficient *Routing* Protocols in *Wireless* Sensor *Network*," *American Journal of Networks and Communications 6.4,* pp. 67-73, 2017.

[26] Annisa, Y. (2017). *PENGARUH MOBILITAS SINK NODE* SENSOR *PADA WIRELESS SENSOR NETWORK (WSN) UNTUK PEMANTAUAN*

AKTIFITAS PERGERAKAN GAJAH DALAM AREA PENANGKARAN. Bandar Lampung: Skripsi.

[27] Lindsey, S., & Raghavendra, C. S., PEGASIS: Power-Efficient Gathering in Sensor Information Systems, Aerospace conference proceedings, 2002. IEEE. Vol. 3. IEEE, 2002.

[28] Z. Ali, A.Anilkumar dan et. all, "NS-3 *Network* Simulator," nsnam, 2011-2019. [Online]. Available: https://www.nsnam.org/. [Diakses 28 November 2019].

[29] A. N. Telkom University, "*Network* Simulation Training by Using NS-3," in *Module 14 & 15 May 2016*, Telkom University, Accsess Net, 2016, pp. 21-22.

[30] Y. Wang1, C. Hu, Z. Feng1, Y. Ren1, "*Wireless* Transmission Module Comparison", 2014 IEEE International Conference on Information and Automation (ICIA), Hailar, China, July, 2014.

[31] Mobasshir Mahbu," Design and Implementation of Multipurpose Radio Controller Unit Using nRF24L01 *Wireless* Transceiver Module and Arduino as MCU", International Journal of Digital Information and *Wireless* Communications (IJDIWC) 9(2): 61-72 The Society of Digital Information and *Wireless* Communications, 2019, pp. 61-72.

[32] N. Semiconductor, " Single chip 2.4 GHz Transceiver NRF24L01," Nordic Semiconductor ASA , Norway, 2006.

[33] P. Christ, B. Neuwinger, F. Werner, U. R¨uckert, "Performance analysis of the nRF24L01 ultra-lowpower transceiver in a multi-transmitter and multireceiver scenario", SENSORS, 2011 IEEE, Limerick, Ireland, October, 2011.

[34] K. Amelia, D. Yendri,M.Kom, R. Aisuwarya,M.Eng, "PERANCANGAN SISTEM MONITORING SUHU, KELEMBABAN DAN TITIK EMBUN UDARA SECARA REALTIME MENGGUNAKAN MKROKONTROLER ARDUINO DENGAN LOGIKA FUZZY YANG

DAPAT DIAKSES MELALUI INTERNET", Available: http://repo.unand.ac.id/id/eprint/782 [Diakses 20 Maret 2020].

[35] Aosong, "Temperature and humidity module DHT11 Product Manual", Aosong(Guangzhou) Electronics Co.,Ltd.

[36] Resign Desain Lab, "*Soil moisture* Sensor", Resign Desain Lab, 2008.

[37] Radiospares, "Arduino Nano Paket datasheet", Radiospares, pp.1-7.

[38] Radiospares, "Arduino Uno Paket datasheet", Radiospares, pp.1-7.

[39] K. Setiawan, Supriyadin, I. Santoso, R. Buana, "MENGHITUNG RUTE TERPENDEK MENGGUNAKAN ALGORITMA A* DENGAN FUNGSI EUCLIDEAN DISTANCE," Seminar Nasional Teknologi Informasi dan Komunikasi 2018 (SENTIKA 2018) ISSN:2089-9815 Yogyakarta, 23-24 Maret 2018, pp. 70-78.

Halaman ini sengaja dikosongkan

LAMPIRAN

1. **Source Coding BS**

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
#include <SoftwareSerial.h>

RF24 radio(8, 9);
SoftwareSerial paket data_serial(6, 7);

const byte address[6] = "00002";
String tujuan[1] = {"101"};
int channel[1] = {5};
unsigned long waktuKirim, timeout = 4000;
bool paket dataKirim;
String paket data;

void setup() {
  Serial.begin(9600);
  paket data_serial.begin(9600);
  radio.begin();
  radio.openWritingPipe(address);
  radio.setPALevel(RF24_PA_MAX);
  radio.stopListening();
  radio.powerUp();
  Serial.println("----Server----");
}

void loop() {
  for (int i = 0; i < 4; i++) {
```

```
    radio.setChannel(channel[i]);
    kirimPaket data(tujuan[i]);
    radio.openReadingPipe(1, address); delay(50);
    radio.startListening(); delay(50);
    waktuKirim = millis();
    while (1) {
     terimaPaket data();
     if (paket dataKirim == true) {
       paket data_serial.print(paket data);
       paket data_serial.println("");
       paket dataKirim = false;
     }

     if (millis() - waktuKirim >= timeout) {

       radio.openWritingPipe(address);
       radio.stopListening();
       break;
     }
    }
  }

}

void kirimPaket data(String pesan) {
  int ppaket data = pesan.length();
  char text[ppaket data];
  for (int i = 0; i < ppaket data; i++) {
    text[i] = pesan.charAt(i);
  }
  radio.write(&text, sizeof(text));
  Serial.print("Kirim Paket data " );
```

```
  Serial.println(pesan);
  pesan = "";
}

void terimaPaket data() {
  if (radio.available()) {
    char text[32] = "";
    radio.read(&text, sizeof(text));
    Serial.println(text);
    paket data = text;
    paket dataKirim = true;
  }
}
```

2. Source Coding *Node* sensor Sensor

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
#include <DHT.h>;

#define DHTPIN A3
#define SOILPIN A0
#define DHTTYPE DHT22

DHT dht(DHTPIN, DHTTYPE);
RF24 radio(8, 9);

float hum, temp, soil;
unsigned long waktuTampil = 0, jepaket datampil = 200;
const byte address[6] = "00002";
String alamat*Node* sensor = "101";
String paket data;
```

```
void setup() {
  Serial.begin(9600);
  radio.begin();
  radio.openReadingPipe(1, address);
  radio.setChannel(5);
  radio.setPALevel(RF24_PA_MAX);
  radio.startListening();
  radio.powerUp();
  pinMode(SOILPIN, INPUT);
  Serial.println("----Client----");
}

void loop() {
  terimaPaket data();
  if (millis() - waktuTampil >= jepaket datampil) {
    waktuTampil = millis();
    bacaSensor();
  }
}

void terimaPaket data(){
  radio.openReadingPipe(0,address);delay(50);
  radio.startListening();delay(50);
  if (radio.available()) {
    char text[32] = "";
    radio.read(&text, sizeof(text));
    Serial.println(text);
    String perintah = text;
    delay(300);
    radio.openWritingPipe(address);delay(50);
    radio.stopListening();delay(50);
```

```
    if (perintah == alamatNode  sensor){
      delay(200);
      kirimPaket data(paket data);
    }
  }
}

void kirimPaket data(String pesan){
  int ppaket data=pesan.length();
  char text[ppaket data];
  for (int i=0; i<ppaket data; i++){
    text[i]=pesan.charAt(i);
  }
  radio.write(&text,sizeof(text));
  Serial.print("Kirim Paket data " );
  Serial.println(pesan);
  pesan="";
}

void bacaSensor(){
  soil = analogRead(SOILPIN);
  //c1 - c3
  soil = map(soil, 625, 245, 0 * 100, 10 * 100) / 100.0;
  //c4
  //soil = map(soil, 675, 250, 0 * 100, 10 * 100) / 100.0;
  if (soil <= 0 ) {
    soil = 0;
  }
  else {
    soil = soil;
  }
  hum = dht.readHumidity();
```

```
  temp= dht.readTemperature();
  paket data = "Nilai Sensor";
  paket data = "_"+String(alamatNode
sensor)+"_"+String(soil)+"_"+String(hum)+"_"+String(temp);
}

void tampilSerial(){
  Serial.print("Soil: ");
  Serial.print(soil);
  Serial.print("\t");
  Serial.print("Humidity: ");
  Serial.print(hum);
  Serial.print("\t");
  Serial.print(" %, Temp: ");
  Serial.print(temp);
  Serial.print("\t");
  Serial.println(" Celsius");
}
```

3. Source Coding Skenario BS di Pojok PEGASIS

```
#include "ns3/netanim-module.h"
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/flow-monitor-helper.h"
#include "ns3/ipv4-global-routing-helper.h"
#include "ns3/energy-module.h"
#include "ns3/simple-device-energy-model.h"
#include "ns3/wifi-radio-energy-model-helper.h"
#include <iostream>
```

```
using namespace ns3;
NS_LOG_COMPONENT_DEFINE ("FirstScriptExample");
int main (int argc, char *argv[])
{
 //jarak
 /*

 */
 Config::SetDefault ("ns3::OnOffApplication::PaketSize", UintegerValue
(210));
 Config::SetDefault ("ns3::OnOffApplication::Paket dataRate",
StringValue ("448kb/s"));

 Time::SetResolution (Time::NS);
 LogComponentEnable ("UdpEcho*Client*Application",
LOG_LEVEL_INFO);
 LogComponentEnable ("UdpEchoServerApplication",
LOG_LEVEL_INFO);

 Node  sensorContainer node  sensors;
 node  sensors.Create (6);

 Node  sensorContainer n3n2 = Node  sensorContainer(node
sensors.Get(3), node  sensors.Get(2));
 Node  sensorContainer n2n1 = Node  sensorContainer(node
sensors.Get(2), node  sensors.Get(1));
 Node  sensorContainer n1n4 = Node  sensorContainer(node
sensors.Get(1), node  sensors.Get(4));
 Node  sensorContainer n4n5 = Node  sensorContainer(node
sensors.Get(4), node  sensors.Get(5));
 Node  sensorContainer n5n0 = Node  sensorContainer(node
sensors.Get(5), node  sensors.Get(0));
```

```
  InternetStackHelper stack;
  stack.Install (node  sensors);

  PointToPointHelper pointToPoint;
  pointToPoint.SetDeviceAttribute ("Paket dataRate", StringValue
("5Mbps"));
  pointToPoint.SetChannelAttribute ("Delay", StringValue ("1.4ms"));

  NetDeviceContainer d3d2 = pointToPoint.Install(n3n2);
  NetDeviceContainer d2d1 = pointToPoint.Install(n2n1);
  NetDeviceContainer d1d4 = pointToPoint.Install(n1n4);
  NetDeviceContainer d4d5 = pointToPoint.Install(n4n5);
  NetDeviceContainer d5d0 = pointToPoint.Install(n5n0);

  //NetDeviceContainer devices;
  //devices = pointToPoint.Install (node  sensors);
  //devices = pointToPoint.Install(node  sensors.Get(0), node
sensors.Get(1));

  Ipv4AddressHelper address;
  address.SetBase ("10.1.1.0", "255.255.255.0");
  Ipv4InterfaceContainer i3i2 = address.Assign(d3d2);
  address.SetBase("10.1.2.0", "255.255.255.0");
  Ipv4InterfaceContainer i2i1 = address.Assign(d2d1);
  address.SetBase("10.1.3.0", "255.255.255.0");
  Ipv4InterfaceContainer i1i4 = address.Assign(d1d4);
  address.SetBase("10.1.4.0", "255.255.255.0");
  Ipv4InterfaceContainer i4i5 = address.Assign(d4d5);
  address.SetBase("10.1.5.0", "255.255.255.0");
  Ipv4InterfaceContainer i5i0 = address.Assign(d5d0);
```

```
  Ipv4Global*Routing*Helper::Populate*Routing*Tables();
  uint16_t port = 9;

  //koneksi1
  OnOffHelper onoff("ns3::UdpSocketFactory",
                    Address(InetSocketAddress(i3i2.GetAddress(0), port)));
  onoff.SetConstantRate(Paket dataRate("448kb/s"),329*2);
  ApplicationContainer apps = onoff.Install(node  sensors.Get(2));
  apps.Start(Seconds(1.0));
  apps.Stop(Seconds(10.0));
    PaketSinkHelper sink("ns3::UdpSocketFactory",
                      Address(InetSocketAddress(Ipv4Address::GetAny(),
port)));
  apps = sink.Install(node  sensors.Get(3));
  apps.Start(Seconds(1.0));
  apps.Stop(Seconds(10.0));

  //koneksi2
  OnOffHelper onoff2("ns3::UdpSocketFactory",
                    Address(InetSocketAddress(i2i1.GetAddress(0), port)));
  onoff2.SetConstantRate(Paket dataRate("448kb/s"),329*2);
  ApplicationContainer apps2 = onoff2.Install(node  sensors.Get(1));
  apps2.Start(Seconds(1.0));
  apps2.Stop(Seconds(10.0));

  PaketSinkHelper sink2("ns3::UdpSocketFactory",
                      Address(InetSocketAddress(Ipv4Address::GetAny(),
port)));
  apps2 = sink2.Install(node  sensors.Get(2));
  apps2.Start(Seconds(1.0));
  apps2.Stop(Seconds(10.0));
```

```
  //koneksi3
  OnOffHelper onoff3("ns3::UdpSocketFactory",
                  Address(InetSocketAddress(i1i4.GetAddress(0), port)));
  onoff3.SetConstantRate(Paket dataRate("448kb/s"),329*2);
  ApplicationContainer apps3 = onoff3.Install(node  sensors.Get(4));
  apps3.Start(Seconds(1.0));
  apps3.Stop(Seconds(10.0));

  PaketSinkHelper sink3("ns3::UdpSocketFactory",
                    Address(InetSocketAddress(Ipv4Address::GetAny(),
port)));
  apps3 = sink3.Install(node  sensors.Get(1));
  apps3.Start(Seconds(1.0));
  apps3.Stop(Seconds(10.0));

  //koneksi4
  OnOffHelper onoff4("ns3::UdpSocketFactory",
                  Address(InetSocketAddress(i4i5.GetAddress(0), port)));
  onoff4.SetConstantRate(Paket dataRate("448kb/s"),329*2);
  ApplicationContainer apps4 = onoff4.Install(node  sensors.Get(5));
  apps4.Start(Seconds(1.0));
  apps4.Stop(Seconds(10.0));
    PaketSinkHelper sink4("ns3::UdpSocketFactory",
                    Address(InetSocketAddress(Ipv4Address::GetAny(),
port)));
  apps4 = sink4.Install(node  sensors.Get(4));
  apps4.Start(Seconds(1.0));
  apps4.Stop(Seconds(10.0));

  //koneksi5

  OnOffHelper onoff5("ns3::UdpSocketFactory",
```

```
                Address(InetSocketAddress(i5i0.GetAddress(0), port)));
  onoff5.SetConstantRate(Paket dataRate("448kb/s"), 329);
  ApplicationContainer apps5 = onoff5.Install(node  sensors.Get(0));
  apps5.Start(Seconds(1.0));
  apps5.Stop(Seconds(10.0));

   PaketSinkHelper sink5("ns3::UdpSocketFactory",
                  Address(InetSocketAddress(Ipv4Address::GetAny(),
port)));
  apps5 = sink5.Install(node  sensors.Get(5));
  apps5.Start(Seconds(1.0));
  apps5.Stop(Seconds(10.0));

  /*
  //koneksi1
  OnOffHelper onoff("ns3::UdpSocketFactory",
                Address(InetSocketAddress(i0i1.GetAddress(0), port)));
  onoff.SetConstantRate(Paket dataRate("448kb/s"),329*2);
  ApplicationContainer apps = onoff.Install(node  sensors.Get(1));
  apps.Start(Seconds(1.0));
  apps.Stop(Seconds(10.0));

  PaketSinkHelper sink("ns3::UdpSocketFactory",
                  Address(InetSocketAddress(Ipv4Address::GetAny(),
port)));
  apps = sink.Install(node  sensors.Get(0));
  apps.Start(Seconds(1.0));
  apps.Stop(Seconds(10.0));

   //koneksi2
  OnOffHelper onoff2("ns3::UdpSocketFactory",
                Address(InetSocketAddress(i1i2.GetAddress(0), port)));
```

```
  onoff2.SetConstantRate(Paket dataRate("448kb/s"),329*2);
  ApplicationContainer apps2 = onoff2.Install(node  sensors.Get(2));
  apps2.Start(Seconds(1.0));
  apps2.Stop(Seconds(10.0));

  PaketSinkHelper sink2("ns3::UdpSocketFactory",
                  Address(InetSocketAddress(Ipv4Address::GetAny(),
port)));
  apps2 = sink2.Install(node  sensors.Get(1));
  apps2.Start(Seconds(1.0));
  apps2.Stop(Seconds(10.0));

   //koneksi3
  OnOffHelper onoff3("ns3::UdpSocketFactory",
                Address(InetSocketAddress(i2i3.GetAddress(0), port)));
  onoff3.SetConstantRate(Paket dataRate("448kb/s"),329*2);
  ApplicationContainer apps3 = onoff3.Install(node  sensors.Get(3));
  apps3.Start(Seconds(1.0));
  apps3.Stop(Seconds(10.0));

  PaketSinkHelper sink3("ns3::UdpSocketFactory",
                  Address(InetSocketAddress(Ipv4Address::GetAny(),
port)));
  apps3 = sink3.Install(node  sensors.Get(2));
  apps3.Start(Seconds(1.0));
  apps3.Stop(Seconds(10.0));

  //koneksi4
  OnOffHelper onoff4("ns3::UdpSocketFactory",
                Address(InetSocketAddress(i3i4.GetAddress(0), port)));
  onoff4.SetConstantRate(Paket dataRate("448kb/s"),329*2);
  ApplicationContainer apps4 = onoff4.Install(node  sensors.Get(4));
```

```
  apps4.Start(Seconds(1.0));
  apps4.Stop(Seconds(10.0));

  PaketSinkHelper sink4("ns3::UdpSocketFactory",
                    Address(InetSocketAddress(Ipv4Address::GetAny(),
port)));
  apps4 = sink4.Install(node  sensors.Get(3));
  apps4.Start(Seconds(1.0));
  apps4.Stop(Seconds(10.0));

  //koneksi5
    OnOffHelper onoff5("ns3::UdpSocketFactory",
                  Address(InetSocketAddress(i4i5.GetAddress(0), port)));
  onoff5.SetConstantRate(Paket dataRate("448kb/s"), 329);
  ApplicationContainer apps5 = onoff5.Install(node  sensors.Get(5));
  apps5.Start(Seconds(1.0));
  apps5.Stop(Seconds(10.0));

  PaketSinkHelper sink5("ns3::UdpSocketFactory",
                    Address(InetSocketAddress(Ipv4Address::GetAny(),
port)));
  apps5 = sink5.Install(node  sensors.Get(4));
  apps5.Start(Seconds(1.0));
  apps5.Stop(Seconds(10.0));

  */

  /*
  //koneksi 1
  Ipv4InterfaceContainer interfaces = address.Assign (d0d1);
  UdpEchoServerHelper echoServer (port);
```

```
  ApplicationContainer serverApps = echoServer.Install (node  sensors.Get
(1));
  serverApps.Start (Seconds (1.0));
  serverApps.Stop (Seconds (10.0));

  UdpEchoClientHelper echoClient (interfaces.GetAddress (1), port);
  echoClient.SetAttribute ("MaxPakets", UintegerValue (1));
  echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
  echoClient.SetAttribute ("PaketSize", UintegerValue (1024));

  ApplicationContainer clientApps = echoClient.Install (node  sensors.Get
(0));
  clientApps.Start (Seconds (2.0));
  clientApps.Stop (Seconds (10.0));

  //koneksi 2
  Ipv4InterfaceContainer interfaces2 = address.Assign (d1d2);
  UdpEchoServerHelper echoServer2 (port);
  ApplicationContainer serverApps2 = echoServer2.Install (node
sensors.Get (2));
  serverApps2.Start (Seconds (1.0));
  serverApps2.Stop (Seconds (10.0));

  UdpEchoClientHelper echoClient2 (interfaces2.GetAddress (1), port);
  echoClient2.SetAttribute ("MaxPakets", UintegerValue (1));
  echoClient2.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
  echoClient2.SetAttribute ("PaketSize", UintegerValue (1024));

  ApplicationContainer clientApps2 = echoClient2.Install (node
sensors.Get (1));
  clientApps2.Start (Seconds (2.0));
  clientApps2.Stop (Seconds (10.0));
```

```
 */

  AnimationInterface anim ("animation.xml");
  anim.SetConstantPosition(node  sensors.Get(0), 0.5, 0.7);
  anim.SetConstantPosition(node  sensors.Get(1), 5.57, 6.86);
  anim.SetConstantPosition(node  sensors.Get(2), 12.2, 3.73);
  anim.SetConstantPosition(node  sensors.Get(3), 13.36, 10.89);
  anim.SetConstantPosition(node  sensors.Get(4), 1.35, 12.77);
  anim.SetConstantPosition(node  sensors.Get(5), 1.62, 1.39);
  anim.UpdateNode  sensorDescription(node  sensors.Get(0), "Base
Station");
  anim.UpdateNode  sensorDescription(node  sensors.Get(1), "N3");
  anim.UpdateNode  sensorDescription(node  sensors.Get(2), "N2");
  anim.UpdateNode  sensorDescription(node  sensors.Get(3), "N5");
  anim.UpdateNode  sensorDescription(node  sensors.Get(4), "N4");
  anim.UpdateNode  sensorDescription(node  sensors.Get(5), "Leader
Node  sensor");

  anim.UpdateNode  sensorColor(node  sensors.Get(0),255,255,0);
  anim.UpdateNode  sensorColor(node  sensors.Get(5),0,0,255);

  //Simulator::Stop(Seconds(11));
  Simulator::Run ();
  Simulator::Destroy ();

  //Perhitungan Energi
    float arus[] = {(float)0.044,
   (float)0.047,
   (float)0.047,
   (float)0.044,
   (float)0.044};
```

```
float battery = 0.594; //mAh
 int habisMenit[] = {
        (int)(battery/arus[0]*60.0)*3,
        (int)(battery/arus[1]*60.0)*3,
        (int)(battery/arus[2]*60.0)*3,
        (int)(battery/arus[3]*60.0)*3,
        (int)(battery/arus[4]*60.0)*3,
};
int menitMax = 0;
for(int i=0;i<5;i++){
        if(menitMax<habisMenit[i])
                menitMax = habisMenit[i];
}
for(int i=0;i<menitMax+100;i++){
        int jumlahHidup = 0;
        int jumlahMati = 0;
        if(i<habisMenit[0])jumlahHidup++;
        else jumlahMati++;
        if(i<habisMenit[1])jumlahHidup++;
        else jumlahMati++;
        if(i<habisMenit[2])jumlahHidup++;
        else jumlahMati++;
        if(i<habisMenit[3])jumlahHidup++;
        else jumlahMati++;
        if(i<habisMenit[4])jumlahHidup++;
        else jumlahMati++;
        if(i%30==0)
                std::cout<<jumlahHidup<<","<<jumlahMati<<std::endl;
}
/*
float jouleBat = 0.594;
float jouleAlat = 0.0455;
```

```
 int j = 0;
 for(;jouleBat>0;jouleBat-=jouleAlat){
        //if(j%1800==0)
        std::cout<<jouleBat<<std::endl;
        j++;
 }
 for(int i=0;i<100;i++)
        std::cout<<0<<std::endl;
 */
 return 0;
}
```

4. Source Coding Skenario BS di Tengah PEGASIS

```
#include "ns3/netanim-module.h"
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/flow-monitor-helper.h"
#include "ns3/ipv4-global-routing-helper.h"
#include "ns3/energy-module.h"
#include "ns3/simple-device-energy-model.h"
#include "ns3/wifi-radio-energy-model-helper.h"
#include <iostream>

 using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("FirstScriptExample");

int main (int argc, char *argv[])
{
 float nDelay = 1.4;
```

```
  //jarak
  /*

  */
  Config::SetDefault ("ns3::OnOffApplication::PaketSize", UintegerValue
(210));
  Config::SetDefault ("ns3::OnOffApplication::Paket dataRate",
StringValue ("448kb/s"));

  Time::SetResolution (Time::NS);
  LogComponentEnable ("UdpEcho*Client*Application",
LOG_LEVEL_INFO);
  LogComponentEnable ("UdpEchoServerApplication",
LOG_LEVEL_INFO);

  *Node*  sensorContainer *node*  sensors;
  *node*  sensors.Create (6);

  *Node*  sensorContainer n0n1 = *Node*  sensorContainer(*node*
sensors.Get(0), *node*  sensors.Get(1));
  *Node*  sensorContainer n1n2 = *Node*  sensorContainer(*node*
sensors.Get(1), *node*  sensors.Get(2));
  *Node*  sensorContainer n2n3 = *Node*  sensorContainer(*node*
sensors.Get(2), *node*  sensors.Get(3));
  *Node*  sensorContainer n3n4 = *Node*  sensorContainer(*node*
sensors.Get(3), *node*  sensors.Get(4));
  *Node*  sensorContainer n4n5 = *Node*  sensorContainer(*node*
sensors.Get(4), *node*  sensors.Get(5));

  InternetStackHelper stack;
  stack.Install (*node*  sensors);
```

```
  PointToPointHelper pointToPoint;
  pointToPoint.SetDeviceAttribute ("Paket dataRate", StringValue
("5Mbps"));
  pointToPoint.SetChannelAttribute ("Delay", StringValue ("1.4ms"));

  NetDeviceContainer d0d1 = pointToPoint.Install(n0n1);
  NetDeviceContainer d1d2 = pointToPoint.Install(n1n2);
  NetDeviceContainer d2d3 = pointToPoint.Install(n2n3);
  NetDeviceContainer d3d4 = pointToPoint.Install(n3n4);
  NetDeviceContainer d4d5 = pointToPoint.Install(n4n5);

    //NetDeviceContainer devices;
  //devices = pointToPoint.Install (node  sensors);
  //devices = pointToPoint.Install(node sensors.Get(0), node
sensors.Get(1));

  Ipv4AddressHelper address;
  address.SetBase ("10.1.1.0", "255.255.255.0");
  Ipv4InterfaceContainer i0i1 = address.Assign(d0d1);
  address.SetBase("10.1.2.0", "255.255.255.0");
  Ipv4InterfaceContainer i1i2 = address.Assign(d1d2);
  address.SetBase("10.1.3.0", "255.255.255.0");
  Ipv4InterfaceContainer i2i3 = address.Assign(d2d3);
  address.SetBase("10.1.4.0", "255.255.255.0");
  Ipv4InterfaceContainer i3i4 = address.Assign(d3d4);
  address.SetBase("10.1.5.0", "255.255.255.0");
  Ipv4InterfaceContainer i4i5 = address.Assign(d4d5);

  Ipv4Global*Routing*Helper::Populate*Routing*Tables();
  uint16_t port = 9;

  //koneksi1
```

```
  OnOffHelper onoff("ns3::UdpSocketFactory",
                    Address(InetSocketAddress(i0i1.GetAddress(0), port)));
  onoff.SetConstantRate(Paket dataRate("448kb/s"),329*2);
  ApplicationContainer apps = onoff.Install(node  sensors.Get(1));
  apps.Start(Seconds(1.0));
  apps.Stop(Seconds(10.0));

  PaketSinkHelper sink("ns3::UdpSocketFactory",
                       Address(InetSocketAddress(Ipv4Address::GetAny(),
port)));
  apps = sink.Install(node  sensors.Get(0));
  apps.Start(Seconds(1.0));
  apps.Stop(Seconds(10.0));

  //koneksi2
  OnOffHelper onoff2("ns3::UdpSocketFactory",
                    Address(InetSocketAddress(i1i2.GetAddress(0), port)));
  onoff2.SetConstantRate(Paket dataRate("448kb/s"),329*2);
  ApplicationContainer apps2 = onoff2.Install(node  sensors.Get(2));
  apps2.Start(Seconds(1.0));
  apps2.Stop(Seconds(10.0));
    PaketSinkHelper sink2("ns3::UdpSocketFactory",
                       Address(InetSocketAddress(Ipv4Address::GetAny(),
port)));
  apps2 = sink2.Install(node  sensors.Get(1));
  apps2.Start(Seconds(1.0));
  apps2.Stop(Seconds(10.0));

    //koneksi3
  OnOffHelper onoff3("ns3::UdpSocketFactory",
                    Address(InetSocketAddress(i2i3.GetAddress(0), port)));
  onoff3.SetConstantRate(Paket dataRate("448kb/s"),329*2);
```

```
  ApplicationContainer apps3 = onoff3.Install(node  sensors.Get(3));
  apps3.Start(Seconds(1.0));
  apps3.Stop(Seconds(10.0));

  PaketSinkHelper sink3("ns3::UdpSocketFactory",
                    Address(InetSocketAddress(Ipv4Address::GetAny(),
port)));
  apps3 = sink3.Install(node  sensors.Get(2));
  apps3.Start(Seconds(1.0));
  apps3.Stop(Seconds(10.0));

  //koneksi4
  OnOffHelper onoff4("ns3::UdpSocketFactory",
                 Address(InetSocketAddress(i3i4.GetAddress(0), port)));
  onoff4.SetConstantRate(Paket dataRate("448kb/s"),329*2);
  ApplicationContainer apps4 = onoff4.Install(node  sensors.Get(4));
  apps4.Start(Seconds(1.0));
  apps4.Stop(Seconds(10.0));

  PaketSinkHelper sink4("ns3::UdpSocketFactory",
                    Address(InetSocketAddress(Ipv4Address::GetAny(),
port)));
  apps4 = sink4.Install(node  sensors.Get(3));
  apps4.Start(Seconds(1.0));
  apps4.Stop(Seconds(10.0));

    //koneksi5

  OnOffHelper onoff5("ns3::UdpSocketFactory",
                 Address(InetSocketAddress(i4i5.GetAddress(0), port)));
  onoff5.SetConstantRate(Paket dataRate("448kb/s"), 329);
  ApplicationContainer apps5 = onoff5.Install(node  sensors.Get(5));
```

```
  apps5.Start(Seconds(1.0));
  apps5.Stop(Seconds(10.0));

  PaketSinkHelper sink5("ns3::UdpSocketFactory",
                    Address(InetSocketAddress(Ipv4Address::GetAny(),
port)));
  apps5 = sink5.Install(node  sensors.Get(4));
  apps5.Start(Seconds(1.0));
  apps5.Stop(Seconds(10.0));

  /*
  //koneksi1
  OnOffHelper onoff("ns3::UdpSocketFactory",
                 Address(InetSocketAddress(i0i1.GetAddress(0), port)));
  onoff.SetConstantRate(Paket dataRate("448kb/s"),329*2);
  ApplicationContainer apps = onoff.Install(node  sensors.Get(1));
  apps.Start(Seconds(1.0));
  apps.Stop(Seconds(10.0));

  PaketSinkHelper sink("ns3::UdpSocketFactory",
                    Address(InetSocketAddress(Ipv4Address::GetAny(),
port)));
  apps = sink.Install(node  sensors.Get(0));
  apps.Start(Seconds(1.0));
  apps.Stop(Seconds(10.0));

  //koneksi2
  OnOffHelper onoff2("ns3::UdpSocketFactory",
                 Address(InetSocketAddress(i1i2.GetAddress(0), port)));
  onoff2.SetConstantRate(Paket dataRate("448kb/s"),329*2);
  ApplicationContainer apps2 = onoff2.Install(node  sensors.Get(2));
  apps2.Start(Seconds(1.0));
```

```
  apps2.Stop(Seconds(10.0));

  PaketSinkHelper sink2("ns3::UdpSocketFactory",
                        Address(InetSocketAddress(Ipv4Address::GetAny(),
port)));
  apps2 = sink2.Install(node  sensors.Get(1));
  apps2.Start(Seconds(1.0));
  apps2.Stop(Seconds(10.0));

    //koneksi3
  OnOffHelper onoff3("ns3::UdpSocketFactory",
                     Address(InetSocketAddress(i2i3.GetAddress(0), port)));
  onoff3.SetConstantRate(Paket dataRate("448kb/s"),329*2);
  ApplicationContainer apps3 = onoff3.Install(node  sensors.Get(3));
  apps3.Start(Seconds(1.0));
  apps3.Stop(Seconds(10.0));
     PaketSinkHelper sink3("ns3::UdpSocketFactory",
                        Address(InetSocketAddress(Ipv4Address::GetAny(),
port)));
  apps3 = sink3.Install(node  sensors.Get(2));
  apps3.Start(Seconds(1.0));
  apps3.Stop(Seconds(10.0));


  //koneksi4
  OnOffHelper onoff4("ns3::UdpSocketFactory",
                     Address(InetSocketAddress(i3i4.GetAddress(0), port)));
  onoff4.SetConstantRate(Paket dataRate("448kb/s"),329*2);
  ApplicationContainer apps4 = onoff4.Install(node  sensors.Get(4));
  apps4.Start(Seconds(1.0));
  apps4.Stop(Seconds(10.0));
```

```
  PaketSinkHelper sink4("ns3::UdpSocketFactory",
                    Address(InetSocketAddress(Ipv4Address::GetAny(),
port)));
  apps4 = sink4.Install(node  sensors.Get(3));
  apps4.Start(Seconds(1.0));
  apps4.Stop(Seconds(10.0));

  //koneksi5

  OnOffHelper onoff5("ns3::UdpSocketFactory",
                 Address(InetSocketAddress(i4i5.GetAddress(0), port)));
  onoff5.SetConstantRate(Paket dataRate("448kb/s"), 329);
  ApplicationContainer apps5 = onoff5.Install(node  sensors.Get(5));
  apps5.Start(Seconds(1.0));
  apps5.Stop(Seconds(10.0));
     PaketSinkHelper sink5("ns3::UdpSocketFactory",
                    Address(InetSocketAddress(Ipv4Address::GetAny(),
port)));
  apps5 = sink5.Install(node  sensors.Get(4));
  apps5.Start(Seconds(1.0));
  apps5.Stop(Seconds(10.0));

  */

  /*
  //koneksi 1
  Ipv4InterfaceContainer interfaces = address.Assign (d0d1);
  UdpEchoServerHelper echoServer (port);
  ApplicationContainer serverApps = echoServer.Install (node  sensors.Get
(1));
  serverApps.Start (Seconds (1.0));
```

```
  serverApps.Stop (Seconds (10.0));

  UdpEcho*Client*Helper echo*Client* (interfaces.GetAddress (1), *port*);
  echo*Client*.SetAttribute ("MaxPakets", UintegerValue (1));
  echo*Client*.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
  echo*Client*.SetAttribute ("PaketSize", UintegerValue (1024));

  ApplicationContainer *client*Apps = echo*Client*.Install (*node*  sensors.Get
(0));
  *client*Apps.Start (Seconds (2.0));
  *client*Apps.Stop (Seconds (10.0));

  //koneksi 2
  Ipv4InterfaceContainer interfaces2 = address.Assign (d1d2);
  UdpEchoServerHelper echoServer2 (*port*);
  ApplicationContainer serverApps2 = echoServer2.Install (*node*
sensors.Get (2));
  serverApps2.Start (Seconds (1.0));
  serverApps2.Stop (Seconds (10.0));
  UdpEcho*Client*Helper echo*Client*2 (interfaces2.GetAddress (1), *port*);
  echo*Client*2.SetAttribute ("MaxPakets", UintegerValue (1));
  echo*Client*2.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
  echo*Client*2.SetAttribute ("PaketSize", UintegerValue (1024));

  ApplicationContainer *client*Apps2 = echo*Client*2.Install (*node*
sensors.Get (1));
  *client*Apps2.Start (Seconds (2.0));
  *client*Apps2.Stop (Seconds (10.0));
  */

  AnimationInterface anim ("animation.xml");
  anim.SetConstantPosition(*node*  sensors.Get(0), 7.4, 7.3);
```

```
  anim.SetConstantPosition(node  sensors.Get(1), 5.57, 6.86);
  anim.SetConstantPosition(node  sensors.Get(2), 12.2, 3.73);
  anim.SetConstantPosition(node  sensors.Get(3), 13.36, 10.89);
  anim.SetConstantPosition(node  sensors.Get(4), 1.35, 12.77);
  anim.SetConstantPosition(node  sensors.Get(5), 1.62, 1.39);

  anim.UpdateNode  sensorDescription(node  sensors.Get(0), "Base
Station");
  anim.UpdateNode  sensorDescription(node  sensors.Get(1), "Leader
Node  sensor");
  anim.UpdateNode  sensorDescription(node  sensors.Get(2), "N2");
  anim.UpdateNode  sensorDescription(node  sensors.Get(3), "N5");
  anim.UpdateNode  sensorDescription(node  sensors.Get(4), "N4");
  anim.UpdateNode  sensorDescription(node  sensors.Get(5), "N1");

  anim.UpdateNode  sensorColor(node  sensors.Get(0),255,255,0);
  anim.UpdateNode  sensorColor(node  sensors.Get(1),0,0,255);

  //Simulator::Stop(Seconds(11));
  Simulator::Run ();
  Simulator::Destroy ();

  //Perhitungan Energi

  float arusPerangkat = 110; //mAh
  float arusTerima = 12.3;
  int jumlahNode  sensor = 5;
  float arus[] = {(float)7+arusPerangkat+arusTerima,
   (float)7.5+arusPerangkat+arusTerima,
   (float)7+arusPerangkat+arusTerima,
   (float)7+arusPerangkat+arusTerima,
   (float)7+arusPerangkat+arusTerima};
```

```
  //float totalArus[] = {0,0,0,0,0};
  //float tDelay = 1.5;
  float battery = 2200; //mAh
  float sisaArus[] = {battery,battery,battery,battery,battery};
  int node  sensorLoop [] = {0,0,0,0};

  int iter = 0;
  int checkZero = 0;
  for(iter=0;;iter++){
    for(int i=0;i<jumlahNode  sensor;i++){
      sisaArus[i]-=arus[i];
    }
    checkZero = 0;
    for(int i=0;i<jumlahNode  sensor;i++){
      if(sisaArus[i]<=0)checkZero++;
      else node  sensorLoop[i]++;
    }
    if(checkZero==5)break;
  }
  for(int i=0;i<4;i++){
    std::cout<<"Total loop on Node  sensor "<<i+1<<": "<<node
sensorLoop[i]<<std::endl;
  }
  std::cout<<"Total loop until no power left: "<<iter<<std::endl;

  std::cout<<"Node  sensor Delay: "<<nDelay<<std::endl;

  return 0;
}
```

5. Source Coding Skenario BS di Pojok IMHRP-P

```
#include "ns3/netanim-module.h"
```

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/flow-monitor-helper.h"
#include "ns3/ipv4-global-routing-helper.h"
#include "ns3/energy-module.h"
#include "ns3/simple-device-energy-model.h"
#include "ns3/wifi-radio-energy-model-helper.h"
#include <iostream>

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("FirstScriptExample");

int main (int argc, char *argv[])
{
  float nDelay = 1.4;
  //jarak
  /*

  */
  Config::SetDefault ("ns3::OnOffApplication::PaketSize", UintegerValue
(210));
  Config::SetDefault ("ns3::OnOffApplication::Paket dataRate",
StringValue ("448kb/s"));

  Time::SetResolution (Time::NS);
  LogComponentEnable ("UdpEchoClientApplication",
LOG_LEVEL_INFO);
```

```
  LogComponentEnable ("UdpEchoServerApplication",
LOG_LEVEL_INFO);

  Node  sensorContainer node  sensors;
  node  sensors.Create (6);

  Node  sensorContainer n0n5 = Node  sensorContainer(node
sensors.Get(0), node  sensors.Get(5));
  Node  sensorContainer n5n1 = Node  sensorContainer(node
sensors.Get(5), node  sensors.Get(1));

  Node  sensorContainer n0n2 = Node  sensorContainer(node
sensors.Get(0), node  sensors.Get(2));
  Node  sensorContainer n2n3 = Node  sensorContainer(node
sensors.Get(2), node  sensors.Get(3));
  Node  sensorContainer n0n4 = Node  sensorContainer(node
sensors.Get(0), node  sensors.Get(4));

  InternetStackHelper stack;
  stack.Install (node  sensors);

  PointToPointHelper pointToPoint;
  pointToPoint.SetDeviceAttribute ("Paket dataRate", StringValue
("5Mbps"));
  pointToPoint.SetChannelAttribute ("Delay", StringValue ("1.4ms"));

  NetDeviceContainer d0d5 = pointToPoint.Install(n0n5);
  NetDeviceContainer d5d1 = pointToPoint.Install(n5n1);
  NetDeviceContainer d0d2 = pointToPoint.Install(n0n2);
  NetDeviceContainer d2d3 = pointToPoint.Install(n2n3);
  NetDeviceContainer d0d4 = pointToPoint.Install(n0n4);
```

```
  //NetDeviceContainer devices;
  //devices = pointToPoint.Install (node  sensors);
  //devices = pointToPoint.Install(node  sensors.Get(0), node
sensors.Get(1));

  Ipv4AddressHelper address;
  address.SetBase ("10.1.1.0", "255.255.255.0");
  Ipv4InterfaceContainer i0i5 = address.Assign(d0d5);
  address.SetBase("10.1.2.0", "255.255.255.0");
  Ipv4InterfaceContainer i5i1 = address.Assign(d5d1);
  address.SetBase("10.1.3.0", "255.255.255.0");
  Ipv4InterfaceContainer i0i2 = address.Assign(d0d2);
  address.SetBase("10.1.4.0", "255.255.255.0");
  Ipv4InterfaceContainer i2i3 = address.Assign(d2d3);
  address.SetBase("10.1.5.0", "255.255.255.0");
  Ipv4InterfaceContainer i0i4 = address.Assign(d0d4);

  Ipv4Global Routing Helper::Populate Routing Tables();
  uint16_t port = 9;

  //koneksi1
  OnOffHelper onoff("ns3::UdpSocketFactory",
                    Address(InetSocketAddress(i0i5.GetAddress(0), port)));
  onoff.SetConstantRate(Paket dataRate("448kb/s"),329*2);
  ApplicationContainer apps = onoff.Install(node  sensors.Get(5));
  apps.Start(Seconds(1.0));
  apps.Stop(Seconds(3.0));

    PaketSinkHelper sink("ns3::UdpSocketFactory",
                      Address(InetSocketAddress(Ipv4Address::GetAny(),
port)));
```

```
  apps = sink.Install(node  sensors.Get(0));
  apps.Start(Seconds(1.0));
  apps.Stop(Seconds(3.0));

  //koneksi2
  OnOffHelper onoff2("ns3::UdpSocketFactory",
                Address(InetSocketAddress(i5i1.GetAddress(0), port)));
  onoff2.SetConstantRate(Paket dataRate("448kb/s"),329*2);
  ApplicationContainer apps2 = onoff2.Install(node  sensors.Get(1));
  apps2.Start(Seconds(1.0));
  apps2.Stop(Seconds(3.0));

  PaketSinkHelper sink2("ns3::UdpSocketFactory",
                  Address(InetSocketAddress(Ipv4Address::GetAny(),
port)));
  apps2 = sink2.Install(node  sensors.Get(5));
  apps2.Start(Seconds(1.0));
  apps2.Stop(Seconds(3.0));

  //koneksi3
  OnOffHelper onoff3("ns3::UdpSocketFactory",
                Address(InetSocketAddress(i0i2.GetAddress(0), port)));
  onoff3.SetConstantRate(Paket dataRate("448kb/s"),329*2);
  ApplicationContainer apps3 = onoff3.Install(node  sensors.Get(2));
  apps3.Start(Seconds(3.0));
  apps3.Stop(Seconds(5.0));

  PaketSinkHelper sink3("ns3::UdpSocketFactory",
                  Address(InetSocketAddress(Ipv4Address::GetAny(),
port)));
  apps3 = sink3.Install(node  sensors.Get(0));
  apps3.Start(Seconds(3.0));
```

```
  apps3.Stop(Seconds(5.0));

  //koneksi4
  OnOffHelper onoff4("ns3::UdpSocketFactory",
                  Address(InetSocketAddress(i2i3.GetAddress(0), port)));
  onoff4.SetConstantRate(Paket dataRate("448kb/s"),329*2);
  ApplicationContainer apps4 = onoff4.Install(node  sensors.Get(3));
  apps4.Start(Seconds(3.0));
  apps4.Stop(Seconds(5.0));

  PaketSinkHelper sink4("ns3::UdpSocketFactory",
                    Address(InetSocketAddress(Ipv4Address::GetAny(),
port)));
  apps4 = sink4.Install(node  sensors.Get(2));
  apps4.Start(Seconds(1.0));
  apps4.Stop(Seconds(10.0));

  //koneksi5

  OnOffHelper onoff5("ns3::UdpSocketFactory",
                  Address(InetSocketAddress(i0i4.GetAddress(0), port)));
  onoff5.SetConstantRate(Paket dataRate("448kb/s"), 329);
  ApplicationContainer apps5 = onoff5.Install(node  sensors.Get(4));
  apps5.Start(Seconds(5.0));
  apps5.Stop(Seconds(6.0));

  PaketSinkHelper sink5("ns3::UdpSocketFactory",
                    Address(InetSocketAddress(Ipv4Address::GetAny(),
port)));
  apps5 = sink5.Install(node  sensors.Get(0));
  apps5.Start(Seconds(5.0));
  apps5.Stop(Seconds(6.0));
```

```
/*
//koneksi1
OnOffHelper onoff("ns3::UdpSocketFactory",
                  Address(InetSocketAddress(i0i1.GetAddress(0), port)));
onoff.SetConstantRate(Paket dataRate("448kb/s"),329*2);
ApplicationContainer apps = onoff.Install(node  sensors.Get(1));
apps.Start(Seconds(1.0));
apps.Stop(Seconds(10.0));

PaketSinkHelper sink("ns3::UdpSocketFactory",
                     Address(InetSocketAddress(Ipv4Address::GetAny(),
port)));
apps = sink.Install(node  sensors.Get(0));
apps.Start(Seconds(1.0));
apps.Stop(Seconds(10.0));

//koneksi2
OnOffHelper onoff2("ns3::UdpSocketFactory",
                  Address(InetSocketAddress(i1i2.GetAddress(0), port)));
onoff2.SetConstantRate(Paket dataRate("448kb/s"),329*2);
ApplicationContainer apps2 = onoff2.Install(node  sensors.Get(2));
apps2.Start(Seconds(1.0));
apps2.Stop(Seconds(10.0));

  PaketSinkHelper sink2("ns3::UdpSocketFactory",
                     Address(InetSocketAddress(Ipv4Address::GetAny(),
port)));
apps2 = sink2.Install(node  sensors.Get(1));
apps2.Start(Seconds(1.0));
apps2.Stop(Seconds(10.0));
```

```
  //koneksi3
  OnOffHelper onoff3("ns3::UdpSocketFactory",
                  Address(InetSocketAddress(i2i3.GetAddress(0), port)));
  onoff3.SetConstantRate(Paket dataRate("448kb/s"),329*2);
  ApplicationContainer apps3 = onoff3.Install(node  sensors.Get(3));
  apps3.Start(Seconds(1.0));
  apps3.Stop(Seconds(10.0));

   PaketSinkHelper sink3("ns3::UdpSocketFactory",
                    Address(InetSocketAddress(Ipv4Address::GetAny(),
port)));
  apps3 = sink3.Install(node  sensors.Get(2));
  apps3.Start(Seconds(1.0));
  apps3.Stop(Seconds(10.0));

  //koneksi4
  OnOffHelper onoff4("ns3::UdpSocketFactory",
                  Address(InetSocketAddress(i3i4.GetAddress(0), port)));
  onoff4.SetConstantRate(Paket dataRate("448kb/s"),329*2);
  ApplicationContainer apps4 = onoff4.Install(node  sensors.Get(4));
  apps4.Start(Seconds(1.0));
  apps4.Stop(Seconds(10.0));

  PaketSinkHelper sink4("ns3::UdpSocketFactory",
                    Address(InetSocketAddress(Ipv4Address::GetAny(),
port)));
  apps4 = sink4.Install(node  sensors.Get(3));
  apps4.Start(Seconds(1.0));
  apps4.Stop(Seconds(10.0));

  //koneksi5
```

```
  OnOffHelper onoff5("ns3::UdpSocketFactory",
                     Address(InetSocketAddress(i4i5.GetAddress(0), port)));
  onoff5.SetConstantRate(Paket dataRate("448kb/s"), 329);
  ApplicationContainer apps5 = onoff5.Install(node  sensors.Get(5));
  apps5.Start(Seconds(1.0));
  apps5.Stop(Seconds(10.0));

  PaketSinkHelper sink5("ns3::UdpSocketFactory",
                        Address(InetSocketAddress(Ipv4Address::GetAny(),
port)));
  apps5 = sink5.Install(node  sensors.Get(4));
  apps5.Start(Seconds(1.0));
  apps5.Stop(Seconds(10.0));

  */
    /*
  //koneksi 1
  Ipv4InterfaceContainer interfaces = address.Assign (d0d1);
  UdpEchoServerHelper echoServer (port);
  ApplicationContainer serverApps = echoServer.Install (node  sensors.Get
(1));
  serverApps.Start (Seconds (1.0));
  serverApps.Stop (Seconds (10.0));

  UdpEchoClientHelper echoClient (interfaces.GetAddress (1), port);
  echoClient.SetAttribute ("MaxPakets", UintegerValue (1));
  echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
  echoClient.SetAttribute ("PaketSize", UintegerValue (1024));

  ApplicationContainer clientApps = echoClient.Install (node  sensors.Get
(0));
  clientApps.Start (Seconds (2.0));
```

```
  clientApps.Stop (Seconds (10.0));

  //koneksi 2
  Ipv4InterfaceContainer interfaces2 = address.Assign (d1d2);
  UdpEchoServerHelper echoServer2 (port);
  ApplicationContainer serverApps2 = echoServer2.Install (node
sensors.Get (2));
  serverApps2.Start (Seconds (1.0));
  serverApps2.Stop (Seconds (10.0));

  UdpEchoClientHelper echoClient2 (interfaces2.GetAddress (1), port);
  echoClient2.SetAttribute ("MaxPakets", UintegerValue (1));
  echoClient2.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
  echoClient2.SetAttribute ("PaketSize", UintegerValue (1024));

  ApplicationContainer clientApps2 = echoClient2.Install (node
sensors.Get (1));
  clientApps2.Start (Seconds (2.0));
  clientApps2.Stop (Seconds (10.0));
  */
     AnimationInterface anim ("animation.xml");
  anim.SetConstantPosition(node  sensors.Get(0), 0.5, 0.7);
  anim.SetConstantPosition(node  sensors.Get(1), 5.57, 6.86);
  anim.SetConstantPosition(node  sensors.Get(2), 12.2, 3.73);
  anim.SetConstantPosition(node  sensors.Get(3), 13.36, 10.89);
  anim.SetConstantPosition(node  sensors.Get(4), 1.35, 12.77);
  anim.SetConstantPosition(node  sensors.Get(5), 1.62, 1.39);

  anim.UpdateNode  sensorDescription(node  sensors.Get(0), "Base
Station");
  anim.UpdateNode  sensorDescription(node  sensors.Get(1), "N3");
  anim.UpdateNode  sensorDescription(node  sensors.Get(2), "N2");
```

```
anim.UpdateNode  sensorDescription(node  sensors.Get(3), "N5");
anim.UpdateNode  sensorDescription(node  sensors.Get(4), "N4");
anim.UpdateNode  sensorDescription(node  sensors.Get(5), "N5");

anim.UpdateNode  sensorColor(node  sensors.Get(0),255,255,0);
//anim.UpdateNode  sensorColor(node  sensors.Get(5),0,0,255);

//Simulator::Stop(Seconds(11));
Simulator::Run ();
Simulator::Destroy ();

//Perhitungan Energi

float arusPerangkat = 110; //mAh
float arusTerima = 12.3;
int jumlahNode  sensor = 5;
float arus[] = {(float)7.5+arusPerangkat+arusTerima,
 (float)7.5+arusPerangkat+arusTerima,
 (float)7+arusPerangkat+arusTerima,
 (float)7+arusPerangkat+arusTerima,
 (float)7+arusPerangkat+arusTerima};
//float totalArus[] = {0,0,0,0,0};
//float tDelay = 1.5;
float battery = 2200; //mAh
float sisaArus[] = {battery,battery,battery,battery,battery};
int node  sensorLoop [] = {0,0,0,0};

int iter = 0;
int checkZero = 0;
for(iter=0;;iter++){
  for(int i=0;i<jumlahNode  sensor;i++){
    sisaArus[i]-=arus[i];
```

```
    }
    checkZero = 0;
    for(int i=0;i<jumlahNode  sensor;i++){
      if(sisaArus[i]<=0)checkZero++;
      else node  sensorLoop[i]++;
    }
    if(checkZero==5)break;
  }
  for(int i=0;i<4;i++){
    std::cout<<"Total loop on Node  sensor "<<i+1<<": "<<node
sensorLoop[i]<<std::endl;
  }
  std::cout<<"Total loop until no power left: "<<iter<<std::endl;

  std::cout<<"Node  sensor Delay: "<<nDelay<<std::endl;

  int habisMenit[] = {
          (int)(battery/arus[0]*60.0),
          (int)(battery/arus[1]*60.0),
          (int)(battery/arus[2]*60.0),
          (int)(battery/arus[3]*60.0),
          (int)(battery/arus[4]*60.0),
  };
  int menitMax = 0;
  for(int i=0;i<5;i++){
          if(menitMax<habisMenit[i])
                  menitMax = habisMenit[i];
  }
  for(int i=0;i<menitMax+100;i++){
          int jumlahHidup = 0;
          int jumlahMati = 0;
          if(i<habisMenit[0])jumlahHidup++;
```

```
            else jumlahMati++;
            if(i<habisMenit[1])jumlahHidup++;
            else jumlahMati++;
            if(i<habisMenit[2])jumlahHidup++;
            else jumlahMati++;
            if(i<habisMenit[3])jumlahHidup++;
            else jumlahMati++;
            if(i<habisMenit[4])jumlahHidup++;
            else jumlahMati++;
            if(i%30==0)
            std::cout<<jumlahHidup<<","<<jumlahMati<<std::endl;
        }
        /*
        float jouleBat = (int)(battery*3.7*3.6);
        float jouleAlat = 0.59*5;
        int j = 0;
        for(;jouleBat>0;jouleBat-=jouleAlat){
            if(j%1800==0)std::cout<<jouleBat<<std::endl;
            j++;
        }
        for(int i=0;i<100;i++)
            std::cout<<0<<std::endl;
        */
        return 0;
    }
```

6. Source Coding Skenario BS di Tengah IMHRP-P

```
#include "ns3/netanim-module.h"
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
```

```
#include "ns3/applications-module.h"
#include "ns3/flow-monitor-helper.h"
#include "ns3/ipv4-global-routing-helper.h"
#include "ns3/energy-module.h"
#include "ns3/simple-device-energy-model.h"
#include "ns3/wifi-radio-energy-model-helper.h"
#include <iostream>

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("FirstScriptExample");

int main (int argc, char *argv[])
{

  Config::SetDefault ("ns3::OnOffApplication::PaketSize", UintegerValue (210));
  Config::SetDefault ("ns3::OnOffApplication::DataRate", StringValue
("448kb/s"));

  Time::SetResolution (Time::NS);
  LogComponentEnable ("UdpEcho*Client*Application", LOG_LEVEL_INFO);
  LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);
  *Node*  sensorContainer *node*  sensors;
  *node*  sensors.Create (6);
   *Node*  sensorContainer n0n1 = *Node*  sensorContainer(*node*  sensors.Get(0), *node*
sensors.Get(1));
  *Node*  sensorContainer n1n5 = *Node*  sensorContainer(*node*  sensors.Get(1), *node*
sensors.Get(5));

  *Node*  sensorContainer n0n2 = *Node*  sensorContainer(*node*  sensors.Get(0), *node*
sensors.Get(2));
```

```
  Node  sensorContainer n2n3 = Node  sensorContainer(node  sensors.Get(2), node
sensors.Get(3));
  Node  sensorContainer n0n4 = Node  sensorContainer(node  sensors.Get(0), node
sensors.Get(4));

  InternetStackHelper stack;
  stack.Install (node  sensors);

  PointToPointHelper pointToPoint;
  pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
  pointToPoint.SetChannelAttribute ("Delay", StringValue ("1.4ms"));

  NetDeviceContainer d0d1 = pointToPoint.Install(n0n1);
  NetDeviceContainer d1d5 = pointToPoint.Install(n1n5);
  NetDeviceContainer d0d2 = pointToPoint.Install(n0n2);
  NetDeviceContainer d2d3 = pointToPoint.Install(n2n3);
  NetDeviceContainer d0d4 = pointToPoint.Install(n0n4);

  Ipv4AddressHelper address;
  address.SetBase ("10.1.1.0", "255.255.255.0");
  Ipv4InterfaceContainer i0i1 = address.Assign(d0d1);
  address.SetBase("10.1.2.0", "255.255.255.0");
  Ipv4InterfaceContainer i1i5 = address.Assign(d1d5);
  address.SetBase("10.1.3.0", "255.255.255.0");
  Ipv4InterfaceContainer i0i2 = address.Assign(d0d2);
  address.SetBase("10.1.4.0", "255.255.255.0");
  Ipv4InterfaceContainer i2i3 = address.Assign(d2d3);
  address.SetBase("10.1.5.0", "255.255.255.0");
  Ipv4InterfaceContainer i0i4 = address.Assign(d0d4);

  Ipv4Global*Routing*Helper::Populate*Routing*Tables();
  uint16_t port = 9;
```

```
 //koneksi1
OnOffHelper onoff("ns3::UdpSocketFactory",
                Address(InetSocketAddress(i0i1.GetAddress(0), port)));
onoff.SetConstantRate(DataRate("448kb/s"),329*2);
ApplicationContainer apps = onoff.Install(node  sensors.Get(1));
apps.Start(Seconds(1.0));
apps.Stop(Seconds(2.0));

PaketSinkHelper sink("ns3::UdpSocketFactory",
                  Address(InetSocketAddress(Ipv4Address::GetAny(), port)));
apps = sink.Install(node  sensors.Get(0));
apps.Start(Seconds(1.0));
apps.Stop(Seconds(2.0));

//koneksi2
OnOffHelper onoff2("ns3::UdpSocketFactory",
                Address(InetSocketAddress(i1i5.GetAddress(0), port)));
onoff2.SetConstantRate(DataRate("448kb/s"),329*2);
ApplicationContainer apps2 = onoff2.Install(node  sensors.Get(5));
apps2.Start(Seconds(1.0));
apps2.Stop(Seconds(2.0));

PaketSinkHelper sink2("ns3::UdpSocketFactory",
                  Address(InetSocketAddress(Ipv4Address::GetAny(), port)));
apps2 = sink2.Install(node  sensors.Get(1));
apps2.Start(Seconds(1.0));
apps2.Stop(Seconds(2.0));

//koneksi3
OnOffHelper onoff3("ns3::UdpSocketFactory",
                Address(InetSocketAddress(i0i2.GetAddress(0), port)));
onoff3.SetConstantRate(DataRate("448kb/s"),329*2);
```

```
ApplicationContainer apps3 = onoff3.Install(node  sensors.Get(2));
apps3.Start(Seconds(2.0));
apps3.Stop(Seconds(3.0));


PaketSinkHelper sink3("ns3::UdpSocketFactory",
                Address(InetSocketAddress(Ipv4Address::GetAny(), port)));
apps3 = sink3.Install(node  sensors.Get(0));
apps3.Start(Seconds(2.0));
apps3.Stop(Seconds(3.0));

  //koneksi4
OnOffHelper onoff4("ns3::UdpSocketFactory",
              Address(InetSocketAddress(i2i3.GetAddress(0), port)));
onoff4.SetConstantRate(DataRate("448kb/s"),329*2);
ApplicationContainer apps4 = onoff4.Install(node  sensors.Get(3));
apps4.Start(Seconds(2.0));
apps4.Stop(Seconds(3.0));

PaketSinkHelper sink4("ns3::UdpSocketFactory",
                Address(InetSocketAddress(Ipv4Address::GetAny(), port)));
apps4 = sink4.Install(node  sensors.Get(2));
apps4.Start(Seconds(2.0));
apps4.Stop(Seconds(3.0));

//koneksi5

OnOffHelper onoff5("ns3::UdpSocketFactory",
              Address(InetSocketAddress(i0i4.GetAddress(0), port)));
onoff5.SetConstantRate(DataRate("448kb/s"), 329);
ApplicationContainer apps5 = onoff5.Install(node  sensors.Get(4));
apps5.Start(Seconds(3.0));
```

```
apps5.Stop(Seconds(4.0));

PaketSinkHelper sink5("ns3::UdpSocketFactory",
                Address(InetSocketAddress(Ipv4Address::GetAny(), port)));
apps5 = sink5.Install(node  sensors.Get(0));
apps5.Start(Seconds(3.0));
apps5.Stop(Seconds(4.0));

AnimationInterface anim ("animation.xml");
anim.SetConstantPosition(node  sensors.Get(0), 7.4, 7.3);
anim.SetConstantPosition(node  sensors.Get(1), 5.57, 6.86);
anim.SetConstantPosition(node  sensors.Get(2), 12.2, 3.73);
anim.SetConstantPosition(node  sensors.Get(3), 13.36, 10.89);
anim.SetConstantPosition(node  sensors.Get(4), 1.35, 12.77);
anim.SetConstantPosition(node  sensors.Get(5), 1.62, 1.39);

anim.UpdateNode  sensorDescription(node  sensors.Get(0), "Base Station");
anim.UpdateNode  sensorDescription(node  sensors.Get(1), "N3");
anim.UpdateNode  sensorDescription(node  sensors.Get(2), "N2");
anim.UpdateNode  sensorDescription(node  sensors.Get(3), "N5");
anim.UpdateNode  sensorDescription(node  sensors.Get(4), "N4");
anim.UpdateNode  sensorDescription(node  sensors.Get(5), "N5");

anim.UpdateNode  sensorColor(node  sensors.Get(0),255,255,0);




//Simulator::Stop(Seconds(11));
Simulator::Run ();
Simulator::Destroy ();
```

```
//Perhitungan Energi
float arusPerangkat = 110; //mAh
float arusTerima = 12.3;
int jumlah*Node*  sensor = 5;
float arus[] = {(float)0.044,
 (float)0.044,
 (float)0.044,
 (float)0.044,
 (float)0.044};

int habisMenit[] = {
        (int)(battery/arus[0]*60.0),
        (int)(battery/arus[1]*60.0),
        (int)(battery/arus[2]*60.0),
        (int)(battery/arus[3]*60.0),
        (int)(battery/arus[4]*60.0),
};
int menitMax = 0;
for(int i=0;i<5;i++){
        if(menitMax<habisMenit[i])
                menitMax = habisMenit[i];
}
for(int i=0;i<menitMax+100;i++){
        int jumlahHidup = 0;
        int jumlahMati = 0;
        if(i<habisMenit[0])jumlahHidup++;
        else jumlahMati++;
        if(i<habisMenit[1])jumlahHidup++;
        else jumlahMati++;
        if(i<habisMenit[2])jumlahHidup++;
        else jumlahMati++;
```

```
        if(i<habisMenit[3])jumlahHidup++;
        else jumlahMati++;
        if(i<habisMenit[4])jumlahHidup++;
        else jumlahMati++;
        if(i%30==0)
        std::cout<<jumlahHidup<<","<<jumlahMati<<std::endl;
  }

  return 0;
}
```

RIWAYAT PENULIS

Windi Puspitasari lahir di Malang pada tanggal 05 Desember 1995. Penulis mengenyam pendidikan SD hingga SMK di Malang yaitu SD Negeri Tunjungtirto I, SMP Negeri 1 Karangploso dan SMK Negeri 8 Malang, hingga akhirnya memutuskan untuk melanjutkan pendidikan tinggi di Politeknik Negeri Malang jurusan Jaringan Telekomunikasi Digital. Setelah lulus dari jenjang diploma, maka penulis berkeinginan untuk menuntut ilmu lebih lanjut di Institut Teknologi Sepuluh Nopember jurusan Telekomunikasi Multimedia. Selama aktif berkuliah di ITS, penulis mengikuti kepanitian konferensi internasional yang diselenggarakan oleh ITS yaitu ICAMIMIA 2019. Semua kegiatan yang dilakukan di ITS turut berkonstribusi dalam mengembangkan kemampuan penulis.