

**TUGAS AKHIR – IF184802**

**RANCANG BANGUN CLUSTER-BASED PROTOKOL UNTUK PENGIRIMAN DATA SECARA ADAPTIF DENGAN PENGATURAN KEKUATAN TRANSMISI DAN MONITORING KETERSEDIAN ENERGI PADA LINGKUNGAN WIRELESS SENSOR NETWORK DENGAN nRF24L01**

ZAHRI RUSLI

NRP 05111540000108

Dosen Pembimbing I

Waskhito Wibisono, S.Kom., M.Eng., Ph.D.

Dosen Pembimbing II

Tohari Ahmad, S.Kom., MIT., Ph.D.

DEPARTEMEN INFORMATIKA

Fakultas Teknologi Informasi dan Komunikasi

Institut Teknologi Sepuluh Nopember

Surabaya 2019



**TUGAS AKHIR – IF184802**

**RANCANG BANGUN CLUSTER-BASED PROTOKOL UNTUK PENGIRIMAN DATA SECARA ADAPTIF DENGAN PENGATURAN KEKUATAN TRANSMISI DAN MONITORING KETERSEDIAN ENERGI PADA LINGKUNGAN WIRELESS SENSOR NETWORK DENGAN nRF24L01**

**ZAHRI RUSLI**

**NRP 05111540000108**

**Dosen Pembimbing I**

**Waskhito Wibisono, S.Kom., M.Eng., Ph.D.**

**Dosen Pembimbing II**

**Tohari Ahmad, S.Kom., MIT., Ph.D.**

**DEPARTEMEN INFORMATIKA**

**Fakultas Teknologi Informasi dan Komunikasi**

**Institut Teknologi Sepuluh Nopember**

**Surabaya 2019**

***(Halaman ini sengaja dikosongkan)***

******

**UNDERGRADUATE THESIS – IF184802**

**ADAPTIVE ZONING IMPLEMENTATION IN ZONE ROUTING PROTOCOL BASED ON NEIGHBOR NODE DENSITY LEVEL**

**ZAHRI RUSLI**

**NRP 05111540000108**

First Advisor

**Waskhito Wibisono, S.Kom., M.Eng., Ph.D.**

**Second Advisor**

**Tohari Ahmad, S.Kom., MIT., Ph.D.**

**INFORMATICS DEPARTMENT**

**Faculty of Information Communication and Technology**

**Institut Teknologi Sepuluh Nopember**

**Surabaya 2019**

***(Halaman ini sengaja dikosongkan)***

**LEMBAR PENGESAHAN**

**RANCANG BANGUN *CLUSTER-BASED* PROTOKOL UNTUK PENGIRIMAN DATA SECARA ADAPTIF DENGAN PENGATURAN KEKUATAN TRANSMISI DAN *MONITORING* KETERSEDIAN ENERGI PADA LINGKUNGAN WIRELESS SENSOR NETWORK DENGAN nRF24L01**

**TUGAS AKHIR**

Diajukan untuk Memenuhi Salah Satu Syarat

Memperoleh Gelar Sarjana Komputer

pada

Bidang Studi Arsitektur Jaringan Komputer

Program Studi S-1 Departemen Informatika

Fakultas Teknologi Informasi dan Komunikasi

Institut Teknologi Sepuluh Nopember

Oleh:

**ZAHRI RUSLI**

**NRP: 05111540000108**

Disetujui oleh Pembimbing tugas akhir:

1. Waskitho Wibisono, S.Kom.,M.Eng.,Ph.D. ..........................

(NIP. 197410222000031001) (Pembimbing 1)

1. Dr.Eng. Radityo Anggoro, S.Kom., M.Sc. ..........................

(NIP. 198410162008121002) (Pembimbing 2)

**SURABAYA**

**JUNI, 2019**

***(Halaman ini sengaja dikosongkan)***

**RANCANG BANGUN *CLUSTER-BASED* PROTOKOL UNTUK PENGIRIMAN DATA SECARA ADAPTIF DENGAN PENGATURAN KEKUATAN TRANSMISI DAN *MONITORING* KETERSEDIAN ENERGI PADA LINGKUNGAN WIRELESS SENSOR NETWORK DENGAN nRF24L01**

|  |  |  |
| --- | --- | --- |
| **Nama Mahasiswa** | **:** | **Zahri Rusli** |
| **NRP** | **:** | **05111540000108** |
| **Departemen** | **:** | **Informatika FTIK ITS** |
| **Dosen Pembimbing 1** | **:** | **Waskitho Wibisono, S.Kom., M.Eng., Ph.D.** |
| **Dosen Pembimbing 2** | **:** | **Tohari Ahmad S.Kom., MIT., Ph.D.** |

# Abstrak

*Mobile Ad Hoc Network* (MANET) adalah jaringan nirkabel ad hoc yang melakukan konfigurasi mandiri secara kontinyu dan tidak memiliki infrastruktur yang tetap. Salah satu algoritma *routing* pada MANET adalah *Zone Routing Protocol* (ZRP). ZRP merupakan algoritma *routing* hibrid yang menggabungkan kelebihan dari *routing protocol* proaktif dan reaktif, yaitu mengurangi *control overhead* dari *routing protocol* proaktif dan mengurangi *latency* dari *routing protocol* reaktif. Algoritma ZRP membagi *node* ke dalam zona-zona menggunakan radius yang ditentukan secara manual. Penentuan radius secara manual terbilang kurang fleksibel dan optimal mengingat MANET merupakan jaringan ad hoc yang topologinya berubah-ubah dalam periode tertentu.

Modifikasi akan dilakukan pada konsep penentuan zona, yang semula bersifat statis diubah menjadi adaptif berdasarkan tingkat kepadatan *node* tetangga pada zona di dalam radius. Hal ini dilakukan dengan cara menghitung jumlah kepadatan *node* tetangga dari setiap *node*. Jika kepadatan *node* tetangga kurang dari *threshold* yang ditentukan, maka radius akan bertambah satu. Sebaliknya, jika kepadatan *node* tetangga melebihi *threshold* yang ditentukan, maka radius akan berkurang satu.

Hasil dari implementasi menggunakan skenario simulasi NS-2berupa peningkatan rata-rata *Packet Delivery Ratio* sebesar 2,21 %, peningkatan rata-rata *Routing Overhead* sebesar 18,66 %, dan peningkatan rata-rata *Delivery Delay* sebesar 323 %.

Dari hasil di atas, dapat diketahui bahwa modifikasi zonasi adaptif pada ZRP dapat diimplementasikan dan memengaruhi kinerja algoritma *routing* ZRP pada lingkungan MANET.

**Kata kunci :****Kepadatan *Node* Tetangga, MANET*,* NS-2*,* ZRP.**

**ADAPTIVE ZONING IMPLEMENTATION IN ZONE ROUTING PROTOCOL BASED ON NEIGHBOR NODE DENSITY LEVEL**

**Student’s Name : Zahri Rusli**

**Student’s ID : 05111540000108**

**Department : Informatika FTIK-ITS**

**First Advisor : Waskitho Wibisono, S.Kom., M.Eng., Ph.D.**

**Second Advisor : Tohari Ahmad, S.Kom., MIT., Ph.D.**

# Abstract

*Mobile Ad Hoc Network (MANET) is an ad hoc wireless network that performs continuous self-configuration and does not have a fixed infrastructure. One of the routing algorithms in MANET is Zone Routing Protocol (ZRP). ZRP is a hybrid routing algorithm that combines the advantages of proactive and reactive routing protocols, which reduces the control overhead of proactive routing protocols and reduces the latency of reactive routing protocols [4]. The ZRP algorithm divides nodes into zones using a manually determined radius. Manually determining radius is less flexible and optimal considering MANET is an Ad Hoc network whose topology changes over a period of time.*

*Modifications will be made to the concept of zone determination, which was originally statically converted to adaptive based on the density of neighboring nodes in the zone within the radius. This is done by calculating the density of neighboring nodes from each node. If the density of neighboring nodes is less than the specified threshold, then the radius will increase by one. Conversely, if the density of neighboring nodes exceeds the specified threshold, then the radius will decrease by one.*

*The results of the implementation using the NS-2 simulation scenario in the form of an increase in the average Packet Delivery Ratio of 2,21%, an increase in the Routing Overhead average of 18,66%, and an increase in the average Delivery Delay of 323%.*

*From the results above, it can be seen that modification of adaptive zoning in ZRP can be implemented and affect the performance of the ZRP routing algorithm in the MANET environment.*

***Keyword: Density of Neighbor Node, MANET, NS-2, ZRP.***

# KATA PENGANTAR



Puji syukur kepada Allah Yang Maha Esa atas segala karunia dan rahmat-Nya sehingga penulis dapat menyelesaikan tugas akhir yang berjudul

***“*RANCANG BANGUN *CLUSTER-BASED* PROTOKOL UNTUK PENGIRIMAN DATA SECARA ADAPTIF DENGAN PENGATURAN KEKUATAN TRANSMISI DAN *MONITORING* KETERSEDIAN ENERGI PADA LINGKUNGAN WIRELESS SENSOR NETWORK DENGAN nRF24L01”.**

Harapan dari penulis, semoga apa yang tertulis di dalam buku tugas akhir ini dapat bermanfaat bagi pengembangan ilmu pengetahuan saat ini dan ke depannya, serta dapat memberikan kontribusi yang nyata.

Dalam pelaksanaan dan pembuatan tugas akhir ini tentunya sangat banyak bantuan yang penulis terima dari berbagai pihak, tanpa mengurangi rasa hormat penulis ingin mengucapkan terima kasih sebesar-besarnya kepada:

1. Allah SWT. dan Nabi Muhammad SAW. yang telah membimbing penulis selama hidup.
2. Keluarga penulis (Ayah, Ibu, Mbak Fina, Ifa, Gifar, Yusuf, dan keluarga penulis yang lain) yang selalu memberikan dukungan baik berupa doa, moral, dan material yang tak terhingga kepada penulis, sehingga penulis dapat menyelesaikan tugas akhir ini.
3. Bapak Ir. F.X. Arunanto, M.Sc. dan Bapak Dr.Eng. Radityo Anggoro, S.Kom., M.Sc. selaku Dosen Pembimbing penulis yang telah membimbing, memberikan nasihat, dan memotivasi penulis sehingga penulis dapat menyelesaikan tugas akhir ini.
4. Bapak Dr.Eng. Darlis Herumurti, S.Kom., M.Kom. selaku kepala Departemen Informatika ITS.
5. Bapak dan Ibu Dosen yang telah memberikan ilmunya selama penulis berkuliah di Informatika ITS.
6. Teman-teman penulis DaPur Emak Pede (Nahda Fauziyah Zahrah, Purina Qurota Ayunin, dan Anisah Putri Diana) yang selalu memberikan semangat secara tidak langsung kepada penulis, selalu memberikan hiburan, selalu menemani hari-hari penulis saat senang maupun susah, dan juga menjadi keluarga baru penulis saat berkuliah di Departemen Informatika ITS.
7. Teman-teman KP PLN Hero, Purina, Huda, dan Eritha yang telah menemani kerja praktik penulis dan mau mem-*back up* penulis dikala sakit.
8. Teman-teman penulis Ronald, Rezky, Ufa dan lain-lain yang telah mewarnai kehidupan penulis.
9. Teman-teman dari keluarga besar Laboratorium NCC (Hero, Sisil, Yoga, Dely, Zulfa, Ubut, Zayn, Ical, Azki, Nuza, Akmal, Siraj, Adin, Wasil) yang telah menemani, memberi semangat, doa, serta hiburan dikala penulis sedang jenuh saat pengerjaan tugas akhir ini.
10. Teman-teman dari Kabinet Semangat Berpadu BEM FTIK periode 2017/2018 yang telah mewarnai hari-hari penulis dan mengajarkan makna ikhlas dan sabar, yang sudah berjuang bersama penulis selama kurang lebih satu setengah tahun, yang terkadang membuat penulis kesal namun tidak apa-apa.
11. Teman-teman EA Bahagia BEM FTIf Presisi Bermanfaat yang telah menjadi teman organisasi penulis, menambah rumah baru bagi penulis.
12. Teman-teman Hublu Inspirasi HMTC Inspirasi yang telah menjadi teman berhimpun penulis.
13. Teman-teman Pemandu Phoenix yang telah menjadi rekan mandu penulis selama berkiprah di kepemanduan fakultas.
14. Teman-teman Saman TC yang telah memberikan hiburan, mengajarkan penulis menari saman, dan menjadikan penulis lebih bijak dalam menghadapi adik tingkat.
15. Teman-teman SSL yang telah menjadikan Minggu pagi penulis lebih bermanfaat.
16. Teman-teman pejuang SW 119 yang selalu memberikan informasi penting dan semangat kepada penulis untuk menyelesaikan tugas akhir.
17. Teman-teman angkatan 2015 (Masamalas) yang sudah menjadi saksi hidup perjalanan karir penulis selama berkuliah di Informatika ITS.
18. Untuk orang-orang yang tidak dapat disebutkan satu persatu oleh penulis dan pembaca buku tugas akhir ini.

Penulis telah berusaha sebaik-baiknya dalam menyusun tugas akhir ini. Namun, penulis memohon maaf apabila terdapat kekurangan, kesalahan maupun kelalaian yang telah penulis lakukan. Kritik dan saran yang membangun dapat disampaikan sebagai bahan perbaikan selanjutnya. Tetap semangat dalam menjalani kehidupan, jangan menyerah, karena Allah masih ingin melihat kita berjuang. Semoga kita semua selalu diberi kebahagiaan lahir dan batin dan kesuksesan dunia akhirat. Aamiin.

Surabaya, 14 Januari 2019

Hania Maghfira

***(Halaman ini sengaja dikosongkan)***

# DAFTAR ISI

Abstrak vii

Abstract ix

KATA PENGANTAR xi

DAFTAR ISI xv

DAFTAR GAMBAR xix

DAFTAR TABEL xxi

KODE SUMBER xxiii

BAB I PENDAHULUAN 1

1.1 Latar Belakang 1

1.2 Rumusan Masalah 2

1.3 Batasan Permasalahan 2

1.4 Tujuan 2

1.5 Manfaat 2

1.6 Metodologi 3

1.6.1 Penyusunan Proposal Tugas Akhir 3

1.6.2 Studi Literatur 3

1.6.3 Implementasi Sistem 3

1.6.4 Pengujian dan Evaluasi 3

1.6.5 Penyusunan Buku 4

1.7 Sistematika Penulisan Laporan 4

BAB II TINJAUAN PUSTAKA 7

2.1 MANET 7

2.2 *Zone Routing Protocol* (ZRP) 8

2.3 *Network Simulator 2* (NS-2) 10

2.3.1 Instalasi 11

2.3.2 *Trace File* 14

2.4 AWK 15

BAB III PERANCANGAN 17

3.1 Deskripsi Umum 17

3.2 Daftar Istilah 18

3.3 Perancangan Skenario *Threshold* Kepadatan *Node* Tetangga 20

3.4 Analisis dan Perancangan Modifikasi *Routing Protocol* ZRP 20

3.4.1 Perancangan Penghitungan Kepadatan *Node* Tetangga untuk Setiap Radius 20

3.4.2 Perancangan Pemilihan Radius 21

3.5 Perancangan Simulasi pada NS-2 21

3.6 Perancangan Metrik Analisis 22

3.6.1 *Packet Delivery Ratio* (PDR) 22

3.6.2 *Routing Overhead* (RO) 23

3.6.3 *Delivery Delay* (DD) 23

BAB IV IMPLEMENTASI 25

4.1 Lingkungan Pembangunan Sistem 25

4.2 Instalasi *Routing Protocol* ZRP pada NS-2 25

4.3 Implementasi Modifikasi *Routing Protocol* ZRP 26

4.3.1 Implementasi Penghitungan Kepadatan *Node* Tetangga untuk Setiap Radius 27

4.3.2 Implementasi Pemilihan Radius 31

4.4 Implementasi Simulasi pada NS-2 34

4.5 Implementasi Metrik Analisis 43

4.5.1 Implementasi *Packet Delivery Ratio* 43

4.5.2 Implementasi *Routing Overhead* 44

4.5.3 Implementasi *Delivery Delay* 44

BAB V UJI COBA DAN EVALUASI 47

5.1 Lingkungan Uji Coba 47

5.2 Hasil Uji Coba 47

5.2.1 Hasil Pra-Uji Coba Penentuan *Threshold* 48

5.2.2 Hasil Uji Coba Simulasi NS-2 49

BAB VI KESIMPULAN DAN SARAN 55

6.1 Kesimpulan 55

6.2 Saran 55

DAFTAR PUSTAKA 57

LAMPIRAN 59

1. Kode Konfigurasi Kelas *NDPAgent* 59

2. Kode Konfigurasi Kelas *IARPAgent* 60

3. Kode Konfigurasi *constant.h* 61

4. Kode Fungsi NDPAckTimer::handle(Event\* e) 63

5. Kode Fungsi IARPPeriodicUpdateTimer::handle (Event\* e) 65

6. Kode Konfigurasi Posisi *Node* pada Simulasi NS-2 68

7. Kode Skrip AWK *Packet Delivery Ratio* 70

8. Kode Skrip AWK *Routing Overhead* 71

9. Kode Skrip AWK *Delivery Delay* 72

BIODATA PENULIS 73

***(Halaman ini sengaja dikosongkan)***

# DAFTAR GAMBAR

[**Gambar 2.1** Contoh Zona *Routing* Node S dengan Radius 2[4] 9](file:///D:\Kuliah\IF\TA-Buku\BUKU\Buku%20TA_ver%204.docx#_Toc535423290)

[**Gambar 2.2** Arsitektur ZRP[4] 10](#_Toc535423291)

[**Gambar 3.1** Diagram Rancangan Simulasi ZRP Modifikasi 18](#_Toc535423292)

[**Gambar 3.2** *Pseudocode* Penghitungan Kepadatan *Node* Tetangga 21](#_Toc535423293)

[**Gambar 3.3** *Pseudocode* Penghitungan Radius 21](#_Toc535423294)

[**Gambar 5.1** Grafik Rata-rata *Packet Delivery Ratio* pada Simulasi NS-2 50](#_Toc535423295)

[**Gambar 5.2** Grafik Rata-rata *Routing Overhead* pada Simulasi NS-2 52](#_Toc535423296)

[**Gambar 5.3** Grafik Rata-rata *Delivery Delay* pada Simulasi NS2 53](#_Toc535423297)

***(Halaman ini sengaja dikosongkan)***

# DAFTAR TABEL

[**Tabel 2.1** Detail Penjelasan *Trace File* ZRP 14](#_Toc535423275)

[**Tabel 3.1** Daftar Istilah 19](#_Toc535423276)

[**Tabel 3.2** Parameter Lingkungan Simulasi 22](#_Toc535423277)

[**Tabel 4.1** Tabel Lingkungan Pembangunan Sistem 25](#_Toc535423278)

[**Tabel 5.1** Spesifikasi Perangkat yang Digunakan 47](#_Toc535423279)

[**Tabel 5.2** Hasil Pra-Uji Coba Penentuan *Threshold* 49](#_Toc535423280)

***(Halaman ini sengaja dikosongkan)***

# KODE SUMBER

[**Kode Sumber 4.1** Modifikasi Fungsi print\_tables dalam Kelas NDPAgent 27](#_Toc535423252)

[**Kode Sumber 4.2** Menghitung *Node* di dalam IARP 28](#_Toc535423253)

[**Kode Sumber 4.3** Inisialisasi Fungsi *print\_tables* dalam Kelas *NDPAgent* 29](#_Toc535423254)

[**Kode Sumber 4.4** Inisialisasi Fungsi *count\_node\_in\_rad* 29](#_Toc535423255)

[**Kode Sumber 4.5** Modifikasi Dokumen *constant.h* 29](#_Toc535423256)

[**Kode Sumber 4.6** Memanggil Fungsi print\_tables dalam Kelas NDPAckTimer 30](#_Toc535423257)

[**Kode Sumber 4.7** Memanggil Fungsi *count\_node\_in\_rad* dalam Kelas *IARPPeriodicUpdateTimer* 31](#_Toc535423258)

[**Kode Sumber 4.8** Pemilihan Radius dalam Kelas NDPAckTimer 32](#_Toc535423259)

[**Kode Sumber 4.9** Pemilihan Radius dalam Kelas IARPPeriodicTimer 33](#_Toc535423260)

[**Kode Sumber 4.10** Pengaturan Parameter Lingkungan Simulasi 34](file:///D:\Kuliah\IF\TA-Buku\BUKU\Buku%20TA_ver%204.docx#_Toc535423261)

[**Kode Sumber 4.11** Pengaturan Inisialisasi NS-2 35](#_Toc535423262)

[**Kode Sumber 4.12** Pengaturan Konfigurasi Parameter Node 36](#_Toc535423263)

[**Kode Sumber 4.13** Posisi Statis Node Sumber dan Node Tujuan 37](#_Toc535423264)

[**Kode Sumber 4.14** Posisi Acak Node Intermediate 38](#_Toc535423265)

[**Kode Sumber 4.15** Pemberian Label dan Warna pada Node 39](#_Toc535423266)

[**Kode Sumber 4.16** Konfigurasi Pergerakan Node 40](#_Toc535423267)

[**Kode Sumber 4.17** Konfigurasi Koneksi Simulasi 41](#_Toc535423268)

[**Kode Sumber 4.18** Konfigurasi Menyimpan Hasil Simulasi 41](#_Toc535423269)

[**Kode Sumber 4.19** Konfigurasi Mengakhiri Simulasi 42](#_Toc535423270)

[**Kode Sumber 4.20** Perintah Menjalankan Simulasi TCL 42](file:///D:\Kuliah\IF\TA-Buku\BUKU\Buku%20TA_ver%204.docx#_Toc535423271)

[**Kode Sumber 4.21** Pseudocode Menghitung PDR 43](#_Toc535423272)

[**Kode Sumber 4.22** Pseudocode Menghitung RO 44](#_Toc535423273)

[**Kode Sumber 4.23** Pseudocode Menghitung Delivery Delay 45](#_Toc535423274)

***(Halaman ini sengaja dikosongkan)***

# BAB I PENDAHULUAN

## Latar Belakang

Perkembangan teknologi komunikasi saat ini teleh berkembang dengan sangat pesat, salah satunya dibidang *Wireless Sensor Network* (WSN). WSN adalah sebuah jaringan yang terdiri dari node-node yang didistribusikan baik secara strategi maupun secara acak yang bertujuan untuk mengamati suatu kejadian tertentu. WSN sudah dianggap sebagai salah satu penemuan yang sangat penting karena harganya murah dan handal.

Modul Wireless nRF24L01 adalah sebuah modul komunikasi *short range* yang memanfaatkan pita gelombang RF 2.4GHz ISM (Industrial, Scientific and Medical) sebagai media komunikasinya. Modul ini menggunakan anatarmuka SPI (Serial Peripheral Interface)untuk berkomunikasi dengan mikrokontroler sehingga modul dapat digunakan dengan mikrokontroler manapun yang mendukung SPI. Modul nRF24L01 menggunakan daya sangat rendah sehingga cocok digunakan pada node-node dalam jaringan sensor yang biasanya ditenagai daya baterai yang terbatas.

Pada jaringan sensor yang akan dibuat, diharapkan semua node dapat mengirimkan data ke sebuah server atau *coordinator*. Namun, apabila semua node mengirmkan langsung ke server atau *coordinator*, maka penggunaan energi dalam jaringan akan menjadi boros dikarenakan semua node akan menggunakan kekuatan transmisi tinggi untuk mengirim data ke server atau *coordinator.* Sehingga, perlu adanya node yang menjadi *Cluster Head* dalam jaringan untuk mengumpulkan data dari node lain sebelum mengirimkannya ke server atau *coordinator*, sehingga cukup satu node yang menggunakan kekuatan transmisi tinggi. Namun, node yang menjadi cluster head akan cepat kehabisan energi yang mengakibatkan energi dalam jaringan menjadi tidak seimbang. Maka diperlukan sebuah metoda pemilihan *cluster head* yang efisian untuk menangani permasalahan tersebut.

## Rumusan Masalah

Rumusan masalah yang diangkat dalam tugas akhir ini dapat dipaparkan sebagai berikut:

1. Bagaimana mengatur kekuatan transmisi *cluster* member dan *cluster head* secara *adaptive*  berdasarkan ketersedian energi pada node ?
2. Bagaimana melakukan monitoring terhadap perubahan energi pada masing-masing *node* ?
3. Bagaimana menantukan time slot untuk penjadwalan transmisi tiap node dalam satu *cluster* ?

## Batasan Permasalahan

Berdasarkan masalah yang diuraikan oleh penulis, maka batasan masalah pada tugas akhir ini adalah:

* 1. Menggunakan *platform*  Arduino sebagai mikrokontroller
  2. Menggunakan modul nRF24L01 sebagai media transmisi data antar  *node* pada jaringan sensor
  3. Menggunakan *Sensor INA219* sebagai sensor untuk menguku jumlah energi yang terdapat pada masing-masing node.
  4. Menggunakan baterai sebagai sumber energi pada setiap *node*

## Tujuan

Tujuan dari pembuatan tugas akhir ini adalah sebagai berikut:

1. Membangun sebuah jaringan sensor nirkabel yang dapat berkoordinasi secara *adaptive* dan *dinamis* dalam pemilohan *cluster head* yang daoat menyeimbangkan penggunaan energi dalam jaringan tersebut.
2. Melakukan monitoring terhadap perubahan energi pada masing-masing *node.*

## Manfaat

Manfaat yang diperoleh dari pengerjaan tugas akhir ini adalah menghasilkan sebuah jaringan sensor nirkabel yang dapat berkoordinasi secara *adaptive* dan *dinamis* sert memiliki penggunaan energi yang seimbang dan tahan lama.

## Metodologi

Pembuatan tugas akhir ini dilakukan dengan menggunakan metodologi sebagai berikut:

### Penyusunan Proposal Tugas Akhir

Proposal tugas akhir ini berisi gambaran tentang tugas akhir yang akan dibuat. Pendahuluan proposal tugas akhir meliputi hal yang menjadi latar belakang diajukannya usulan tugas akhir, rumusan masalah yang diangkat, batasan masalah yang menjadi konstrain dari tugas akhir, tujuan pembuatan tugas akhir, dan manfaat dari hasil tugas akhir. Di dalam proposal tugas akhir juga dijabarkan mengenai tinjauan pustaka yang menjadi referensi pendukung dalam pembuatan tugas akhir ini. Terdapat pula sub bab jadwal kegiatan yang menjelaskan jadwal pengerjaan tugas akhir.

### Studi Literatur

Studi literatur yang dilakukan dalam pengerjaan Tugas Akhir ini adalah mengenai Rancang Bangun *Cluster-Based* Protokol untuk pengiriman data secara adaptif dengan pengaturan kekuatan transmisi dan *monitoring* ketersedian energy pada lingkungan *Wireless Sensor Network* dengan Menggunakan nRF2L01. Sehingga, studi literature ini dapat diterapkan pada perancangan jaringan sensor nirkabel yang dapat melakukan pemilihan cluster head secara *adaptive* dan *dinamis*.

### Implementasi Sistem

Implementasi merupakan tahap untuk mengimplementasikan metode-metode yang sudah diajukan pada proposal tugas akhir. Untuk membangun algoritma yang telah dirancang sebelumnya, implementasi dilakukan dengan menggunakan NS-2 sebagai simulator jaringan, bahasa C/C++ sebagai bahasa pemrograman untuk uji coba mengimplementasikan metode yang sudah diajukan.

### Pengujian dan Evaluasi

Pengujian dan evaluasi dari hasil Tugas Akhir ini akan diujicobakan dengan membangun jaringan sensor nirkabel menggunakan modul wireless nRF2L01 yang berlokasi di Departement Informatika Institut Teknologi Sepuluh Nopember Surabaya.

### Penyusunan Buku

Pada tahap ini dilakukan penyusunan buku sebagai dokumentasi dari pelaksanaan tugas akhir yang mencakup seluruh konsep, teori, implementasi, serta hasil yang telah dikerjakan.

## Sistematika Penulisan Laporan

Sistematika penulisan laporan tugas akhir adalah sebagai berikut:

* + 1. Bab I. Pendahuluan

Bab ini berisi penjelasan mengenai latar belakang, rumusan masalah, batasan permasalahan, tujuan, manfaat, metodologi, dan sistematika penulisan dari pembuatan tugas akhir.

* + 1. Bab II. Tinjauan Pustaka

Bab ini berisi kajian teori atau penjelasan dari metode, algoritma, *library*,dan *tools* yang digunakan dalam penyusunan tugas akhir ini. Kajian teori yang dimaksud berisi tentang penjelasan singkat mengenai *Mobile Ad Hoc Network* (MANET), *Zone Routing Protocol* (ZRP), *Network Simulator* 2 (NS-2), dan AWK.

* + 1. Bab III. Perancangan Perangkat Lunak dan Perangkat Keras

Bab ini berisi pembahasan mengenai desain dari jaringan sensor nirkable yang akan dibuat, meliputi arsitektur dann proses perangkat lunak dan perangkat keras.

* + 1. Bab IV. Implementasi

Bab ini menjelaskan implementasi dari desain dari jaringan yang akan dilakukan pada tahan desain, meliputi potongan *pseudocode* yang terdapat dalam perangkat ludan dan perangkat keras yang digunakan.

* + 1. Bab V. Pengujian dan Evaluasi

Bab ini berisi hasil uji coba dan evaluasi dari implementasi jaringan sensor nirkabel yang dibuat dengan melihat keluaran yang dihasilkan, analisa dan eveluasi untuk mengatahui kemampuan jaringan dan penggunaan energy pada jaringan sensor nirkabel.

* + 1. Bab VI. Kesimpulan dan Saran

Bab ini merupakan bab yang menyampaikan kesimpulan dari hasil uji coba yang dilakukan, masalah-masalah yang dialami pada proses pengerjaan tugas akhir, dan saran untuk pengembangan tugas akhir ke depannya.

* + 1. Daftar Pustaka

Bab ini berisi daftar pustaka yang dijadikan literatur dalam tugas akhir.

* + 1. Lampiran

Dalam lampiran terdapat kode sumber program secara keseluruhan.

***(Halaman ini sengaja dikosongkan)***

# BAB II TINJAUAN PUSTAKA

Bab ini berisi pembahasan mengenai teori-teori dasar atau penjelasan dari metode dan alat yang digunakan dalam tugas akhir. Penjelasan ini bertujuan untuk memberikan gambaran secara umum terhadap program yang dibuat dan berguna sebagai penunjang dalam pengembangan riset yang berkaitan.

## *Wireless Communication*

*Wireless Communication* atau Komunikasi nirkabel merupakan komunikasi terjadi antara dua devices yang di sebut *transmitter dan receiver* tanpa menggunakan perantara kabel.[1]

*Transmiter* pada komunikasi nirkabel memiliki fungsi sebagai pengirim pesan informasi sedangkan *receiver* berfungsi sebagai penerima pesan informasi. *Transmiter* dan *receiver* pada perangkat *wireless* umumnya sudah terintegrasi menjadi satu perangkat. Sehingga dalam satu perangkat tersebut bisa berperan sebagai *transmitter* dan *receiver.*[1]

Komunikasi nirkabel memiliki beberapa kelebihan yaitu penghematan dari sisi perangkat keras (hardware). Hal ini disebabkan karena komunikasi nirkabel tidak memerlukan kabel ketika berkomunikasi sehingga dapat menghemat biaya komunikasi kabel. Selain itu komunikasi nirkabel juga memiliki skalabilitas yang tinggi. Setiap devices yang ingin berkomunikasi dengan device yang lain tidak memerlukan instalasi atau persiapan hardware yang banyak seperti kabel dan alokasi port.[1]

Walaupun komunikasi nirkabel mempunnyai kelebihan dari sisi hardware dan kemudahan akses, komunikasi nirkabel lemah terhadap intervensi dari gangguan external. Gangguan external bisa seperti radiasi atau juga transmisi dari alat nirkabel lainnya. Selain itu untuk melayani banyak client jaringan nirkabel tidak bisa menerima seluruh transmisi dalam 1 waktu, sehingga dibutuhkan sebuah penjadwalan transmisi agar tidak terjadi tabrakan transmisi.

*Mobile Ad Hoc Network* (MANET) adalah jaringan nirkabel ad hoc yang melakukan konfigurasi mandiri secara kontinyu dan tidak memiliki infrastruktur yang tetap. Setiap komponen pada MANET dapat bergerak secara bebas ke segala arah, hal ini menyebabkan perubahan pada topologi secara acak. Karakteristik MANET yang dinamis membuat MANET banyak dimanfaatkan sebagai alat bantu komunikasi dalam situasi yang tidak memiliki infrastruktur seperti upaya rekonstruksi bencana alam, evakuasi Tim SAR, hingga operasi militer [1].

Tipe jaringan MANET adalah desentralisasi, maknanya setiap *node* memiliki tanggung jawab dalam proses pengiriman dan penerimaan paket data. *Node*-*node* tersebut akan meneruskan paket data ke *node* lain secara dinamis berdasarkan konektivitas jaringan dan algoritma *routing* yang digunakan. Algoritma *routing* pada MANET sendiri diklasifikasikan ke dalam tiga kategori, yaitu *routing protocol* proaktif, reaktif, dan hibrid. *Node*-*node* pada *routing protocol* proaktif bekerja secara kontinyu untuk mencari rute ke *node* tujuan sehingga tabel *routing* akan selalu ter-*update*. Algoritma *routing protocol* proaktif antara lain *Destination-Sequenced Distance-Vector Routing* (DSDV), *Optimized Link State Routing* (OLSR), *Fisheye State Routing* (FSR), dan *Fuzzy Sighted Link State* (FSLS). Pada *routing protocol* reaktif, *node* baru akan bekerja ketika ada permintaan pengiriman data. Algoritma yang termasuk ke dalam *routing protocol* reaktif antara lain *Dynamic Source Routing* (DSR) dan *Ad Hoc On Demand Distance Vector Routing* (AODV). *Routing protocol* hibrid merupakan kombinasi dari *routing protocol* proaktif dan reaktif. Protokol ini menggunakan klister-klasteruntuk menentukan rute dari setiap *node*. *Routing protocol* proaktif digunakan untuk pengiriman paket data antar *node* dalam satu klaster, sedangkan *routing protocol* reaktif digunakan untuk pengiriman paket data antar klaster. Algoritma yang termasuk ke dalam *routing protocol* hibrid salah satunya adalah *Zone Routing Protocol* (ZRP) [2] [3].

## *Wireless Sensor Network*

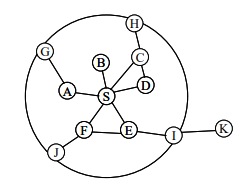
*Wireless sensor network (WSN)* merupakan sekumpulan dari beberapa device kecil yang dilengkapi dengan sensing yang terintegrasi dan kemampuan komunikasi nirkabel, yang diharapkan dapat digunakan secara luas dalam berbagai aplikasi. Sensor ini dioperasikan dengan daya baterai dan energinya tidak selalu diperbaharui karena masalah biaya, lingkungan dan bentuknya.[3]

Energi dalam WSN nodes digunakan pada CPU, sensor, dan radio yang dimana merupakan mengomsumsi energi terbanyak. Untuk mengoptimalkan penggunaan energi, identifikasi source yang paling besar menghabiskan energy dalam komunikasi sangatlah penting, seperti *collision, overhearing, control packet overhead*, dan *idle listening*. Salah satu tantangan untuk mencapai teknologi potensial ini yaitu dengan manajemen konsumsi energi yang efektif dalam device ini untuk memaksimalkan lifespan sebuah node dan akhirnya lifespan jaringan pada saat yang sama juga cukup memelihara kualitas dan kuantitas service. [3]

*Zone Routing* *Protocol* (ZRP) adalah salah satu algoritma *routing* hibrid yang menggunakan parameter berupa *routing* zona. ZRP mulai dikembangkan pada tahun 1997 oleh Haas dan Pearlman. Protokol ini menggabungkan kelebihan dari *routing protocol* proaktif dan reaktif, yaitu mengurangi *control overhead* dari *routing protocol* proaktif dan mengurangi *latency* dari *routing protocol* reaktif.

ZRP membagi jaringannya menjadi dua zona yang berbeda, yaitu zona dalam atau *routing protocol* proaktif lokal yang disebut *IntrA-zone Routing Protocol* (IARP) dan zona luar atau *routing protocol* reaktif global yang disebut *IntEr-zone Routing Protocol* (IERP). Setiap *node* memiliki zona sendiri dan berpotensi berada dalam zona yang tumpang tindih, dan setiap zona berpotensi pula memiliki ukuran yang berbeda-beda. Perbedaan ukuran zona dipengaruhi oleh besarnya radius atau *hop* antar *node*. Penentuan besarnya radius dilakukan secara manual sesuai dengan kebutuhan.

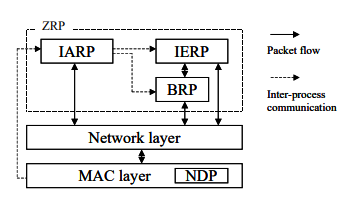
Cara kerja ZRP ditunjukan pada Gambar 2.1, di mana radius yang digunakan adalah dua. *Node* yang termasuk dalam zona *routing* *node* S adalah semua *node* kecuali *node* K, karena posisi *node* K yang berada diluar zona *routing* *node* S. Penentuan zona *routing* tidak ditentukan dari jarak fisik, tetapi dari *hop* (lompatan).

Pada zona *routing* terdapat dua jenis *node*, yaitu *peripheral node* dan *interior node*. *Peripheral node* adalah *node* yang memiliki jarak minimum dari *node* sumber sama dengan radius. Sedangkan *interior* *node* adalah *node* yang memiliki jarak minimum dari *node* sumber kurang dari radius (berada di dalam zona radius). Pada Gambar 2.1, *node* G, H, I, J merupakan *peripheral* *node*, *node* A, B, C, D, E, F merupakan *interior* *node*, dan *node* K merupakan *node* di luar zona *routing* *node* S. Pada Gambar 2.1, terlihat bahwa *node* H dapat dicapai melalui dua jalur, yang pertama sepanjang dua *hop* melalui *node* S-C-H, yang kedua sepanjang tiga *hop* melalui *node* S-D-C-H. Namun, bagaimanapun juga *node* H tetap berada di dalam zona *routing* selama jarak terpendek yang dapat ditempuh adalah kurang dari atau sama dengan zona radius.

**Gambar 2.1** Contoh Zona Routing Node S dengan Radius 2[4]

Ketika *node* S (*node* sumber) ingin mengirim paket ke *node* tujuan, langkah pertama yang dilakukan adalah mengirimkan permintaan rute ke *peripheral* *node* miliknya. Permintaan rute berisi alamat sumber, alamat tujuan dan *sequence* *number*, lalu setiap *peripheral* *node* akan mengecek zonanya apakah *node* tujuan berada di dalamnya. Jika pada zona *peripheral* *node* tidak terdapat *node* tujuan, maka *peripheral* *node* akan menambahkan alamatnya kepada paket *route-request* dan meneruskan paket kepada *peripheral* *node*. Namun, ketika *node* tujuan berada di dalam zonanya, maka *peripheral* *node* akan mengirimkan *route-reply* kembali menuju *node* sumber. *Node* sumber menggunakan jalur yang disimpan dalam paket *route-reply* untuk mengirim paket data ke *node* tujuan.

Arsitektur *routing protocol* ZRP ditunjukkan pada Gambar 2.2. IARP memelihara informasi *routing* dari *node*-*node* yang berada dalam zona *routing* sebuah *node*. *Route discovery* dan *route maintenance* dilakukan oleh IERP. Bila diperlukan penemuan global, jika topologi zona lokal diketahui, maka hal tersebut bisa digunakan untuk mengurangi lalu lintas. Untuk mengirim *broadcast* paket, ZRP menggunakan konsep *bordercasting*, yang layanannya disediakan oleh *Bordercasting Resolution Protocol* (BRP). BRP menggunakan zona *routing* yang sudah diperluas, yang disediakan oleh IARP lokal untuk membangun *bordercast tree* bersamaan dengan paket permintaan yang diarahkan. BRP menggunakan mekanisme *query control* yang sangat khusus untuk mengarahkan permintaan rute dari area jaringan yang telah dicakup oleh kueri sebelumnya [4].



**Gambar 2.2** Arsitektur ZRP[4]

## Clustering Jaringan Sensor Nirkabel

Jaringan sensor nirkabel biasanya terdiri dari banyak node bisa mencapai puluhan bahkan ratusan node. Node-node ini dapat memperluas jangkuan sink dalam melakukan sensing, karena node yang berada diluar jangkauan dapat membantu memberi data dengan melakukan multihop ke node lainnya sampai kepada sink.[4]

Melakukan multihop atau tranmisi jauh secara masive tanpa adanya organisasi tertentu akan menguras energi sangat besar yang malah membuat *lifespan* dari pada jaringan sensor terlalu pendek karena pemakaian energi yang tidak efisien. Karena itu jaringan sensor nirkabel dilakukan *clustering* dimana membagi node menjadi beberapa kelompok kecil. Di dalam *cluster* terdapat sebuah node yang menjadi komando node lain yang ada di dalam *cluster* yang disebut *cluster head*. *Cluster head* berfungsi dalam mengatur *cluster* node dari penjadwalan transmisi maupun sampai menjadi hop ke sink. Namun ada kalanya *cluster head* kehabisan daya atau kehilangan fungsi sehingga dapat mengancam keseluruhan sistem, maka sangat penting untuk melakukan rotasi cluster head agar terhindar dari mati nya jaringan sensor nirkabel.[4]

*Network Simulator* (NS) pertama kali dibangun sebagai varian dari REAL *Network Simulator* pada tahun 1989 di UCB (University of California Berkeley). NS sendiri bersifat *open source* di bawah GPL (*Gnu Public License*), sehingga NS dapat diunduh dan digunakan secara gratis. Sifat *open source* ini juga berdampak pada pengembangan NS menjadi lebih dinamis. Pemodelan media, protokol, komponen jaringan, dan perilaku trafik cukup lengkap bila dibandingkan dengan perangkat lunak lain yang sejenis. Hal ini disebabkan oleh pengembangan NS yang dilakukan oleh banyak periset dunia [5].

*Network simulator* memiliki beberapa versi, salah satunya adalah *Network Simulator* 2 (NS-2). NS-2 merupakan alat bantu simulasi berbasis aktivitas yang banyak digunakan pada penelitian jaringan kabel dan nirkabel. NS-2 menggunakan dua bahasa utama, yaitu C++ dan *Object-oriented Tool Command Language* (OTCL). Bahasa C++ digunakan untuk menggambarkan mekanisme internal (*backend*) pada objek simulasi, sedangkan OTCL digunakan untuk menggambarkan lingkungan simulasi eksternal (*frontend*) pada perakitan dan konfigurasi objek simulasi. Hasil dari simulasi menggunakan NS-2 berupa *trace* *file* (\*.tr) dan *network animator* (\*.nam) [6] [7].

Pada tugas akhir ini digunakan NS-2 versi 2.35 untuk melakukan simulasi lingkungan MANET menggunakan *routing protocol* ZRP yang sudah dimodifikasi. *Trace file* yang dihasilkan oleh NS-2 juga digunakan sebagai informasi untuk mengukur performa *routing protocol* ZRP yang sudah dimodifikasi.

### Instalasi

Sebelum melakukan instalasi NS-2, terlebih dahulu harus memastikan bahwa *compiler* yang terpasang sudah sesuai dengan standar yang digunakan oleh NS-2, yaitu gcc-4.8. Untuk memasang *compiler* tersebut dapat dilakukan dengan perintah sebagai berikut:

sudo apt install gcc-4.8 g++-4.8

NS-2 membutuhkan beberapa *package* yang harus sudah terpasang pada perangkat komputer sebelum memulai instalasi NS-2. Untuk memasang *dependency* yang dibutuhkan dapat dilakukan dengan perintah sebagai berikut :

sudo apt update

sudo apt-get install build-essential autoconf automake libxmu-dev

Kemudian, unduh NS-2 versi 2.35 yang dapat dilakukan melalui tautan berikut :

https://sourceforge.net/projects/nsnam/files/allinone/ns-allinone-2.35/ns-allinone-2.35.tar.gz/download

lalu ekstrak berkas unduhanNS-2 pada direktori yang diinginkan. Selanjutnya, masuk ke dalam direktori ns-allinone-2.35/ns-2.35, lalu ubah baris kode ke-137 pada *file* ls.h di folder linkstate menjadi seperti perintah berikut :

gedit Makefile.in

ubah baris kode ke-36 dan 37 menjadi:

CC = gcc-4.8

CPP = g++-4.8

gedit linkstate/ls.h

ubah baris kode ke-137 menjadi:

void eraseAll(){**this->**erase (baseMap::begin(),baseMap::end());}

Setelah mengubah baris kode, kembali ke direktori ns-allinone-2.35 dan instalNS-2 dengan menjalankan perintah berikut:

./install

Kemudian, *environment* sistem pada NS-2 diatur agar NS-2 dapat dijalankan. Pengaturan tersebut dapat dilakukan melalui perintah sebagai berikut:

sudo gedit ~/.bashrc

Copy skrip berikut diakhir file:

# LD\_LIBRARY\_PATH

OTCL\_LIB=/home/hania/ns-allinone-2.35/otcl-1.14

NS2\_LIB=/home/hania/ns-allinone-2.35/lib

X11\_LIB=/usr/X11R6/lib

USR\_LOCAL\_LIB=/usr/local/lib

export LD\_LIBRARY\_PATH=$LD\_LIBRARY\_PATH:$OTCL\_LIB:$NS2\_LIB:$X11\_LIB:$USR\_LOCAL\_LIB

# TCL\_LIBRARY

TCL\_LIB=/home/hania/ns-allinone-2.35/tcl8.5.10/library

USR\_LIB=/usr/lib

export TCL\_LIBRARY=$TCL\_LIB:$USR\_LIB

# PATH

XGRAPH=/home/hania/ns-allinone-2.35/bin:/home/hania/ns-allinone-2.35/tcl8.5.10/unix:/home/hania/ns-allinone-2.35/tk8.5.10/unix

NS=/home/hania/ns-allinone-2.35/ns-2.35/

NAM=/home/hania/ns-allinone-2.35/nam-1.15/

PATH=$PATH:$XGRAPH:$NS:$NAM

Lalu masukkan perintah source ~/.bashrc dan ns pada terminal.

### *Trace File*

*Trace file* merupakan dokumen hasil simulasi NS-2 yang berisikan informasi detail mengenai pengiriman paket data. *Trace file* digunakan untuk menganalisis performa *routing protocol* yang disimulasikan. Penjelasan mengenai *trace* *file* ditunjukkan pada Tabel 2.1.

**Tabel 2.1** Detail Penjelasan Trace File ZRP

| **Kolom ke-** | **Penjelasan** | **Isi** |
| --- | --- | --- |
| 1 | *Event* | s: *sent*  r: *received*  f: *forwarded*  D: *dropped* |
| 2 | *Time* | Waktu terjadinya *event* |
| 3 | ID *Node* | \_x\_: dari 0 hingga banyak *node* pada topologi |
| 4 | *Layer* | AGT: *application*  RTR: *routing*  LL: *link layer*  IFQ: *packet queue*  MAC: MAC  PHY: *physical* |
| 5 | *Flag* | ---: Tidak ada |
| 6 | *Sequence Number* | Nomor paket |
| 7 | Tipe Paket | ZRP: paket *routing ZRP*  Cbr: berkas paket CBR (*Constant Bit Rate*)  RTS: *Request To Send* yang dihasilkan MAC 802.11  CTS: *Clear To Send* yang dihasilkan MAC 802.11  ACK: MAC ACK  ARP: Paket *link layer Address Resolution Protocol* |
| 8 | Ukuran | Ukuran paket pada *layer* saat itu |
| 9 | Detail MAC | [a b c d]  a: perkiraan waktu paket  b: alamat penerima  c: alamat asal  d: IP *header* |
| 10 | *Flag* | ------: Tidak ada |
| 11 | Detail IP *source*, *destination*, dan *nexthop* | [a:b c:d e f]  a: IP *source node*  b: *port source node*  c: IP *destination node* (jika -1 berarti *broadcast*)  d: *port destination node*  e: IP *header* ttl  f: IP *nexthop* (jika 0 berarti *node* 0 atau *broadcast)* |

## *Cluster Head*

Dalam jaringan sensor nirkabel terdapat node yang nantinya akan menjadi *cluster head* yang bertugas untuk mengumpulkan data dari node-node yang tergabung dan kemudian mengirimkannya ke node server / coordinator. Cluster head akan dipilih dan diganti di setiap putaran untuk meratakan penggunaan energi.[4]

*Cluster Head* juga bertugas menjadi komando dalam clsunter node. Cluster head juga mengatur penjadwalan transmisisetiapnodeuntukmenghindaritubrukantransmisiyang berujung packet lost. Karena fungsinya yang vital cluster head harus awas terhadap penurunan energi yang telah digunakan. Saat penurunan energi melewati batas cluster head akan memberi instruksi kepada cluster node untuk mengganti cluster head denganyangbarudarinodelain.

merupakan sebuah program filter untuk teks berupa bahasa pemrograman pada *shell* atau C yang memiliki karakteristik sebagai alat untuk melakukan filter atau manipulasi data. AWK bersifat *data-driven* yang berisikan kumpulan perintah yang akan dijalankan pada data tekstural baik secara langsung pada *file* atau digunakan sebagai bagian dari *pipeline*. Secara umum AWK dapat digunakan untuk mengelola *database* sederhana, membuat laporan, memvalidasi data, dan membuat algoritma yang digunakan untuk mengubah bahasa komputer ke bahasa lainnya. Dengan kata lain, AWK menyediakan fasilitas yang dapat memudahkan untuk memecah bagian data untuk proses selanjutnya, mengurutkan data, dan menampilkan komunikasi jaringan yang sederhana. Pada tugas akhir ini, AWK digunakan untuk membuat skrip menghitung metrik analisis berupa *Packet Delivery Ratio* (PDR), *Routing Overhead* (RO), dan *Delivery Delay* (DD) dari hasil simulasi menggunakan NS-2 [8].

***(Halaman ini sengaja dikosongkan)***

# BAB III PERANCANGAN

Bab ini membahas mengenai perancangan implementasi sistem yang dibangun pada tugas akhir. Bagian yang akan dijelaskan pada bab ini berawal dari deskripsi umum sistem, perancangan skenario, alur, hingga gambaran implementasi sistem yang diterapkan pada *Network Simulator* 2 (NS-2).

## Deskripsi Umum

Pada tugas akhir ini akan dilakukan implementasi *routing protocol* ZRP yang dimodifikasi dengan menerapkan metode zonasi yang adaptif terhadap tingkat kepadatan *node* tetangga. Pembuatan skenario MANET ini menggunakan simulator yaitu *Network Simulator* 2 (NS-2).

Perancangan skenario uji coba diawali dengan menetapkan *threshold* kepadatan *node* tetangga. *Threshold* ini digunakan sebagai batas maksimal jumlah kepadatan *node* tetangga yang boleh dimiliki *node* saat simulasi dijalankan. Selanjutnya, melakukan inisialisasi nilai radius yang akan digunakan sebagai nilai awal saat simulasi dijalankan, dan penentuan nilai radius maksimal yang boleh digunakan dalam simulasi. Nilai awal dan maksimal radius ini bersifat tetap. Kemudian melakukan modifikasi pemilihan radius yang akan digunakan untuk proses simulasi. Jika kepadatan *node* tetangga pada zona di dalam radius (zona IARP) kurang dari *threshold* dan besar radius kurang dari radius maksimal, maka radius akan bertambah satu. Sebaliknya, jika kepadatan *node* tetangga melebihi *threshold* yang ditentukan, maka radius akan berkurang satu.

Skenario pengiriman paket data menggunakan *routing protocol* ZRP akan berjalan setelah besar radius terpilih. Simulasi yang dilakukan akan menghasilkan *trace file,* yang kemudian dianalisis menggunakan skrip AWK untuk mendapatkan *Packet Delivery Ratio* (PDR)*, Routing Overhead* (RO), dan *Delivery Delay* (DD)*.* Analisis tersebut dapat mengukur performa *routing protocol* ZRP yang telah dimodifikasi dibandingkan dengan *routing protocol* ZRP asli. Diagram rancangan simulasi dengan *routing protocol* ZRP yang telah dimodifikasi dapat dilihat pada Gambar 3.1.

Skenario aktivitas *node* (.tcl)

Protokol ZRP modifikasi

Simulasi MANETs pada NS-2

*Trace file*

Skrip analisis (.awk)

*Routing Overhead*

*Packet Delivery Ratio*

*Delivery Delay*

Digunakan

Digunakan

Menghasilkan

Dianalisis oleh

Menghasilkan

Menghasilkan

Menghasilkan

**Gambar 3.1** Diagram Rancangan Simulasi ZRP Modifikasi

## Daftar Istilah

Daftar istilah yang sering digunakan pada buku tugas akhir ini dapat dilihat pada Tabel 3.1.

**Tabel 3.1** Daftar Istilah

| **No.** | **Istilah** | **Penjelasan** |
| --- | --- | --- |
| 1 | *Mobile Adhoc Network*  (MANET) | *Mobile Ad Hoc Network* (MANET) merupakan jaringan nirkabel ad hoc yang melakukan konfigurasi mandiri secara kontinyu dan tidak memiliki infrastruktur yang tetap. |
| 2 | *Zone Routing Protocol* (ZRP) | *Zone Routing Protocol* (ZRP) merupakan salah satu algoritma *routing* hibrid yang menggunakan parameter berupa *routing* zona dan menggabungkan kelebihan dari *routing* *protocol* proaktif dan reaktif. |
| 3 | *Network Simulator* 2 (NS-2) | NS-2 merupakan alat bantu simulasi berbasis aktivitas pada penelitian jaringan kabel maupun nirkabel. |
| 4 | *Packet Delivery Ratio* (PDR) | PDR merupakan teknik penghitungan perbandingan jumlah paket yang diterima oleh *node* tujuan dengan jumlah paket yang dikirim oleh *node* sumber. |
| 5 | *Routing Overhead* (RO) | RO merupakan teknik penghitungan jumlah *routing* paket kontrol yang ditransmisikan ke *node* tujuan selama simulasi terjadi. |
| 6 | *Delivery Delay* (DD) | DD merupakan teknik penghitungan waktu yang diperlukan mulai dari paket dikirimkan oleh *node* sumber sampai paket data tersebut berhasil diterima *node* tujuan. |
| 7 | *Threshold* | Batas nilai yang dijadikan sebagai acuan. |
| 8 | *Node* tetangga | *Node* di sekitar *node* sumber yang berada di dalam zona *IntrA-zone Routing Protocol* (IARP). |
| 9 | Kepadatan *Node* Tetangga | Perbandingan jumlah *node* tetangga dengan jumlah seluruh *node*. |

## Perancangan Skenario *Threshold* Kepadatan *Node* Tetangga

Perancangan skenario *threshold* kepadatan *node* tetangga pada tugas akhir ini dilakukan melalui pra-uji coba menghitung nilai metrik analisis untuk setiap besaran kepadatan *node* tetangga. Besar kepadatan *node* tetangga yang digunakan adalah 0 hingga 1. Simulasi yang dilakukan menggunakan ZRP yang telah dimodifikasi. Hasil dari penghitungan metrik analisis nantinya akan dilihat nilai metrik analisis yang tertinggi, dengan nilai PDR yang menjadi parameter utama penentuan *threshold*. Besar kepadatan *node* tetangga yang memiliki nilai PDR paling tinggi akan dijadikan sebagai nilai *threshold* untuk semua skenario simulasi.

## Analisis dan Perancangan Modifikasi *Routing Protocol* ZRP

Perancangan modifikasi *routing protocol* pada tugas akhir ini mencakup pada perancangan penghitungan kepadatan *node* tetangga untuk setiap radius dan perancangan pemilihan radius yang adaptif terhadap jumlah kepadatan *node* tetangga.

### Perancangan Penghitungan Kepadatan *Node* Tetangga untuk Setiap Radius

Ketika nilai *threshold* kepadatan *node* tetangga serta nilai radius awal dan maksimal telah ditentukan, maka ZRP yang telah dimodifikasi akan menghitung kepadatan *node* tetangga. Kepadatan *node* tetangga yang dimaksud adalah kepadatan *node* di sekitar *node* sumber yang berada di dalam zona IARP.

Langkah awal untuk menghitung kepadatan *node* tetangga adalah dengan menghitung terlebih dahulu jumlah *node* di sekitar *node* sumber yang berada di dalam zona IARP. Setelah itu, membagi jumlah *node* tetangga dengan jumlah seluruh *node* yang digunakan pada simulasi. Sehingga, akan didapat besar kepadatan *node* tetangga di dalam zona IARP. *Pseudocode* untuk penghitungan kepadatan *node* tetangga dapat dilihat pada Gambar 3.2.

total\_*node*\_IARP = jumlah *node* tetangga

density = total\_*node*\_IARP/total\_*node*\_simulasi

**Gambar 3.2** Pseudocode Penghitungan Kepadatan Node Tetangga

### Perancangan Pemilihan Radius

Pemilihan radius akan dimulai setelah kepadatan *node* tetangga diketahui. Selanjutnya, jumlah kepadatan *node* tetangga dibandingkan dengan nilai *threshold* yang telah ditentukan. Jika jumlah kepadatan *node* tetangga kurang dari nilai *threshold* dan besar radius kurang dari besar radius maksimal, maka besar radius bertambah satu. Sebaliknya, jika kepadatan *node* tetangga melebihi *threshold* yang ditentukan dan besar radius lebih dari radius minimal, maka radius akan berkurang satu. *Pseudocode* untuk penghitungan radius dapat dilihat pada Gambar 3.3.

if (density < threshold && radius < radius\_max)

then

radius++

if (density > threshold && radius > radius\_min)

then

radius--

**Gambar 3.3** Pseudocode Penghitungan Radius

## Perancangan Simulasi pada NS-2

Simulasi MANET pada NS-2 dilakukan dengan menggunakan skrip TCL dengan parameter yang berisi konfigurasi untuk membangun lingkungan simulasi. Parameter simulasi perancangan sistem MANET dapat dilihat pada Tabel 3.2 [9].

**Tabel 3.2** Parameter Lingkungan Simulasi

|  |  |  |
| --- | --- | --- |
| **No.** | **Parameter** | **Spesifikasi** |
| 1. | *Network Simulator* | NS-2 versi 2.35 |
| 2. | *Routing protocol* | ZRP |
| 3. | Radius | radius awal = 1,  radius maksimal = 6 |
| 4. | Waktu Simulasi | 200 detik |
| 5. | Waktu Pengiriman Paket Data | 2,5 – 200 detik |
| 5. | Area Simulasi | 1500 m x 1500 m |
| 6. | Banyak *Node* | 25, 50, 100 |
| 7. | Kecepatan Pergerakan *Node* | 5 m/s |
| 8. | Radius Transmisi | 250 m |
| 9. | Tipe Koneksi | UDP |
| 10. | Tipe Data | *Constant Bit Rate* (CBR) |
| 11. | *Source/Destination* | Dinamis |
| 12. | Kecepatan Generasi Paket | 1 paket per detik |
| 13. | Ukuran Paket Data | 512 *bytes* |
| 14. | Protokol MAC | IEEE 802.11 |
| 15. | Tipe Antena | OmniAntenna |
| 16. | Tipe Interface Queue | Droptail/PreQueue |
| 17. | Tipe Kanal | *Wireless Channel* |
| 18. | Tipe Trace | Old Format Trace |

## Perancangan Metrik Analisis

Berikut ini merupakan parameter-parameter yang akan dianalisis pada tugas akhir ini untuk mengetahui pengaruh dari implementasi zonasi yang adaptif terhadap tingkat kepadatan *node* tetangga pada *routing protocol* ZRP.

### *Packet Delivery Ratio* (PDR)

*Packet Delivery Ratio* (PDR) merupakan teknik penghitungan perbandingan antara jumlah paket yang diterima oleh *node* tujuan dengan jumlah paket yang dikirim oleh *node* sumber. PDR dihitung dengan persamaan 3.1.

**(3.1)**

Pada persamaan 3.1, didapatkan informasi bahwa :

***Packet Received*** : jumlah total paket yang diterima oleh *node* tujuan.

***Packet Sent*** : jumlah total paket yang dikirim oleh *node* sumber.

### *Routing Overhead* (RO)

*Routing Overhead* (RO) merupakan teknik penghitungan jumlah *routing* paket kontrol yang ditransmisikan per data paket ke *node* tujuan selama simulasi terjadi. *Routing* paket kontrol yang ditransmisikan berupa paket *route request* (RREQ), *route reply* (RREP), dan *route error* (RERR). Penghitungan RO digunakan untuk melihat banyaknya rute yang dicari dalam jaringan selama simulasi dijalankan. RO dihitung dengan persamaan 3.2.

**(3.2)**

### *Delivery Delay* (DD)

*Delivery Delay* (DD) merupakan teknik penghitungan waktu yang diperlukan mulai dari paket dikirimkan oleh *node* sumber sampai paket data tersebut berhasil diterima oleh *node* tujuan. Penghitungan DD digunakan untuk melihat berapa lama waktu yang diperlukan paket untuk sampai ke *node* tujuan. DD dihitung dengan persamaan 3.3.

**(3.3)**

Pada persamaan 3.3, didapatkan informasi bahwa :

***Time Packet Received*** : waktu ketika paket diterima oleh *node* tujuan.

***Time Packet Sent*** : waktu ketika paket dikirim oleh *node* sumber.

***(Halaman ini sengaja dikosongkan)***

# BAB IV IMPLEMENTASI

Bab ini membahas mengenai implementasi sistem yang dilakukan berdasarkan rancangan yang telah dijabarkan pada bab sebelumnya. Implementasi berupa kode sumber untuk membangun program. Implementasi yang dijelaskan meliputi lingkungan pembangunan sistem, implementasi skenario *threshold* kepadatan *node* tetangga, implementasi modifikasi *routing protocol* ZRP, implementasi simulasi pada NS-2, dan implementasi metrik analisis.

## Lingkungan Pembangunan Sistem

Lingkungan sistem yang digunakan untuk membangun implementasi skenario dapat dilihat pada Tabel 4.1.

**Tabel 4.1** Tabel Lingkungan Pembangunan Sistem

|  |  |  |
| --- | --- | --- |
| **Perangkat** | **Komponen** | **Spesifikasi** |
| **Perangkat Keras** | *Processor* | Intel(R) Core(TM) i5-7400 CPU @ 3.00GHz 3.00 GHz |
| Memori | RAM 8,00 GB |
| Penyimpanan | 928 GB |
| **Perangkat Lunak** | Sistem Operasi | Ubuntu Bionic 18.04 LTS 64 bit |
| Simulator | *Network Simulator* 2 versi 2.35 |

## Instalasi *Routing Protocol* ZRP pada NS-2

Pada umumnya, *routing protocol* ZRP tidak tersedia pada bawaan instalasi NS-2 versi 2.35 standar. Maka dari itu, perlu dilakukan instalasi *patch routing protocol* ZRP agar *routing protocol* ZRP dapat dijalankan pada NS-2. Sebelum melakukan instalasi, pastikan bahwa NS-2 sudah diinstal pada perangkat. Langkah-langkah instalasi *routing protocol* ZRP adalah sebagai berikut :

1. Unduh *patch* ZRP pada tautan berikut <http://bit.ly/PatchZRP-NS235>.
2. Masuk ke dalam direktori *ns-allinone-2.35.*
3. Salin *file* *patch* yang diunduh tadi (*file* bernama *zrp-ns235.patch*) ke dalam direktori *ns-allinone-2.35.*
4. Setelah disalin, jalankan perintah berikut untuk melakukan *patch* *routing protocol* ZRP.

patch -p0 < zrp-ns235.patch

1. Setelah melakukan *patch*, lakukan instalasi pada direktori *ns-allinone-2.35* menggunakan perintah berikut.

./install

1. Atur *environment variables* pada .bashrc agar NS-2 dapat dijalankan pada semua lingkungan Linux.

## Implementasi Modifikasi *Routing Protocol* ZRP

*Routing protocol* ZRP merupakan salah satu *routing protocol* hibrid yang umum digunakan dalam simulasi MANET. *Routing protocol* ZRP ini membagi *node* ke dalam zona-zona menggunakan radius yang ditentukan secara manual. Penentuan radius secara manual terbilang kurang fleksibel mengingat MANET merupakan jaringan ad hoc yang topologinya berubah-ubah dalam periode tertentu.

Pada tugas akhir ini dilakukan modifikasi pada *routing protocol* ZRP untuk konsep zonasinya, di mana nilai radius akan berubah dinamis berdasarkan tingkat kepadatan *node* tetangga sesuai dengan *threshold* yang telah ditentukan. Sehingga, pada tugas akhir ini dapat dilihat performa pada *routing protocol* ZRP yang telah dimodifikasi.

Implementasi modifikasi *routing protocol* ZRP ini dibagi menjadi dua bagian, yaitu

1. Implementasi penghitungan kepadatan *node* tetangga untuk setiap radius, dan
2. Implementasi pemilihan radius.

Kode program implementasi *routing protocol* ZRP pada NS-2 versi 2.35 terletak di dalam direktori *ns-2.35/zrp*. Pada direktori tersebut terdapat beberapa dokumenseperti *constants.h*, *zrp.cc*, *zrp.h*, dan *zrp.o*. Pada tugas akhir ini, penulis hanya melakukan modifikasi pada dokumen *zrp.cc*, *zrp.h*, dan *constants.h* untuk menghitung jumlah kepadatan *node* tetangga dan memilih radius sesuai dengan tingkat kepadatan *node* tetangganya*.* Pada bagian ini, penulis akan menjelaskan langkah-langkah dalam mengimplementasikan modifikasi *routing protocol* ZRP agar radius menjadi adaptif terhadap tingkat kepadatan *node* tetangga.

### Implementasi Penghitungan Kepadatan *Node* Tetangga untuk Setiap Radius

Langkah awal yang dilakukan untuk menghitung jumlah kepadatan *node* tetangga seperti yang telah dirancang pada subbab 3.4.1 adalah dengan menghitung jumlah *node* tetangga yang berada di dalam zona IARP.

Pada *routing protocol* ZRP, terdapat fungsi *print\_tables* dalam kelas *NDPAgent* yang berfungsi untuk mencatat *node* tetangga yang berjarak satu *hop* dari *node* sumber. Pada tugas akhir ini, penulis memodifikasi fungsi tersebut agar bisa mengeluarkan informasi berisi *node* tetangga yang berjarak satu *hop* dari *node* sumber. Potongan kode modifikasi dapat dilihat pada Kode Sumber 4.1.

int NDPAgent::print\_tables() {

...

return neighborLst\_.numNeighbors\_;

}

**Kode Sumber 4.1** Modifikasi Fungsi print\_tables dalam Kelas NDPAgent

Dalam *routing protocol* ZRP, terdapat juga kelas *IARPAgent* yang berisi fungsi-fungsi untuk mengatur proses pada IARP ketika simulasi dijalankan. Pada tugas akhir ini, penulis memanfaatkan kelas tersebut untuk membuat fungsi baru guna menghitung jumlah *node* yang berada di dalam zona IARP, yang jaraknya lebih dari satu *hop* dari *node* sumber. Fungsi tersebut penulis beri nama *count\_node\_in\_rad* dan diletakkan di dalam dokumen *zrp.cc*. Potongan kode modifikasi dapat dilihat pada Kode Sumber 4.2.

int IARPAgent::count\_*node*\_in\_rad(){

InnerRoute \*cur = irLst\_.head\_;

int result = 0;

printf("\nIARP Routes: %d, [ ", agent\_->myaddr\_);

if(irLst\_.numRoutes\_ == 0){

printf("EMPTY");

}

for(int i=0; i<irLst\_.numRoutes\_; i++){

if(cur->numHops\_<= agent\_->radius\_){

printf("(%d:%d), ", cur->addr\_, cur->nextHop\_);

result++;

}

cur = cur->next\_;

}

printf("]");

return result;

}

**Kode Sumber 4.2** Menghitung *Node* di dalam IARP

Setelah melakukan modifikasi pada fungsi *print\_tables* dan membuat fungsi baru, selanjutnya penulis melakukan modifikasi pada kelas *NDPAgent* dan *IARPAgent* yang terletak pada dokumen *zrp.h*. Pada kelas *NDPAgent*, modifikasi yang dilakukan adalah mengganti tipe data fungsi *print\_tables* menjadi integer. Potongan kode modifikasi dapat dilihat pada Kode Sumber 4.3. Kode selengkapnya dapat dilihat pada lampiran 1.

class NDPAgent{

public:

...

int print\_tables();

}

**Kode Sumber 4.3** Inisialisasi Fungsi *print\_tables* dalam Kelas *NDPAgent*

Sedangkan pada kelas *IARPAgent*, modifikasi yang dilakukan adalah menambah fungsi yang baru dibuat agar fungsi dapat dikenali. Potongan kode modifikasi dapat dilihat pada Kode Sumber 4.4. Kode selengkapnya dapat dilihat pada lampiran 2.

class IARPAgent{

public:

...

int count\_*node*\_in\_rad();

}

**Kode Sumber 4.4** Inisialisasi Fungsi *count\_node\_in\_rad*

Penulis juga melakukan modifikasi pada dokumen *constant.h* dengan mengubah nilai DEBUG menjadi 1 agar jumlah *node* tetangga dapat dikeluarkan pada dokumen lain. Potongan kode modifikasi dapat dilihat pada Kode Sumber 4.5. Kode selengkapnya dapat dilihat pada lampiran 3.

**Kode Sumber 4.5** Modifikasi Dokumen *constant.h*

...

#define DEBUG 1

...

Pada *routing protocol* ZRP, terdapat kelas *NDPAckTimer* yang terletak pada dokumen *zrp.cc*. Fungsi tersebut digunakan untuk melakukan *update* *node* tetangga yang berjarak satu *hop* dari *node* sumber secara periodik. Apabila sebuah *node* tidak menerima ACK dari *node* tetangga yang berjarak satu *hop* darinya, maka *node* tersebut dihapus dari tabel *node* tetangga dan kelas *NDPAckTimer* akan memberitahu kelas *IARP* untuk melakukan *update* rute. Di dalam fungsi *NDPAckTimer*, penulis memanggil fungsi *print\_tables* yang telah dimodifikasi dan menyimpannya ke dalam variabel jumlah *node* tetangga yang berjarak satu *hop*. Selanjutnya, penulis menambah kode untuk menghitung kepadatan *node* tetangga, yaitu dengan membagi jumlah *node* tetangga dengan jumlah seluruh *node* yang digunakan dalam simulasi. Potongan kode modifikasi untuk menghitung kepadatan *node* tetangga dapat dilihat pada Kode Sumber 4.6. Kode selengkapnya dapat dilihat pada lampiran 4.

void NDPAckTimer::handle(Event\* e){

...

if (agent\_->radius\_==1) {

int num\_neighbor = (agent\_-> ndpAgt\_) .print\_tables();

float density = (float)num\_neighbor / jumlah\_*node*\_simulasi;

}

}

**Kode Sumber 4.6** Memanggil Fungsi print\_tables dalam Kelas NDPAckTimer

Pada *routing protocol* ZRP, terdapat juga kelas *IARPPeriodicUpdateTimer* yang terletak pada dokumen *zrp.cc*. Fungsi tersebut digunakan untuk melakukan *update* IARP secara periodik. Di dalam fungsi ini, penulis memanggil fungsi yang telah dibuat dan menyimpannya ke dalam variabel jumlah *node* tetangga. Selanjutnya, penulis juga menambah kode untuk menghitung kepadatan *node* tetangga, yaitu dengan membagi jumlah *node* tetangga dengan jumlah seluruh *node* yang digunakan dalam simulasi. Potongan kode modifikasi untuk menghitung kepadatan *node* tetangga dapat dilihat pada Kode Sumber 4.7. Kode selengkapnya dapat dilihat pada lampiran 5.

**Kode Sumber 4.7** Memanggil Fungsi *count\_node\_in\_rad* dalam Kelas *IARPPeriodicUpdateTimer*

void IARPPeriodicUpdateTimer::handle(Event\* e){

...

int *node*\_in\_rad = (agent\_-> iarpAgt\_) .count\_*node*\_in\_rad()-1;

float kepadatan = (float)*node*\_in\_rad / jumlah\_*node*\_simulasi;

...

}

### Implementasi Pemilihan Radius

Langkah awal yang dilakukan untuk memilih radius secara adaptif terhadap kepadatan *node* tetangga seperti yang telah dirancang pada subbab 3.4.2 adalah membandingkan kepadatan *node* tetangga dengan nilai *threshold*. Proses membandingkan kepadatan *node* tetangga dilakukan pada kelas *NDPAckTimer* dan *IARPPeriodicUpdateTimer* dalam dokumen *zrp.cc*.

Pada kelas *NDPAckTimer*, apabila kepadatan *node* tetangga kurang dari *threshold*, maka radius bertambah satu. Penambahan baris kode diletakkan di dalam baris kode untuk menghitung kepadatan *node* tetangga. Potongan kode untuk proses pemilihan radius dapat dilihat pada Kode Sumber 4.8. Kode selengkapnya dapat dilihat pada lampiran 4.

void NDPAckTimer::handle(Event\* e){

...

if (agent\_->radius\_==1) {

...

float th = nilai\_threshold;

printf("\nNDP UPDATED FOR *NODE* NUMBER : %d, radius : %d, kepadatan : %f, num\_neighbor : %d\n\n", agent\_->myaddr\_, agent\_->radius\_, density, num\_neighbor);

if (density < th && num\_neighbor > 0) {

agent\_->radius\_ +=1;

}

}

}

**Kode Sumber 4.8** Pemilihan Radius dalam Kelas NDPAckTimer

Selanjutnya pada kelas *IARPPeriodicTimer*, terdapat dua kondisi dalam implementasi radius yang adaptif. Kondisi pertama, yaitu apabila kepadatan *node* tetangga kurang dari *threshold* serta besar radius lebih dari satu dan kurang dari radius maksimal, maka radius bertambah satu. Kondisi kedua, yaitu apabila kepadatan *node* tetangga lebih dari *threshold* dan besar radius lebih dari satu, maka radius akan bekurang satu. Penambahan baris kode diletakkan tepat setelah penghitungan kepadatan *node* tetangga. Potongan kode untuk proses pemilihan radius dapat dilihat pada Kode Sumber 4.9. Kode selengkapnya dapat dilihat pada lampiran 5.

void IARPPeriodicUpdateTimer::handle(Event\* e){

...

int *node*\_in\_rad = (agent\_-> iarpAgt\_) .count\_*node*\_in\_rad()-1;

float kepadatan = (float)*node*\_in\_rad / jumlah\_*node*\_simulasi;

float treshold = nilai\_threshold;

printf("\nIARP UPDATED FOR *NODE* NUMBER : %d, radius : %d, kepadatan : %f, *node*\_in\_rad : %d\n\n", agent\_->myaddr\_, agent\_->radius\_, kepadatan, *node*\_in\_rad);

if (kepadatan < treshold && agent\_->radius\_ < radius\_maksimal && agent\_->radius\_ > 1 && *node*\_in\_rad > 0){

agent\_->radius\_ +=1;

}

if(kepadatan>treshold && agent\_->radius\_>1){

agent\_->radius\_ -=1;

}

(agent\_->iarpAgt\_).build*Routing*Table();

...

}

**Kode Sumber 4.9** Pemilihan Radius dalam Kelas IARPPeriodicTimer

## Implementasi Simulasi pada NS-2

Implementasi simulasi pada NS-2 dijalankan menggunakan kode program TCL. Dokumen ini berisi konfigurasi yang dibutuhkan untuk setiap *node* dan langkah-langkah yang dilakukan selama simulasi. Kode program untuk konfigurasi lingkungan simulasi dapat dilihat pada Kode Sumber 4.10.

#Channel Type

set val(chan) Channel/WirelessChannel;

#radio-propagationmodel

set val(prop) Propagation/TwoRayGround;

#network interface type

set val(netif) Phy/WirelessPhy;

#MAC type

set val(mac) Mac/802\_11;

#interface queue type

set val(ifq) Queue/DropTail/PriQueue;

#link layer type

set val(ll) LL;

#antenna model

set val(ant) Antenna/OmniAntenna;

#max packet in ifq

set val(ifqlen) 1000;

#number of mobile*node*s

set val(nn) 100;

#*Routing* protocol

set val(rp) ZRP;

#Dimension of topography

set val(x) 1500;

set val(y) 1500;

#Simulation time

set val(stop) 200.0;

#Setting ZRP initial radius

Agent/ZRP set radius\_ 1;

**Kode Sumber 4.10** Pengaturan Parameter Lingkungan Simulasi

Setelah melakukan konfigurasi lingkungan simulasi, langkah selanjutnya adalah melakukan inisialisasi program untuk skenario MANET. Kode program untuk inisialisasi skenario MANET dapat dilihat pada Kode Sumber 4.11.

#Create a ns simulator

set ns\_ [new Simulator]

#Set up topography object

set topo [new Topography]

$topo load\_flatgrid $val(x) $val(y)

#Nam File Creation nam – network animator

set namtrace [open file\_name.nam w]

#Tracing all the events and configuration

$ns\_ namtrace-all $namtrace

$ns\_ namtrace-all-wireless $namfile $val(x) $val(y)

#Trace File creation

set tracefile [open file\_name.tr w]

#Tracing all the events and cofiguration

$ns\_ trace-all $tracefile

#general operational descriptor- storing the hop details in the network

create-god $val(nn)

#Create wireless channel

set chan [new $val(chan)]

**Kode Sumber 4.11** Pengaturan Inisialisasi NS-2

Pada Kode Sumber 4.11, ns\_ merupakan variabel baru untuk simulator. Pengaturan topografi juga dilakukan untuk menentukan luas area yang akan digunakan saat simulasi, di mana dalam simulasi ini digunakan topografi dengan koordinat x dan y. Untuk menyimpan hasil simulasi, digunakan perintah *trace-all* di mana nantinya hasil simulasi akan dimasukkan ke dalam dokumen berekstensi .tr. Pada NS-2 juga disediakan *Network Animator* (biasa disebut NAM), yang memiliki sistem kerja hampir sama dengan dokumen .tr. Keduanya sama-sama melakukan *trace* terhadap skenario pada NS-2. Perbedaannya terletak pada hasil keluarannya, di mana NAM berbentuk visual dengan dokumen berekstensi .nam, sedangkan .tr berbentuk teks.

Sebelum membuat *node* untuk simulasi, hal pertama yang dilakukan adalah melakukan konfigurasi parameter *node* yang akan digunakan untuk simulasi. Konfigurasi *node* dapat terdiri dari tipe ad hoc *routing protocol*, *link layer*, MAC *layer*, dan lain sebagainya. Kode program untuk konfigurasi parameter *node* dapat dilihat pada Kode Sumber 4.12.

$ns\_ *node*-config

-adhoc*Routing* $val(rp) \

-llType $val(ll) \

-macType $val(mac) \

-ifqType $val(ifq) \

-ifqLen $val(ifqlen) \

-antType $val(ant) \

-propType $val(prop) \

-phyType $val(netif) \

-topoInstance $topo \

-agentTrace ON \

-routerTrace ON \

-macTrace ON \

-movementTrace ON \

-channel $chan

**Kode Sumber 4.12** Pengaturan Konfigurasi Parameter Node

Setelah melakukan konfigurasi parameter *node*, selanjutnya adalah membuat *node* dengan mendefinisikan *node*-*node* yang digunakan seperti menentukan posisi awal *node*, pergerakan *node*, dan pemberian label untuk masing-masing *node*. Pada tugas akhir ini, posisi *node* sumber dan tujuan diatur secara statis, sedangkan *node* *intermediate* diatur secara acak. Kode program untuk mendefinisikan *node* sumber dan tujuan dapat dilihat pada Kode Sumber 4.13.

#*Node* creation

for {set i 0} {$i < $val(nn)} {incr i} {

set *node*\_($i) [$ns\_ *node*]

$*node*\_($i) random-motion 0

$*node*\_($i) color black

$ns\_ initial\_*node*\_pos $*node*\_($i) 50

}

#set *node* source and destination

set val(source) [expr "$val(nn) - 2"]

set val(destination) [expr "$val(nn) - 1"]

#set source position

$*node*\_($val(source)) set X\_ 0.0

$*node*\_($val(source)) set Y\_ 750.0

$*node*\_($val(source)) set Z\_ 0.0

#set destination position

$*node*\_($val(destination)) set X\_ 1500.0

$*node*\_($val(destination)) set Y\_ 750.0

$*node*\_($val(destination)) set Z\_ 0.0

**Kode Sumber 4.13** Posisi Statis Node Sumber dan Node Tujuan

Pada Kode Sumber 4.13, dapat diketahui bahwa *node* sumber didefinisikan dengan jumlah *node* dikurangi 2, sedangkan untuk *node* tujuan adalah jumlah *node* dikurangi 1. Selanjutnya, pada Kode Sumber 4.13 juga dijelaskan bahwa posisi *node* sumber mempunyai koordinat posisi yaitu 0, 750 (sumbu x, y) dan kedalaman 0 (sumbu z). Sedangkan untuk *node* tujuan mempunyai koordinat posisi yaitu 1500, 750 (sumbu x, y) dan kedalaman 0 (sumbu z).

Selanjutnya, penulis mengatur posisi *node* *intermediate* secara acak dengan aturan untuk sumbu x acak mulai dari 0 hingga 1500, sumbu y dibuat statis dengan variasi titik koordinat dari 0 hingga 1500, dan sumbu z tetap pada titik 0. Penentuan *node* secara acak tergantung pada jumlah *node* simulasinya. Kode program untuk mendefinisikan *node* sumber dan tujuan dapat dilihat pada Kode Sumber 4.14. Kode selengkapnya dapat dilihat pada lampiran 6.

set *node* 0

for {set i 0} {$i < 5} {incr i} {

set distance 250

$*node*\_($*node*) set X\_ [expr {($distance\*$i) + $distance}]

$*node*\_($*node*) set Y\_ 750.0

$*node*\_($*node*) set Z\_ 0.0

$ns\_ initial\_*node*\_pos $*node*\_($*node*) 50

set *node* [expr $*node*+1]

}

for {set i 0} {$i < 6} {incr i} {

set distance 250

$*node*\_($*node*) set X\_ [expr {($distance\*$i) + 125}]

$*node*\_($*node*) set Y\_ 925.0

$*node*\_($*node*) set Z\_ 0.0

$ns\_ initial\_*node*\_pos $*node*\_($*node*) 50

set *node* [expr $*node*+1]

}

**Kode Sumber 4.14** Posisi Acak Node Intermediate

Berikutnya adalah pemberian warna sebagai penanda untuk mempermudah pengecekan pada NAM saat melakukan simulasi. Pemberian warna *node* sesuai dengan peran dari masing-masing *node*. Warna hijau menyatakan bahwa *node* tersebut adalah *node* sumber, warna biru menyatakan *node* tujuan, dan warna hitam menyatakan *node* *intermediate*. Kode program untuk memberi label dan warna pada *node* sumber, tujuan, dan *intermediate* dapat dilihat pada Kode Sumber 4.15.

# Label and coloring

for {set i 0} {$i < $val(nn)-2} {incr i} {

$ns\_ at 0.0 "$*node*\_($i) color black"

$ns\_ at 0.0 "$*node*\_($i) label *Node*$i"

}

$*node*\_($val(source)) color green

$ns\_ at 0.0 "$*node*\_($val(source)) color green"

$ns\_ at 0.0 "$*node*\_($val(source)) label Source"

$*node*\_($val(destination)) color blue

$ns\_ at 0.0 "$*node*\_($val(destination)) color blue"

$ns\_ at 0.0 "$*node*\_($val(destination)) label Destination"

**Kode Sumber 4.15** Pemberian Label dan Warna pada Node

Langkah selanjutnya adalah melakukan konfigurasi untuk mengatur pergerakan *node* pada simulasi. Dalam tugas akhir ini, pergerakan *node* dibuat secara acak. Pergerakan *node* secara acak menggunakan fitur *setdest* pada NS-2, dan menggunakan fitur *rand*() yang merupakan fungsi bawaan dari bahasa pemrograman C++. Waktu dimulainya perpindahan untuk setiap *node* juga diatur secara acak yang dimulai dari detik ke-1. Kode program untuk mengatur pergerakan *node* dapat dilihat pada Kode Sumber 4.16.

$ns\_ at 0.0 "destination"

proc destination {} {

global ns\_ val *node*\_

set time 1.0

set now [$ns\_ now]

for {set i 0} {$i < $val(nn)} {incr i} {

if {$i%3 == 0} {

$ns\_ at $now "$*node*\_($i) setdest [expr rand()\*$val(x)] [expr rand()\*$val(y)] 5.0"

}

if {$i%3 == 1} {

$ns\_ at $now "$*node*\_($i) setdest [expr rand()\*$val(x)] [expr rand()\*$val(y)] 5.0"

}

if {$i%3 == 2} {

$ns\_ at $now "$*node*\_($i) setdest [expr rand()\*$val(x)] [expr rand()\*$val(y)] 5.0"

}

}

$ns\_ at [expr $now+$time] "destination"

}

**Kode Sumber 4.16** Konfigurasi Pergerakan Node

Setelah melakukan konfigurasi posisi, label, warna, dan pergerakan *node* untuk simulasi, langkah selanjutnya adalah mendefinisikan koneksi untuk simulasi, di mana pada tugas akhir ini penulis menggunakan koneksi UDP. Dalam pengaturan koneksi UDP ini, ditentukan juga ukuran paket sebesar 512 *bytes* dengan interval waktu pengiriman sebesar 0,25 detik dan waktu simulasi yang dimulai pada detik ke 2,5 hingga detik ke 200. Kode program konfigurasi koneksi UDP dapat dilihat pada Kode Sumber 4.17.

**Kode Sumber 4.17** Konfigurasi Koneksi Simulasi

#Setup a UDP connection

set udp0 [new Agent/UDP]

$ns\_ attach-agent $*node*\_($val(source)) $udp0

set null0 [new Agent/Null]

$ns\_ attach-agent $*node*\_($val(destination)) $null0

set cbr0 [new Application/Traffic/CBR]

$cbr0 set packetSize\_ 512

$cbr0 set interval\_ 0.25

$cbr0 set random\_ 1

$cbr0 set maxpkts\_ 10000

$cbr0 attach-agent $udp0

$ns\_ connect $udp0 $null0

$ns\_ at 2.5 "$cbr0 start"

$ns\_ at [expr $val(stop)] "$cbr0 stop"

Langkah selanjutnya adalah melakukan pengaturan untuk menyimpan hasil simulasi dalam bentuk .tr dan .nam. Kode program pengaturan untuk menyimpan hasil simulasi dapat dilihat pada Kode Sumber 4.18.

**Kode Sumber 4.18** Konfigurasi Menyimpan Hasil Simulasi

proc finish {} {

global ns\_ trfile namtrace

$ns\_ flush-trace

close $namtrace

exec nam file\_name.nam &

exit 0

}

Setelah melakukan konfigurasi menyimpan hasil simulasi, penulis melakukan konfigurasi untuk mengakhiri simulasi. Kode program untuk mengakhiri simulasi dapat dilihat pada Kode Sumber 4.19.

**Kode Sumber 4.19** Konfigurasi Mengakhiri Simulasi

for {set i 0} {$i < $val(nn)} {incr i} {

$ns\_ at $val(stop) "$*node*\_($i) reset";

}

$ns\_ at $val(stop) "$ns\_ nam-end-wireless $val(stop)"

$ns\_ at $val(stop) "stop"

$ns\_ at $val(stop) "finish"

$ns\_ at $val(stop) "puts \"NS EXITING...\"; $ns\_ halt"

proc stop {} {

global ns\_ trfile

$ns\_ flush-trace

close $trfile

}

Implementasi simulasi pada dokumenTCL telah selesai. Untuk menjalankan skrip TCL, digunakan perintah yang dijalankan di terminal seperti pada Kode Sumber 4.20.

$ ns namafile.tcl

$ nam namafile.nam

**Kode Sumber 4.20** Perintah Menjalankan Simulasi TCL

Ketika skenario selesai dijalankan, maka akan dihasilkan dua dokumen, yaitu dokumen berekstensi .tr sebagai dokumen *trace route*, dan dokumen berekstensi .nam sebagai *network animator*.

## Implementasi Metrik Analisis

Simulasi yang telah dijalankan oleh NS-2 menghasilkan keluaran berupa *trace file* yang berisikan data mengenai aktivitas yang terjadi selama simulasi dalam bentuk *plain text* berekstensi .tr. Dari data tersebut, dapat dilakukan analisis performa *routing protocol* ZRPdengan mengukur beberapa metrik. Pada tugas akhir ini, metrik yang akan dianalisis adalah *Packet Delivery Ratio* (PDR), *Routing Overhead* (RO), dan *Delivery Delay* (DD).

### Implementasi *Packet Delivery Ratio*

Pada subbab 3.6.1, telah ditunjukkan contoh struktur data *event* dalam *trace file* oleh NS-2 dan telah dijelaskan cara menghitung PDR. Skrip AWK untuk menghitung PDR dapat dilihat pada lampiran 7.

Dalam menghitung PDR, kata kunci yang perlu diperhatikan pada *trace file* adalah *layer* AGT, karena kata kunci tersebut menunjukkan *event* yang bersangkutan dengan paket komunikasi data. Kemudian menghitung jumlah paket yang dikirimkan dan paket yang diterima dengan menggunakan karakter pada kolom pertama sebagai filter, karena kolom pertama yang menunjukkan *event* yang terjadi dari sebuah paket. Setelah itu, nilai PDR dapat dihitung dengan persamaan yang telah dijelaskan pada subbab 3.6.1. *Pseudocode* untuk menghitung PDR dapat dilihat pada Kode Sumber 4.21.

sent = 0

received = 0

for i = 1 to the number of rows

if in a row contains “s” and AGT then

sent++

else if in a row contains “r” and AGT then

eeceived++

end if

pdr = received / sent

**Kode Sumber 4.21** Pseudocode Menghitung PDR

Penghitungan PDR pada tugas akhir ini dilakukan setelah simulasi dijalankan. Untuk menjalankan skrip AWK PDR menggunakan perintah awk -f pdr.awk *nama\_trace\_file*.tr.

### Implementasi *Routing Overhead*

Pada subbab 3.6.2, telah ditunjukkan contoh struktur data *event* dalam *trace file* oleh NS-2 dan telah dijelaskan cara menghitung RO. Skrip AWK untuk menghitung RO dapat dilihat pada lampiran 8.

Seperti yang telah dijelaskan sebelumnya, RO merupakan jumlah dari *routing* paket kontrol baik itu paket RREQ, paket RREP, maupun paket RERR. Sehingga, untuk mendapatkan *route request* hal yang perlu dilakukan adalah menjumlahkan tiap paket dengan filter *event sent* pada kolom pertama dan *event layer* RTR pada kolom ke-4. *Pseudocode* untuk menghitung RO dapat dilihat pada Kode Sumber 4.22.

ro = 0

for i = 1 to the number of rows

if in a row contains “s” or “f” and RTR then

ro++

end if

**Kode Sumber 4.22** Pseudocode Menghitung RO

Pada tugas akhir ini, penghitungan RO juga dilakukan setelah simulasi dijalankan Untuk menjalankan skrip awk RO menggunakan perintah awk -f ro.awk *nama\_trace\_file*.tr.

### Implementasi *Delivery Delay*

Pada subbab 3.6.3, telah ditunjukkan contoh struktur data *event* dalam *trace file* oleh NS-2 dan telah dijelaskan cara menghitung *Delivery Delay*. Skrip AWK untuk menghitung *Delivery Delay* dapat dilihat pada lampiran 9.

Dalam menghitung *Delivery Delay*, kata kunci yang perlu diperhatikan pada *trace file* adalah *layer* AGT, karena kata kunci tersebut menyatakan bahwa aktivitas terjadi pada layer *agent* atau aplikasi di mana pengiriman sudah berupa paket asli. Selain itu, untuk mengetahui waktu yang dibutuhkan selama pengiriman paket data dapat dilihat pada kolom kedua *trace file,* dengan merujuk pada kolom pertama filter *event* di mana kata kunci *sent* (s) menandakan pengiriman paket dan *received* (r) yang berarti penerimaan paket. Setelah itu nilai *Delay Delivery* dapat dihitung dengan persamaan yang telah dijelaskan pada subbab 3.6.3. *Pseudocode* untuk menghitung *Delivery Delay* dapat dilihat pada Kode Sumber 4.23.

time\_sent = 0

time\_received = 0

for i=1 to the number of rows

if in a row contains “s” and AGT then

time\_sent++

else in a row contains “r” and AGT then

time\_received++

end if

dd = time\_received - time\_sent

**Kode Sumber 4.23** Pseudocode Menghitung Delivery Delay

Pada tugas akhir ini, penghitungan *Delivery Delay* juga dilakukan setelah simulasi dijalankan. Untuk menjalankan skrip AWK *Delivery Delay* menggunakan perintah awk -f dd.awk *nama-trace-file*.tr.

***(Halaman ini sengaja dikosongkan)***

# BAB V UJI COBA DAN EVALUASI

Bab ini membahas mengenai uji coba yang dilakukan dan evaluasi sesuai dengan rancangan dan implementasi. Dari hasil yang didapatkan setelah melakukan uji coba, akan dilakukan evaluasi sehingga dapat diambil kesimpulan untuk bab selanjutnya.

## Lingkungan Uji Coba

Uji coba dilakukan pada perangkat dengan spesifikasi seperti yang tertera pada Tabel 5.1.

**Tabel 5.1** Spesifikasi Perangkat yang Digunakan

|  |  |
| --- | --- |
| **Komponen** | **Spesifikasi** |
| **CPU** | Intel(R) Core(TM) i5-7400 CPU @ 3.00GHz 3.00 GHz |
| **Sistem Operasi** | Ubuntu Bionic 18.04 LTS 64 bit |
| **Linux Kernel** | Linux kernel 4.4 |
| **Memori** | RAM 8 GB |
| **Penyimpanan** | 928 GB |

Pengujian dilakukan dengan menjalankan skenario yang disimulasikan menggunakan NS-2. Dari simulasi tersebut dihasilkan sebuah *trace file* dengan ekstensi .tr yang akan dianalisis menggunakan skrip AWK untuk mendapatkan *Packet Delivery Ratio*, *Routing Overhead*, dan *Delivery Delay* dengan kode masing-masing pada lampiran 7. Kode Skrip AWK *Packet Delivery Ratio*, lampiran 8. Kode Skrip AWK *Routing* *Overhead*, dan lampiran 9. Kode Skrip AWK *Delivery Delay*.

## Hasil Uji Coba

Simulasi yang telah dijalankan oleh NS-2 hingga proses analisis menggunakan skrip AWK menghasilkan keluaran berupa nilai metrik analisis. Sebelumnya, pada subbab 3.3 telah dibahas mengenai skenario penentuan *threshold* yang menggunakan *routing protocol* ZRP modifikasi. Pada bagian ini, akan dibahas mengenai hasil pra-uji coba penentuan *threshold* dan hasil uji coba skenario simulasi NS-2 menggunakan *routing protocol* ZRP yang telah dimodifikasi.

### Hasil Pra-Uji Coba Penentuan *Threshold*

Sebelum memasuki hasil uji coba skenario simulasi NS-2, dilakukan terlebih dahulu pra-uji coba penentuan nilai *threshold* sesuai dengan skenario yang telah dirancang pada subbab 3.3. Dalam penentuan *threshold*, parameter lingkungan simulasi diatur sama dengan parameter lingkungan yang telah dijelaskan dalam subbab 3.5. Perbedaannya terletak pada jumlah dan posisi *node*, di mana dalam pra-uji coba penentuan *threshold* ini jumlah *node* yang digunakan adalah 25 *node* serta posisi *node* dibuat statis dan tidak bergerak. Selain itu, pada penentuan *threshold* ini menggunakan *routing protocol* ZRP yang telah dimodifikasi.

Penentuan *threshold* dilakukan dengan menggunakan besar kepadatan *node* tetanggadari 0 sampai batas tertentu, yaitu ketika nilai *Packet Delivery Ratio, Routing Overhead,* dan *Delivery Delay* menghasilkan angka yang sama untuk kepadatan yang berbeda. Simulasi dilakukan satu kali untuk setiap besaran *threshold*. Hal itu dilakukan karena posisi dan pergerakan *node* dalam pra-uji coba ini dibuat statis dan tidak bergerak, sehingga akan menghasilkan nilai metrik analisis yang sama pada besar kepadatan *node* tetangga yang sama.

Dari simulasi pra-uji coba yang telah dilakukan, hasil metrik analisis menunjukkan bahwa pada saat kepadatan *node* tetangga sebesar 0,52, nilai PDR mencapai angka 1. Begitupun dengan metrik analisis untuk kepadatan *node* tetangga hingga 1,00 menunjukkan hasil metrik analisis yang sama untuk nilai PDR, RO, dan DD. Sehingga, hasil pra-uji coba penentuan *threshold* didapatkan angka *threshold* sebesar 0,52. Hasil pra-uji coba penentuan *threshold* selengkapnya dapat dilihat pada Tabel 5.2.

**Tabel 5.2** Hasil Pra-Uji Coba Penentuan Threshold

|  |  |  |  |
| --- | --- | --- | --- |
| *Density* | *Packet Delivery Ratio* | *Routing Overhead* | *Delivery Delay* |
| 0,04 | 0,9975 | 11.613 | 83,5084 |
| 0,08 | 0,9975 | 12.721 | 88,3990 |
| 0,12 | 0,6013 | 10.117 | 194,1120 |
| 0,16 | 0,6146 | 10.243 | 44,4640 |
| 0,20 | 0,6184 | 10.436 | 41,9038 |
| 0,24 | 0,6247 | 10.766 | 41,9407 |
| 0,28 | 0,1043 | 8.449 | 33,5361 |
| 0,32 | 0,6038 | 12.019 | 509,8210 |
| 0,36 | 0,6090 | 12.290 | 52,8119 |
| 0,40 | 0,6112 | 12.529 | 498,0540 |
| 0,44 | 0,3654 | 11.294 | 40,0569 |
| 0,48 | 0,6197 | 13.065 | 42,3226 |
| 0,52 | 1,0000 | 15.052 | 47,6143 |

Dari hasil pra-uji coba penentuan *threshold*, dapat diketahui bahwa ketika *density* atau kepadatan *node* tetangga sebesar 0,52, nilai *Packet Delivery Ratio* menunjukkan angka 1, di mana angka tersebut merupakan nilai tertinggi dari setiap percobaan pada besar kepadatan *node* tetangga yang dilakukan. Sehingga, pada tugas akhir ini 0,52 dipilih sebagai nilai *threshold* kepadatan *node* tetangga. Selanjutnya, nilai *threshold* tersebut digunakan dalam simulasi NS-2 untuk mengetahui performa metrik analisisnya.

### Hasil Uji Coba Simulasi NS-2

Pengujian pada skenario simulasi NS-2 dilakukan untuk melihat perbandingan metrik analisis berupa *Packet Delivery Ratio, Routing Overhead,* dan *Delivery* *Delay* antara *routing protocol* ZRP asli dan ZRP yang telah dimodifikasi. Pengujian dilakukan sesuai dengan perancangan skenario yang telah dijelaskan pada subbab 3.5

Pengambilan data uji metrik analisis dengan luas area 1500 m x 1500 m dan *node* sebanyak 25, 50, dan 100 menggunakan nilai *threshold* yang telah didapatkan pada hasil pra-uji coba yaitu sebesar 0,52. Nilai *threshold* ini bermakna bahwa maksimal *node* di dalam zona IARP yang dimiliki oleh suatu *node* tidak boleh melebihi nilai *threshold*, dan nilai radius minimal adalah 1. Uji coba simulasi NS-2 dilakukan sebanyak sepuluh kali, kemudian dihitung nilai rata-rata untuk setiap parameter metrik analisis yang telah ditentukan.

Hasil pengambilan data metrik analisis berupa *Packet Delivery Ratio* pada *node* 25, 50, dan 100 dengan menggunakan *routing protocol* ZRP asli dan ZRP yang telah dimodifikasi dapat dilihat pada Gambar 5.1.

**Gambar 5.1** Grafik Rata-rata Packet Delivery Ratio pada Simulasi NS-2

Berdasarkan grafik pada Gambar 5.1, dapat dilihat bahwa *routing protocol* ZRP asli dan ZRP yang telah dimodifikasi mengalami kenaikan nilai *Packet Delivery Ratio*. Pada lingkungan dengan jumlah *node* 25, menghasilkan perbedaan selisih PDR sebesar 0,03024, atau naik sekitar 3,16 % di mana *routing protocol* ZRP yang telah dimodifikasi unggul dalam nilai PDR tersebut. Pada lingkungan dengan jumlah *node* 50, menghasilkan perbedaan selisih PDR sebesar 0,01426, atau naik sekitar 1,48 % di mana *routing protocol* ZRP yang telah dimodifikasi juga unggul dalam nilai PDR tersebut. Pada lingkungan dengan jumlah *node* 100, menghasilkan perbedaan selisih PDR sebesar 0,01151, atau naik sekitar 1,99 % di mana *routing protocol* ZRP yang telah dimodifikasi juga unggul dalam nilai PDR tersebut.

Dapat dilihat rata-rata kenaikan yang terjadi untuk nilai PDR adalah 2,21 %. Jika ketiga lingkungan variasi jumlah *node* tersebut dibandingkan, dapat dilihat bahwa dengan jumlah *node* 25 menghasilkan PDR yang lebih bagus daripada di lingkungan dengan jumlah *node* yang lebih banyak untuk *routing protocol* ZRP yang telah dimodifikasi. Sedangkan pada ZRP asli, nilai PDR yang lebih bagus dihasilkan oleh jumlah *node* 50. Dapat dilihat pula bahwa ZRP yang telah dimodifikasi menghasilkan PDR yang lebih bagus atau dalam hal ini lebih tinggi daripada *routing protocol* ZRP asli.

Hasil pengambilan data metrik analisis berupa *Routing Overhead* pada *node* 25, 50, dan 100 dengan menggunakan *routing protocol* ZRP asli dan ZRP yang telah dimodifikasi dapat dilihat pada Gambar 5.2.

**Gambar 5.2** Grafik Rata-rata Routing Overhead pada Simulasi NS-2

Berdasarkan grafik pada Gambar 5.2, dapat dilihat bahwa *routing protocol* ZRP asli dan ZRP yang telah dimodifikasi mengalami kenaikan nilai *Routing Overhead*. Pada lingkungan dengan jumlah *node* 25, menghasilkan perbedaan selisih RO sebesar 1436,2, atau naik sekitar 7,84 % di mana *routing protocol* ZRP yang telah dimodifikasi lebih unggul dalam nilai RO tersebut. Pada lingkungan dengan jumlah *node* 50, menghasilkan perbedaan selisih RO sebesar 7297,8, atau naik sekitar 18,85 % di mana *routing protocol* ZRP yang telah dimodifikasi juga unggul dalam nilai RO tersebut. Pada lingkungan dengan jumlah *node* 100, menghasilkan perbedaan selisih RO sebesar 27688,5, atau naik sekitar 29,28 % di mana *routing protocol* ZRP yang telah dimodifikasi juga unggul dalam nilai RO tersebut.

Dapat dilihat rata-rata kenaikan yang terjadi untuk nilai RO adalah 18,66 %. Jika ketiga lingkungan variasi jumlah *node* tersebut dibandingkan, dapat dilihat bahwa dengan jumlah *node* 25 menghasilkan RO yang lebih bagus atau lebih sedikit daripada lingkungan dengan jumlah *node* yang lebih banyak untuk *routing protocol* ZRP asli maupun yang telah dimodifikasi.

Hasil pengambilan data metrik analisis berupa *Delivery* *Delay* pada *node* 25, 50, dan 100 dengan menggunakan *routing protocol* ZRP asli dan ZRP yang telah dimodifikasi dapat dilihat pada Gambar 5.3.

**Gambar 5.3** Grafik Rata-rata Delivery Delay pada Simulasi NS2

Berdasarkan grafik pada Gambar 5.3, dapat dilihat bahwa *routing protocol* ZRP asli dan ZRP yang telah dimodifikasi mengalami kenaikan nilai *Delivery Delay*. Pada lingkungan dengan jumlah *node* 25, menghasilkan perbedaan selisih *Delivery* *Delay* sebesar 601,22, atau naik sekitar 345 % di mana *routing protocol* ZRP yang telah dimodifikasi lebih unggul dalam nilai *Delivery* *Delay* tersebut. Pada lingkungan dengan jumlah *node* 50, menghasilkan perbedaan selisih *Delivery* *Delay* sebesar 652,26, atau naik sekitar 420 % di mana *routing protocol* ZRP yang telah dimodifikasi juga unggul dalam nilai *Delivery* *Delay* tersebut. Pada lingkungan dengan jumlah *node* 100, menghasilkan perbedaan selisih *Delivery* *Delay* sebesar 1688,46, atau naik sekitar 205 % di mana *routing protocol* ZRP yang telah dimodifikasi juga unggul dalam nilai *Delivery* *Delay* tersebut.

Dapat dilihat rata-rata kenaikan yang terjadi untuk nilai *Delivery* *Delay* adalah 323 %. Jika ketiga lingkungan variasi jumlah *node* tersebut dibandingkan, dapat dilihat bahwa dengan jumlah *node* 25 menghasilkan *Delivery* *Delay* yang lebih bagus atau lebih sedikit daripada lingkungan dengan jumlah *node* yang lebih banyak untuk *routing protocol* ZRP yang telah dimodifikasi. Sedangkan untuk *routing protocol* ZRP asli, nilai *Delivery* *Delay* yang lebih bagus dihasilkan oleh *node* 50.

# BAB VI KESIMPULAN DAN SARAN

Bab ini membahas mengenai kesimpulan yang diperoleh dari tugas akhir yang telah dikerjakan dan saran terkait pengembangan dari tugas akhir ini yang dapat dilakukan pada masa yang akan datang.

## Kesimpulan

Kesimpulan yang diperoleh dari hasil uji coba dan evaluasi pada tugas akhir ini adalah sebagai berikut:

1. Dengan menambahkan penghitungan kepadatan *node* tetangga dan membandingkannya dengan nilai *threshold*, maka radius yang adaptif terhadap kepadatan *node* tetangga dapat diimplementasikan.
2. Pengaruh zonasi yang adaptifterhadap performa *routing protocol* ZRP secara keseluruhan untuk skenario simulasi NS-2 menghasilkan peningkatan rata-rata *Packet Delivery Ratio* sebesar 2,21 %, peningkatan rata-rata *Routing Overhead* sebesar 18,66%, dan peningkatan rata-rata *Delivery Delay* sebesar 323%.

## Saran

Saran yang diberikan dari hasil uji coba dan evaluasi pada tugas akhir ini adalah sebagai berikut:

1. Untuk mendapatkan hasil uji coba yang lebih baik, dapat dilakukan uji coba yang lebih banyak, misalnya sepuluh kali.
2. Untuk mendapatkan hasil uji coba yang lebih baik, dapat dilakukan penentuan posisi dan pergerakan *node* yang lebih teratur atau dengan kata lain semi-acak.
3. Ide untuk implementasi zonasi yang adaptif terhadap kepadatan *node* tetanggasudah bagus, kedepannya bisa mengimplementasikan dengan menambahkan aspek lain yang dijadikan untuk parameter zonasi yang adaptif seperti kecepatan, arah, dan lain sebagainya.

***(Halaman ini sengaja dikosongkan)***

# DAFTAR PUSTAKA

|  |  |
| --- | --- |
| [1] | S. Adiwicaksono, "DETEKSI MALICIOUS *NODE* PADA ZONE *ROUTING*," Teknik Informatika, Institut Teknologi Sepuluh Nopember, Surabaya, 2017. |
| [2] | M. F. Rahman, K. S. N. Ripon, S. M. M. Karim and M. I. H. Suvo, "Performance Analysis of Estimation of Distribution," *International Journal of Computer Science and Information Security,* vol. 8, August 2010. |
| [3] | A. Nasipuri, "Mobile Ad Hoc Networks," The University of North Carolina, Charlotte. |
| [4] | N. Beijar, "Zone *Routing* Protocol (ZRP)," Finland, 2015. |
| [5] | J. Sasongko, "Network Simulator dan Network Animator Menggunakan Cygnus Windows dalam Windows XP," vol. XIV, Januari 2009. |
| [6] | Y. Joshi and D. Parekh, 2016. [Online]. Available: https://academic.csuohio.edu/yuc/mobile/mcproj/5d-MC%20Final%20Report%20.pdf. [Accessed 19 October 2018]. |
| [7] | P. R. L. D. Neha Singh and V. Mathur, "Network Simulator NS2-2.35," *International Journal of Advanced Research in Computer Science and Software Engineering,* vol. 2, May 2012. |
| [8] | A. V. Aho, B. W. Kerninghan and P. J. Weinberger, The AWK Programming Language, Boston: Addison-Wesley, 1988. |
| [9] | Surateno, "OPTIMASI PENENTUAN ZONA PADA *ROUTING PROTOCOL* HOPNET DENGAN TEKNIK MIN-SEARCHING," Pasca Sarjana Teknik Informatika, Intitut Teknologi Sepuluh Nopember, Surabaya, 2011. |

***(Halaman ini sengaja dikosongkan)***

# LAMPIRAN

## Kode Konfigurasi Kelas *NDPAgent*

class NDPAgent{

public:

ZRPAgent \*agent\_;

NeighborList neighborLst\_;

NDPBeaconTransmitTimer BeaconTransmitTimer\_;

NDPAckTimer AckTimer\_;

int startup\_jitter\_;

NDPAgent(ZRPAgent \*agent) : agent\_(agent), BeaconTransmitTimer\_(agent),

AckTimer\_(agent), startup\_jitter\_(DEFAULT\_STARTUP\_JITTER) {}

void startUp();

void recv\_NDP\_BEACON(Packet\* p);

void recv\_NDP\_BEACON\_ACK(Packet\* p);

int print\_tables();

}

## Kode Konfigurasi Kelas *IARPAgent*

class IARPAgent{

public:

ZRPAgent \*agent\_;

int updateSendFlag\_;

LinkStateList lsLst\_;

Peripheral*Node*List pnLst\_;

InnerRouteList irLst\_;

IARPUpdateDetectedList upLst\_;

int linkLifeTime\_;

int updateLifeTime\_;

IARPPeriodicUpdateTimer PeriodicUpdateTimer\_;

IARPExpirationTimer ExpirationTimer\_;

int startup\_jitter\_;

IARPAgent(ZRPAgent \*agent) : agent\_(agent), updateSendFlag\_(TRUE),

linkLifeTime\_(DEFAULT\_LINK\_LIFETIME),

updateLifeTime\_(DEFAULT\_UPDATE\_LIFETIME),PeriodicUpdateTimer\_(agent),ExpirationTimer\_(agent),startup\_jitter\_(DEFAULT\_STARTUP\_JITTER){}

void startUp();

void build*Routing*Table();

void addRouteInPacket(nsaddr\_t dest,Packet \*p);

void recv\_IARP\_UPDATE(Packet\* p);

void recv\_IARP\_DATA(Packet\* p);

int count\_*node*\_in\_rad();

};

## Kode Konfigurasi *constant.h*

#ifndef \_constants\_h\_

#define \_constants\_h\_

#define DEBUG 1

#define ROUTER\_PORT 0xff

#define TRUE 1

#define FALSE 0

#define LINKUP 1

#define LINKDOWN 0

#define DEFAULT\_STARTUP\_JITTER 1

#define DEFAULT\_EXPIRATION\_CHECK\_PERIOD 1

#define MAX\_SEQUENCE\_ID 1000000

#define MIN\_PACKET\_DROP\_TIME 3

#define MAX\_PACKET\_DROP\_TIME 10

#define IERP\_TTL 50

#define DEFAULT\_BEACON\_PERIOD 3

#define DEFAULT\_BEACON\_PERIOD\_JITTER 1

#define DEFAULT\_NEIGHBOR\_ACK\_TIMEOUT 2

#define DEFAULT\_MAX\_ACK\_TIMEOUT 2

#define DEFAULT\_MIN\_IARP\_UPDATE\_PERIOD 3

#define DEFAULT\_MAX\_IARP\_UPDATE\_PERIOD 9

#define DEFAULT\_LINK\_LIFETIME 10

#define DEFAULT\_UPDATE\_LIFETIME 30

#define DEFAULT\_BRP\_XMIT\_POLICY 0

#define IERP\_REPLY\_SNOOP 1

#define IERP\_ERROR\_SNOOP 1

#define IERP\_XMIT\_JITTER 1

#define DEFAULT\_QUERY\_LIFETIME 30

#define DEFAULT\_QUERY\_RETRY\_TIME 5

#define DEFAULT\_ROUTE\_LIFETIME 120

#define DEFAULT\_MAX\_IERP\_REPLY 3

#define ZRP\_DEFAULT\_HDR\_LEN 10

#define DEFAULT\_ZONE\_RADIUS 2

#define DEFAULT\_SEND\_BUFFER\_SIZE 100

#define DEFAULT\_PACKET\_LIFETIME 20

#define DEFAULT\_INTERPKT\_JITTER 1

typedef double Time;

typedef int32\_t Query\_ID;

enum ZRPTYPE {

NDP\_BEACON, NDP\_BEACON\_ACK, IARP\_UPDATE, IARP\_DATA, IERP\_REPLY, IERP\_REQUEST, IERP\_ROUTE\_ERROR, IERP\_DATA

};

enum BRP\_XMIT\_POLICY {

BRP\_UNICAST, BRP\_MULTICAST

};

#endif

## Kode Fungsi NDPAckTimer::handle(Event\* e)

void NDPAckTimer::handle(Event\* e) {

if((agent\_->ndpAgt\_) .neighborLst\_ .isEmpty()){

if(DEBUG) {

Time now = Scheduler::instance() .clock();

printf("\n\_%d\_ [%6.6f] | NDP\_ISOLATED\_*NODE* | No NDP\_BEACON\_ACK Detected | ",agent\_->myaddr\_, now); agent\_->print\_tables(); printf("\n");

}return;

}

Time now = Scheduler::instance() .clock();

Neighbor \*prev, \*curNb;

prev = NULL;

curNb = (agent\_->ndpAgt\_).neighborLst\_ .head\_;

for(; curNb!=NULL; ) {

if((now - curNb->lastack\_) > neighbor\_ack\_timeout\_) {

curNb->AckTOCount\_++;

if(curNb->AckTOCount\_ >= DEFAULT\_MAX\_ACK\_TIMEOUT) {

(agent\_->iarpAgt\_). updateSendFlag\_ = TRUE;

Linkstate \*handleToFoundLS = NULL;

int foundFlag;

foundFlag = (agent\_->iarpAgt\_).lsLst\_.findLink(curNb->addr\_, agent\_->myaddr\_, &handleToFoundLS);

if( foundFlag == TRUE ) {

handleToFoundLS->isup\_ = LINKDOWN;

(agent\_->iarpAgt\_) .build*Routing*Table();}

curNb->linkStatus\_ = LINKDOWN;

}

if(DEBUG) {

Time now = Scheduler::instance() .clock();

printf("\n\_%d\_ [%6.6f] | NDP\_BEACON\_ACK\_TIMEOUT | -S %d -Nb %d | -TimeOut %d | ",agent\_->myaddr\_, now, agent\_->myaddr\_, curNb->addr\_, neighbor\_ack\_timeout\_);

agent\_->print\_tables(); printf("\n");

}

}

prev = curNb;

curNb=curNb->next\_;

}

if (agent\_->radius\_==1) {

int num\_neighbor = (agent\_-> ndpAgt\_) .print\_tables();

float density = (float)num\_neighbor / jumlah\_*node*\_simulasi;

float th = nilai\_threshold;

printf("\nNDP UPDATED FOR *NODE* NUMBER : %d, radius : %d, kepadatan : %f, num\_neighbor : %d\n\n", agent\_->myaddr\_, agent\_->radius\_, density, num\_neighbor);

if (density < th && num\_neighbor > 0){

agent\_->radius\_ +=1;

}

}

}

## Kode Fungsi IARPPeriodicUpdateTimer::handle (Event\* e)

void IARPPeriodicUpdateTimer::handle(Event\* e){

if(agent\_->radius\_ == 1) {

Scheduler::instance().schedule(this, &intr\_, min\_iarp\_update\_period\_);

return;

}

Time now = Scheduler::instance().clock();

if((agent\_->iarpAgt\_) .updateSendFlag\_ ==FALSE && (now-lastUpdateSent\_) <max\_iarp\_update\_period\_) {

if(DEBUG) {

Time now = Scheduler::instance() .clock();

printf("\n\_%d\_ [%6.6f] | MISS\_IARP\_UPDATE | No Neighbor Changes | -Tlsu %d | ",agent\_->myaddr\_, now, min\_iarp\_update\_period\_);

agent\_->print\_tables();

printf("\n");

}

Scheduler::instance().schedule(this, &intr\_, min\_iarp\_update\_period\_);

return;

}

if((agent\_->ndpAgt\_).neighborLst\_.isEmpty() == FALSE) {

Packet\* p;

p = (agent\_->pktUtil\_). pkt\_create (IARP\_UPDATE, IP\_BROADCAST, agent\_->radius\_-1);

(agent\_->pktUtil\_).pkt\_add\_LSU\_space(p, (agent\_->ndpAgt\_) .neighborLst\_ .numNeighbors\_);

hdr\_zrp \*hdrz = HDR\_ZRP(p);

Neighbor \*cur = (agent\_->ndpAgt\_). neighborLst\_.head\_;

for(int i=0; i<(agent\_->ndpAgt\_) .neighborLst\_.numNeighbors\_; i++) {

hdrz->links\_[i].src\_ = agent\_->myaddr\_;

hdrz->links\_[i].dest\_ = cur->addr\_;

hdrz->links\_[i].isUp\_ = cur->linkStatus\_;

cur = cur->next\_;}

hdrz->numlinks\_ = (agent\_->ndpAgt\_). neighborLst\_.numNeighbors\_;

(agent\_->pktUtil\_).pkt\_broadcast(p, 0.00);

if(DEBUG) {

Time now = Scheduler::instance() .clock();

hdr\_zrp \*hdrz = HDR\_ZRP(p);

hdr\_ip \*hdrip = HDR\_IP(p);

printf("\n\_%d\_ [%6.6f] | XMIT\_IARP\_UPDATE | -S %d -IS %d | -SEQ %d | ",agent\_->myaddr\_, now, hdrz->src\_, hdrip->saddr(), hdrz->seq\_);

(agent\_->pktUtil\_) .pkt\_print\_links(p);

agent\_->print\_tables(); printf("\n");}

(agent\_->iarpAgt\_).upLst\_.addUpdate (hdrz->src\_, hdrz->seq\_,now+(agent\_->iarpAgt\_).updateLifeTime\_);

(agent\_->iarpAgt\_).updateSendFlag\_ = FALSE;

lastUpdateSent\_ = now;

(agent\_->ndpAgt\_).neighborLst\_ .purgeDownNeighbors();}

else{

if(DEBUG) {

Time now = Scheduler::instance().clock();

printf("\n\_%d\_ [%6.6f] | IARP\_ISOLATED\_*NODE* | No Neighbors | ",agent\_->myaddr\_, now);

agent\_->print\_tables(); printf("\n");

}

}

int *node*\_in\_rad = (agent\_-> iarpAgt\_) .count\_*node*\_in\_rad()-1;

float kepadatan = (float)*node*\_in\_rad / jumlah\_*node*\_simulasi;

float treshold = nilai\_threshold;

printf("\nIARP UPDATED FOR *NODE* NUMBER : %d, radius : %d, kepadatan : %f, *node*\_in\_rad : %d\n\n", agent\_->myaddr\_, agent\_->radius\_, kepadatan, *node*\_in\_rad);

if(kepadatan < treshold && agent\_->radius\_ < radius\_maksimal && agent\_->radius\_ > 1 && *node*\_in\_rad > 0){

agent\_->radius\_ +=1;

}

if(kepadatan>treshold && agent\_->radius\_>1){

agent\_->radius\_ -=1;

}

(agent\_->iarpAgt\_).build*Routing*Table();

Scheduler::instance().schedule(this, &intr\_, min\_iarp\_update\_period\_);

}

## Kode Konfigurasi Posisi *Node* pada Simulasi NS-2

set *node* 0

for {set i 0} {$i < 5 } { incr i } {

set distance 250

$*node*\_($*node*) set X\_ [expr {($distance\*$i) + $distance}]

$*node*\_($*node*) set Y\_ 750.0

$*node*\_($*node*) set Z\_ 0.0

$ns\_ initial\_*node*\_pos $*node*\_($*node*) 50

set *node* [expr $*node*+1]

}

for {set i 0} {$i < 6 } { incr i } {

set distance 250

$*node*\_($*node*) set X\_ [expr {($distance\*$i) + 125}]

$*node*\_($*node*) set Y\_ 925.0

$*node*\_($*node*) set Z\_ 0.0

$ns\_ initial\_*node*\_pos $*node*\_($*node*) 50

set *node* [expr $*node*+1]

}

for {set i 0} {$i < 6 } { incr i } {

set distance 250

$*node*\_($*node*) set X\_ [expr {($distance\*$i) + 125}]

$*node*\_($*node*) set Y\_ 525.0

$*node*\_($*node*) set Z\_ 0.0

$ns\_ initial\_*node*\_pos $*node*\_($*node*) 50

set *node* [expr $*node*+1]

}

for {set i 0} {$i < 3 } { incr i } {

set distance 250

$*node*\_($*node*) set X\_ [expr {($distance\*$i) + 500}]

$*node*\_($*node*) set Y\_ 1150.0

$*node*\_($*node*) set Z\_ 0.0

$ns\_ initial\_*node*\_pos $*node*\_($*node*) 50

set *node* [expr $*node*+1]

}

for {set i 0} {$i < 3 } { incr i } {

set distance 250

$*node*\_($*node*) set X\_ [expr {($distance\*$i) + 500}]

$*node*\_($*node*) set Y\_ 350.0

$*node*\_($*node*) set Z\_ 0.0

$ns\_ initial\_*node*\_pos $*node*\_($*node*) 50

set *node* [expr $*node*+1]

}

## Kode Skrip AWK *Packet Delivery Ratio*

BEGIN {

sendLine = 0;

recvLine = 0;

fowardLine = 0;

}

$0 ~/^s.\* AGT/ {

sendLine ++ ;

}

$0 ~/^r.\* AGT/ {

recvLine ++ ;

}

$0 ~/^f.\* RTR/ {

fowardLine ++ ;

}

END {

printf "cbr s:%d r:%d, r/s Ratio:%.4f, f:%d \n", sendLine, recvLine, (recvLine/sendLine),fowardLine;

}

## Kode Skrip AWK *Routing Overhead*

BEGIN {

rt\_pkts = 0;

}

{

if ($2 >= 101) {

if (($1 == "s" || $1 == "f") && ($4 == "RTR") && ($7 == "DSR")) {

rt\_pkts++;

}

}

}

END {

printf "*Routing* Packets \t= %d \n", rt\_pkts;

}

## Kode Skrip AWK *Delivery Delay*

BEGIN{

sum\_delay = 0; count = 0;

}

{

if ($2 >= 101) {

if($4=="AGT" && $1=="s" && seqno < $6) {

seqno = $6;

}

if($4 == "AGT" && $1 == "s") {

start\_time[$6] = $2;

}

else if(($7 == "cbr") && ($1 == "r")) {

end\_time[$6] = $2;

}

else if($1 == "D" && $7 == "cbr") {

end\_time[$6] = -1;

}

}

}

END {

for(i=0; i<=seqno; i++) {

if(end\_time[i] > 0) {

delay[i] = end\_time[i]-start\_time[i];

count++;}

else {

delay[i] = -1;}

}

for(i=0; i<=seqno; i++) {

if(delay[i] > 0) {

n\_to\_n\_delay=n\_to\_n\_delay+delay[i];

}

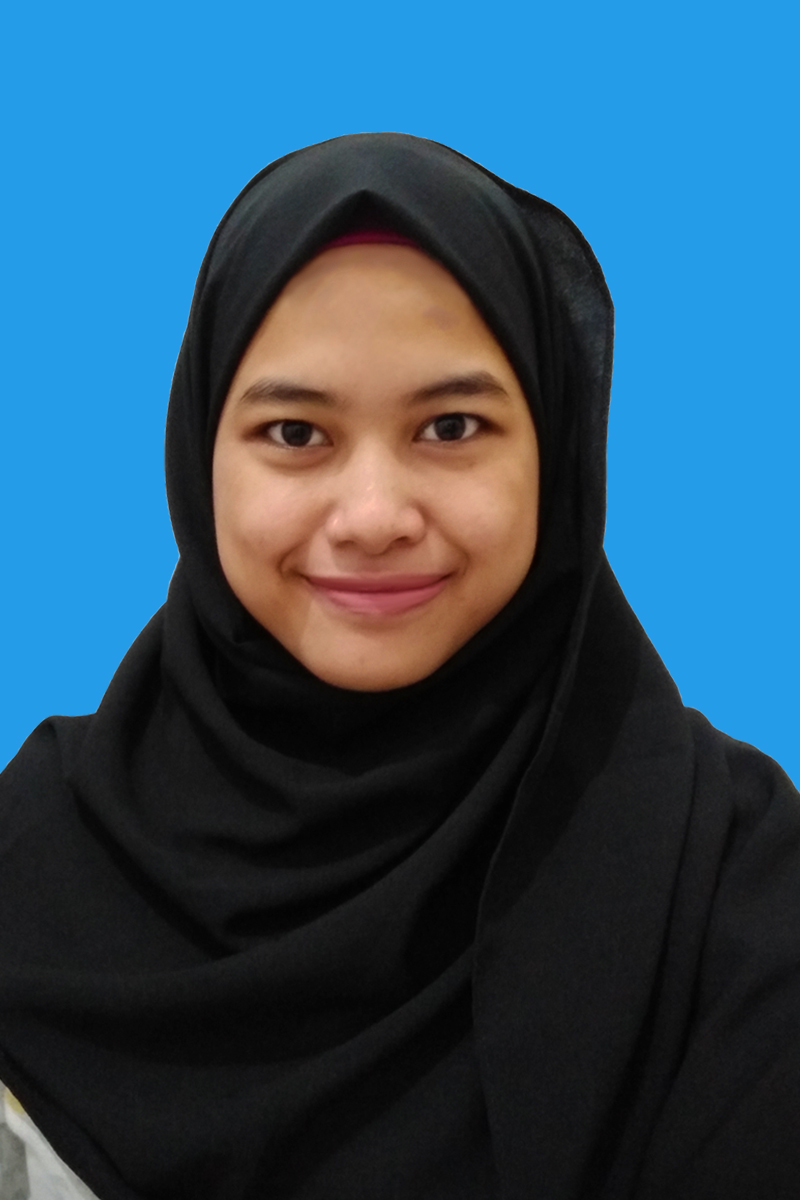
}

n\_to\_n\_delay = n\_to\_n\_delay/count;

printf "End-to-End Delay \t= "n\_to\_n\_delay \* 1000 " ms \n";

}

# BIODATA PENULIS

 Hania Maghfira lahir di Pasuruan pada tanggal 19 Agustus 1997. Penulis menempuh pendidikan formal di TK Bustanul Athfal Purwokerto (2001-2003), SDN Dukuwaluh II Purwokerto (2003-2004), SDN Kebagusan 04 Pg. Jakarta Selatan (2004-2006), SDN Purutrejo II Pasuruan (2006-2007), MI Nurul Huda Gondol Buleleng Bali (2007-2009), MTsN PATAS Buleleng Bali (2009-2012), SMAN 2 Pasuruan (2012-2015), dan Informatika ITS Surabaya (2015-2019). Bidang studi yang diambil oleh penulis saat berkuliah di Departemen Informatika ITS adalah Arsitektur Jaringan Komputer (AJK). Penulis aktif dalam organisasi Badan Eksekutif Mahasiswa Fakultas Teknologi Informasi dan Komunikasi sebagai Wakil Ketua BEM periode 2017-2018, Himpunan Mahasiswa Teknik Computer-Informatika (2016-2017), dan Jamaah Masjid Manarul Ilmi (2016-2018). Penulis juga aktif dalam kegiatan kepanitian seperti SCHEMATICS 2016 - 2017 Divisi Humas, FTIF FESTIVAL 2017 sebagai Bendahara, dan INTEGRALISTIK FESTIVAL 2016 sebagai Sekretaris. Penulis pernah menjalani kerja praktik di PT PLN (Persero) Kantor Pusat Jakarta periode Januari 2018, magang di Kementerian Luar Negeri RI periode Juli-Agustus 2017 dan PT Toyota Astra Motor Jakarta periode Juni-Agustus 2018. Selama berkuliah, penulis juga menjadi administrator di Laboratorium Komputasi Berbasis Jaringan. Penulis dapat dihubungi melalui nomor *handphone* 082139497057 atau di email if.hania@outlook.com.

***(Halaman ini sengaja dikosongkan)***