TUGAS AKHIR – IF184802

PEMUTAKHIRAN ALGORITMA ANT COLONY OPTIMIZATION (ACO) UNTUK PEMILIHAN RUTE PADA ROUTING PROTOCOL AD-HOC ON-DEMAND DISTANCE VECTOR (AODV) DI LINGKUNGAN VANET

SON ARDHYNATA SUKARNO MUDHA
NRP 05111340000107

Dosen Pembimbing I
Dr. Eng. Radityo Anggoro, S.Kom., M.Sc.

Dosen Pembimbing II
Ary Mazharuddin Shiddiqi, S.Kom., M.Comp.Sc., Ph.D.

Departemen Teknik Informatika
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2020

(Halaman ini sengaja dikosongkan)

TUGAS AKHIR – IF184802

PEMUTAKHIRAN ALGORITMA ANT COLONY OPTIMIZATION (ACO) UNTUK PEMILIHAN RUTE PADA ROUTING PROTOCOL AD-HOC ON-DEMAND DISTANCE VECTOR (AODV) DI LINGKUNGAN VANET

SON ARDHYNATA SUKARNO MUDHA
NRP 05111340000107

Dosen Pembimbing I
Dr. Eng. Radityo Anggoro, S.Kom., M.Sc.

Dosen Pembimbing II
Ary Mazharuddin Shiddiqi, S.Kom., M.Comp.Sc., Ph.D.

Departemen Teknik Informatika
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2020

(Halaman ini sengaja dikosongkan)

UNDERGRADUATE THESES – IF184802

THE ENHANCEMENT OF ANT COLONY OPTIMIZATION (ACO) ALGORITHM FOR ROUTES SELECTION ON AD-HOC ON-DEMAND DISTANCE VECTOR (AODV) PROTOCOL IN VANET ENVIRONMENT

SON ARDHYNATA SUKARNO MUDHA
NRP 05111340000107

First Advisor
Dr. Eng. Radityo Anggoro, S.Kom, M.Sc.

Second Advisor
Ary Mazharuddin Shiddiqi, S.Kom., M.Comp.Sc., Ph.D.

Department of Informatics
Faculty of Intelligenent Electrical and Informatics Technology
Sepuluh Nopember Institute of Technology
Surabaya 2020

(Halaman ini sengaja dikosongkan)

LEMBAR PENGESAHAN

Pemutakhiran Algoritma Ant Colony Optimization (ACO) untuk Pemilihan Rute pada Routing Protocol Ad-hoc On-demand Distnce Vector (AODV) di Lingkungan VANET

TUGAS AKHIR
Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Komputasi Berbasis Jaringan
Program Studi S-1 Teknik Informatika
Departemen Teknik Informatika
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember

Oleh:
SON ARDHYNATA SUKARNO MUDHA
NRP: 05111340000107

Disetujui oleh Pembimbing Tugas Akhir:

1. Dr.Eng. Radityo Anggoro, S.Kom., M.Sc. .....
   (NIP. 19841016 200812 1 002) (Pembimbing 1)

2. Ary Mazharuddin Shiddiqi, S.Kom., M.Comp.Sc., Ph.D. .....
   (NIP. 19810620 200501 1 003) (Pembimbing 2)

SURABAYA
JULI, 2020

(Halaman ini sengaja dikosongkan)

PEMUTAKHIRAN ALGORITMA ANT COLONY OPTIMIZATION (ACO) UNTUK PEMILIHAN RUTE PADA ROUTING PROTOCOL AD-HOC ON-DEMAND DISTANCE VECTOR (AODV) DI LINGKUNGAN VANET

Nama Mahasiswa	**:**	**Son Ardhynata Sukarno Mudha**
NRP	**:**	**05111340000107**
Departemen	**:**	**Informatika FTEIC-ITS**
Dosen Pembimbing 1	**:**	**Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.**
Dosen Pembimbing 2	**:**	**Ary Mazharuddin Shiddiqi, S.Kom., M.Comp.Sc., Ph.D.**

Abstrak

Vehicular Ad-hoc Networks (VANET) merupakan pengembangan dari *Mobile Ad-hoc Networks* (MANET) yang memiliki karakteristik *node* dengan mobilitas yang tinggi. Seleksi rute optimal menjadi isu menantang pada *platform* jaringan VANET yang memiliki mobilitas node lebih tinggi. Di sinilah *peran routing protocol* berpengaruh terhadap performa VANET. Salah satu contoh *routing protocol* yang digunakan pada jaringan VANET adalah *Ad-hoc On-demand Distance Vector* (AODV). Meskipun populer, AODV memiliki kelemahan dalam menjaga kestablilan komunikasi antar *node*. Kelemahan AODV ini dapat diatasi dengan melakukan modifikasi pada *routing protocol* ini.

Pada Tugas Akhir ini akan diajukan modifikasi AODV pada mekanisme pencarian rute dengan menggunakan algoritma *Ant Colony Optimization* (ACO) yang dimutakhirkan untuk memilih node tetangga. Node tetangga diseleksi berdasarkan kekuatan sinyal

yang diterima, energi residu pada node, kecepatan rata-rata node, dan jarak node relatif dengan node tujuan. Parameter-parameter tersebut digunakan untuk membangun sebuah fungsi heuristik dalam penghitungan nilai *pheromone* sebuah rute. Rute yang memiliki nilai *pheromone* yang tinggi mengindikasikan rute yang stabil dan optimal yang akan digunakan untuk *packet forwarding*.

Penggunaan algoritma *Ant Colony Optimization* terhadap performa *routing protocol AODV* menunjukkan adanya perubahan yang cukup signifikan pada skenario *real* dengan rata-rata kenaikan PDR sebesar 4,38%, penurunan *End-to-End Delay* sebesar 17,79%, rata-rata penurunan *Routing Overhead* sebesar 28,77%, rata-rata penurunan *Forwarded Route Request* sebesar 28,4% dan rata-rata kenaikan *Throughput* sebesar 3,64%

Kata kunci: VANETs, AODV, ACO, NS-2, Node Tetangga, Pheromone

THE ENHANCEMENT OF ANT COLONY OPTIMIZATION (ACO) ALGORITHM FOR ROUTES SELECTION ON AD-HOC ON-DEMAND DISTANCE VECTOR (AODV) PROTOCOL IN VANET ENVIRONMENT

Student's Name	**:**	**Son Ardhynata Sukarno Mudha**
Student's ID	**:**	**05111340000107**
Department	**:**	**Informatics, Faculty of ELECTICS-ITS**
First Advisor	**:**	**Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.**
Second Advisor	**:**	**Ary Mazharuddin Shiddiqi, S.Kom., M.Comp.Sc., Ph.D.**

Abstract

Vehicular Ad-hoc Networks (VANET) is the development of Mobile Ad-hoc Networks (MANET) which have a characteristic of high mobility nodes. Optimal route selection is a challenging issue on the VANET network platform which has higher mobility nodes. This is where the routing protocol plays a role in VANET performance. One example of the routing protocol used on VANET networks is Ad-hoc On-demand Distance Vector (AODV). Although well-known, AODV has a weakness in maintaining the stability of communication between nodes. This weakness can be overcome by modifying its routing protocol.

This final project will propose the AODV modification on the route discovery using the enhanced Ant Colony Optimization (ACO) algorithm to select the neighboring nodes. The neighboring nodes are selected based on the received signal, the residual energy at the node, the average speed of the node, and the distance of the node

relative to the destination node. These parameters are used to construct a heuristic function in calculating the pheromone value of a route. Routes that have high pheromone values determine the stable and optimal routes to be used for packet forwarding.

The use of Ant Colony Optimization algorithm to the AODV performance routing protocol shows that there is a significant change in the real scenario with an average increase in PDR of 4.38%, an decrease in End-to-End Delay of 17.79%*, an average decrease in Routing Overhead amounted to* 28.77%*, the average decrease in Canal Route Request was* 28.4% *and the average increase in Throughput was 3.64%.*

Keyword: VANETs, AODV, ACO, NS-2, Intermediate Node, Pheromone

KATA PENGANTAR

Puji syukur kepada Allah Yang Maha Esa atas segala karunia dan rahmat-Nya penulis dapat menyelesaikan Tugas Akhir yang berjudul **"Pemutakhiran Algoritma Ant Colony Optimization (ACO) untuk Pemilihan Rute pada Routing Protocol Ad-hoc On-demand Distance Vector (AODV) di Lingkungan VANET".**

Penulis berharap penelitian di dalam buku Tugas Akhir ini dapat bermanfaat bagi pengembangan ilmu pengetahuan saat ini dan masa mendatang.

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

1. Allah SWT atas semua rahmat yang diberikan sehingga penulis dapat menyelesaikan Tugas Akhir ini.
2. Keluarga penulis (Bapak Sukarno, Ibu Emy Supriyati, dan Dheetyas Glibson Rajindra Azizi) yang selalu memberikan dukungan baik berupa doa, moral, dan material yang tak terhingga kepada penulis, sehingga penulis dapat menyelesaikan Tugas Akhir ini.
3. Bapak Dr. Eng. Radityo Anggoro, S.Kom., M.Sc., dan Bapak Ary Mazharuddin Shiddiqi, S.Kom., M.Comp.Sc., Ph.D. selaku dosen pembimbing, atas arahan dan bantuannya dalam pengerjaan Tugas Akhir ini.
4. Sahabat-sahabat terbaikku: M. Alfian Nur, Candra Aditya, Andre Abdirrosyid, Hasan Busri, Vian Rafsanjani, Adib Fuad Firdaus, dan Hani Brilianti Rochmanto atas dukungan moral yang tidak henti-hentinya.
5. Kawan-kawan Alumni SMA Negeri 10 Malang Sampoerna Academy di manapun kalian berada.

6. Asma Khedri dan kawan-kawan Erasmus di Eropa, atas waktu-waktu menyenangkan dalam berbagi cerita, ilmu, dan pemikiran-pemikiran luar biasa.
7. Juga tidak lupa kepada semua pihak yang belum sempat disebutkan satu per satu yang telah membantu penulis dalam menyelesaikan Tugas Akhir ini.

Penulis menyadari bahwa masih terdapat kekurangan, kesalahan, maupun kelalaian yang telah penulis lakukan. Oleh karena itu, saran dan kritik yang membangun sangat dibutuhkan untuk penyempurnaan Tugas Akhir ini.

Surabaya, Juli 2020

Son Ardhynata Sukarno Mudha

DAFTAR ISI

ABSTRAK ..... **VII**
ABSTRACT ..... **IX**
KATA PENGANTAR ..... **XI**
DAFTAR ISI ..... **XIII**
DAFTAR GAMBAR ..... **XVI**
DAFTAR TABEL ..... **XVIII**
BAB I PENDAHULUAN **1**
1.1 LATAR BELAKANG 1
1.2 RUMUSAN MASALAH 3
1.3 BATASAN MASALAH 3
1.4 TUJUAN 3
1.5 MANFAAT 4
1.6 METODOLOGI 4
1.6.1 Penyusunan Proposal Tugas Akhir *4*
1.6.2 Studi Literatur *5*
1.6.3 Analisis dan Desain Perangkat Lunak *5*
1.6.4 Implementasi Perangkat Lunak *5*
1.6.5 Pengujian dan Evaluasi *5*
1.6.6 Penyusunan Buku Tugas Akhir *6*
1.7 SISTEMATIKA PENULSIAN LAPORAN 6
BAB II TINJAUAN PUSTAKA **9**
2.1 VEHICULAR AD-HOC NETWORKS (VANET) 9
2.2 AD-HOC ON-DEMAND DISTANCE VECTOR (AODV) 10
2.3 ANT COLONY OPTIMIZATION (ACO) 12
2.4 *NETWORK SIMULATOR-2* (NS-2) 14
2.5 *OPENSTREETMAP* (OSM) 15
2.6 *JAVA OPENSTREETMAP* (JOSM) 16
2.7 *SIMUALTION OF URBAN MOBILITY* (SUMO) 16
BAB III PERANCANGAN **17**
3.1 DESKRIPSI UMUM 17
3.2 PERANCANGAN SKENARIO *GRID* 20
3.3 PERANCANGAN SKENARIO *REAL* 21
3.4 PERANCANGAN MODIFIKASI *ROUTING PROTOCOL* AODV 23

3.4.1 Perancangan Pengiriman Route Request *27*
3.4.2 Perancangan Ant Colony Optimization (ACO) *28*
3.4.3 Perancangan Pengiriman Route Reply *29*
3.5 PERANCANGAN SIMULASI PADA NS-2 30
3.6 PERANCANGAN METRIK ANALISIS 30
3.6.1 Packet Delivery Rasio (PDR) *30*
3.6.2 Rata-rata End-to-End Delay (E2E) *31*
3.6.3 Routing Overhead (RO) *31*
3.6.4 Forwarded Route Request (RREQ F) *32*
3.6.5 Throughput *32*
BAB IV IMPLEMENTASI 35
4.1 IMPLEMENTASI SKENARIO *GRID* 35
4.2 IMPLEMENTASI SKENARIO *REAL* 39
4.3 IMPLEMENTASI MODIFIKASI *ROUTING PROTOCOL* AODV 42
4.3.1 Implementasi Modifikasi Struktur *43*
4.3.2 Implementasi Inisiasi Nilai Variabel *46*
4.3.3 Implementasi Pengiriman Route Request *48*
4.3.4 Implementasi Ant Colony Optimization (ACO) *49*
4.3.5 Implementasi Pengiriman Route Request *50*
4.4 IMPLEMENTASI SIMULSI PADA NS-2 52
4.5 IMPLEMENTASI METRIK ANALISIS 53
4.5.1 Implementasi Packet Delivery Ratio *53*
4.5.2 Implementasi Rata-rata End-to-End Delay (E2E) *54*
4.5.3 Implementasi Routing Overhead (RO) *55*
4.5.4 Implementasi Forwarded Route Request *56*
4.5.5 Implementasi Throughput *57*
BAB V UJI COBA DAN EVALUASI 59
5.1 LINGKUNGAN UJI COBA 59
5.2 HASIL UJI COBA DAN EVALUASI 60
5.2.1 Hasil Uji Coba dan Evaluasi Skenario Grid *61*
5.2.2 Hasil Uji Coba dan Evaluasi Skenario Real *73*
BAB VI KESIMPULAN DAN SARAN 87
6.1 KESIMPULAN 87
6.2 SARAN 87
DAFTAR PUSTAKA 89

LAMPIRAN.....93
BIODATA PENULIS.....109

DAFTAR GAMBAR

Gambar 3.1 Diagram rancangan simulasi AODV modifikasi 18
Gambar 3.2 Alur perancangan skenario grid 21
Gambar 3.3 Alur perancangan skenario real..... 23
Gambar 3.4 Flow chart modifikasi routing protocol AODV dengan menggunakan algoritma ACO..... 26
Gambar 4.1 Perintah netgenerate..... 35
Gambar 4.2 Hasil Generate peta grid..... 36
Gambar 4.3 Perintah randomTrips.py..... 37
Gambar 4.4 Perintah duarouter..... 37
Gambar 4.5 File skrip .sumocfg..... 38
Gambar 4.6 Perintah SUMO untuk membuat skenario .xml..... 39
Gambar 4.7 Perintah traceExporter.py..... 39
Gambar 4.8 Ekspor peta dari OpenStreetMap 40
Gambar 4.9 Perintah netconvert 40
Gambar 4.10 Hasil konversi peta real..... 41
Gambar 4.11 Potongan modifikasi struktur aodv_paket.h..... 44
Gambar 4.12 Potongan modifikasi struktur aodv_rtable.h 46
Gambar 4.13 Potongan kode inisiasi nilai variabel 47
Gambar 4.14 Potongan modifikasi kode fungsi sendReply()..... 51
Gambar 4.15 Implementasi simulasi NS-2 52
Gambar 4.16 Pseudocode penghitungan PDR..... 54
Gambar 4.17 Pseudocode penghitungan E2E Delay 55
Gambar 4.18 Pseudocode penghitungan Routing Overhead 56
Gambar 4.19 Pseudocode penghitungan RREQ F 57
Gambar 4.20 Pseudocode penghitungan Throughput..... 58
Gambar 5.1 Grafik PDR skenario grid 63
Gambar 5.2 Grafik E2E Delay skenario grid..... 65
Gambar 5.3 Grafik RO skenario grid..... 67

Gambar 5.4 Grafik RREQ F skenario grid 69
Gambar 5.5 Grafik Throughput skenario grid 71
Gambar 5.6 Grafik PDR skenario real 75
Gambar 5.7 Grafik E2E Delay skenario real 77
Gambar 5.8 Grafik RO skenario real 79
Gambar 5.9 Grafik RREQ F skenario real 81
Gambar 5.10 Grafik Throughput skenario real 83

DAFTAR TABEL

Tabel 3.1 Daftar istilah 19
Tabel 3.2 Struktur paket RREQ 24
Tabel 3.3 Struktur paket RREP 24
Tabel 3.4 Struktur routing table 24
Tabel 5.1 Spesifikasi perangkat 59
Tabel 5.2 Lingkungan uji coba 60
Tabel 5.3 Hasil rata-rata PDR skenario grid 61
Tabel 5.4 Hasil rata-rata E2E Delay skenario grid 62
Tabel 5.5 Hasil rata-rata RO skenario grid 62
Tabel 5.6 Hasil rata-rata RREQ F skenario grid 62
Tabel 5.7 Hasil rata-rata Throughput skenario grid 62
Tabel 5.8 Hasil rata-rata PDR skenario real 73
Tabel 5.9 Hasil rata-rata E2E Delay skenario real 74
Tabel 5.10 Hasil rata-rata RO skenario real 74
Tabel 5.11 Hasil rata-rata RREQ F skenario real 74
Tabel 5.12 Hasil rata-rata Throughput skenario real 74

BAB I
PENDAHULUAN

1.1 Latar Belakang

Kebutuhan informasi di era teknologi kian berkembang seiring waktu. Berkat dukungan perangkat keras, perangkat lunak dan teknologi komunikasi yang beragam, infrastruktur jaringan dapat merambah ke segala bidang dan aspek kehidupan manusia, sehingga memudahkan kita untuk mendapatkan informasi dengan mudah, cepat, di mana pun dan kapan pun. Perkembangan infrastruktur jaringan telah memajukan banyak bidang tidak terkecuali bidang transportasi. Dikembangkan dari *Mobile Ad-Hoc Network* (MANET) yang memanfaatkan perangkat bergerak genggam nirkabel sebagai *node* komunikasi, *Vehicular Ad-Hoc Network* (VANET) mengadopsi konsep MANET untuk memanfaatkan berbagai macam moda transportasi sebagai *node* komunikasi. Tentunya, penerapan VANET ini bukan tanpa tantangan. Tingkat mobilitas *node* yang tinggi menyebabkan perubahan topologi yang sangat dinamis, sehingga diperlukan strategi *routing* yang tepat dan stabil agar sebuah data dapat tersampaikan dari *source node* ke *destination node* dengan baik dan tetap di dalam jangkauan sinyal transmisi perangkat-perangkat yang ada dalam topologi tersebut.

Seperti teknologi jaringan lainnya, kerja VANET didefinisikan oleh protokol yang mengatur lalu lintas data pada sebuah topologi jaringan. *Routing protocol* merupakan protokol terpenting yang memungkinkan terjadinya komunikasi antar *node*. Protokol ini dibedakan menjadi dua berdasarkan cara pembentukan rutenya, yakni protokol proaktif dan protokol reaktif. Protokol proaktif, seperti *Destinantion Sequenced*

Distance Vector (DSDV) dan *Optimized Link State Routing* (OLSR), bekerja dengan cara membuat sebuah *routing table* untuk menyimpan *state* sebuah node dan rute, yang diperbarui tiap selang waktu tertentu tanpa memperhatikan beban jaringan, *bandwidth*, dan ukuran jaringan. Di sisi lain, protokol reaktif, seperti *Ad-Hoc On-Demand Distance Vector* (AODV) dan *Dynamic Source Routing* (DSR), bekerja dengan membentuk *routing table* hanya jika ada permintaan pengiriman data atau ketika terjadi perubahan rute.

Pada Tugas Akhir ini, diusulkan sebuah modifikasi pada protokol AODV untuk meningkatkan *Quality of Service* (QoS) pada lingkungan VANET. Modifikasi difokuskan pada mekanisme pemilihan rute (*routing discovery*) menggunakan algortima *Ant Colony Optimization* (ACO) untuk memilih *node* perantara yang memiliki konektivitas yang stabil. Modifikasi ini juga ditujukan untuk memutakhirkan algoritma ACO yang sebelumnya telah diajukan [1]. Pada [1], pemilihan node perantara dilakukan dengan mempertimbangkan nilai *pheromone* sebuah rute yang didapatkan dengan menggunakan *heuristic function* yang disebut *pheromone counter* (PC) yang mengambil parameter kekuatan sinyal yang diterima, energi residu pada *node* tetangga, *congestion* pada *node* tetangga dan *hop count* pada *node* tetangga [2]. Pada Tugas Akhir, modifikasi akan dilakukan pada fungsi tersebut dengan penambahan dua parameter yakni kecepatan rata-rata *node* tetangga dan arah *node* tetangga. Hasil akhir yang diharapkan adalah mengetahui perbandingan kinerja AODV, AODV dengan ACO, dan AODV dengan ACO yang dimutakhirkan. Pengukuran performa dilakukan dengan mengetahui performa *Packet Delivery Ratio* (PDR), *End-to-End Delay, Routing Overhead* (RO), jumlah *forwarded* RREQ, dan *Throughput*.

1.2 Rumusan Masalah

Berikut beberapa hal yang menjadi rumusan masalah pada Tugas Akhir ini:

1. Bagaimana membatasi jumlah forwarding node dalam proses pengiriman Rebroadcast Route Request (RREQ) di lingkungan VANET?

2. Bagaimana dampak pemutakhiran algoritma *Ant Colony Optimization* yang dimutakhirkan terhadap performa protokol AODV secara keseluruhan diukur berdasarkan *Packet Delivery Ratio* (PDR), *End-to-End Delay*, *Routing Overhead* (RO), jumlah *forwarded RREQ*, dan *Throughput* ?

1.3 Batasan Masalah

Batasan masalah pada Tugas Akhir ini adalah sebagai berikut:

1. Jaringan yang digunakan adalah jaringan *Vehicular Ad hoc Networks* (VANETs).

2. *Routing protocol* yang diujicoba yakni AODV.

3. Simulasi pengujian jaringan menggunakan *Network Simulator* 2 (NS-2).

4. Pembuatan skenario uji coba menggunakan *Simulation of Urban Mobility* (SUMO).

1.4 Tujuan

Tujuan dari tugas akhir ini adalah sebagai berikut:

1. Mereduksi jumlah *forwarding node* yang bertanggung jawab untuk meneruskan pesan *Route Request* (RREQ) di lingkungan VANETs.

2. Menganalisis performa Algoritma *Ant Colony Optimization* pada AODV yang telah dimutakhirkan berdasarkan matriks berdasarkan *Packet Delivery Ratio* (PDR), *End-to-End Delay*, *Routing Overhead* (RO), jumlah *forwarded RREQ*, dan *Throughput*.

1.5 Manfaat

Dengan dibuatnya Tugas Akhir ini akan memberikan sebuah manfaat dalam mengetahui dampak performa AODV terhadap modifikasi dan pemutakhiran yang dilakukan.

1.6 Metodologi

Pembuatan Tugas Akhir ini dilakukan dengan menggunakan metodologi sebagai berikut:

1.6.1 Penyusunan Proposal Tugas Akhir

Proposal tugas akhir ini berisi tentang deskripsi pendahuluan dari tugas akhir yang akan dibuat. Pendahuluan ini terdiri atas hal yang menjadi latar belakang diajukannya usulan tugas akhir, rumusan masalah yang diangkat, batasan masalah untuk tugas akhir, tujuan dari pembuatan tugas akhir, dan manfaat dari hasil pembuatan tugas akhir. Selain itu, dijabarkan pula tinjauan pustaka yang digunakan sebagai referensi pendukung pembuatan tugas akhir. Metodologi berisi penjelasan mengenai tahapan penyusunan tugas akhir mulai dari penyusunan proposal hingga penyusunan buku tugas akhir. Terakhir dijabarkan pula jadwal kegiatan yang menjelaskan jadwal pengerjaan tugas akhir ini.

1.6.2 Studi Literatur

Pada studi literatur ini, akan dipelajari sejumlah referensi yang diperlukan dalam melakukan implementasi yaitu mengenai NS-2 Network Simulator, *Vehicular Ad-hoc Networks* (VANETs), dan *reactive routing protocol* AODV.

1.6.3 Analisis dan Desain Perangkat Lunak

Pada tahap ini dilakukan analisis dari hasil percobaan modifikasi reactive routing protocol AODV dan pemutakhiran algoritma ACO yang dibuat. Data yang dianalisis berasal dari penghitungan *Packet Delivery Ratio* (PDR), *End-to-End Delay*, *Routing Overhead* (RO), jumlah *forwarded RREQ*, dan *Throughput*..

1.6.4 Implementasi Perangkat Lunak

Pada tahap ini dilakukan perancangan implementasi model jaringan yang dibuat berupa modifikasi dari protokol AODV agar mendapatkan rute yang stabil dan optimal.

1.6.5 Pengujian dan Evaluasi

Pengujian dilakukan dengan menjalankan simulasi yang dihasilkan oleh SUMO menggunakan NS-2 Network Simulator yang nantinya akan menghasilkan sebuah *trace file*. Dari *trace file* tersebut akan dilakukan penghitungan *Packet Delivery Ratio* (PDR), *End-to-End* Delay, *Routing Overhead* (RO), jumlah *forwarded* RREQ, dan Throughput, untuk menguji performa *routing protocol* yang telah dimodifikasi.

1.6.6 Penyusunan Buku Tugas Akhir

Pada tahap ini dilakukan penyusunan buku yang menjelaskan seluruh konsep, dasar teori dari metode yang digunakan, implementasi serta hasil dari modifikasi yang telah didapatkan sebagai dokumentasi dari pelaksanaan Tugas Akhir.

1.7 Sistematika Penulsian Laporan

Sistematika penulisan laporan Tugas Akhir adalah sebagai berikut:

1. Bab I. Pendahuluan

   Bab ini berisikan penjelasan mengenai latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, metodologi, dan sistematika penulisan laporan Tugas Akhir.

2. Bab II. Tinjuan Pustaka

   Bab ini berisi kajian teori atau penjelasan dari metode, algoritma, *library*, dan *tools* yang digunakan dalam penyusunan Tugas Akhir ini. Bab ini berisi tentang penjelasan singkat mengenai, VANET, AODV, *Network Simulator 2* (NS-2), *OpenStreetMap*, *Java OpenStreetMap* (JOSM), *Simulation of Urban Moubility* (SUMO), dan AWK

3. Bab III. Perancangan

   Bab ini berisi pembahasan mengenai perancangan skenario mobilitas grid dan real, perancangan simulasi

pada NS2, perancangan modifikasi AODV, serta perancangan metrik analisis berupa Packet Delivery Ratio (PDR), rata-rata End-to-End Delay (E2E), Routing Overhead, Forwarded RREQ, dan Throughput.

4. Bab IV. Implementasi

   Bab ini menjelaskan implementasi yang berbentuk kode sumber dari proses modifikasi routing protocol AODV, pembuatan simulasi pada NS2, SUMO, dan penghitungan metrik analisis.

5. Bab V. Uji Coba dan Evaluasi

   Bab ini berisikan hasil uji coba dan evaluasi dari implementasi yang telah dilakukan untuk menyelesaikan masalah yang dibahas pada Tugas Akhir.

6. 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 solusi ke depannya.

7. Daftar Pustaka

   Bab ini berisi daftar pustaka yang dijadikan literatur dalam Tugas Akhir.

8. Lampiran

   Dalam lampiran terdapat tabel-tabel data hasil uji coba dan kode sumber program secara keseluruhan.

(Halaman ini sengaja dikosongkan)

BAB II
TINJAUAN PUSTAKA

2.1 Vehicular Ad-Hoc Networks (VANET)

Vehicular Ad-hoc Network (VANET) adalah sebuah jaringan komunikasi nirkabel berbasis kendaraan yang merupakan pengembangan dari *Mobile Ad-hoc Network* (MANET). VANET umumnya digunakan di *Intelligent Transportation System* (ITS) untuk keamanan pengendara [3]. Bentuk pertukaran informasi di VANET memungkinkan sistem ini untuk diaplikasikan pada bidang keamanan maupun non-keamanan. Secara umum, komunikasi di VANET dapat dikategorikan menjadi dua jenis: komunikasi *vehicle-to-vehicle* (V2V) dan komunikasi *vehicle-to-infrastructure* (V2I) [4, 5]. Terlepas dari jenis komunikasi yang digunakan, *medium access control* (MAC) *layer* tetap menjadi acuan fundamental untuk mengakses medium transmisi nirkabel (seperti Wi-Fi atau jaringan seluler) agar kendaraan atau *node* dapat melakukan *sharing* dengan kendaraan atau *node* lainnya secara efisien. MAC *layer* dalam hal ini menjadi perhatian utama karena peran pentingnya untuk mengatur *flow* data agar tidak terjadi *packet loss* yang disebabkan karena *packet collision* [6]. Protokol MAC yang digunakan dalam VANET diharapkan dapat memenuhi kebutuhan spesifik VANET seperti mobilitas *node* yang tinggi, frekuensi *link discontinuity* yang besar, perubahan topologi yang cepat, dan transportasi data dengan waktu yang terbatas. Oleh karena itu, performa MAC *protocol* berkontribusi besar dalam efektivitas penerapan VANET yang baik dan aman [7].

VANET memiliki beberapa karakteristik yang membedakannya dengan MANET seperti mobilitas *node* yang dapat diprediksi

dikarenakan pergerakan kendaraan yang dibatasi oleh struktur jalan dan peraturan lalu lintas yang harus dipatuhi; kepadatan jaringan yang bervariasi yang dipengaruhi oleh *traffic* kendaraan; jaringan dengan skala besar seperti di kota dan pemukiman padat penduduk; kemampuan komputasi tinggi berkat dukungan sensor, prosesor, *Global Positioning System* (GPS), kapasitas memori besar, dan sistem antena yang mampu memberikan informasi mendetail mengenai kendaraan seperti kecepatan, arah, dan posisi; dan sumber tenaga yang banyak yang disuplai oleh baterai kendaraan [8].

2.2 Ad-hoc On-demand Distance Vector (AODV)

Ad-hoc On-demand Distance Vector (AODV) merupakan sebuah metode *routing* yang digunakan untuk mengirimkan data atau pesan antar perangkat bergerak. AODV dapat mengirim data secara *unicast, multicast,* dan *broadcast* [9]. AODV memungkinkan perangkat atau *node* untuk mengirim data ke *node* yang tidak terjangkau secara langsung karena keterbatasan *range* transmisi dengan cara meneruskan data tersebut melewati *node-node* tetangga yang dapat dijangkau. Hal ini dilakukan dengan cara mencari rute yang dapat dilewati oleh data sehingga dapat sampai ke *node* tujuan. Sebuah *node* merupakan tetangga dari *node* lain apabila *node* tersebut dapat berkomunikasi secara langsung dan berada pada *range* transmisi. *Node* akan mengecek keberadaan *node* tetangga dengan mendengarkan pesan HELLO yang di-*broadcast* oleh setiap node pada selang waktu tertentu. [10]. Secara umum mekanisme fundamental AODV terbagi menjadi dua yakni *route discovery mechanism* dan *route maintenance mechanism* [11].

Route discovery mechanism diawali dengan *source node* yang ingin mengirimkan data ke *destination node*. *Source node* men-*generate* pesan *Route Request* (RREQ). Pesan RREQ ini berisi informasi penting seperti *source node*, *destination node,* jangka waktu validitas RREQ dan *sequence number* yang juga digunakan sebagai ID unik. Ketika *node-node* tetangga dari *source node* menerima RREQ, *node-node* ini memiliki dua pilihan; jika *node-node* ini mengetahui rute ke *destination node* atau *node-node* tersebut adalah *destination node*, maka *node* akan mengirimkan pesan *Route Reply* (RREP) kembali ke *source node*, jika tidak *node-node* ini akan me-*rebroadcast* RREQ ke tetangga masing-masing seterusnya hingga jangka waktu validitas RREQ habis. Apabila *source node* tidak menerima RREP dalam jangka waktu yang ditentukan, *source node* akan me-*rebroadcast* RREQ namun dengan jangka waktu validitas yang lebih lama dan ID yang baru. Semua *node* menggunakan *sequence number* di RREQ untuk memastikan semua node tidak me-*rebroadcast* RREQ yang telah kadaluarsa.

Sequence number berperan sebagai *time stamp* yang memungkinkan *node* untuk membandingkan seberapa *"fresh"* informasi yang ada di *node-node* lain. Tiap kali *node-node* mengirim berbagai tipe pesan, *node-node* tersebut akan menaikkan *sequence number*-nya. Tiap node akan mencatat *sequence number* semua *node* uang berkomunkasi dengannya. *Sequence number* yang lebih tinggi mengindikasikan rute yang lebih baru dan informasi yang lebih akurat.

Route maintenance mechanism menggunakan pesan *Route Error* (RERR) yang memungkinkan AODV untuk menyesuaikan rute ketika *node* bergerak. Ketika *node* menerima pesan RERR, *node* tersebut akan memperbarui

routing table dengan menghapus rute yang memiliki *bad node*. Terdapat tiga skenario yang menyebabkan sebuah *node* mengirimkan pesan RERR. Skenario pertama, ketika *node* menerima paket data yang harus diteruskan namun tidak memiliki rute ke *node* tujuan. Permasalahan sebenarnya bukanlah *node* yang tidak memiliki rute, permasalahannya adalah satu atau beberapa *node* mungkin berpikir bahwa rute yang benar adalah untuk sampai ke *node* tujuan adalah dengan melewati node tersebut. Skenario kedua, *node* menerima RERR yang menyebabkan setidaknya satu rute yang dimilikinya menjadi invalid. Bila ini terjadi, *node* akan mengirim RERR dengan *node-node* yang sekarang tidak dapat dijangkau. Skenario ketiga, ketika *node* tidak bisa berkomunikasi dengan salah satu tetangganya. Ketika ini terjadi, *node* akan melihat *routing table* untuk rute yang memakai *node* tetangga tersebut sebagai *next hop* dan menandainya sebagai rute invalid. Lalu *node* ini akan mengirim RERR dengan tetangga dan rute yang tidak valid.

2.3 Ant Colony Optimization (ACO)

Ant Colony Optimization (ACO) merupakan metode optimasi stokastik berbasis populasi yang diperkenalkan pertama kali oleh Marco Dorigo pada tahun 1991 [12, 13, 14] terinspirasi dari tingkah laku koloni semut yang hidup secara berkelompok. Koloni semut bekerja sama mencari sumber daya makanan, menandai kekayaan sumber daya, rekrutmen dan berkomunikasi dengan memanfaatkan ciri fisiologis mereka yakni menghasilkan *trail pheromone* [15]. Algoritma ACO mengonversi dan memformulasikan perilaku koloni semut dalam memanfaatkan *pheromone* menjadi bentuk matematis yang dipahami oleh komputer. Algoritma ACO banyak diadopsi untuk menyelesaikan kasus-kasus kombinatorial

seperti penjadwalan, penentuan rute, *assignment problem* (seperti *Satisifiability problem* (SAT)) dan sebagainya.

Algoritma ACO menginisiasi populasi semut artifisial untuk melakukan penjelajahan rute. Semut artifisial ini merupakan agen yang akan mencari solusi yang baik pada sebuah problem optimasi. Untuk mengaplikasikan algoritma ACO, problem optimasi perlu terlebih dahulu dikonversikan menjadi problem pencarian rute dengan *weighted graph*.

Pergerakan semut dalam memilih rute ditentukan oleh nilai probabilitas yang dipengaruhi oleh nilai *pheromone* dan *heuristic function* yang digunakan. Secara umum, probabilitas pemilihan rute ini dapat dirumuskan sebagai berikut.

$$p_j = \frac{\tau(i,j)^{\alpha}\eta(i,j)^{\beta}}{\sum_{h \epsilon J}\tau(i,h)^{\alpha}\eta(i,h)^{\beta}} \qquad (2.1)$$

dengan p_j adalah probabilitas mengunjungi node j, $\tau(i,j)$ adalah level *pheromone* pada *edge* (i,j), $\eta(i,j)$ adalah *heuristic function* yang dapat didefinisikan sesuai kebutuhan problem, h adalah *node-node* yang boleh dikunjungi pada sebuah persimpangan J, serta α dan β adalah parameter pembobotan yang dapat diset sesuai kebutuhan untuk menentukan tingkat pengaruh *pheromone* dan *heuristic function* dalam menentukan probabilitas p_j.

Nilai *pheromone* pada *edge* umumnya akan diperbarui ketika semua semut telah memenuhi solusi yang ditentukan, naik turunnya sebuah nilai *pheromone* pada sebuah *edge* berkaitan erat dengan kualitas *edge* tersebut untuk menuju ke solusi.

Berikut adalah contoh *rule* untuk memperbarui nilai *pheromone* secara global.

$$\tau(i,j) = (1-\rho)\cdot\tau(i,j) + \sum_{k=1}^{m}\Delta\tau_k(i,j) \qquad (2.2)$$

dengan ρ adalah laju evaporasi *pheromone* yang nilainya dapat diset sesuai kebutuhan, $\Delta\tau_k(i,j)$ adalah kuantitas nilai *pheromone* pada *edge* (i,j) yang ditambahkan oleh semut k, $\Delta\tau_k(i,j)$ bernilai Q/L_k jika semut k menggunakan *edge* (i,j) saat menjelajah, dan 0 apabila semut tidak menggunakan *edge* (i,j), dengan Q adalah konstanta dan L_k adalah *cost* yang digunakan semut untuk menjelajah (contohnya jarak tempuh, energi, dan sebagainya).

Langkah-langkah di atas di ulang dengan populasi semut yang diperbarui tiap iterasi hingga ditemukan konvergensi ke arah solusi yang ingin dicapai.

2.4 *Network Simulator-2* (NS-2)

Network Simulator 2 atau banyak dikenal dengan NS-2, adalah sebuah perangkat lunak simulasi jaringan *open source* (*GNU Public License*) yang berguna untuk mempelajari sifat dinamis jaringan komunikasi, simulasi fungsi jaringan kabel, nirkabel, dan protokol. NS-2 merupakan perangkat lunak yang menampilkan bagaimana proses komunikasi dalam jaringan terjadi. Fitur-fitur di NS-2 memungkinkan pengguna untuk melakukan *modelling*, pengujian dan memvalidasi pemodelan jaringan seperti VANET dan MANET [16].

NS-2 menggunakan dua bahasa utama yaitu bahasa C++ dan *Object-orientated Tool Command Language* (OTCL). Di NS-2 mendefinisikan mekanisme internal (*back end*) dari objek simulasi, dan OTCL mendefinisikan lingkungan simulasi eksternal (*fron tend*) untuk perakitan dan konfigurasi objek. Setelah simulasi, NS-2 memberikan hasil simulasi dalam bentuk *file* NAM atau *trace file* [17].

2.5 *OpenStreetMap* (OSM)

OpenStreetMap adalah sebuah proyek *open source* kolaboratif berbasis web untuk membuat peta dunia yang dibangun sepenuhnya oleh sukarelawan dengan melakukan survei menggunakan GPS, digitalisasi citra satelit, dan mengumpulkan data geografis yang tersedia di publik. Melalui *Open Data Commons Open Database License 1.0*, kontributor OSM dapat memiliki, memodifikasi, dan membagikan data peta kepada khalayak luas.

Terdapat beragam jenis peta digital yang tersedia di internet, namun sebagian besar memiliki keterbatasan secara legal maupun teknis. Hal ini membuat masyarakat, pemerintah, peneliti dan akademisi, inovator, dan banyak pihak lainnya tidak dapat menggunakan data yang tersedia di dalam peta tersebut secara luas. Hal inilah yang memotivasi OSM untuk menyediakan layanannya. Baik peta dasar OSM maupun data yang tersedia di dalamnya dapat diunduh secara gratis dan terbuka yang dapat digunakan untuk didistribusikan kembali.

Di banyak tempat di dunia ini, terutama di daerah terpencil dan terbelakang secara ekonomi, tidak terdapat insentif komersial sama sekali bagi perusahaan pemetaan untuk mengembangkan data di tempat ini. OSM dapat menjadi jawaban di banyak tempat seperti ini, baik itu pengembangan ekonomi, tata kota,

kontingensi bencana, maupun untuk berbagai tujuan lainnya [18].

2.6 *Java OpenStreetMap* (JOSM)

Java OpenStreetMap atau biasa disingkat JOSM merupakan sebuah aplikasi desktop berbasis *Java* dan dapat dioperasikan pada sistem operasi seperti *Windows*, *Mac OS*, dan *Linux* untuk menyunting data yang didapatkan dari *OpenStreetMap*. JOSM pertama kali dikembangkan oleh Immanuel Scholz pada tahun 2005. Aplikasi JOSM tidak membutuhkan koneksi internet saat menyunting data OSM dan dapat diunduh pada halaman https://josm.openstreetmap.de untuk mendapatkan versi terbaru dari aplikasi ini [19].

2.7 *Simualtion of Urban Mobility* (SUMO)

Simulation of Urban Mobility (SUMO) merupakan sebuah simulator *open source* mengenai lalu lintas jalan yang memungkinkan pengguna membangun topologi jaringan yang disesuaikan sesuai kebutuhan. Fitur-fitur yang tersedia membantu menghasilkan sebuah topologi jalan yang bisa dirancang sesuai keadaan di dunia nyata. SUMO juga memiliki dukungan untuk model simulasi mikroskopis, seperti menerapkan batas kecepatan, mendefinisikan jumlah jalur, persimpangan, lampu lalu lintas, dan lain-lain.

Hal ini juga memungkinkan pengguna untuk merancang kendaraan dengan atribut tertentu seperti panjang kendaraan, kecepatan maksimum, percepatan, dan perlambatannya. SUMO juga menyediakan pilihan bagi pengguna untuk menentukan rute acak untuk kendaraan. Ada juga pilihan yang tersedia untuk model sistem transportasi umum, di mana setiap kendaraan datang dan berangkat sesuai dengan jadwal [20].

BAB III
PERANCANGAN

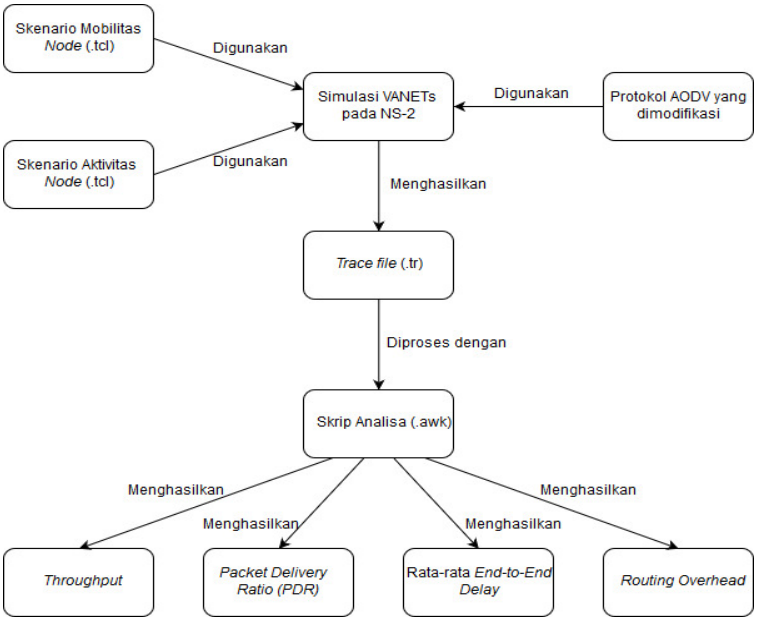
Pada bab ini dijelaskan mengenai perancangan sistem yang dibuat dalam Tugas Akhir. Secara khusus akan dibahas mengenai deskripsi umum, perancangan skenario, alur, dan implementasi.

3.1 Deskripsi Umum

Dalam Tugas Akhir ini akan dilakukan implementasi dan analisis *routing protocol* AODV yang telah dilakukan modifikasi pada bagian proses *route discovery.* Proses tersebut dijalankan pada *Network Simulator-2* (NS-2). Rancangan simulasi dapat dilihat pada diagram dalam Gambar 3.1.

Dalam Tugas Akhir ini, digunakan dua jenis skenario sebagai perbandingan pengukuran, yaitu peta berbentuk *grid* dan *real.* Pembuatan skenario *grid* dibuat dengan bantuan aplikasi SUMO dan pembuatan skenario *real* dibuat berdasarkan peta lalu lintas kota Surabaya yang diambil dari situs resmi *OpenStreetMap* (OSM) yang kemudian dirapikan menggunakan aplikasi *Java OpenStreetMap Editor* (JOSM). Setelah itu, dilakukan simulasi lalu lintas dengan menggunakan bantuan aplikasi SUMO dari *file* peta yang berhasil terbentuk. Hasil simulasi lalu lintas tersebut kemudian digunakan untuk simulasi *routing protocol* AODV yang telah dimodifikasi pada *Network Simulator-2* (NS-2). Simulasi NS-2 menghasilkan sebuah *trace file* yang dianalisis untuk menghitung *Packet Delivery Ratio* (PDR), rata-rata *End-to-End Delay* (E2E)*, Routing Overhead*, *Forwarded Route Request* (RREQ), dan *Throughput,* dengan menggunakan skrip AWK. Penghitungan metrik tersebut dilakukan untuk membandingkan performa dari

routing protocol AODV *default* yang belum dimodifikasi, AODV dengan AODV yang telah dimodifikasi.



Gambar 3.1 Diagram rancangan simulasi AODV modifikasi

Daftar istilah yang sering digunakan pada buku Tugas Akhir ini dapat dilihat pada Tabel 3.1.

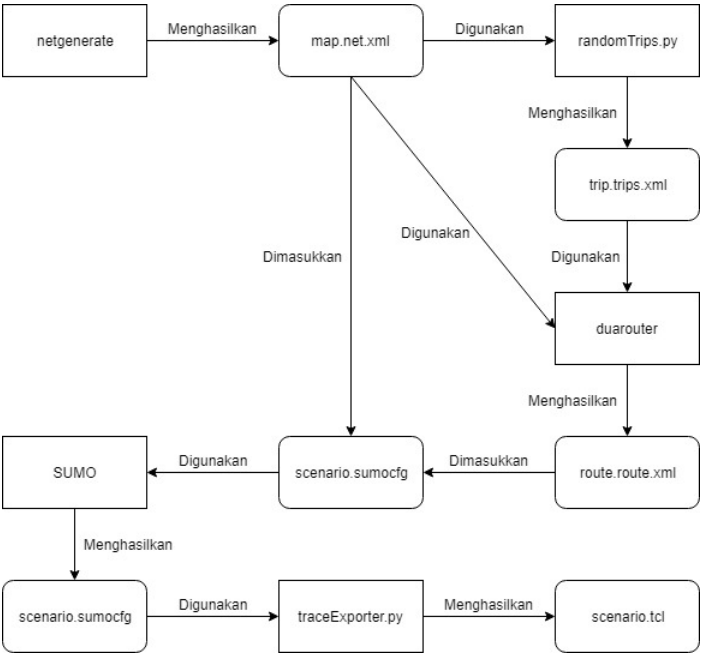
Tabel 3.1 Daftar istilah

No	Istilah	Penjelasan
1	AODV	Singkatan dari *Ad hoc On-demand Distance Vector. Routing protocol* yang digunakan pada Tugas Akhir ini.
2	*Throughput*	Rata-rata dari jumlah data yang diterima oleh *node* tujuan yang terukur pada waktu tertentu.
3	PDR	*Packet Delivery Ratio*. Salah satu metrik analisis yang diukur. Berupa rasio jumlah pengiriman paket yang terkirim.
4	E2E	Rata-rata *End-to-End Delay*. Jeda waktu yang diukur saat paket terkirim.
5	RO	*Routing Overhead*. Jumlah *control packet* yang terkirim.
6	RREQ F	*Forwarded Route Request*. Jumlah *forwarding node* yang bertugas untuk mengirim ulang (*rebroadcast*) RREQ.
7	RREQ	*Route Request*. Paket *request* pada AODV yang dikirim untuk mendapatkan rute.
8	RREP	*Route Reply*. Paket *reply* pada AODV yang dikirim ke *node* sumber melalui rute yang telah terbentuk.
9	*Threshold*	Batas nilai yang dijadikan sebagai acuan.

3.2 Perancangan Skenario *Grid*

Dalam pembuatan peta *grid*, terlebih dahulu ditentukan panjang jalan dan jumlah titik persimpangan jalan. Peta *grid* memiliki bentuk segi empat. Luas area yang dibutuhkan untuk simulasi adalah 910 m x 910 m, maka ditentukan panjang jalan tiap *grid* lebih kurang 300 meter dengan 4 titik persimpangan yang akan membentuk 9 petak. Panjang jalan dan jumlah titik persimpangan jalan tersebut kemudian digunakan sebagai argumen untuk *netgenerate.* Selain itu, argumen *netgenerate* juga dapat digunakan untuk menentukan batas kecepatan setiap jalan. Hasil dari netgenerate digunakan sebagai argumen untuk *randomTrips.py*. Fungsi dari program *randomTrips.py* tersebut adalah untuk mendefinisikan rute perjalanan dari seluruh *node*. *Output* atau keluaran dari *randomTrips.py* diproses oleh *duarouter* untuk memperbaiki masalah konektivitas rute (jika ada). Setelah itu dilakukan simulasi lalu lintas dengan menggunakan SUMO.

Simulasi tersebut akan menghasilkan *file* berekstensi .xml. *File* tersebut akan diekspor menjadi sebuah *file* baru yang dapat diproses oleh NS-2. Proses ekspor *file* tersebut dapat dilakukan dengan menggunakan bantuan *traceExporter.py* dan menghasilkan *file* skenario mobilitas *grid* dari setiap *node* yang berekstensi .tcl. Alur pembuatan skenario *grid* dapat dilihat pada Gambar 3.2.



Gambar 3.2 Alur perancangan skenario *grid*

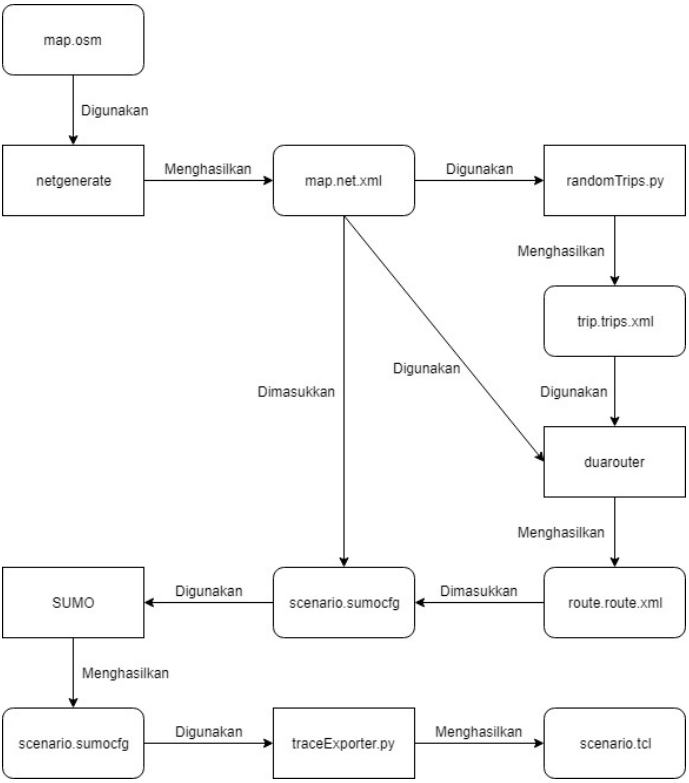
3.3 Perancangan Skenario *Real*

Untuk perancangan skenario *real*, terlebih dahulu dilakukan pemilihan area yang nantinya digunakan sebagai model untuk simulasi. Setelah memilih area, unduh area tersebut dari situs resmi *OpenStreetMap* (OSM) menggunakan fitur *download*. Potongan peta yang telah dipilih dan diunduh tersebut kemudian dirapikan dengan menggunakan aplikasi *Java OpenStreetMap Editor* (JOSM). Dilakukan juga pengaturan interval lampu lalu lintas yang bertujuan untuk menyerupai dunia nyata. Selanjutnya, dilakukan penghapusan jalan yang

terputus dan jalan yang tidak diinginkan dari potongan peta tersebut sehingga menjadi peta sebuah area yang tertutup.

Setelah potongan peta yang dipilih selesai dirapikan, dibuat sebuah *file* yang berisi definisi dari spesifikasi batasan simulasi lalu lintas. Batasan yang dimaksud seperti batas kecepatan pada kelas jalan tertentu dan batasan lainnya. Berdasarkan *file* batasan simulasi yang telah dibuat, peta dikonversi dengan menggunakan netconvert. Hasil konversi peta tersebut kemudian digunakan untuk membuat *file* rute pergerakan kendaraan melalui randomTrips.py dan duarouter. Setelah didapatkan *file* peta yang telah dikonversi dan *file* rute pergerakan kendaraan, dilakukan simulasi dengan menggunakan SUMO.

Hasil simulasi tersebut adalah *file* berekstensi .xml yang akan diekspor menjadi *file* yang dapat diproses oleh NS-2. Proses ekspor dapat dilakukan dengan bantuan *traceExporter.py* dan menghasilkan *file* skenario mobilitas *real* dari setiap *node* yang berekstensi .tcl. Alur pembuatan skenario *real* dapat dilihat pada Gambar 3.3.



Gambar 3.3 Alur perancangan skenario *real*

3.4 Perancangan Modifikasi *Routing Protocol* AODV

Pada Tugas Akhir ini, *routing protocol* yang diajukan adalah modifikasi dari *routing protocol* AODV. Modifikasi tersebut mengubah mekanisme pada proses *route discovery* dari *routing protocol* AODV. Mekanisme pencarian *node* untuk pengiriman ulang paket *route request* (RREQ) pada *routing protocol* AODV langsung dikirim begitu saja. Berbeda dengan *routing protocol* AODV, pada AODV yang dimodifikasi terdapat tahap

seleksi *node*. Terdapat juga penghitungan nilai *pheromone.* Nilai *pheromone* adalah suatu nilai yang diadopsi dari penggunaan algoritma *Ant Colony Optimization* (ACO). Nilai tersebut digunakan untuk menentukan rute pengiriman paket *route reply* (RREP) oleh *node* tujuan. Hasil penghitungan nilai *pheromone* disimpan pada header paket RREQ dan RREP serta *routing table*. Oleh karena itu, diperlukan modifikasi dengan menambahkan *field* pada struktur paket tersebut dan pada *routing table.* Perubahan struktur dapat dilihat pada Tabel 3.2, Tabel 3.3, dan Tabel 3.4.

Tabel 3.2 Struktur paket RREQ

Src_address	Src_sequence no	Request_id	Dest_address	Dest_sequence no	Hop_count	Signal_strength	Pheromone_count

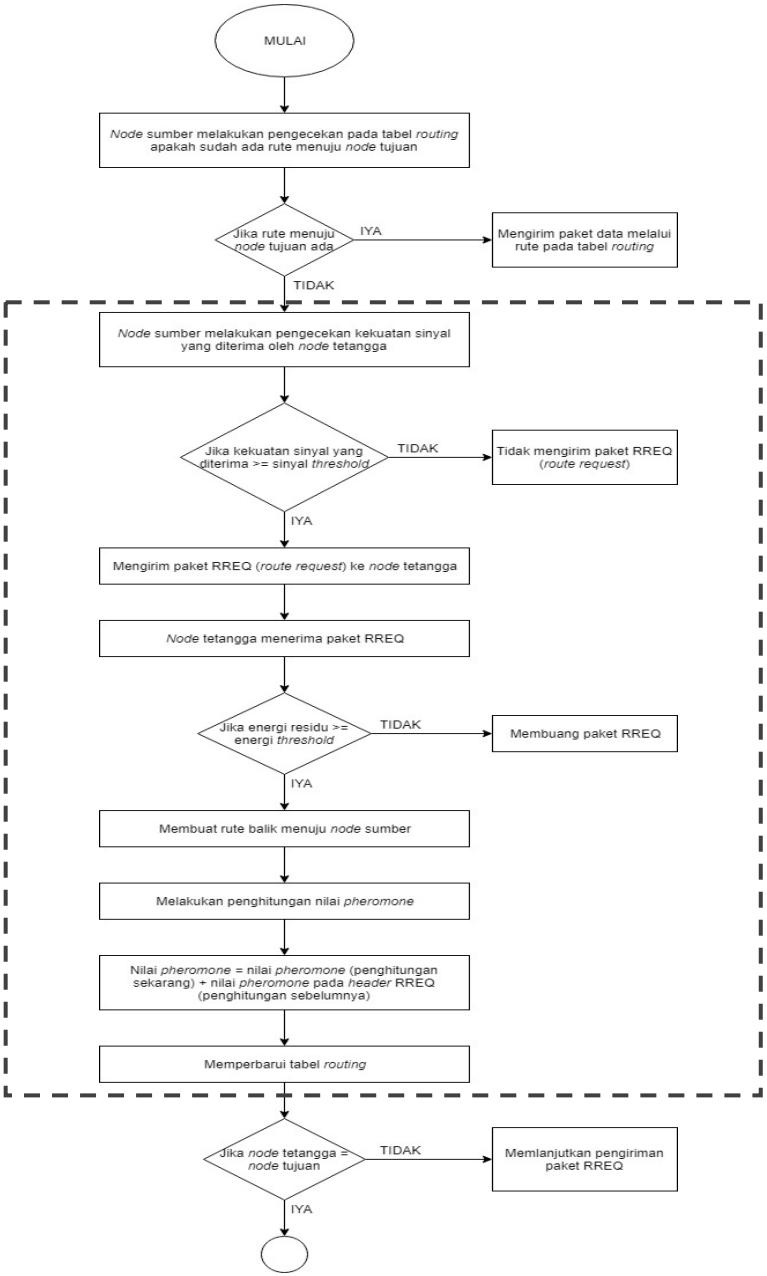
Tabel 3.3 Struktur paket RREP

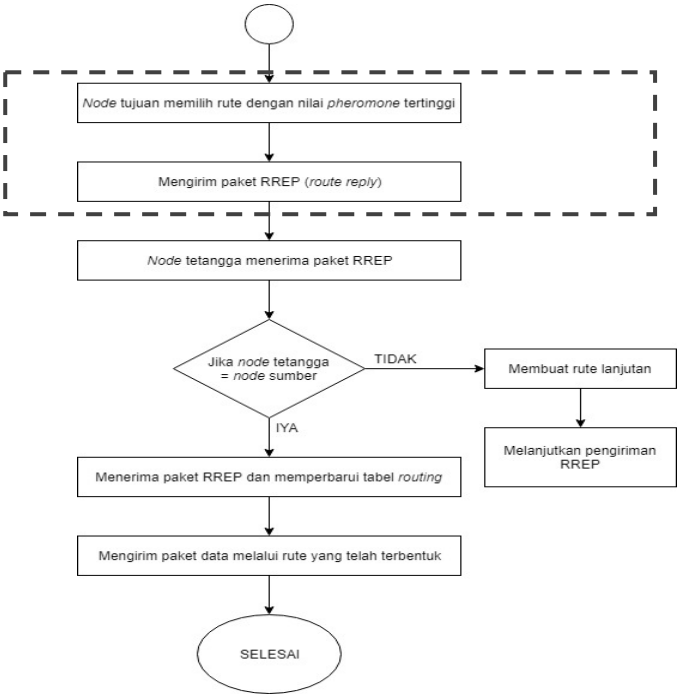
Src_address	Dest_address	Dest_sequenceno	Lifetime	Hop_count	Pheromone_count

Tabel 3.4 Struktur *routing table*

Dest_address	Sequence_no	Hop_count	Next_hop	RSSM	Residual Energy	Congestion Metric	Speed	Distance	Pheromone_count

Gambar 3.4. menunjukkan *flow chart* dari modifikasi *routing protocol* AODV dengan menggunakan algoritma ACO pada bagian yang berada di dalam kotak bergaris putus-putus.


MULAI
Node sumber melakukan pengecekan pada tabel routing apakah sudah ada rute menuju node tujuan
Jika rute menuju node tujuan ada
IYA
Mengirim paket data melalui rute pada tabel routing
TIDAK
Node sumber melakukan pengecekan kekuatan sinyal yang diterima oleh node tetangga
Jika kekuatan sinyal yang diterima >= sinyal threshold
TIDAK
Tidak mengirim paket RREQ (route request)
IYA
Mengirim paket RREQ (route request) ke node tetangga
Node tetangga menerima paket RREQ
Jika energi residu >= energi threshold
TIDAK
Membuang paket RREQ
IYA
Membuat rute balik menuju node sumber
Melakukan penghitungan nilai pheromone
Nilai pheromone = nilai pheromone (penghitungan sekarang) + nilai pheromone pada header RREQ (penghitungan sebelumnya)
Memperbarui tabel routing
Jika node tetangga = node tujuan
TIDAK
Memlanjutkan pengiriman paket RREQ
IYA



Gambar 3.4 *Flow chart* modifikasi *routing protocol* AODV dengan menggunakan algoritma ACO

Modifikasi mekanisme pengiriman paket RREQ dilakukan setelah *node* sumber melakukan pengecekan terhadap rute menuju *node* tujuan pada *routing table*. Modifikasi yang ditambahkan adalah *node* sumber melakukan pengecekan kekuatan sinyal yang diterima dan kemudian melakukan seleksi *node* tetangga. Modifikasi yang ditambahkan selanjutnya adalah ketika *node* tetangga menerima paket RREQ. *Node* tetangga sendiri yang melakukan seleksi berdasarkan energi residu atau energi tersisa pada *node* tersebut. *Node* yang berhasil memenuhi kondisi energi residu akan melakukan

penghitungan nilai *pheromone*. Penghitungan nilai tersebut adalah bentuk dari penggunaan algoritma *Ant Colony Optimization* (ACO) yang dilakukan pada mekanisme *route discovery*. Ketika *node* tujuan menerima paket RREQ, maka mekanisme berikutnya adalah pengiriman *route reply* (RREP). Modifikasi pada mekanisme pengiriman paket RREP adalah *node* tujuan memilih rute dengan nilai *pheromone* tertinggi.

3.4.1 Perancangan Pengiriman *Route Request*

Modifikasi *routing protocol* AODV ini memiliki 2 tahap seleksi *node* pada mekanisme pengiriman paket *route request* (RREQ). Sebelum melakukan seleksi *node*, *node* sumber terlebih dahulu melakukan pengecekan *routing table*. Pengecekan tersebut bertujuan untuk mengetahui apakah terdapat rute menuju *node* tujuan. Jika rute tersedia, paket data akan dikirim melalui rute tersebut dan tidak dilakukan mekanisme *route discovery* baru. Namun, jika rute tidak tersedia, mekanisme *route discovery* akan dilakukan.

Tahap pertama dari seleksi *node* adalah *node* sumber mengetahui nilai kekuatan sinyal yang diterima oleh *node* tetangga. Terdapat nilai sinyal *threshold* yang ditentukan sebelumnya, yaitu sebesar -75 dBm. *Node* sumber tidak akan mengirimkan paket RREQ apabila nilai kekuatan sinyal yang diterima oleh *node* tetangga lebih kecil dari nilai sinyal *threshold* tersebut. Namun jika nilai kekuatan sinyal yang diterima oleh *node* tetangga lebih besar atau sama dengan nilai sinyal *threshold* tersebut, maka *node* sumber akan mengirimkan paket RREQ. Setelah *node* tetangga menerima paket RREQ dilakukan seleksi *node* tahap kedua.

Pada seleksi *node* tahap kedua, nilai kecepatan *node* dan jarak antara *node* dengan *node* tujuan akan di-*fetch* dan pada tahap ini

juga akan dilakukan pengecekan energi yang terdapat pada *node* tersebut atau biasa disebut juga dengan energi residu. Terdapat nilai energi *threshold* yang ditentukan, yaitu sebesar 10 Joule. Jika nilai energi residu pada *node* tetangga lebih kecil dari nilai energi *threshold* tersebut*,* paket RREQ akan dibuang. Namun, jika nilai energi residu pada *node* tetangga lebih besar atau sama dengan nilai energi *threshold,* dilanjutkan dengan melakukan 3 langkah yaitu pembuatan rute balik menuju *node* sumber, penghitungan nilai *pheromone*, dan memperbarui *routing table*. Penghitungan nilai *pheromone* dilakukan berdasarkan besarnya kekuatan sinyal yang diterima, energi residu, kecepatan *node*, kemacetan, jumlah *hop*, dan jarak *node* tersebut relatif dengan *node* tujuan pada selang waktu tertentu. Nilai *pheromone* disimpan pada *header* paket RREQ. Nilai tersebut akan dijumlahkan secara terus-menerus hingga paket RREQ diterima oleh *node* tujuan. Apabila *node* tetangga bukan merupakan *node* tujuan, maka pengiriman paket RREQ akan dilanjutkan. Jika paket telah mencapai *node* tujuan, *node* akan mengembalikannya dengan paket *route reply* (RREP).

3.4.2 Perancangan *Ant Colony Optimization* (ACO)

Pemilihan *next hop* untuk meneruskan *route request* (RREQ) ke tujuan tidak selalu sempurna. Hal ini bisa dikarenakan antrean (kepadatan) dan kurangnya energi pada *node* tersebut. Karena hal tersebut, Tugas Akhir ini mempertimbangkan nilai *pheromone* yang diadopsi dari penggunaan algoritma *Ant Colony Optimization* (ACO) yang digunakan untuk memilih *next hop*. Penghitungan nilai *pheromone* dinyatakan sebagai PC dan dapat dilihat pada persamaan 3.1.

$$PC = \frac{RSSM \times RE \times n_Speed}{Congestion \times HC \times n_Distance} \quad (3.1)$$

dengan PC adalah nilai *pheromone* yang digunakan untuk pemilihan rute, RSSM adalah metrik kekuatan sinyal yang ditermia, RE adalah nilai energi residu pada *node* tetangga, *n_Speed* adalah nilai kecepatan pada *node* tetangga, *Congestion* adalah nilai antrain atau kepadatan pada *node* tetangga, HC adalah nilai *hop count* pada *node* tetangga , dan *n_Distance* adalah nilai jarak *node* tetangga relatif dengan *node* tujuan.

RSSM sendiri dihitung dengan persamaan 3.2.

$$RSSM = 1 - \frac{SignalThr_i + c_s}{RSS_i + c_r} \quad (3.2)$$

dengan $SignalThr_i$ adalah nilai *threshold* sinyal pada node *i*, c_s adalah nilai konstanta penyesuai nilai *threshold*, RSS_i adalah nilai kekuatan sinyal pada node *i*, dan c_r adalah nilai konstanta penyesuai nilai sinyal.

3.4.3 Perancangan Pengiriman *Route Reply*

Paket *route reply* (RREP) dikirimkan dengan hasil penghitungan nilai *pheromone* sebagai pertimbangan. Karena itu, pada *node* tujuan terdapat pengecekan nilai *pheremone*. Pengecekan tersebut dilakukan dengan membandingkan nilai *pheromone* yang terdapat pada *routing table* dengan nilai *pheromone* yang baru diterima melalui paket *route request* (RREQ). Jika terdapat rute baru dengan nilai *pheromone* yang

lebih tinggi, rute baru tersebut akan disimpan pada *routing table*. Penentuan rute baru dilakukan berdasarkan nilai *pheromone* tertinggi yang dipilih oleh *node* tujuan untuk mengirimkan paket RREP.

3.5 Perancangan Simulasi pada NS-2

File skenario yang telah dibuat dengan menggunakan SUMO dan *file* skrip dengan ekstensi .tcl yang berisi konfigurasi lingkungan simulasi digabungkan untuk menjalankan simulasi VANETs pada *Network Simulator-2* (NS-2). Saat simulasi NS-2 dijalankan, *routing protocol* AODV akan melakukan seleksi *node* yang berhak melakukan *rebroadcast* paket RREQ.

3.6 Perancangan Metrik Analisis

Untuk dapat membandingkan performa dari *routing protocol* AODV yang asli dengan *routing protocol* AODV yang telah dimodifikasi pada Tugas Akhir ini, dilakukan analisis terhadap parameter-parameter sebagai berikut:

3.6.1 *Packet Delivery Rasio* (PDR)

Metrik *Packet Delivery Ratio* (PDR) didapatkan dengan membandingkan banyaknya paket data yang dikirim dengan paket data yang diterima. PDR menunjukkan nilai keberhasilan paket yang dikirimkan. Semakin tinggi nilai PDR yang didapatkan, maka semakin tinggi pula tingkat keberhasilan pengiriman paket data. Rumus untuk menghitung PDR dapat dilihat pada persamaan 3.3. Nilai PDR dinyatakan dalam bentuk persen.

$$PDR = \frac{received}{sent} x\ 100\ \% \qquad (3.3)$$

dengan PDR adalah *Packet Delivery Rasio*, *received* adalah banyak paket data yang diterima, dan *sent* adalah banyak paket data yang dikirimkan.

3.6.2 Rata-rata *End-to-End Delay* (E2E)

Rata-rata *End-to-End Delay* didapatkan dengan menghitung rata-rata *delay* antara waktu paket data ketika diterima dan waktu paket dikirimkan. Nilai dari E2E dinyatakan dalam satuan detik. Delay tiap paket didapat dari rentang waktu antara *node* asal saat mengirimkan paket dan *node* tujuan menerima paket. Delay tiap paket tersebut semua dijumlahkan dan dibagi dengan jumlah paket yang berhasil diterima. Rumus penghitungan nilai E2E dapat dilihat pada persamaan 3.4.

$$E2E = \frac{\sum_{m=1}^{recvnum} CBRRecvTime\ - CBRSentTime}{recvnum} \qquad (3.4)$$

dengan E2E adalah *End-to-End Delay*, *CBRRecvTime* adalah waktu *node* asal mengirimkan paket, *CBRSentTIme* adalah waktu *node* tujuan menerima paket, dan *recvnum* adalah jumlah paket yang berhasil diterima.

3.6.3 *Routing Overhead* (RO)

Routing Overhead (RO) adalah jumlah paket kontrol *routing* yang ditransmisikan per data paket yang terkirim ke *node* tujuan selama simulasi terjadi. Nilai RO didapatkan dengan menjumlahkan semua paket kontrol *routing* yang ditransmisikan, baik itu paket *route request* (RREQ), *route*

reply (RREP), maupun *route error* (RERR). Penghitungan *Routing Overhead* dapat dilihat pada persamaan 3.5.

$$RO = RREQ_{sent} + RREP_{sent} + RERR_{sent} \qquad (3.5)$$

dengan RO adalah *Routing Overhead*, $RREQ_{sent}$ adalah paket *route request* (RREQ) yang berhasil terkirim, $RREP_{sent}$ adalah paket *route reply* (RREP) yang berhasil terkirim, dan $RERR_{sent}$ adalah paket *route error* (RERR) yang berhasil terkirim.

3.6.4 *Forwarded Route Request* (RREQ F)

Forwarded Route Request (RREQ F) adalah jumlah paket kontrol *route request* (RREQ) yang di *forward* atau di *rebroadcast* per data paket ke *node* tujuan selama simulasi terjadi. Nilai RREQ F didapatkan dengan menjumlahkan semua paket *control routing* yang ditransmisikan khususnya *route request* bagian *forwarding* (RREQ F). Penghitungan *Routing Overhead* dapat dilihat pada persamaan 3.6.

$$Forwarded\ Route\ Request = \sum_{n=1}^{rreqsent} packet\ sent \qquad (3.6)$$

dengan *rreqsent* adalah jumlah *forwarded route request* yang terkirim, dan *n* adalah jumlah paket yang dikirimkan.

3.6.5 Throughput

Throughput adalah rata-rata jumlah data (*bits*) yang diterima oleh *node* tujuan pada ukuran waktu tertentu yang terukur

dengan menggunakan rute jaringan. Pada Tugas Akhir ini, *throughput* dinyatakan dalam satuan *kilobits* per detik (Kbps). Penghitungan *throughput* dapat dilihat pada persamaan 3.7.

$$Throughput = \frac{bytes\ recieved\ \times 8}{simulation\ time\ \times 1024}\ Kbps \qquad (3.7)$$

dengan *throughput* adalah jumlah data yang diterima oleh *node* tujuan, *bytes received* adalah nilai *bytes* yang diterima oleh *node* tujuan, dan *simulation time* adalah waktu simulasi untuk satu skenario.

(Halaman ini sengaja dikosongkan)

BAB IV
IMPLEMENTASI

Pada bab ini akan diberikan pembahasan mengenai implementasi dari perancangan sistem yang sudah dijelaskan pada bab sebelumnya. Implementasi berupa cara pembuatan skenario mobilitas VANETs yang terbagi menjadi dua, yaitu skenario *grid* dan *real* serta kode sumber untuk membangun program.

4.1 Implementasi Skenario *Grid*

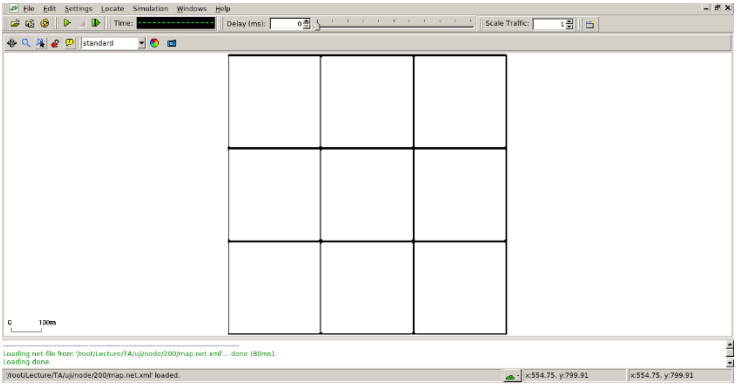
Pada Tugas Akhir ini, penulis menggunakan *tool* netgenerate dari SUMO untuk mengimplementasikan skenario *grid* berupa peta jalan berpetak. Peta skenario *grid* dibuat dengan panjang jalan 300 meter dan luas peta 910 m x 910 m. Dengan melebihkan 10 meter untuk menghindari terjadinya kesalahan pada saat simulasi. Jumlah titik persimpangan antara jalan vertikal dan jalan horizontal sebanyak 4 titik x 4 titik. Dengan jumlah titik persimpangan sebanyak 4 titik tersebut, maka terbentuk 9 buah petak. Kecepatan *node* (kendaraan) secara *default* diatur agar bernilai acak.

Perintah netgenerate untuk membuat peta *grid* dengan spesifikasi tersebut dapat dilihat pada Gambar 4.1.

```
netgenerate --grid --grid.number=4 --
grid.length=300 --tls.guess=1 --output-
file=map.net.xml
```

Gambar 4.1 Perintah netgenerate

Setelah itu, akan dihasilkan *file* peta berekstensi .xml. Gambar peta yang telah dibuat dengan *netgenerate* dapat dilihat pada Gambar 4.2.



Gambar 4.2 Hasil Generate peta *grid*

Setelah peta terbentuk, maka dilakukan pembuatan *node* dan pergerakan *node* dengan menentukan titik asal dan titik tujuan setiap *node* secara random menggunakan *tool* randomTrips.py yang terdapat pada SUMO. Perintah penggunaan *tool* randomTrips.py untuk membuat *node* sebanyak n *node* dengan pergerakannya dapat dilihat pada Gambar 4.3. Opsi -e diisi dengan jumlah kendaraan yang diinginkan saat simulasi.

```
python3
$SUMO_HOME/tools/randomTrips.py      -n
map.net.xml    -e    188    -l    --trip-
attributes="departLane=\"best\"
departSpeed=\"random\"
departPos=\"random_free\""           -o
trip.trips.xml
```

Gambar 4.3 Perintah randomTrips.py

Setelah titik asal dan titik akhir didefinisikan, selanjutnya dilakukan pembuatan rute yang akan digunakan oleh kendaraan menggunakan *tool* duarouter. Perintah penggunaan *tool* duarouter dapat dilihat pada Gambar 4.4.

```
duarouter -n map.net.xml --route-files
trip.trips.xml   -o   route.rou.xml   --
ignore-errors --repair
```

Gambar 4.4 Perintah duarouter

Ketika menggunakan *tool* duarouter, SUMO memastikan bahwa jalur untuk *node-node* yang di*generate* tidak akan melenceng dari jalur peta yang sudah di*generate* menggunakan *tool* randomTrips.py. Selanjutnya untuk menjadikan peta dan pergerakan *node* yang telah di*generate* menjadi sebuah skenario dalam bentuk *file* berekstensi .net.xml, dibutuhkan sebuah *file* skrip dengan ekstensi .sumocfg untuk menggabungkan *file* peta

dan rute pergerakan *node*. Isi dari *file* skrip .sumocfg dapat dilihat pada Gambar 4.5

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration
xmlns:xsi="http://www.w3.org/2001/XMLSchema
-instance"
xsi:noNamespaceSchemaLocation="http://sumo.
dlr.de/xsd/sumoConfiguration.xsd">
   <input>
      <net-file value="map.net.xml"/>
      <route-files value="routes.rou.xml"/>
   </input>
   <time>
      <begin value="0"/>
      <end value="200"/>
   </time>
</configuration>
```

Gambar 4.5 File skrip .sumocfg

File .sumocfg disimpan dalam direktori yang sama dengan *file* peta berekstensi .net.xml dan *file* rute pergerakan *node* berekstensi .trips.xml. *File* .sumocfg digunakan untuk mendefinisikan lokasi *file* .net.xml dan .trips.xml serta durasi simulasi. Untuk melihat visualisasi lalu lintas, *file* .sumocfg dapat dibuka dengan menggunakan *tool* sumo-gui. Kemudian lakukan simulasi lalu lintas dengan menggunakan perintah SUMO yang dapat dilihat pada Gambar 4.6.

```
sumo -c file.sumocfg --fcd-output
scenario.xml
```

Gambar 4.6 Perintah SUMO untuk membuat skenario .xml

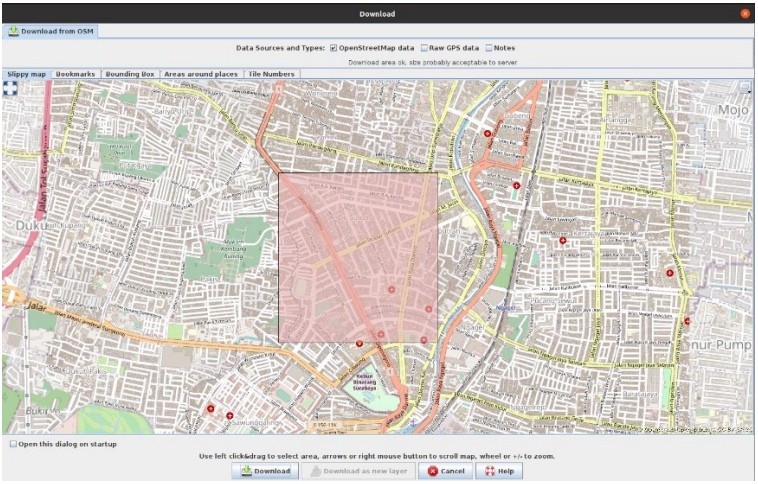
Agar dapat disimulasikan pada NS-2, *file* skenario berekstensi .xml harus dikonversi menjadi *file* berekstensi .tcl dengan menggunakan *tool* traceExporter.py. Perintah untuk menggunakan *tool* traceExporter.py dapat dilihat pada Gambar 4.7.

```
python3
$SUMO_HOME/tools/traceExporter.py --fcd-
input=scenario.xml --ns2mobility-
output=scenario.tcl
```

Gambar 4.7 Perintah traceExporter.p*y*

4.2 Implementasi Skenario *Real*

Pada Tugas Akhir ini, penulis mengimplementasikan skenario *real* menggunakan peta hasil pengambilan suatu area di kota Surabaya, yaitu area jalan sekitar Jl. Dr. Soetomo Surabaya. Setelah menentukan area simulasi, ekspor data peta dari situs *OpenStreetMap* seperti yang ditunjukkan pada Gambar 4.8



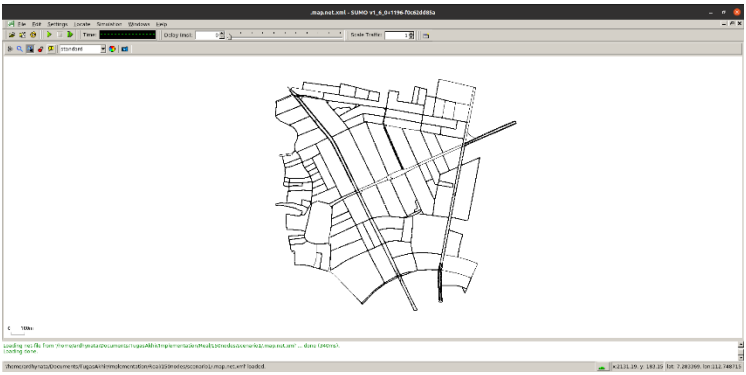
Gambar 4.8 Ekspor peta dari *OpenStreetMap*

File hasil ekspor dari *OpenStreetMap* tersebut adalah *file* peta dengan ekstensi .osm. Peta disunting terlebih dahulu dengan menggunakan *Java OpenStreetMap Editor* (JOSM) untuk menghapus jalan yang tidak digunakan. Kemudian *file* .osm hasil suntingan tersebut dikonversi menjadi peta dalam bentuk *file* berekstensi .xml menggunakan *tool* netconvert dari SUMO. Perintah untuk menggunakan netconvert dapat dilihat pada Gambar 4.9.

```
netconvert --osm-files map.osm --output-
file map.net.xml
```

Gambar 4.9 Perintah netconvert

Hasil konversi peta dari *file* berekstensi .osm menjadi *file* berekstensi .xml dapat dilihat menggunakan *tool* sumo-gui seperti yang ditunjukkan pada Gambar 4.10



Gambar 4.10 Hasil konversi peta real

Setelah peta terbentuk, langkah selanjutnya sama dengan ketika membuat skenario *grid*, yaitu membuat *node* asal dan *node* tujuan menggunakan *tool* randomTrips.py. Lalu menggunakan *tool* duarouter untuk membuat rute *node* untuk sampai ke tujuan. Kemudian menggunakan *tool* SUMO dengan bantuan *file* skrip berekstensi .sumocfg untuk membuat *file* skenario berekstensi .xml. Agar dapat disimulasikan pada NS-2, *file* skenario berekstensi ,xml tersebut dikonversi menjadi *file* skenario berekstensi .tcl dengan menggunakan *tool* traceExporter.py. Perintah untuk menggunakan *tool* tersebut sama dengan ketika membuat skenario *grid* di atas.

4.3 Implementasi Modifikasi *Routing Protocol* AODV

Pada tugas akhir ini dilakukan modifikasi pada *routing protocol* AODV agar dapat mengurangi jumlah *forwarding node*, yaitu *node* yang bertugas untuk melakukan *rebroadcast* paket *route request* (RREQ). Hal tersebut dilakukan dengan cara melakukan seleksi *node* tetangga berdasarkan kekuatan sinyal yang diterima dan energi residu yang terdapat pada *node* tetangga pada waktu *t.* Pada *routing protocol* AODV, juga ditambahkan penghitungan nilai *pheromone* yang digunakan oleh *node* tujuan untuk menentukan rute pengiriman pake *route reply* (RREP). Rute yang dipilih adalah rute yang memiliki nilai *pheromone* tertinggi dengan harapan rute tersebut memiliki tingkat stabilitas yang tinggi. Sehingga dapat dilihat peningkatan performa pada *routing protocol* AODV yang telah dimodifikasi dibandingkan dengan performa pada *routing protocol* AODV asli.

Implementasi modifikasi routing protocol AODV ini dibagi menjadi empat bagian yaitu:

- Implementasi Modifikasi Struktur
- Implementasi Pengiriman *Route Reply*
- Implementasi Penghitungan *Pheromone*
- Implementasi Pengiriman *Route Reply*

Kode implementasi dari routing protocol AODV pada NS-2 versi 2.35 berada pada direktori ns-2.35/aodv. Pada direktori tersebut terdapat beberapa file diantaranya seperti aodv.cc, aodv.h, aodv_packet.h, aodv_rtable.h, dan lain-lain. Pada Tugas Akhir ini, penulis memodifikasi *file* aodv.cc yang terdapat pada *folder* ns-2.35/aodv untuk menyeleksi *node* tetangga yang bertugas melakukan *rebroadcast*, menghitung nilai *pheromone* dan menentukan rute yang akan dipilih. Sedangkan *file* aodv.h yang ada pada *folder* ns-2.35/aodv untuk menginisiasi variabel

posisi *node*, sinyal, energi, *threshold*, dan antrian (kepadatan) pada *node*. Penulis juga menginisiasi variabel *pheromone* pada *file* aodv_packet.h dan aodv_rtable.h untuk menyimpan hasil penghitungan nilai *pheromone*. Pada bagian ini, penulis menjelaskan langkah - langkah dalam mengimplementasikan modifikasi *routing protocol* AODV.

4.3.1 Implementasi Modifikasi Struktur

Pada implementasi *routing protocol* AODV modifikasi ini, terdapat dua *field* yang perlu ditambahkan ke dalam struktur paket *route request* (RREQ). Terdapat satu *field* yang perlu ditambahkan ke dalam struktur paket *route reply* (RREP) dan *routing table*. Untuk itu dilakukan penambahan atribut $rq_pheromone_count$ pada *struct* $hdr_aodv_request$ dan atribut $rp_pheromone_count$ pada *struct* hdr_aodv_reply di skrip aodv_packet.h seperti pada Gambar 4.11.

```
struct hdr_aodv_request {
  u_int8_t        rq_type;
  u_int8_t        reserved[2];
  u_int8_t        rq_hop_count;
  u_int32_t       rq_bcast_id;

  nsaddr_t        rq_dst;
  u_int32_t       rq_dst_seqno;
  nsaddr_t        rq_src;
  u_int32_t       rq_src_seqno;
  double          rq_pheromone_count;
  double          rq_timestamp;
  inline int size() {
    int sz = 0;
       sz = 8*sizeof(u_int32_t);
       assert (sz >= 0);
       return sz;
  }
};

struct hdr_aodv_reply {
        u_int8_t        rp_type;
        u_int8_t        reserved[2];
        u_int8_t        rp_hop_count;
        nsaddr_t        rp_dst;
        u_int32_t       rp_dst_seqno;
        nsaddr_t        rp_src;
        double          rp_lifetime;
        double          rp_pheromone_count;
        double          rp_timestamp;

  inline int size() {
  int sz = 0;
       sz = 6*sizeof(u_int32_t);
       assert (sz >= 0);
       return sz;
  }
};
```

Gambar 4.11 Potongan modifikasi struktur aodv_paket.h

Serta dilakukan juga penambahan atribut $rt_pheromone_count$ dan beberapa atribut pendukung penghitungan *pheromone* pada *class* $aodv_rt_entry$ di skrip aodv_rtable.h seperti pada Gambar 4.12.

```
class aodv_rt_entry {
  friend class aodv_rtable;
  friend class AODV;
  friend class LocalRepairTimer;
  .
  .
  protected:
        LIST_ENTRY(aodv_rt_entry) rt_link;

        nsaddr_t         rt_dst;
        u_int32_t        rt_seqno;
        u_int16_t        rt_hops;
       int                   rt_last_hop_count;
        nsaddr_t         rt_nexthop;
        double               rt_rssm;
        double               rt_cm;

        double               rt_rem;
        double               rt_distance;
        double               rt_speed;
        double               rt_pheromone_count;

        aodv_precursors rt_pclist;
        double           rt_expire;
        u_int8_t         rt_flags;
       double
rt_disc_latency[MAX_HISTORY];
       char           hist_indx;
       int            rt_req_last_ttl;
       aodv_ncache rt_nblist;
};
```

Gambar 4.12 Potongan modifikasi struktur aodv_rtable.h

4.3.2 Implementasi Inisiasi Nilai Variabel

Pada *routing protocol* AODV modifikasi, terdapat penghitungan *pheromone* dan proses penyeleksian *node*

tetangga yang membandingkan nilai kekuatan sinyal, energi residu, kecepatan *node* tetangga, dan jarak *node* tetangga relatif dengan *node* tujuan dengan nilai - nilai *threshold* yang sudah dijelaskan pada subbab 3.4.1. Inisiasi nilai variabel perlu dilakukan dengan memberikan nilai nol sehingga variabel tersebut tidak bernilai kosong. Hali ini dilakukan agar dapat digunakan untuk penghitungan *pheromone*. Nilai – nilai *threshold* juga perlu diinisiasikan agar dapat digunakan untuk proses penyeleksian *node* tetangga. Nilai – nilai tersebut diinisiasikan pada kode sumber aodv.cc yang terdapat dalam direktori ns2.3.5/aodv. Untuk potongan kode tersebut bisa dilihat pada Gambar 4.13.

```
AODV::AODV(nsaddr_t    id)    :    Agent(PT_AODV),
btimer(this),     htimer(this),     ntimer(this),
rtimer(this), lrtimer(this), rqueue() {
  .
  SIGNAL_THR          = -75.00;
  RE_THR              = 10.00;
  rx_power            = 0.0;
  RSSix               = 0.0;
  RSSM                = 0.0;
  pheromone_count     = 0.0;
  n_energy            = 0.0;
  n_congestion        = 0.0;
  rt_phcount          = 0.0;
  t_x = t_y = t_z     = 0.0;
  d_x = d_y = d_z     = 0.0;
  n_speed             = 0.0;
  n_distance          = 0.0;  .
  .
  logtarget = 0;
  ifqueue = 0;
}
```

Gambar 4.13 Potongan kode inisiasi nilai variabel

4.3.3 Implementasi Pengiriman *Route Request*

Langkah awal yang dilakukan untuk melakukan pengiriman *route request* (RREQ) seperti yang telah dirancang pada subbab 3.4.1 adalah dengan cara melakukan pengecekan rute pada *routing table.* Jika terdapat rute dari *node* asal ke *node* tujuan, maka paket data dikirim melalui rute tersebut. Jika tidak ada, maka proses pencarian rute akan dilakukan. Pada proses ini, *node* asal melakukan pengecekan kekuatan sinyal yang diterima oleh *node* tetangga sebelum mengirimkan paket RREQ.

Nilai kekuatan sinyal yang diterima digunakan untuk melakukan proses seleksi *node* tetangga tahap pertama. Pada NS-2, kekuatan sinyal yang diterima dari suatu paket pada layer *routing* dapat diakses dengan fungsi $p->txinfo_.RxPr$, dengan p adalah *pointer* dari tipe paket. Pada saat pengimplementasian, perlu juga dilakukan konversi satuan daya dari Watt menjadi dBm agar nilai kekuatan sinyal ini dapat dibandingkan dengan nilai *threshold* sinyal yang dinyatakan dalam dBm.

Dari nilai kekuatan sinyal tersebut, *node* asal dapat mengetahui apakah paket RREQ dapat dikirimkan melalui *node* tetangga tersebut atau tidak. Implementasi ini dilakukan pada fungsi $sendRequest()$ yang terdapat pada kode sumber aodv.cc dalam direktori ns2.3.5/aodv. Untuk kode sumber tersebut dapat dilihat pada lampiran A1.

Pada tahap selanjutnya, dilakukan modifikasi pada fungsi $recvRequest()$. Modifikasi pertama, yakni mencari nilai energi residu, kecepatan *node*, koordinat *node* dari pusat (0,0), dan antrian (kepadatan) yang terdapat pada *node* tetangga. Nilai energi residu, kecepatan *node*, dan koordinat *node*, dapat diperoleh dengan memanfaatkan fungsi ($MobileNode *$

) ($Node::get_node_by_address\ (index)\ ) ->$ $energy_model() -> energy()$. Sedangkan untuk mencari nilai antrian (kepadatan) yang terdapat pada *node* tetangga, dilakukan dengan menambahkan $rqueue.queue_length(index)$.

Modifikasi kedua, yaitu melakukan pengecekan terhadap *node* tetangga. Jika *node* tetangga bukan merupakan *node* tujuan, dilakukan penyeleksian *node* tahap kedua berdasarkan membandingkan nilai energi residu, kecepatan *node*, dan jarak *node* relatif dengan *node* tujuan yang telah diperoleh sebelumnya.

Penghitungan *pheromone* dilakukan setelah proses penyeleksian tahap kedua dengan syarat nilai energi residu pada *node* tetangga harus lebih besar dari nilai *threshold* energi. Jika kurang dari nilai *threshold* energi, maka paket RREQ yang diterima akan dibuang. Hasil dari penghitungan *pheromone*, disimpan pada atribut $rq_pheromone_count$ pada *header* paket RREQ dan $rt_pheromone_count$ pada *routing table*. Sedangkan jika *node* tetangga merupakan *node* tujuan, tidak perlu dilakukan penghitungan *pheromone* lagi. Sehingga hanya dilakukan perbaruan $rt_pheromone_count$ yang terdapat pada *routing table*. Untuk kode sumber tersebut dapat dilihat pada lampiran A2.

4.3.4 Implementasi *Ant Colony Optimization* (ACO)

Implementasi penghitungan *pheromone* sebagai bentuk adopsi dari penggunaan algoritma *Ant Colony Optimization* (ACO) dan dilakukan pada fungsi $recvRequest()$ yang terdapat pada kode sumber aodv.cc dalam direktori ns2.35/aodv. Pada penghitungan ini menggunakan empat variabel, yaitu $iRss$

untuk kekuatan sinyal yang diterima, $iEnergi$ untuk energi residu, $queuelen$ untuk antrian (kepadatan) pada *node*, dan $rq->rq_hop_count$ untuk jumlah *hop count*. Cara penghitungan *pheromone* telah dijelaskan pada persamaan 3.1. Kode sumber penghitungan *pheromone* dapat dilihat pada lampiran A2.

4.3.5 Implementasi Pengiriman *Route Request*

Pada tahap selanjutnya setelah menerima *route request* (RREQ), langkah yang harus dilakukan adalah pemilihan rute yang akan dilalui oleh *route reply* (RREP). *Node* tujuan berhak melakukan pemilihan rute berdasarkan nilai *pheromone* yang tertinggi dengan menggunakan fungsi $max()$. Fungsi tersebut digunakan untuk membandingkan hasil penghitungan *pheromone* pada *header* RREQ dengan nilai *pheromone* yang terdapat pada *routing table*. Implementasi pemilihan rute RREP ini dilakukan pada fungsi $sendReply()$ yang terdapat pada kode sumber aodv.cc dalam direktori ns2.35/aodv. Potongan kode untuk proses pemilihan rute RREP berdasarkan nilai *pheromone* tertinggi dapat dilihat pada Gambar 4.14. Kode selengkapnya dapat dilihat pada lampiran A3.

```
void     AODV::sendReply(nsaddr_t     ipdst,
u_int32_t    hop_count,    nsaddr_t    rpdst,
u_int32_t rpseq, u_int32_t lifetime, double
timestamp)
{
  Packet *p = Packet::alloc();
  struct hdr_cmn *ch = HDR_CMN(p);
  struct hdr_ip *ih = HDR_IP(p);
  struct     hdr_aodv_reply     *rp     =
HDR_AODV_REPLY(p);
  aodv_rt_entry          *rt           =
rtable.rt_lookup(ipdst);

  assert(rt);
  .
  .
  rp->rp_dst = rpdst;

  if(rp->rp_dst == index) {
    rp->rp_pheromone_count              =
max(rt_phcount, rt->rt_pheromone_count);
  }
  .
  .
}
```

Gambar 4.14 Potongan modifikasi kode fungsi sendReply()

$rt_phcount$ untuk hasil penghitungan *pheromone* yang terdapat pada *header* paket RREQ. Sedangkan $rt->rt_pheromone_count$ untuk nilai *pheromone* yang terdapat pada *routing table*.

4.4 Implementasi Simulsi pada NS-2

Implementasi simulasi VANETs pada NS-2 diawali dengan pendeskripsian lingkungan simulasi pada sebuah *file* tcl. *File* ini berisikan konfigurasi setiap *node* dan langkah-langkah yang dilakukan selama simulasi. Potongan konfigurasi lingkungan simulasi dapat dilihat pada Gambar 4.15

```
set val(chan)              Channel/WirelessChannel
set val(prop)              Propagation/TwoRayGround
set val(netif)             Phy/WirelessPhy
set val(mac)               Mac/802_11
set val(ifq)               Queue/DropTail/PriQueue
set val(ll)                LL
set val(ant)               Antenna/OmniAntenna
set opt(x)                 910
set opt(y)                 910
set val(ifqlen)            1000
set val(nn)                200
set val(seed)              1.0
set val(adhocRouting)      AODV
set val(stop)              200
set val(cp)                "cbr.txt"
set val(sc)                "setdest.txt"
set val(energy_mod)        EnergyModel
set val(energy_init)       100
set val(tx_power)          8.0
set val(rx_power)          6.0
set val(idle_power)        0.0
set val(sleep_power)        0.0
```

Gambar 4.15 Implementasi simulasi NS-2

Pada konfigurasi dilakukan pemanggilan terhadap *file traffic* yang berisikan konfigurasi *node* asal, *node* tujuan, pengiriman

paket, serta *file* skenario yang berisi pergerakan *node* yang telah di-*generate* oleh SUMO. Kode implementasi pada NS-2 dapat dilihat pada lampiran A.4 Kode Skenario NS-2.

Konfigurasi untuk *file traffic* bisa dilakukan dengan membuat *file* berekstensi .tcl untuk menyimpan konfigurasi tersebut. Pada *file* konfigurasi lingkungan simulasi, *file traffic* tersebut dimasukkan agar dibaca sebagai *file traffic*. Pada konfigurasi ini, ditentukan *node* sumber dan *node* tujuan pengiriman paket. Pengiriman dimulai pada detik ke 2.556. Implementasi konfigurasi *file traffic* untuk simulasi pada NS-2 dapat dilihat pada lampiran A.5.

4.5 Implementasi Metrik Analisis

Hasil dari simulasi skenario pada NS-2 adalah sebuah *trace file* yang digunakan sebagai bahan analisis pengukuran performa dari *routing protocol* yang disimulasikan. Pada Tugas Akhir ini, terdapat lima metrik yang menjadi parameter analisis, yaitu *Packet Delivery Ratio* (PDR), rata-rata *End-to-End Delay* (E2E), *Routing Overhead* (RO), *Forwarded Route Request*, dan *Throughput*.

4.5.1 Implementasi *Packet Delivery Ratio*

Packet Delivery Ratio (PDR) didapatkan dengan cara menghitung setiap baris terjadinya *event* pengiriman dan penerimaan paket data yang dikirim melalui agen pada *trace file*. Skrip awk untuk menghitung PDR dapat dilihat pada lampiran A.7. Skrip menyaring setiap baris yang mengandung *string* AGT karena kata kunci tersebut menunjukan *event* yang berhubungan dengan paket komunikasi data. Penghitungan dilakukan dengan menjumlahkan paket yang dikirimkan dan paket yang diterima dengan menggunakan karakter pada kolom

pertama sebagai *filter*. Kolom pertama menunjukan event yang terjadi dari sebuah paket. Setelah itu nilai PDR dihitung dengan cara persamaan 3.3. Pseudocode untuk menghitung PDR dapat dilihat pada Gambar 4.16.

```
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 t
          received++
     end if
pdr = received / sent
```

Gambar 4.16 Pseudocode penghitungan PDR

4.5.2 Implementasi Rata-rata *End-to-End Delay* (E2E)

Skrip awk untuk menghitung E2E dapat dilihat pada lampiran A.8 Kode Skrip AWK Rata-rata *End-to-End Delay*.

Dalam penghitungan E2E *Delay*, langkah yang digunakan untuk mendapatkan E2E *Delay* hampir sama dengan ketika mencari PDR, hanya saja yang perlu diperhatikan adalah waktu dari sebuah *event* yang tercatat pada kolom ke-2 dengan *filter event* pada kolom ke-4 adalah layer AGT dan *event* pada kolom pertama guna membedakan paket dikirim atau diterima. Setelah seluruh baris yang memenuhi didapatkan, akan dihitung *delay* dari paket dengan mengurangi waktu dari paket diterima dengan waktu dari paket dikirim dengan syarat memiliki *id* paket yang sama.

Setelah mendapatkan *delay* paket, langkah selanjutnya adalah dengan mencari rata-rata dari *delay* tersebut dengan menjumlahkan semua *delay* paket dan membaginya dengan jumlah paket. Cara penghitungan E2E *Delay* telah dijelaskan pada persamaan 3.4. *Pseudocode* untuk menghitung rata-rata E2E *Delay* dapat dilihat pada Gambar 4.17.

```
sum_delay = 0
counter = 0

for i = 1 to the number of rows
     counter++
     if layer == AGT and event == s then
      start_time[packet_id] = time
     else if layer == AGT and event == r then
      end_time[packet_id] = time
     end if
     delay[packet_id] = end_time[packet_id]
- start_time[packet_id]
     sum_delay += delay[packet_id]
e2e = sum_delay / counter
```

Gambar 4.17 Pseudocode penghitungan E2E Delay

4.5.3 Implementasi *Routing Overhead* (RO)

Seperti yang telah dijelaskan sebelumnya, *routing overhead* merupakan jumlah dari paket kontrol *routing* baik itu RREQ, RREP, maupun RERR. Dengan begitu, untuk mendapatkan RO yang perlu dilakukan adalah menjumlahkan tiap paket dengan *filter event sent* pada kolom pertama dan *event layer* RTR pada kolom ke-4. Penghitungan RO telah dijelaskan pada persamaan 3.5. Skrip AWK untuk menghitung

RO dapat dilihat pada lampiran A.9 Kode Skrip AWK *Routing Overhead*. *Pseudocode* untuk menghitung RO dapat dilihat pada Gambar 4.18.

```
ro = 0
for i = 1 to the number of rows
     if in a row contains "s" and RTR
then
        ro++
     end if
```

Gambar 4.18 Pseudocode penghitungan Routing Overhead

4.5.4 Implementasi *Forwarded Route Request*

Forwarded Route Request (RREQ F) adalah jumlah paket *control route request* yang diteruskan per-data paket ke *node* tujuan selama simulasi terjadi. Dan dapat dikatakan bahwa RREQ F adalah jumlah *forwarding node* yang bertugas untuk mengirim ulang (*rebroadcast*) RREQ. Dengan begitu, untuk mendapatkan RREQ F yang perlu dilakukan adalah menjumlahkan tiap paket dengan *filter event sent* pada kolom pertama, *event layer* RTR pada kolom ke-4, dan *event layer route request* pada kolom-25. Penyaringan juga dilakukan pada kolom ke-3 yang menunjukan *node* id. Selama *node* bukan *node* sumber, maka akan terus dilakukan penjumlahan baris yang terdapat pada *file* dengan ekstensi .tr. Perhitungan RREQ F telah dijelaskan pada persamaan 3.6. Skrip AWK untuk menghitung RREQ F dapat dilihat pada lampiran A.10 Kode Skrip AWK *Forwarded Route Request*. *Pseudocode* untuk menghitung RREQ F dapat dilihat pada Gambar 4.19.

```
rreqf = 0
for i = 1 to the number of rows
     if packet is AODV and RREQ and not
source node then
        rreqf++
     end if
```

Gambar 4.19 Pseudocode penghitungan RREQ F

4.5.5 Implementasi *Throughput*

Throughput didapatkan dengan cara menghitung nilai rata-rata dari jumlah paket yang berhasil diterima oleh *node* tujuan selama waktu pengujian berlangsung. Nilai *throughput* dinyatakan dalam *kilobits* per detik. Cara penghitungan *throughput* telah dijelaskan pada persamaan 3.2. Nilai *throughput* dihitung menggunakan skrip AWK yang memproses *trace file* hasil simulasi. Kode implementasi dari penghitungan *throughput* dapat dilihat pada lampiran A.6. Pseudocode untuk menghitung PDR dapat dilihat pada Gambar 4.20.

```
if level == AGT and event == “s” and flag
== CBR
    sent++
    if !startTime or time < startTime
        startTime = time

if level == AGT and event == "r" and flag
== CBR
    receive++
    if time > stopTime
        stopTime = time
    recvdSize = recvdSize + pkt_size

throughput = recvdSize / (stopTime - startTime)
x (8 / 1000)
```

Gambar 4.20 Pseudocode penghitungan Throughput

BAB V
UJI COBA DAN EVALUASI

Pada bab ini akan dilakukan tahap uji coba dan evaluasi sesuai dengan rancangan dan implementasi. Dari hasil yang didapatkan setelah melakukan uji coba, akan dilakukan evaluasi sehingga dapat ditarik kesimpulan pada bab selanjutnya.

5.1 Lingkungan Uji Coba

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

Tabel 5.1 Spesifikasi perangkat

Komponen	**Spesifikasi**
CPU	Intel(R) Core(TM) i7-7700HQ 2.80GHz
Sistem Operasi	Ubuntu 20.04 (Focal Fossa)
Linux Kernel	Linux kernel 5.4
Memori	16.0 GB

Adapun versi perangkat lunak yang digunakan dalam Tugas Akhir ini adalah sebagai berikut:

- SUMO versi 1.6.0 untuk pembuatan skenario mobilitas VANETs.
- JOSM versi 16731 untuk penyuntingan peta OpenStreetMap.
- NS-2 versi 2.35 untuk simulasi skenario VANETs.

Parameter lingkungan uji coba yang digunakan pada NS-2 dapat dilihat pada Tabel 5.2 Pengujian dilakukan dengan menjalankan skenario yang disimulasikan pada NS-2. Dari simulasi tersebut dihasilkan sebuah *trace file* dengan ekstensi .tr yang akan dianalisis dengan bantuan skrip awk untuk mendapatkan *Throughput,* PDR, E2E, RO, dan RREQ F

menggunakan kode yang terdapat pada lampiran A.6 Kode Skrip AWK *Throughput*, A.7 Kode Skrip AWK *Packet Delivery Ratio*, A.8 Kode Skrip AWK Rata-rata *End-to-End Delay*, A.9 Kode Skrip AWK *Routing Overhead*, dan A.10 Kode Skrip AWK *Forwarded Route Request*.

Tabel 5.2 Lingkungan uji coba

No.	Parameter	Spesifikasi
1	Network simulator	NS-2.35
2	*Routing protocol*	AODV
3	Waktu simulasi	200 detik
4	Area simulasi *grid*	910 m x 910 m
5	Area simulasi *real*	Jl. dr. Soetomo, Surabaya (1734 m x 1780 m)
6	Variasi jumlah n*ode*	150, 200, 250, 300
7	Interval pengiriman paket (paket/detik)	4
8	*Node* sumber dan tujuan	6 pasang *node*
9	Radius transmisi	400 m
10	Protokol MAC	IEEE 802.11p
11	Model propagasi	*Two-ray ground*
12	Energi awal	100 Joule
13	Konsumsi energi saat mengirim (tx_power)	8.0 Watt
14	Konsumsi energi saat menerima (rx_power)	6.0 Watt

5.2 Hasil Uji Coba dan Evaluasi

Pada subbab ini akan dipaparkan hasil uji coba dan evaluasi dari implementasi pemutakhiran algoritma ACO (*Enhanced-Ant AODV)* dengan membandingkan performanya dengan AODV

asli (AODV *Default*) dan AODV dengan ACO versi sebelumnya (*Ant-AODV*).

5.2.1 Hasil Uji Coba dan Evaluasi Skenario *Grid*

Pengujian pada skenario *grid* digunakan untuk melihat perbandingan PDR, E2E, RO, RREQ F, dan *Throughput*, antara *routing protocol* AODV asli dan AODV dengan ACO dan AODV dengan ACO yang dimutakhirkan dalam pemilihan *node* yang dapat menerima paket *route request*. Pengambilan data uji pada skenario *grid* dilakukan sebanyak 20 kali dengan skenario mobilitas *random* pada peta *grid* dengan luas area 910 m x 910 m.

Pada lingkungan uji coba skenario *grid* variasi jumlah *node* menggunakan *node* sebanyak 150, 200, 250, dan 300. Adapun spesifikasi lain, yaitu menggunakan interval pengiriman paket sebanyak 4 paket/detik. Maksimal koneksi dari CBR yang digunakan sebanyak 6 koneksi dimana terdapat 6 *node* pengirim dan 6 *node* tujuan. Hasil analisis dapat dilihat pada Tabel 5.3, Tabel 5.4, Tabel 5.5, Tabel 5.6, Tabel 5.7.

Tabel 5.3 Hasil rata-rata PDR skenario grid

Jumlah *Node*	AODV Default	Ant AODV	Enhanced-Ant AODV
150 *node*	0.74265	0.777515	**0.797995**
200 *node*	0.694365	0.740315	**0.74573**
250 *node*	0.64095	0.686175	**0.695855**
300 *node*	0.617755	0.642475	**0.65642**

Tabel 5.4 Hasil rata-rata E2E Delay skenario grid

Jumlah *Node*	AODV Default	Ant AODV	Enhanced-Ant AODV
150 *node*	742.84865	575.1563	**495.8786**
200 *node*	831.17275	727.96565	**713.95665**
250 *node*	1157.19985	976.73605	**820.27565**
300 *node*	1277.0759	1139.2164	**1079.6547**

Tabel 5.5 Hasil rata-rata RO skenario grid

Jumlah *Node*	AODV Default	Ant AODV	Enhanced-Ant AODV
150 *node*	27876.6	21812.4	**21164.3**
200 *node*	34797.05	29064.25	**28702.75**
250 *node*	46459.3	37149.2	**36302.05**
300 *node*	57212.6	47962.55	**46785.45**

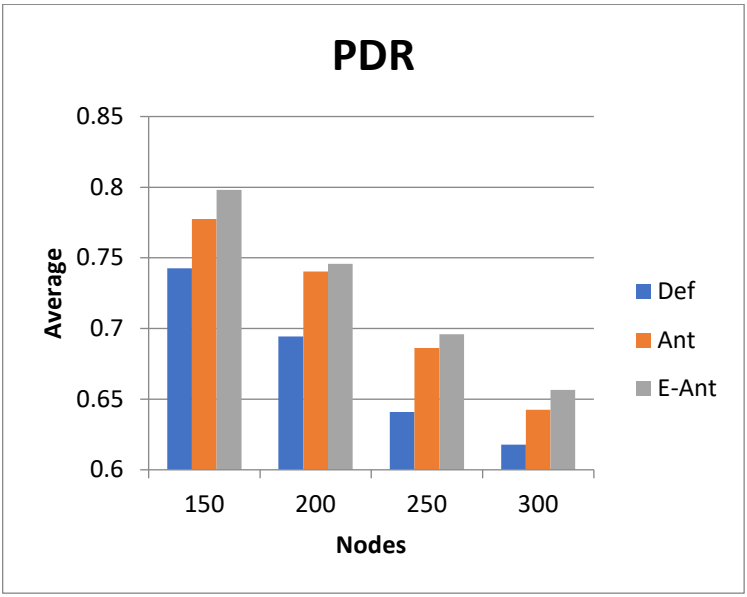
Tabel 5.6 Hasil rata-rata RREQ F skenario grid

Jumlah *Node*	AODV Default	Ant AODV	Enhanced-Ant AODV
150 *node*	26077.75	20449.15	**19828.25**
200 *node*	33030.4	27740.15	**27388.8**
250 *node*	44706.25	35796.15	**34975.45**
300 *node*	55437.65	46585.1	**45442.45**

Tabel 5.7 Hasil rata-rata Throughput skenario grid

Jumlah *Node*	AODV Default	Ant AODV	Enhanced-Ant AODV
150 *node*	63.021095	64.81261	**65.138745**
200 *node*	57.95253	62.09838	**62.54152**
250 *node*	54.312145	58.68655	**59.389115**
300 *node*	52.4731	54.971	**55.942075**

Dari data di atas, dibuat grafik yang merepresentasikan hasil penghitungan *Packet Delivery Ratio*, *End-to-End Delay*, *Routing Overhead*, *Forwarded RREQ*, dan *Throughput*.



Gambar 5.1 Grafik PDR skenario grid

Berdasarkan grafik pada Gambar 5.1, dapat dilihat bahwa secara garis besar, *Enhanced-Ant* AODV mengalami kenaikan performa PDR yang signifikan dibandingkan dengan AODV *Default* dan *Ant*-AODV.

Pada lingkungan dengan jumlah *node* sebanyak 150 *node*, bila dibandingkan dengan AODV *Default*, *Enhanced-Ant* AODV

menghasilkan selisih PDR sebesar 0,055345 atau naik sekitar 7,45% dan bila dibandingkan dengan *Ant* AODV, *Enhanced-Ant* AODV menghasilkan selisih PDR sebesar 0,02048 atau naik sekitar 2,6%. Pada lingkungan simulasi ini *Enhanced-Ant* AODV unggul dibandingkan dua *routing protocol* lainnya.

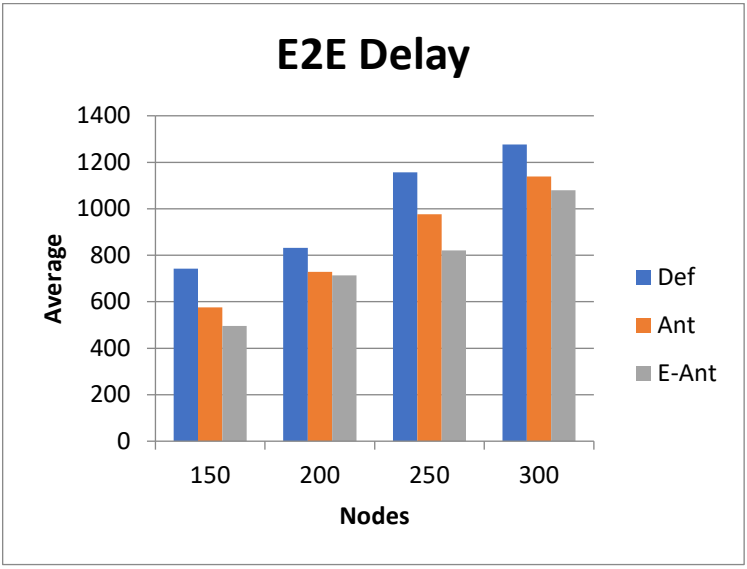
Pada lingkungan dengan jumlah *node* sebanyak 200 *node*, bila dibandingkan dengan AODV *Default*, *Enhanced-Ant* AODV menghasilkan selisih PDR sebesar 0,051365 atau naik sekitar 7,4% dan bila dibandingkan dengan *Ant* AODV, *Enhanced-Ant* AODV menghasilkan selisih PDR sebesar 0,005415 atau naik sekitar 0,73%. Pada lingkungan simulasi ini *Enhanced-Ant* AODV unggul dibandingkan dua *routing protocol* lainnya.

Pada lingkungan dengan jumlah *node* sebanyak 250 *node*, bila dibandingkan dengan AODV *Default*, *Enhanced-Ant* AODV menghasilkan selisih PDR sebesar 0,054905 atau naik sekitar 8,57% dan bila dibandingkan dengan *Ant* AODV, *Enhanced-Ant* AODV menghasilkan selisih PDR sebesar 0,00968 atau naik sekitar 1,41%. Pada lingkungan simulasi ini *Enhanced-Ant* AODV unggul dibandingkan dua *routing protocol* lainnya.

Pada lingkungan dengan jumlah *node* sebanyak 300 *node*, bila dibandingkan dengan AODV *Default*, *Enhanced-Ant* AODV menghasilkan selisih PDR sebesar 0,038665 atau naik sekitar 6,26% dan bila dibandingkan dengan *Ant* AODV, *Enhanced-Ant* AODV menghasilkan selisih PDR sebesar 0,013945 atau naik sekitar 2,17%. Pada lingkungan simulasi ini *Enhanced-Ant* AODV unggul dibandingkan dua *routing protocol* lainnya.

Berdasarkan hasil simulasi pada keempat lingkungan tersebut, dapat dilihat bahwa *Enhanced-Ant* AODV menghasilkan PDR yang selalu lebih baik daripada dua *routing protocol* lainnya.

Rata-rata kenaikan PDR bila dibandingkan dengan AODV *Default* adalah 7,42%, sedangkan bila dibandingkan dengan *Ant* AODV terjadi kenaikan PDR sebesar 1,73%.



Gambar 5.2 Grafik E2E Delay skenario grid

Berdasarkan grafik pada Gambar 5.2, dapat dilihat bahwa secara garis besar, *Enhanced-Ant* AODV mengalami penurunan *End-to-End Delay* yang signifikan dibandingkan dengan AODV *Default* dan *Ant*-AODV.

Pada lingkungan dengan jumlah *node* sebanyak 150 *node*, bila dibandingkan dengan AODV *Default*, *Enhanced-Ant* AODV unggul dalam hal *End-to-End Delay* dengan selisih 246,97005 atau mengalami penurunan *delay* sebesar 49,8%, dan bila dibandingkan dengan *Ant* AODV, *Enhanced-Ant* AODV

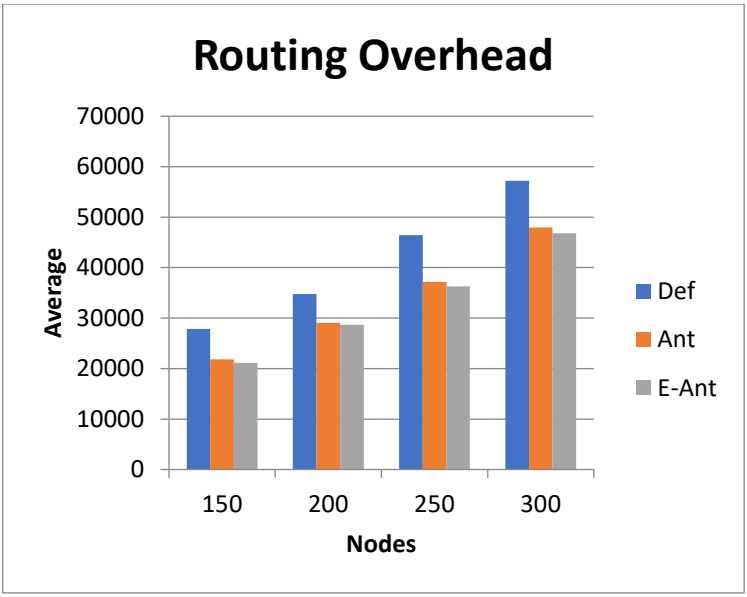
unggul dalam hal *End-to-End Delay* dengan selisih 79,2777 atau mengalami penurunan *delay* sebesar 16%.

Pada lingkungan dengan jumlah *node* sebanyak 200 *node*, bila dibandingkan dengan AODV *Default*, *Enhanced-Ant* AODV unggul dalam hal *End-to-End Delay* dengan selisih 117,2161 atau mengalami penurunan *delay* sebesar 16,42%, dan bila dibandingkan dengan *Ant* AODV, *Enhanced-Ant* AODV unggul dalam hal *End-to-End Delay* dengan selisih 14,009 atau mengalami penurunan *delay* sebesar 1,96%.

Pada lingkungan dengan jumlah *node* sebanyak 250 *node*, bila dibandingkan dengan AODV *Default*, *Enhanced-Ant* AODV unggul dalam hal *End-to-End Delay* dengan selisih 336,9242 atau mengalami penurunan *delay* sebesar 41,07%, dan bila dibandingkan dengan *Ant* AODV, *Enhanced-Ant* AODV unggul dalam hal *End-to-End Delay* dengan selisih 156,4604 atau mengalami penurunan *delay* sebesar 19,07%.

Pada lingkungan dengan jumlah *node* sebanyak 300 *node*, bila dibandingkan dengan AODV *Default*, *Enhanced-Ant* AODV unggul dalam hal *End-to-End Delay* dengan selisih 197,4212 atau mengalami penurunan *delay* sebesar 18,29%, dan bila dibandingkan dengan *Ant* AODV, *Enhanced-Ant* AODV unggul dalam hal *End-to-End Delay* dengan selisih 59,5617 atau mengalami penurunan *delay* sebesar 5,52%.

Bedasarkan simulasi pada keempat lingkungan tersebut, *Enhanced-Ant* AODV selalu lebih unggul dibandingakan kedua *routing protocol* yang lain dalam hal *End-to-End Delay*. Apabila dibandingkan dengan AODV *Default* terjadi rata-rata penurunan *End-to-End Delay* sebesar 31,4%. Apabila dibandingkan dengan *Ant*-AODV terjadi rata-rata penurunan *End-to-End Delay* sebesar 10,64%.



Gambar 5.3 Grafik RO skenario grid

Berdasarkan grafik pada Gambar 5.3, dapat dilihat bahwa *Routing Overhead* pada protokol *Enhanced-Ant* AODV mengalami penurunan yang signifikan bila dibandingkan dengan AODV *Default*, namun kurang signifikan dibandingkan dengan *Ant* AODV meskipun performanya selalu lebih baik.

Pada lingkungan dengan jumlah *node* sebanyak 150 *node*, bila dibandingkan dengan AODV *Default*, *Enhanced-Ant* AODV unggul dalam hal *Routing Overhead* dengan selisih 6712,3 atau mengalami penurunan sebesar 31,72%, dan bila dibandingkan dengan *Ant* AODV, *Enhanced-Ant* AODV unggul dalam hal

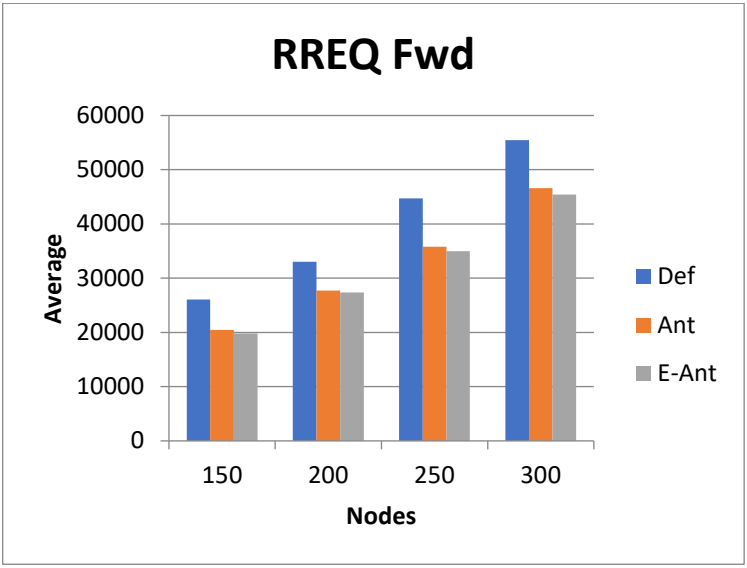
Routing Overhead dengan selisih 648,1 atau mengalami penurunan sebesar 3,06%.

Pada lingkungan dengan jumlah *node* sebanyak 200 *node*, bila dibandingkan dengan AODV *Default*, *Enhanced-Ant* AODV unggul dalam hal *Routing Overhead* dengan selisih 6094,3 atau mengalami penurunan sebesar 21,23%, dan bila dibandingkan dengan *Ant* AODV, *Enhanced-Ant* AODV unggul dalam hal *Routing Overhead* dengan selisih 361,5 atau mengalami penurunan sebesar 1,26%.

Pada lingkungan dengan jumlah *node* sebanyak 250 *node*, bila dibandingkan dengan AODV *Default*, *Enhanced-Ant* AODV unggul dalam hal *Routing Overhead* dengan selisih 10157,25 atau mengalami penurunan sebesar 27,98%, dan bila dibandingkan dengan *Ant* AODV *Enhanced-Ant* AODV unggul dalam hal *Routing Overhead* dengan selisih 847,15 atau mengalami penurunan sebesar 2,33%.

Pada lingkungan dengan jumlah *node* sebanyak 300 *node*, bila dibandingkan dengan AODV *Default*, *Enhanced-Ant* AODV unggul dalam hal *Routing Overhead* dengan selisih 10427,15 atau mengalami penurunan sebesar 22,29%, dan bila dibandingkan dengan *Ant* AODV, *Enhanced-Ant* AODV unggul dalam hal *Routing Overhead* dengan selisih 1177,1 atau mengalami penurunan sebesar 2,51%

Berdasarkan simulasi tersebut dapat disimpulkan bahwa *Enhanced-Ant* AODV selalu unggul dalam hal *Routing Overhead* bila dibandingkan dengan AODV *Default* dengan penurunan rata-rata *Routing Overhead* sebesar 25,805%. Bila dibandingkan dengan *Ant* AODV terjadi penurunan rata-rata *Routing Overhead* hanya sebesar 2,29%.



Gambar 5.4 Grafik RREQ F skenario grid

Berdasarkan grafik pada Gambar 5.4, dapat dilihat bahwa *Routing Overhead* pada protokol *Enhanced-Ant* AODV mengalami penurunan yang signifikan bila dibandingkan dengan AODV *Default*, dan mengalami penurunan yang kurang signifikan bila dibandingkan dengan *Ant* AODV meskipun performanya selalu lebih baik.

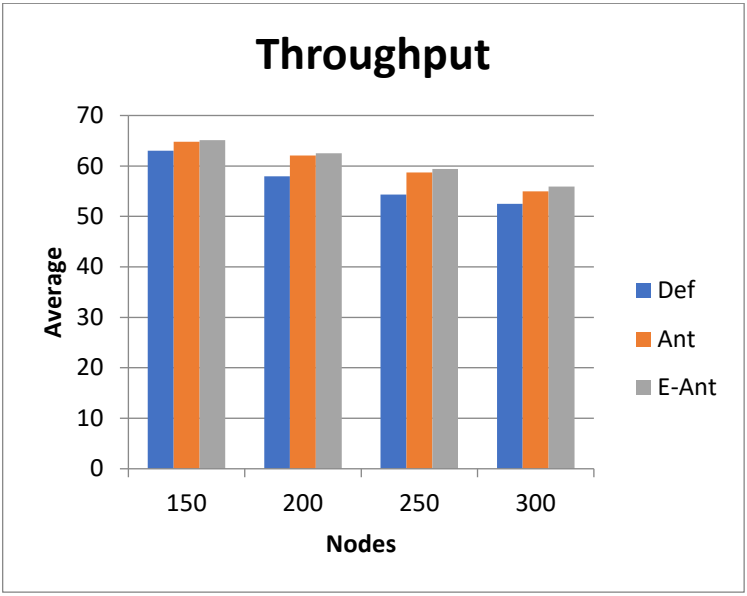
Pada lingkungan dengan jumlah *node* sebanyak 150 *node*, bila dibandingkan dengan AODV *Default*, *Enhanced-Ant* AODV unggul dalam hal *Forwarded RREQ* dengan selisih 6249,5 atau mengalami penurunan sebesar 31,52%, dan bila dibandingkan dengan *Ant* AODV, *Enhanced-Ant* AODV unggul dalam hal *Forwarded RREQ* dengan selisih 620,9 atau mengalami penurunan sebesar 3,1%.

Pada lingkungan dengan jumlah *node* sebanyak 200 *node*, bila dibandingkan dengan AODV *Default*, *Enhanced-Ant* AODV unggul dalam hal *Forwarded RREQ* dengan selisih 5641,6 atau mengalami penurunan sebesar 20,6%, dan bila dibandingkan dengan *Ant* AODV, *Enhanced-Ant* AODV unggul dalam hal *Forwarded RREQ* dengan selisih 351,35 atau mengalami penurunan sebesar 1,28%.

Pada lingkungan dengan jumlah *node* sebanyak 250 *node*, bila dibandingkan dengan AODV *Default*, *Enhanced-Ant* AODV unggul dalam hal *Forwarded RREQ* dengan selisih 9730,8 atau mengalami penurunan sebesar 27,82%, dan bila dibandingkan dengan *Ant* AODV, *Enhanced-Ant* AODV unggul dalam hal *Forwarded RREQ* dengan selisih 820,7 atau mengalami penurunan sebesar 2,34%.

Pada lingkungan dengan jumlah *node* sebanyak 300 *node*, bila dibandingkan dengan AODV *Default*, *Enhanced-Ant* AODV unggul dalam hal *Forwarded RREQ* dengan selisih 9995,2 atau mengalami penurunan sebesar 22%, dan bila dibandingkan dengan *Ant* AODV, *Enhanced-Ant* AODV unggul dalam hal *Forwarded RREQ* dengan selisih 1142,65 atau mengalami penurunan sebesar 2,51%.

Berdasarkan simulasi tersebut dapat disimpulkan bahwa *Enhanced-Ant* AODV selalu unggul dalam hal *Forwarded RREQ* bila dibandingkan dengan AODV *Default* dengan penurunan rata-rata *Forwarded RREQ* sebesar 25,49%. Bila dibandingkan dengan *Ant* AODV terjadi penurunan rata-rata *Forwarded RREQ* sebesar 2,31%.



Gambar 5.5 Grafik Throughput skenario grid

Berdasarkan grafik pada Gambar 5.5, dapat dilihat bahwa secara garis besar, *Enhanced-Ant* AODV mengalami kenaikan performa *Throughput* yang signifikan dibandingkan dengan AODV *Default* dan *Ant*-AODV.

Pada lingkungan dengan jumlah *node* sebanyak 150 *node*, bila dibandingkan dengan AODV *Default*, *Enhanced-Ant* AODV menghasilkan selisih *Throughput* sebesar 2,11765 atau naik sekitar 3,36% dan bila dibandingkan dengan *Ant* AODV, *Enhanced-Ant* AODV menghasilkan selisih *Throughput* sebesar 0,326135 atau naik sekitar 0,5%. Pada lingkungan simulasi ini *Enhanced-Ant* AODV unggul dibandingkan dua *routing protocol* lainnya.

Pada lingkungan dengan jumlah *node* sebanyak 200 *node*, bila dibandingkan dengan AODV *Default*, *Enhanced-Ant* AODV menghasilkan selisih *Throughput* sebesar 4,58899 atau naik sekitar 7,92% dan bila dibandingkan dengan *Ant* AODV, *Enhanced-Ant* AODV menghasilkan selisih *Throughput* sebesar 0,44314 atau naik sekitar 0,71%. Pada lingkungan simulasi ini *Enhanced-Ant* AODV unggul dibandingkan dua *routing protocol* lainnya.

Pada lingkungan dengan jumlah *node* sebanyak 250 *node*, bila dibandingkan dengan AODV *Default*, *Enhanced-Ant* AODV menghasilkan selisih *Throughput* sebesar 5,07697 atau naik sekitar 9,34% dan bila dibandingkan dengan *Ant* AODV, *Enhanced-Ant* AODV menghasilkan selisih *Throughput* sebesar 0,702565 atau naik sekitar 1,2%. Pada lingkungan simulasi ini *Enhanced-Ant* AODV unggul dibandingkan dua *routing protocol* lainnya.

Pada lingkungan dengan jumlah *node* sebanyak 300 *node*, bila dibandingkan dengan AODV *Default*, *Enhanced-Ant* AODV menghasilkan selisih *Throughput* sebesar 3,468975 atau naik sekitar 6,6% dan bila dibandingkan dengan *Ant* AODV, *Enhanced-Ant* AODV menghasilkan selisih *Throughput* sebesar 0,971075 atau naik sekitar 1,77%. Pada lingkungan simulasi ini *Enhanced-Ant* AODV unggul dibandingkan dua *routing protocol* lainnya.

Berdasarkan hasil simulasi pada keempat lingkungan tersebut, dapat dilihat bahwa *Enhanced-Ant* AODV menghasilkan *Throughput* yang selalu lebih baik daripada dua *routing protocol* lainnya. Rata-rata kenaikan *Throughput* bila dibandingkan dengan AODV *Default* adalah 6,65%, sedangkan

bila dibandingkan dengan *Ant* AODV terjadi kenaikan *Throughput* sebesar 1,05%.

5.2.2 Hasil Uji Coba dan Evaluasi Skenario *Real*

Pengujian pada skenario *real* digunakan untuk melihat perbandingan PDR, E2E, RO, RREQ F, dan *Throughput*, antara *routing protocol* AODV asli dan AODV dengan ACO dan AODV dengan ACO yang dimutakhirkan dalam pemilihan *node* yang dapat menerima paket *route request*. Pengambilan data uji pada skenario *grid* dilakukan sebanyak 20 kali dengan skenario mobilitas *random* pada peta *real* dengan luas area 1734 m x 1780 m.

Pada lingkungan uji coba skenario *real* variasi jumlah *node* menggunakan *node* sebanyak 150, 200, 250, dan 300. Adapun spesifikasi lain, yaitu menggunakan interval pengiriman paket sebanyak 4 paket/detik. Maksimal koneksi dari CBR yang digunakan sebanyak 6 koneksi dimana terdapat 6 *node* pengirim dan 6 *node* tujuan. Hasil analisis dapat dilihat pada Tabel 5.8, Tabel 5.9, Tabel 5.10, Tabel 5.11, Tabel 5.12.

Tabel 5.8 Hasil rata-rata PDR skenario real

Jumlah *Node*	**AODV Default**	**Ant AODV**	**Enhanced-Ant AODV**
150 *node*	0.77154	0.792255	**0.80561**
200 *node*	0.78877	0.815655	**0.8193**
250 *node*	0.798785	0.825525	**0.84077**
300 *node*	0.80586	0.82532	**0.837945**

Tabel 5.9 Hasil rata-rata E2E Delay skenario real

Jumlah *Node*	AODV Default	Ant AODV	Enhanced-Ant AODV
150 *node*	650.7805	630.4209	**572.0818**
200 *node*	531.5833	466.4511	**439.5135**
250 *node*	510.4702	457.0301	**421.3254**
300 *node*	522.8311	512.2425	**477.172**

Tabel 5.10 Hasil rata-rata RO skenario real

Jumlah *Node*	AODV Default	Ant AODV	Enhanced-Ant AODV
150 *node*	29956.05	24313.95	**24034.05**
200 *node*	43342.5	33204.15	**32408.25**
250 *node*	49132.65	40330.45	**36941**
300 *node*	53529.1	43346.7	**43275.1**

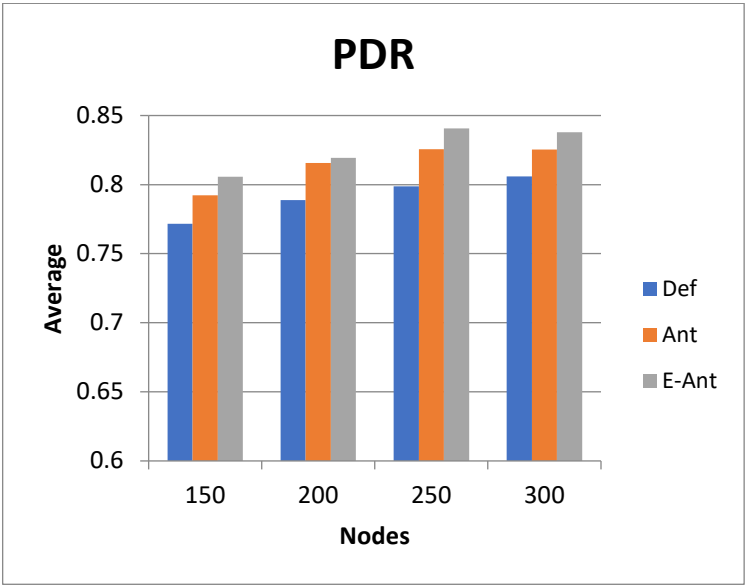
Tabel 5.11 Hasil rata-rata RREQ F skenario real

Jumlah *Node*	AODV Default	Ant AODV	Enhanced-Ant AODV
150 *node*	27373.25	22167.6	**21992**
200 *node*	40005.4	30730.1	**30082**
250 *node*	46035.8	37779.1	**34617.9**
300 *node*	50331.35	40952.1	**40876.5**

Tabel 5.12 Hasil rata-rata Throughput skenario real

Jumlah *Node*	AODV Default	Ant AODV	Enhanced-Ant AODV
150 *node*	45.29127	45.86127	**46.63787**
200 *node*	66.15494	68.39489	**69.27824**
250 *node*	66.86177	68.24426	**68.97183**
300 *node*	65.92102	68.35194	**68.35665**

Dari data di atas, dibuat grafik yang merepresentasikan hasil penghitungan *Packet Delivery Ratio*, *End-to-End Delay*, *Routing Overhead*, *Forwarded RREQ*, dan *Throughput*.



Gambar 5.6 Grafik PDR skenario real

Berdasarkan grafik pada Gambar 5.6, dapat dilihat bahwa secara garis besar, *Enhanced-Ant* AODV mengalami kenaikan performa PDR yang signifikan dibandingkan dengan AODV *Default* dan *Ant*-AODV.

Pada lingkungan dengan jumlah *node* sebanyak 150 *node*, bila dibandingkan dengan AODV *Default*, *Enhanced-Ant* AODV

menghasilkan selisih PDR sebesar 0,03407 atau naik sekitar 4,41% dan bila dibandingkan dengan *Ant* AODV, *Enhanced-Ant* AODV menghasilkan selisih PDR sebesar 0,013355 atau naik sekitar 1,69%. Pada lingkungan simulasi ini *Enhanced-Ant* AODV unggul dibandingkan dua *routing protocol* lainnya.

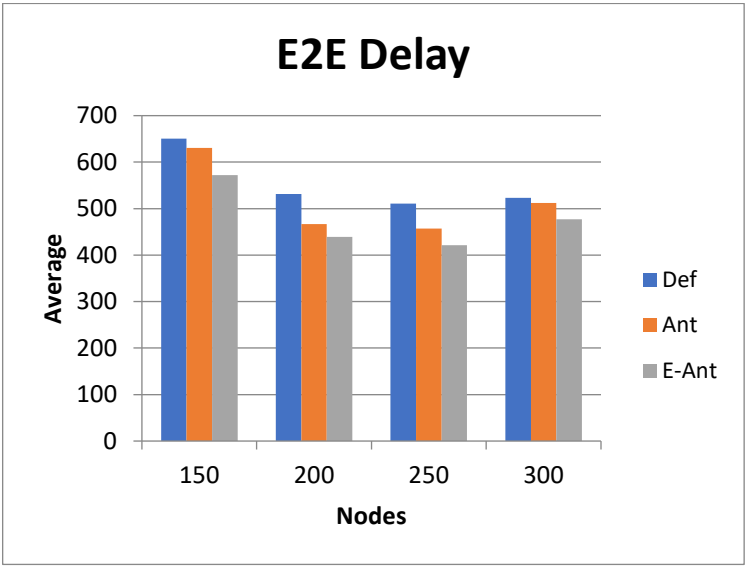
Pada lingkungan dengan jumlah *node* sebanyak 200 *node*, bila dibandingkan dengan AODV *Default*, *Enhanced-Ant* AODV menghasilkan selisih PDR sebesar 0,03053 atau naik sekitar 3,87% dan bila dibandingkan dengan *Ant* AODV, *Enhanced-Ant* AODV menghasilkan selisih PDR sebesar 0,003645 atau naik sekitar 0,45%. Pada lingkungan simulasi ini *Enhanced-Ant* AODV unggul dibandingkan dua *routing protocol* lainnya.

Pada lingkungan dengan jumlah *node* sebanyak 250 *node*, bila dibandingkan dengan AODV *Default*, *Enhanced-Ant* AODV menghasilkan selisih PDR sebesar 0,041985 atau naik sekitar 5,26% dan bila dibandingkan dengan *Ant* AODV, *Enhanced-Ant* AODV menghasilkan selisih PDR sebesar 0,015245 atau naik sekitar 1,85%. Pada lingkungan simulasi ini *Enhanced-Ant* AODV unggul dibandingkan dua *routing protocol* lainnya.

Pada lingkungan dengan jumlah *node* sebanyak 300 *node*, bila dibandingkan dengan AODV *Default*, *Enhanced-Ant* AODV menghasilkan selisih PDR sebesar 0,032085 atau naik sekitar 3,98% dan bila dibandingkan dengan *Ant* AODV, *Enhanced-Ant* AODV menghasilkan selisih PDR sebesar 0,012625 atau naik sekitar 1,53%. Pada lingkungan simulasi ini *Enhanced-Ant* AODV unggul dibandingkan dua *routing protocol* lainnya.

Berdasarkan hasil simulasi pada keempat lingkungan tersebut, dapat dilihat bahwa *Enhanced-Ant* AODV menghasilkan PDR yang selalu lebih baik daripada dua *routing protocol* lainnya. Rata-rata kenaikan PDR bila dibandingkan dengan AODV

Default adalah 4,38%, sedangkan bila dibandingkan dengan *Ant* AODV terjadi kenaikan PDR sebesar 1,38%.



Gambar 5.7 Grafik E2E Delay skenario real

Berdasarkan grafik pada Gambar 5.7, dapat dilihat bahwa secara garis besar, *Enhanced-Ant* AODV mengalami penurunan *End-to-End Delay* yang signifikan dibandingkan dengan AODV *Default* dan *Ant*-AODV.

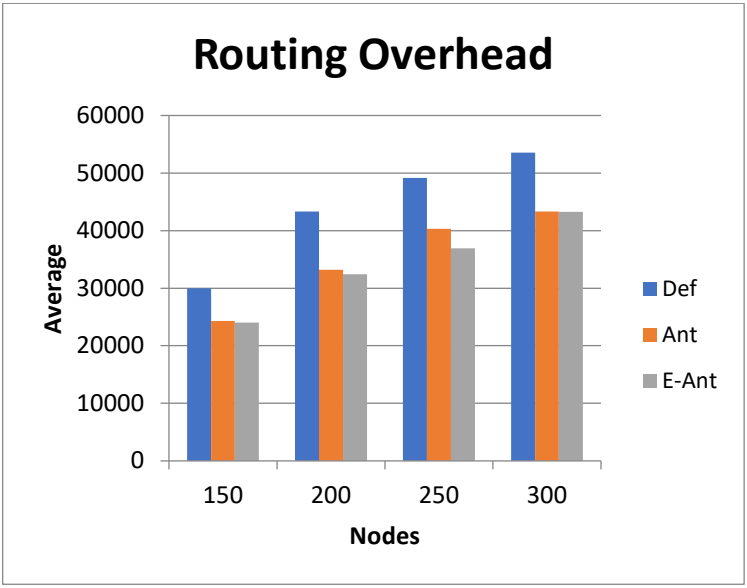
Pada lingkungan dengan jumlah *node* sebanyak 150 *node*, bila dibandingkan dengan AODV *Default*, *Enhanced-Ant* AODV unggul dalam hal *End-to-End Delay* dengan selisih 78,6987 atau mengalami penurunan *delay* sebesar 13,76%, dan bila dibandingkan dengan *Ant* AODV, *Enhanced-Ant* AODV unggul dalam hal *End-to-End Delay* dengan selisih 58,33905 atau mengalami penurunan *delay* sebesar 10,2%.

Pada lingkungan dengan jumlah *node* sebanyak 200 *node*, bila dibandingkan dengan AODV *Default*, *Enhanced-Ant* AODV unggul dalam hal *End-to-End Delay* dengan selisih 92,06985 atau mengalami penurunan *delay* sebesar 20,95%, dan bila dibandingkan dengan *Ant* AODV, *Enhanced-Ant* AODV unggul dalam hal *End-to-End Delay* dengan selisih 26,9376 atau mengalami penurunan *delay* sebesar 6,13%.

Pada lingkungan dengan jumlah *node* sebanyak 250 *node*, bila dibandingkan dengan AODV *Default*, *Enhanced-Ant* AODV unggul dalam hal *End-to-End Delay* dengan selisih 89,14485 atau mengalami penurunan *delay* sebesar 21,16%, dan bila dibandingkan dengan *Ant* AODV, *Enhanced-Ant* AODV unggul dalam hal *End-to-End Delay* dengan selisih 35,70475 atau mengalami penurunan *delay* sebesar 8,47%.

Pada lingkungan dengan jumlah *node* sebanyak 300 *node*, bila dibandingkan dengan AODV *Default*, *Enhanced-Ant* AODV unggul dalam hal *End-to-End Delay* dengan selisih 45,65915 atau mengalami penurunan *delay* sebesar 9,57%, dan bila dibandingkan dengan *Ant* AODV, *Enhanced-Ant* AODV unggul dalam hal *End-to-End Delay* dengan selisih 35,07055 atau mengalami penurunan *delay* sebesar 7,35%.

Bedasarkan simulasi pada keempat lingkungan tersebut, *Enhanced-Ant* AODV selalu lebih unggul dibandingakan kedua *routing protocol* yang lain dalam hal *End-to-End Delay*. Apabila dibandingkan dengan AODV *Default* terjadi rata-rata penurunan *End-to-End Delay* sebesar 17,79%. Apabila dibandingkan dengan *Ant*-AODV terjadi rata-rata penurunan *End-to-End Delay* sebesar 8,04%.



Gambar 5.8 Grafik RO skenario real

Berdasarkan grafik pada Gambar 5.8, dapat dilihat bahwa *Routing Overhead* pada protokol *Enhanced-Ant* AODV mengalami penurunan yang signifikan bila dibandingkan dengan AODV *Default*, namun kurang signifikan dibandingkan dengan *Ant* AODV meskipun performanya selalu lebih baik.

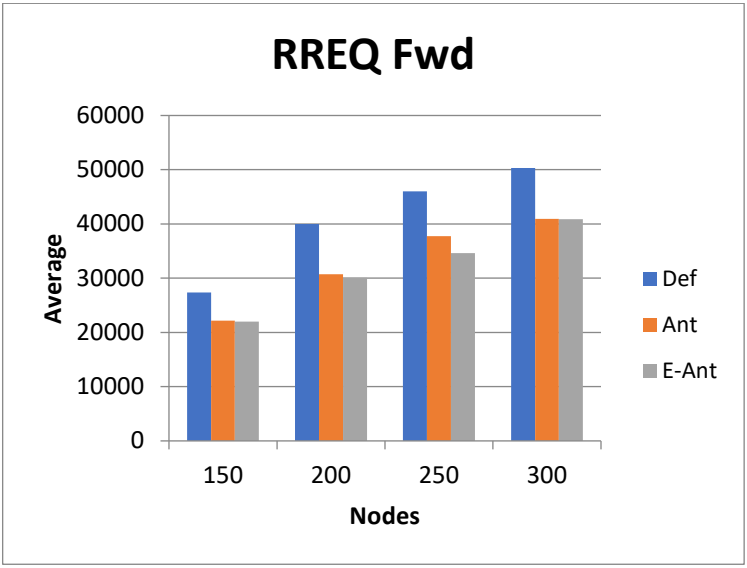
Pada lingkungan dengan jumlah *node* sebanyak 150 *node*, bila dibandingkan dengan AODV *Default*, *Enhanced-Ant* AODV unggul dalam hal *Routing Overhead* dengan selisih 5922 atau mengalami penurunan sebesar 24,64%, dan bila dibandingkan dengan *Ant* AODV, *Enhanced-Ant* AODV unggul dalam hal *Routing Overhead* dengan selisih 279,9 atau mengalami penurunan sebesar 1,16%.

Pada lingkungan dengan jumlah *node* sebanyak 200 *node*, bila dibandingkan dengan AODV *Default*, *Enhanced-Ant* AODV unggul dalam hal *Routing Overhead* dengan selisih 10934,25 atau mengalami penurunan sebesar 33,73%, dan bila dibandingkan dengan *Ant* AODV, *Enhanced-Ant* AODV unggul dalam hal *Routing Overhead* dengan selisih 759,9 atau mengalami penurunan sebesar 2,46%.

Pada lingkungan dengan jumlah *node* sebanyak 250 *node*, bila dibandingkan dengan AODV *Default*, *Enhanced-Ant* AODV unggul dalam hal *Routing Overhead* dengan selisih 12191,65 atau mengalami penurunan sebesar 33%, dan bila dibandingkan dengan *Ant* AODV *Enhanced-Ant* AODV unggul dalam hal *Routing Overhead* dengan selisih 3389,45 atau mengalami penurunan sebesar 9,18%.

Pada lingkungan dengan jumlah *node* sebanyak 300 *node*, bila dibandingkan dengan AODV *Default*, *Enhanced-Ant* AODV unggul dalam hal *Routing Overhead* dengan selisih 10254 atau mengalami penurunan sebesar 23,69%, dan bila dibandingkan dengan *Ant* AODV, *Enhanced-Ant* AODV unggul dalam hal *Routing Overhead* dengan selisih 71,6 atau mengalami penurunan sebesar 0,17%

Berdasarkan simulasi tersebut dapat disimpulkan bahwa *Enhanced-Ant* AODV selalu unggul dalam hal *Routing Overhead* bila dibandingkan dengan AODV *Default* dengan penurunan rata-rata *Routing Overhead* sebesar 28,77%. Bila dibandingkan dengan *Ant* AODV terjadi penurunan rata-rata *Routing Overhead* hanya sebesar 3,24%.



Gambar 5.9 Grafik RREQ F skenario real

Berdasarkan grafik pada Gambar 5.9, dapat dilihat bahwa *Routing Overhead* pada protokol *Enhanced-Ant* AODV mengalami penurunan yang signifikan bila dibandingkan dengan AODV *Default*, dan mengalami penurunan yang kurang signifikan bila dibandingkan dengan *Ant* AODV meskipun performanya selalu lebih baik.

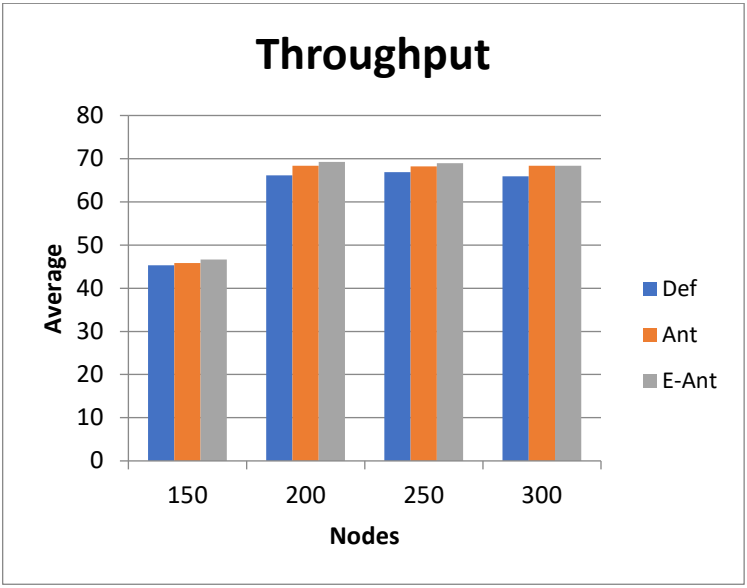
Pada lingkungan dengan jumlah *node* sebanyak 150 *node*, bila dibandingkan dengan AODV *Default*, *Enhanced-Ant* AODV unggul dalam hal *Forwarded RREQ* dengan selisih 5381,25 atau mengalami penurunan sebesar 24,47%, dan bila dibandingkan dengan *Ant* AODV, *Enhanced-Ant* AODV unggul dalam hal *Forwarded RREQ* dengan selisih 175,6 atau mengalami penurunan sebesar 0,8%.

Pada lingkungan dengan jumlah *node* sebanyak 200 *node*, bila dibandingkan dengan AODV *Default*, *Enhanced-Ant* AODV unggul dalam hal *Forwarded RREQ* dengan selisih 9923,4 atau mengalami penurunan sebesar 33%, dan bila dibandingkan dengan *Ant* AODV, *Enhanced-Ant* AODV unggul dalam hal *Forwarded RREQ* dengan selisih 648,1 atau mengalami penurunan sebesar 2,15%.

Pada lingkungan dengan jumlah *node* sebanyak 250 *node*, bila dibandingkan dengan AODV *Default*, *Enhanced-Ant* AODV unggul dalam hal *Forwarded RREQ* dengan selisih 11417,9 atau mengalami penurunan sebesar 33%, dan bila dibandingkan dengan *Ant* AODV, *Enhanced-Ant* AODV unggul dalam hal *Forwarded RREQ* dengan selisih 3161,2 atau mengalami penurunan sebesar 9,13%.

Pada lingkungan dengan jumlah *node* sebanyak 300 *node*, bila dibandingkan dengan AODV *Default*, *Enhanced-Ant* AODV unggul dalam hal *Forwarded RREQ* dengan selisih 9454,85 atau mengalami penurunan sebesar 23,13%, dan bila dibandingkan dengan *Ant* AODV, *Enhanced-Ant* AODV unggul dalam hal *Forwarded RREQ* dengan selisih 75,6 atau mengalami penurunan sebesar 0,18%.

Berdasarkan simulasi tersebut dapat disimpulkan bahwa *Enhanced-Ant* AODV selalu unggul dalam hal *Forwarded RREQ* bila dibandingkan dengan AODV *Default* dengan penurunan rata-rata *Forwarded RREQ* sebesar 28,4%. Bila dibandingkan dengan *Ant* AODV terjadi penurunan rata-rata *Forwarded RREQ* sebesar 3,07%.



Gambar 5.10 Grafik Throughput skenario real

Berdasarkan grafik pada Gambar 5.10, dapat dilihat bahwa secara garis besar, *Enhanced-Ant* AODV mengalami kenaikan performa *Throughput* yang signifikan dibandingkan dengan AODV *Default* dan *Ant*-AODV.

Pada lingkungan dengan jumlah *node* sebanyak 150 *node*, bila dibandingkan dengan AODV *Default*, *Enhanced-Ant* AODV menghasilkan selisih *Throughput* sebesar 1,3466 atau naik sekitar 2,97% dan bila dibandingkan dengan *Ant* AODV, *Enhanced-Ant* AODV menghasilkan selisih *Throughput* sebesar 0.7766 atau naik sekitar 1,69%. Pada lingkungan simulasi ini *Enhanced-Ant* AODV unggul dibandingkan dua *routing protocol* lainnya.

Pada lingkungan dengan jumlah *node* sebanyak 200 *node*, bila dibandingkan dengan AODV *Default*, *Enhanced-Ant* AODV menghasilkan selisih *Throughput* sebesar 3,1233 atau naik sekitar 4,72% dan bila dibandingkan dengan *Ant* AODV, *Enhanced-Ant* AODV menghasilkan selisih *Throughput* sebesar 0,883345 atau naik sekitar 1,29%. Pada lingkungan simulasi ini *Enhanced-Ant* AODV unggul dibandingkan dua *routing protocol* lainnya.

Pada lingkungan dengan jumlah *node* sebanyak 250 *node*, bila dibandingkan dengan AODV *Default*, *Enhanced-Ant* AODV menghasilkan selisih *Throughput* sebesar 2,110055 atau naik sekitar 3,16% dan bila dibandingkan dengan *Ant* AODV, *Enhanced-Ant* AODV menghasilkan selisih *Throughput* sebesar 0,727565 atau naik sekitar 1,06%. Pada lingkungan simulasi ini *Enhanced-Ant* AODV unggul dibandingkan dua *routing protocol* lainnya.

Pada lingkungan dengan jumlah *node* sebanyak 300 *node*, bila dibandingkan dengan AODV *Default*, *Enhanced-Ant* AODV menghasilkan selisih *Throughput* sebesar 2,435625 atau naik sekitar 3,69% dan bila dibandingkan dengan *Ant* AODV, *Enhanced-Ant* AODV menghasilkan selisih *Throughput* sebesar 0,00471 atau naik sekitar 0,01%. Pada lingkungan simulasi ini *Enhanced-Ant* AODV unggul dibandingkan dua *routing protocol* lainnya.

Berdasarkan hasil simulasi pada keempat lingkungan tersebut, dapat dilihat bahwa *Enhanced-Ant* AODV menghasilkan *Throughput* yang selalu lebih baik daripada dua *routing protocol* lainnya. Rata-rata kenaikan *Throughput* bila dibandingkan dengan AODV *Default* adalah 3,64%, sedangkan

bila dibandingkan dengan *Ant* AODV terjadi kenaikan *Throughput* sebesar 1,01%.

(Halaman ini sengaja dikosongkan).

BAB VI
KESIMPULAN DAN SARAN

Pada Bab ini akan diberikan kesimpulan yang diperoleh dari Tugas Akhir yang telah dikerjakan dan saran tentang pengembangan dari Tugas Akhir ini yang dapat dilakukan di masa yang akan datang.

6.1 Kesimpulan

Kesimpulan yang diperoleh pada uji coba dan evaluasi Tugas Akhir ini adalah sebagai berikut:

1. Modifikasi *routing protocol* AODV dengan menggunakan algoritma *Ant Colony Optimization* yang dimutakhirkan telah berhasil membatasi jumlah *forwarding node* yang bertugas untuk *rebroadcast* paket RREQ diihat dari metrik *Forwarded Route Request* (RREQ F).

2. Penggunaan algoritma *Ant Colony Optimization* yang dimutakhirkan terhadap performa *routing protocol AODV* menunjukkan adanya perubahan yang cukup signifikan pada skenario *real* dengan rata-rata kenaikan PDR sebesar 4,38%, penurunan *End-to-End Delay* sebesar 17,79%, rata-rata penurunan *Routing Overhead* sebesar 28,77%, rata-rata penurunan *Forwarded Route Request* sebesar 28,4% dan rata-rata kenaikan *Throughput* sebesar 3,64%.

6.2 Saran

Saran yang diberikan dari hasil uji coba dan evaluasi adalah sebagai berikut:

1. Lebih banyak uji coba yang dilakukan untuk mendapatkan hasil yang akurat.

2. Perhitungkan pula arah gerak *node* apakah mendekati atau menjauhi *node* tujuan

DAFTAR PUSTAKA

[1] I. M. Misbahuddin, "Modifikasi Routing Protocol Ad-hoc On-demand Distance Vector (AODV) Menggunakan Algoritma Ant Colony Optimization (ACO) untuk Pemilihan Rute di Lingkungan VANETs," Tugas Akhir, Departemen Teknik Informatika, Institut Teknologi Sepuluh Nopember, Surabaya, 2019.

[2] D. Sarkar, S. Choudhury and A. Majumder, "Enhanced-Ant-AODV for Optimal Route Selection in Mobile Ad-Hoc Network," *Journal of King Saud University - Computer and Information Sciences,* 2018.

[3] S. Zeadally, R. Hunt, Y. Chen, A. Irwin and A. Hassan, "Vehicular Ad Hoc Networks (VANETS): Status, Results, and Challenges," *Telecommunication Systems,* vol. 50, no. 4, pp. 217-241, 2012.

[4] S. Al-Sultan, M. Al-Doori, A. Al-Bayatti and H. Zedan, "A comprehensive survey on vehicular ad hoc network," *Journal of Network and Computer Application,* vol. 37, no. 1, pp. 380-392, 2014.

[5] J. Gozalvez, M. Sepulcre and R. Bauza, "IEEE 802.11p Vehicle to Infrastructure Communications in Urban Environments," *IEEE Communications Magazine,* vol. 50, no. 5, pp. 176-183, 2012.

[6] C. Campolo, A. Molinaro and R. Scopigno, Vehicular ad hoc Networks: Standards, Solutions, and Research, Heidelberg: Springer International Publishing, 2015.

[7] N. Gupta, A. Prakash and R. Tripathi, "Medium Access Control Protocols for Safety Applications in Vehicular Ad-hoc Network: A Classification and comprehensive survey," *Vehicular Communications,* vol. 2, no. 4, pp. 223-237, 2015.

[8] A. Laouiti, A. Qayyum and M. N. Mohamad Saad, Vehicular Ad-Hoc Networks for Smart Cities, Springer Singapore, 2016.

[9] P. K. Singh and G. Sharma, "An Efficient Prevention of Black Hole Problem in AODV Routing Protocol in MANET," in *11th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, Liverpool, 2012.

[10] L. Klein-Berndt, "A Quick Guide to AODV Routing," National Institute of Standards and Technology, Gaithersburg, 2012.

[11] L. A. Hassnawi, R. B. Ahmad, A. Yahya, S. A. Aljunid and M. Elshaikh, "Performance Analysis of Various Routing Protocols for Motorway Surveillance System Cameras' Network," *International Journal of Computer Science,* vol. 9, pp. 7 - 21, 2012.

[12] M. Dorigo, V. Maniezzo and A. Colorni, "Positive Feedback as a Search Strategy," *Tech. Rep., 91-016,*

Dipartimento di Elettronica, Politecnico di Milano, Italy, 1991.

[13] M. Dorigo, "Optimization, Learning and Natural Algorithm," Disertasi Ph.D., Dipartimento di Elettronica, Politecnico di Milano, Milano, Italy, 1992.

[14] M. Dorigo, V. Maniezzo and A. Colorni, "Ant System: Optimization by a Colony of Cooperating Agents," *IEEE Transactions on Systems, Man, and Cybernetics - Part B,* vol. 26, no. 1, pp. 29-41, 1996.

[15] J. E. Zablotny, Encyclopedia of Insects (Second Edition), New York: Elvesier, 2009.

[16] F. Nutrihadi, R. Anggoro and R. M. Ijtihadie, "Studi Kinerja VANET Scenario Generators: SUMO dan VanetMobisim untuk Implementasi Routing Protocol AODV menggunakan Network Simulator 2 (NS-2)," *Jurnal Teknik ITS,* vol. 5, no. 1, p. 21, 2016.

[17] N. M. Mittal and S. Choudhary, "Comparative Study of Simulators for Vehicular Ad-hoc Networks (VANETs)," *International Journal of Emerging Technology and Advanced Engineering,* vol. 4, no. 4, 2014.

[18] "OpenStreetMap," OpenStreetMap, [Online]. Available: https://www.openstreetmap.org/about. [Accessed 6 Maret 2018].

[19] "JOSM," JOSM, [Online]. Available: http://wiki.openstreetmap.org/wiki/JOSM. [Accessed 6 Maret 2020].

[20] "SUMO," SUMO, [Online]. Available: http://www.sumo.dlr.de/userdoc/Sumo_at_a_Glance.html. [Accessed 6 Maret 2018].

LAMPIRAN

A.1 Kode Fungsi sendRequest()

```
void
AODV::sendRequest(nsaddr_t dst) {
Packet *p = Packet::alloc();
struct hdr_cmn *ch = HDR_CMN(p);
struct hdr_ip *ih = HDR_IP(p);
struct hdr_aodv_request *rq = HDR_AODV_REQUEST(p);
aodv_rt_entry *rt = rtable.rt_lookup(dst);

          t_node    =                  (MobileNode      *)
(Node::get_node_by_address(index));
          n_energy  =   t_node->energy_model()->energy();

          rx_power = p->txinfo_.RxPr;    //RECEIVE SIGNAL
          if (rx_power == 0){
                    RSSix = -200;
          }
          else{
                    RSSix = 10*log10(rx_power) + 30;
          }
          n_congestion       =
rqueue.queue_length(index);
          if(n_congestion == 0) {
                    n_congestion= 1;
          }

          if(RSSix > SIGNAL_THR){
                    RSSM    =    1-((100+SIGNAL_THR)    /
(100+RSSix));
          }else{
                    RSSM = 0.0;
          }

 assert(rt);

 if (rt->rt_flags == RTF_UP) {
   assert(rt->rt_hops != INFINITY2);
   Packet::free((Packet *)p);
   return;
 }

 if (rt->rt_req_timeout > CURRENT_TIME) {
   Packet::free((Packet *)p);
   return;
 }
```

```
 if (rt->rt_req_cnt > RREQ_RETRIES) {
   rt->rt_req_timeout = CURRENT_TIME + MAX_RREQ_TIMEOUT;
   rt->rt_req_cnt = 0;

 Packet *buf_pkt;
   while ((buf_pkt = rqueue.deque(rt->rt_dst))) {
       drop(buf_pkt, DROP_RTR_NO_ROUTE);
   }
   Packet::free((Packet *)p);
   return;
 }

  if(RSSix <= SIGNAL_THR){
    Packet::free((Packet *)p);
    return;
  }


 rt->rt_req_last_ttl    =    max(rt->rt_req_last_ttl,rt-
>rt_last_hop_count);
 if (0 == rt->rt_req_last_ttl) {
   ih->ttl_ = TTL_START;
 }
 else {
   if (rt->rt_req_last_ttl < TTL_THRESHOLD)
     ih->ttl_ = rt->rt_req_last_ttl + TTL_INCREMENT;
   else {
     ih->ttl_ = NETWORK_DIAMETER;
     rt->rt_req_cnt += 1;
   }
 }

 rt->rt_req_last_ttl = ih->ttl_;

 rt->rt_req_timeout   =   2.0   *   (double)   ih->ttl_   *
PerHopTime(rt);
 if (rt->rt_req_cnt > 0)
   rt->rt_req_timeout *= rt->rt_req_cnt;
 rt->rt_req_timeout += CURRENT_TIME;

 rt->rt_rssm = RSSM;
 rt->rt_rem  = n_energy;
 rt->rt_cm   = n_congestion;
 rt->rt_pheromone_count = 0;

 if     (rt->rt_req_timeout     >     CURRENT_TIME     +
MAX_RREQ_TIMEOUT)
```

```
   rt->rt_req_timeout = CURRENT_TIME + MAX_RREQ_TIMEOUT;
 rt->rt_expire = 0;

 ch->ptype() = PT_AODV;
 ch->size() = IP_HDR_LEN + rq->size();
 ch->iface() = -2;
 ch->error() = 0;
 ch->addr_type() = NS_AF_NONE;
 ch->prev_hop_ = index;

 ih->saddr() = index;
 ih->daddr() = IP_BROADCAST;
 ih->sport() = RT_PORT;
 ih->dport() = RT_PORT;

 rq->rq_type = AODVTYPE_RREQ;
 rq->rq_hop_count = 1;
 rq->rq_bcast_id = bid++;
 rq->rq_dst = dst;
 rq->rq_dst_seqno = (rt ? rt->rt_seqno : 0);
 rq->rq_src = index;
 rq->rq_pheromone_count      =         0.0;
 seqno += 2;
 assert ((seqno%2) == 0);
 rq->rq_src_seqno = seqno;
 rq->rq_timestamp = CURRENT_TIME;

 Scheduler::instance().schedule(target_, p, 0.);
}
```

A.2 Kode Fungsi recvRequest()

```
AODV::recvRequest(Packet *p) {
struct hdr_ip *ih = HDR_IP(p);
struct hdr_aodv_request *rq = HDR_AODV_REQUEST(p);
aodv_rt_entry *rt;

       t_node  =                    (MobileNode       *)
(Node::get_node_by_address(index));
       n_energy       =         t_node->energy_model()-
>energy();
       rx_power = p->txinfo_.RxPr;    //RECEIVE SIGNAL

       if (rx_power == 0){
              RSSix = -200; //-(infinite)
       } else {
```

```
                RSSix = 10*log10(rx_power) + 30;  // Watt
to dBm: 1 watt = 10 log 10 + 30 dBm
        }

        n_congestion   = rqueue.queue_length(index);
        if(n_congestion == 0) {
                n_congestion= 1;
        }

        if(RSSix > SIGNAL_THR){
                RSSM     =     1-((100+SIGNAL_THR)     /
(100+RSSix));
        }else{
                RSSM = 0.0;
        }

  if(rq->rq_src == index) {
    Packet::free(p);
    return;
  }

 if (id_lookup(rq->rq_src, rq->rq_bcast_id)) {

   Packet::free(p);
   return;
 }

 id_insert(rq->rq_src, rq->rq_bcast_id);
 aodv_rt_entry *rt0;

   rt0 = rtable.rt_lookup(rq->rq_src);
   if(rt0 == 0) {
     rt0 = rtable.rt_add(rq->rq_src);
   }

   rt0->rt_expire = max(rt0->rt_expire, (CURRENT_TIME +
REV_ROUTE_LIFE));

   if ( (rq->rq_src_seqno > rt0->rt_seqno ) ||
        ((rq->rq_src_seqno == rt0->rt_seqno) &&
         (rq->rq_hop_count < rt0->rt_hops)) ) {
        rt_update(rt0,       rq->rq_src_seqno,       rq-
>rq_hop_count,  rq->rq_pheromone_count,  RSSM,  n_energy,
n_congestion,      ih->saddr(),      max(rt0->rt_expire,
(CURRENT_TIME + REV_ROUTE_LIFE)) );
     if (rt0->rt_req_timeout > 0.0) {
       rt0->rt_req_cnt = 0;
       rt0->rt_req_timeout = 0.0;
```

```
        rt0->rt_req_last_ttl = rq->rq_hop_count;
        rt0->rt_expire       =       CURRENT_TIME       +
ACTIVE_ROUTE_TIMEOUT;
      }
      assert (rt0->rt_flags == RTF_UP);
      Packet *buffered_pkt;
      while ((buffered_pkt = rqueue.deque(rt0->rt_dst)))
{
        if (rt0 && (rt0->rt_flags == RTF_UP)) {
        assert(rt0->rt_hops != INFINITY2);
          forward(rt0, buffered_pkt, NO_DELAY);
        }
      }
    }

 rt = rtable.rt_lookup(rq->rq_dst);

 if(rq->rq_dst == index) {
   pheromone_count = (RSSM*n_energy) / (n_congestion*rq-
>rq_hop_count);

   if(rq->rq_pheromone_count!=0){
     rt_phcount     =     rq->rq_pheromone_count     +
pheromone_count;
   }else{
     rt_phcount = pheromone_count;
   }
   rq->rq_pheromone_count = rt_phcount;
   rt0->rt_pheromone_count = rt_phcount;
   seqno = max(seqno, rq->rq_dst_seqno)+1;
   if (seqno%2) seqno++;

   sendReply(rq->rq_src,
             1,
             index,
             seqno,
             MY_ROUTE_TIMEOUT,
             rq->rq_timestamp,
             rq->rq_pheromone_count);

   Packet::free(p);
 }


 else if (rt && (rt->rt_hops != INFINITY2) &&
               (rt->rt_seqno >= rq->rq_dst_seqno) ) {

   assert(rq->rq_dst == rt->rt_dst);
```

```
   sendReply(rq->rq_src,
             rt->rt_hops + 1,
             rq->rq_dst,
             rt->rt_seqno,
             (u_int32_t) (rt->rt_expire - CURRENT_TIME),
             rq->rq_timestamp,
             rq->rq_pheromone_count);
   rt->pc_insert(rt0->rt_nexthop);
   rt0->pc_insert(rt->rt_nexthop);
        Packet::free(p);
 }

 else {
   if(n_energy <= RE_THR){
        Packet::free(p);
         return;
   } else {
        ih->saddr() = index;
        ih->daddr() = IP_BROADCAST;
        rq->rq_hop_count += 1;

        pheromone_count     =     (RSSM*n_energy)     /
(n_congestion*rq->rq_hop_count);

        if(rq->rq_pheromone_count!=0){
               rt_phcount  =  rq->rq_pheromone_count  +
pheromone_count;
        }else{
                rt_phcount = pheromone_count;
        }
        rq->rq_pheromone_count = rt_phcount;
        rt0->rt_pheromone_count = rt_phcount;

        if (rt) rq->rq_dst_seqno = max(rt->rt_seqno, rq-
>rq_dst_seqno);
        forward((aodv_rt_entry*) 0, p, DELAY);
   }
 }

}
```

A.3 Kode Fungsi sendReply()

```
void    AODV::sendReply(nsaddr_t    ipdst,    u_int32_t
hop_count, nsaddr_t rpdst, u_int32_t rpseq, u_int32_t
lifetime, double timestamp)
{
  Packet *p = Packet::alloc();
  struct hdr_cmn *ch = HDR_CMN(p);
  struct hdr_ip *ih = HDR_IP(p);
  struct hdr_aodv_reply *rp = HDR_AODV_REPLY(p);
  aodv_rt_entry *rt = rtable.rt_lookup(ipdst);

  assert(rt);

  rp->rp_type = AODVTYPE_RREP;
  rp->rp_hop_count = hop_count;
  rp->rp_dst = rpdst;
  rp->rp_dst_seqno = rpseq;
  rp->rp_src = index;
  rp->rp_lifetime = lifetime;
  rp->rp_timestamp = timestamp;
  if(rp->rp_dst == index) {
    rp->rp_pheromone_count    =    max(rt_phcount,    rt-
>rt_pheromone_count);
  }

  ch->ptype() = PT_AODV;
  ch->size() = IP_HDR_LEN + rp->size();
  ch->iface() = -2;
  ch->error() = 0;
  ch->addr_type() = NS_AF_INET;
  ch->next_hop_ = rt->rt_nexthop;
  ch->prev_hop_ = index;
  ch->direction() = hdr_cmn::DOWN;

  ih->saddr() = index;
  ih->daddr() = ipdst;
  ih->sport() = RT_PORT;
  ih->dport() = RT_PORT;
  ih->ttl_ = NETWORK_DIAMETER;

  Scheduler::instance().schedule(target_, p, 0.);
}
```

A.4 Kode Skenario NS-2

```
set val(chan)                   Channel/WirelessChannel
        ;
set val(prop)                   Propagation/TwoRayGround;
set val(netif)                  Phy/WirelessPhy ;
set val(mac)                    Mac/802_11;
set val(ll)                     LL;
set val(ant)                    Antenna/OmniAntenna;
set opt(x)                      1820;
set opt(y)                      1820;

set val(ifqlen)                 1000    ;

set val(nn)                     300     ;

set val(seed)                   1.0     ;

set val(adhocRouting)           AODV    ;

set val(stop)                   200     ;

set val(cp)                     "cbr.tcl";

set val(sc)                     "scenario.tcl"          ;


set val(energy_mod)         EnergyModel                 ;
set val(energy_init)        100                         ;
set val(tx_power)           8.0                         ;
set val(rx_power)           6.0                         ;
set val(idle_power)          0.0                        ;
set val(sleep_power)         0.0                        ;

set ns_         [new Simulator]
set topo        [new Topography]
set tracefd      [open simulation.tr w]
set namtrace     [open simulation.nam w]
$ns_ trace-all $tracefd
$ns_ namtrace-all-wireless $namtrace $opt(x) $opt(y)
set topo                [new Topography]
$topo load_flatgrid $opt(x) $opt(y)
set god_ [create-god $val(nn)]


$ns_ node-config    -adhocRouting $val(adhocRouting) \
                    -llType $val(ll) \
                    -macType $val(mac) \
```

```
                    -ifqType $val(ifq) \
                    -ifqLen $val(ifqlen) \
                    -antType $val(ant) \
                    -propType $val(prop) \
                    -phyType $val(netif) \
                    -channelType $val(chan) \
                    -energyModel $val(energy_mod) \
                    -initialEnergy $val(energy_init) \
                    -txPower $val(tx_power) \
                    -rxPower $val(rx_power) \
                    -idlePower $val(idle_power) \
                    -sleepPower $val(sleep_power) \
                    -topoInstance $topo \
                    -agentTrace ON \
                    -routerTrace ON \
                    -macTrace ON \
                    -movementTrace ON

Phy/WirelessPhy       set    RXThresh_   5.57189e-11   ;
#400m receiver sensitivity range
Phy/WirelessPhy set   CSThresh_   5.57189e-11   ;   #400m
capture threshold  range

for {set i 0} {$i < $val(nn)} {incr i} {
    if {$i < 188} {
        set node_($i) [$ns_ node]
        $node_($i) random-motion 0 ;
    } else {
        $ns_ node-config      adhocRouting
$val(adhocRouting) \
                              -llType $val(ll) \
                              -macType $val(mac) \
                              -ifqType $val(ifq) \
                              -ifqLen $val(ifqlen) \
                              -antType $val(ant) \
                              -propType $val(prop) \
                              -phyType $val(netif) \
                              -channelType $val(chan) \
                              -energyModel
$val(energy_mod) \
                              -initialEnergy 200 \
                              -txPower $val(tx_power) \
                              -rxPower $val(rx_power) \
                              -idlePower
$val(idle_power) \
                              -sleepPower
$val(sleep_power) \
                              -topoInstance $topo \
```

```
                                -agentTrace ON \
                                -routerTrace ON \
                                -macTrace ON \
                                -movementTrace ON
        set node_($i) [$ns_ node]
        $node_($i)  random-motion  0  ;#  disable  random
motion
    }
}

puts "Loading connection pattern..."
source $val(cp)

puts "Loading scenario file..."
source $val(sc)

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

    $ns_ initial_node_pos $node_($i) 20
}

for {set i 0} {$i < $val(nn) } {incr i} {
    $ns_ at $val(stop).0 "$node_($i) reset";
}

$ns_ at  $val(stop).0002 "puts \"NS EXITING...\" ; $ns_
halt"

puts $tracefd "M 0.0 nn $val(nn) x $opt(x) y $opt(y) rp
$val(adhocRouting)"
puts  $tracefd  "M  0.0  sc  $val(sc)  cp  $val(cp)  seed
$val(seed)"
puts $tracefd "M 0.0 prop $val(prop) ant $val(ant)"

puts "Starting Simulation..."
$ns_ run
```

A.5 Kode Konfigurasi *Traffic*

```
set udp_(0) [new Agent/UDP]
$ns_ attach-agent $node_(288) $udp_(0)
set null_(0) [new Agent/Null]
$ns_ attach-agent $node_(299) $null_(0)
set cbr_(0) [new Application/Traffic/CBR]
$cbr_(0) set packetSize_ 512
$cbr_(0) set interval_ 0.25
$cbr_(0) set random_ 1
$cbr_(0) set maxpkts_ 10000
$cbr_(0) attach-agent $udp_(0)
$ns_ connect $udp_(0) $null_(0)
$ns_ at 2.5568388786897245 "$cbr_(0) start"

set udp_(1) [new Agent/UDP]
$ns_ attach-agent $node_(289) $udp_(1)
set null_(1) [new Agent/Null]
$ns_ attach-agent $node_(298) $null_(1)
set cbr_(1) [new Application/Traffic/CBR]
$cbr_(1) set packetSize_ 512
$cbr_(1) set interval_ 0.25
$cbr_(1) set random_ 1
$cbr_(1) set maxpkts_ 10000
$cbr_(1) attach-agent $udp_(1)
$ns_ connect $udp_(1) $null_(1)
$ns_ at 56.333118917575632 "$cbr_(1) start"

set udp_(2) [new Agent/UDP]
$ns_ attach-agent $node_(290) $udp_(2)
set null_(2) [new Agent/Null]
$ns_ attach-agent $node_(297) $null_(2)
set cbr_(2) [new Application/Traffic/CBR]
$cbr_(2) set packetSize_ 512
$cbr_(2) set interval_ 0.25
$cbr_(2) set random_ 1
$cbr_(2) set maxpkts_ 10000
$cbr_(2) attach-agent $udp_(2)
$ns_ connect $udp_(2) $null_(2)
$ns_ at 46.96568928983328 "$cbr_(2) start"

set udp_(3) [new Agent/UDP]
$ns_ attach-agent $node_(291) $udp_(3)
set null_(3) [new Agent/Null]
$ns_ attach-agent $node_(296) $null_(3)
set cbr_(3) [new Application/Traffic/CBR]
$cbr_(3) set packetSize_ 512
$cbr_(3) set interval_ 0.25
```

```
$cbr_(3) set random_ 1
$cbr_(3) set maxpkts_ 10000
$cbr_(3) attach-agent $udp_(3)
$ns_ connect $udp_(3) $null_(3)
$ns_ at 55.634230382570173 "$cbr_(3) start"

set udp_(4) [new Agent/UDP]
$ns_ attach-agent $node_(292) $udp_(4)
set null_(4) [new Agent/Null]
$ns_ attach-agent $node_(295) $null_(4)
set cbr_(4) [new Application/Traffic/CBR]
$cbr_(4) set packetSize_ 512
$cbr_(4) set interval_ 0.25
$cbr_(4) set random_ 1
$cbr_(4) set maxpkts_ 10000
$cbr_(4) attach-agent $udp_(4)
$ns_ connect $udp_(4) $null_(4)
$ns_ at 29.546173154165118 "$cbr_(4) start"

set udp_(5) [new Agent/UDP]
$ns_ attach-agent $node_(293) $udp_(5)
set null_(5) [new Agent/Null]
$ns_ attach-agent $node_(294) $null_(5)
set cbr_(5) [new Application/Traffic/CBR]
$cbr_(5) set packetSize_ 512
$cbr_(5) set interval_ 0.25
$cbr_(5) set random_ 1
$cbr_(5) set maxpkts_ 10000
$cbr_(5) attach-agent $udp_(5)
$ns_ connect $udp_(5) $null_(5)
$ns_ at 7.7030203154790309 "$cbr_(5) start"
```

A.6 Kode Skrip AWK *Throughput*

```
{
          event = $1
          time = 0 + $
          node_id = $3
          pkt_size = 0 + $8
          level = $4

          if (level == "AGT" && event == "s" && $7 ==
"cbr") {
                    sent++
                    if (!startTime || (time < startTime))
{
                              startTime = time
                    }
          }

          if (level == "AGT" && event == "r" && $7 ==
"cbr") {
                    receive++
                    if (time > stopTime) {
                              stopTime = time
                    }
                    recvdSize += pkt_size
          }
}

END {
  printf("%.2f\n",                (recvdSize/(stopTime-
startTime))*(8/1000));
}
```

A.7 Kode Skrip AWK Packet Delivery Rasio

```
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;
}
```

A.8 Kode Skrip AWK Rata-Rata End-to-End 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";
}
```

A.9 Kode Skrip AWK Routing Overhead

```
BEGIN {
        rt_pkts = 0;
}
{
        if (($1 == "s" || $1 == "f") && ($4 == "RTR") &&
($7 == "AODV")) {
                  rt_pkts++;
                }
        }
}
END {
        printf "Routing Packets \t= %d \n", rt_pkts;
}
```

A.10 Kode Skrip AWK *Forwarded Route Request*

```
BEGIN {
        rt_forward = 0;
}
{
                if (($1 == "s") && ($4 == "RTR") && ($7
== "AODV") && ($25 == "(REQUEST)") && ($3 != "_58_")){
                          rt_forward++;
                }
        }
}
END {
        printf  "Forwarded  Route  Request\t=  %d  \n",
rt_forward;
}
```

BIODATA PENULIS

Son Ardhynata Sukarno Mudha, lahir di Magetan, 17 Desember 1994. Penulis adalah anak pertama dari dua bersaudara. Penulis menempuh pendidikan sekolah dasar di SDN Tebon 2 lalu melanjutkan pendidikan sekolah menengah pertama di SMP Negeri 1 Maospati dan meneruskan pendidikan menengah atas di SMA Negeri 10 Malang Sampoerna Academy. Selanjutnya penulis menempuh pendidikan tinggi di Departemen Informatika, Fakultas Teknologi Elektro dan Informatika Cerdas, Institut Teknologi Sepuluh Nopember Surabaya. Selama kuliah, penulis aktif dalam berbagai organisasi intra universitas. Dalam menyelesaikan pendidikan S1, penulis mengambil bidang minat Komputasi Berbasis Jaringan. Sebagai mahasiswa, penulis berperan aktif dalam beberapa organisasi kampus seperti staf Pengembangan Sumber Daya Mahasiswa Himpunan Mahasiswa Teknik-Computer (HMTC) ITS, staf National Seminar of Technology SCHEMATICS 2014, dan staf Pengembangan Sumber Daya Mahasiswa Badan Eksekutif Mahasiswa Fakultas Teknologi Informasi. Penulis pernah melakukan kerja praktik di PT Pertamina cabang Banyuwangi yang bekerjasama dengan PT Bosowa dalam proyek pengembangan perangkat lunak *Monitoring and Consequency Safety Checking* berbasis Android selama 5 bulan. Penulis dapat dihubungi melalui nomor telepon: +6285784364457 atau *email:* atanyhdra@gmail.com.