TUGAS AKHIR – IF184802

IMPLEMENTASI *ROUTING PROTOCOL* AODV DENGAN PENAMBAHAN INFORMASI DUA *HOP* TETANGGA PADA VANETS

FATIMATUS ZULFA
NRP 05111540000073

Dosen Pembimbing I
Ir. F.X. Arunanto, M.Sc.

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

Departemen Informatika
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya 2019

(Halaman ini sengaja dikosongkan)

TUGAS AKHIR – IF184802

IMPLEMENTASI *ROUTING PROTOCOL* AODV DENGAN PENAMBAHAN INFORMASI DUA *HOP* TETANGGA PADA VANETS

FATIMATUS ZULFA
NRP 05111540000073

Dosen Pembimbing I
Ir.F.X. Arunanto, M.Sc.

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

Departemen Informatika
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya 2019

(Halaman ini sengaja dikosongkan)

UNDERGRADUATE THESES – IF184802

IMPLEMENTATION OF ROUTING PROTOCOL AODV WITH ADDITIONAL INFORMATION TWO NEIGHBOR HOPS ON VANETS

FATIMATUS ZULFA
NRP 05111540000073

First Advisor
Ir. F.X. Arunanto, M.Sc.

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

Department of Informatics
Faculty of Information Technology and Communication
Sepuluh Nopember Institute of Technology
Surabaya 2019

(Halaman ini sengaja dikosongkan)

LEMBAR PENGESAHAN

IMPLEMENTASI *ROUTING PROTOCOL* AODV DENGAN PENAMBAHAN INFORMASI DUA *HOP* TETANGGA PADA VANETS

TUGAS AKHIR

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Arsitektur dan Jaringan Komputer
Program Studi S-1 Departemen Informatika
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember

Oleh:
FATIMATUS ZULFA
05111540000073

Disetujui oleh Pembimbing Tugas Akhir:

1. Ir. F.X. Arunanto, M.Sc. .....
(NIP. 19570101 198303 1 004) (Pembimbing 1)

2. Dr.Eng. Radityo Anggoro, S.Kom., M.Sc. .....
(NIP. 198410162008121002) (Pembimbing 2)

SURABAYA
JULI, 2019

(Halaman ini sengaja dikosongkan)

IMPLEMENTASI *ROUTING PROTOCOL* AODV DENGAN PENAMBAHAN INFORMASI DUA *HOP* TETANGGA PADA VANETS

Nama Mahasiswa	**: Fatimatus Zulfa**
NRP	**: 05111540000073**
Departemen	**: Informatika FTIK-ITS**
Dosen Pembimbing 1	**: Ir. F.X. Arunanto M.Sc.**
Dosen Pembimbing 2	**: Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.**

Abstrak

VANETs adalah kepanjangan dari *Vehicular Ad hoc Networks* yang merupakan pengembangan dari *Mobile Ad hoc Network* (MANET). Keduanya merupakan jaringan yang sistemnya tidak memperdulikan adanya sistem jaringan infrastruktur. Pada VANETs, terdapat banyak routing yang dapat diterapkan. Salah satunya adalah AODV, yaitu *Ad hoc On demand Distance Vector*.

AODV merupakan sebuah *routing protocol* yang termasuk dalam klasifikasi *reactive routing protocol. Reactive routing protocol* adalah sebuah *routing protocol* yang akan bekerja hanya ketika dibutuhkan (On demand). Setiap *node* menyimpan tabel *routing next-hop,* yaitu menyimpan informasi tujuan ke *hop* berikutnya dengan *route* tertentu. Ketika *node* asal ingin mengirim paket ke *node* tujuan namun tidak terdapat *node* yang tersedia, *node* tersebut akan memulai proses *route discovery.*

Route discovery merupakan proses pencarian rute yang tepat untuk sebuah *node* asal mengirimkan paketnya ke *node* tujuan. Pada proses *route discovery*, *node* asal mem-*broadcast paket route request* (RREQ) yang disertakan nomer *sequence* tujuan. Ketika *node* tujuan menerima RREQ, maka *node* tersebut akan meneruskan paket *route reply* (RREP). Selain *route discovery,* AODV juga melakukan proses *route maintenance,* yaitu proses untuk mengetahui informasi adanya kerusakan dan kesalahan pada rute.

Untuk mendapatkan informasi tersebut, *node* akan mengirimkan pake *route error* (RRER).

Pada Tugas Akhir ini akan dilakukan pengurangan *broadcast* RREQ dalam jaringan hingga batas tertentu dan mempercepat *route discovery node* tetangganya yang aktif tanpa mengirim pesan tambahan dengan cara menggabungkan *hello message* dengan informasi *node* tetangga yang berdekatan sehingga memungkinkan *node* penerima akan memperbarui table *log*-nya hingga dua segmen yaitu 1-*hop* dan 2-*hop* tetangganya yang aktif. Selain itu untuk menghindari jaringan *overload* akibat pesan RREQ duplikat, pada tugas akhir ini pesan RREQ duplikat tersebut dimanfaatkan sebagai *alternative path. Alternative path* merupakan rute cadangan yang akan dibentuk apabila rute utama mengalami kerusakan atau kesalahan.

Dari hasil uji coba, AODV yang dimodifikasi pada skenario grid berhasil meningkatkan nilai *Packet Delivery Ratio* (PDR) hingga 17.70%, penurunan nilai *End-to-end Delay* hingga 86.95%, penurunan nilai *Routing Overhead* (RO) hingga 0.14%, dan meningkatkan nilai *Throughput* hingga 19.34%, sedangkan pada skenario real berhasil meningkatkan nilai *Packet Delivery Ratio* (PDR) hingga 10.39%, penurunan nilai *End-to-end Delay* hingga 68.92%, penurunan nilai *Routing Overhead* (RO) hingga 3.24%, dan meningkatkan nilai *Throughput* hingga 11.99%.

Kata kunci: VANETs, AODV, Hello Message, RREQ Duplikat, Alternative Path, Node Tetangga.

IMPLEMENTATION OF ROUTING PROTOCOL AODV WITH ADDITIONAL INFORMATION TWO NEIGHBOR HOPS ON VANETS

Student's Name	**:**	**Fatimatus Zulfa**
Student's ID	**:**	**05111540000073**
Department	**:**	**Informatics – FTIK ITS**
First Advisor	**:**	**Ir. F.X. Arunanto M.Sc.**
Second Advisor	**:**	**Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.**

Abstract

VANETs is an abbreviation of Vehicular Ad hoc Networks which is a development of Mobile Ad hoc Network (MANET) where both are networks whose systems do not care about the existence of an infrastructure network system. In VANETs, there is a lot of routing that can be applied. One of them is AODV, namely Ad hoc On demand Distance Vector.

AODV is a routing protocol that is included in the reactive routing protocol classification. Reactive routing protocol is a routing protocol that will work only when needed (On demand). Each node stores the next-hop routing table, which stores the destination information to the next hop with a specific route. When the original node wants to send the packet to the destination node but there are no available nodes, the node will start the route discovery process.

Route discovery is the process of finding the right route for an origin node sending the packet to the destination node. In the route discovery process, the original node broadcasts the route request package (RREQ) which is included in the destination sequence number. When the destination node receives RREQ, then the node will forward the route reply package (RREP). In addition to route

discovery, AODV also performs a route maintenance process, which is a process to find out information about damage and errors on the route. To get this information, the node will send using route error (RRER).

In this Final Project, the RREQ broadcast will be reduced in a network to a certain extent and accelerate route discovery of neighboring active nodes without sending additional messages by combining hello messages with information on adjacent neighbor nodes that allow the receiving node to update the table log up to two segments namely 1-hop and active 2-hop neighbors. In addition, to avoid network overload due to duplicate RREQ messages, in this final assignment the duplicate RREQ message is used as an alternative path. Alternative path is a backup route that will be formed if the main route is damaged or error.

From the results of the trial, AODV modified in the grid scenario succeeded in increasing the value of the Packet Delivery Ratio (PDR) to 17.70%, decreasing the End-to-end Delay value to 86.95%, decreasing the Overhead Routing value (RO) to 0.14%, and increasing the value Throughput is up to 19.34%, while in real scenario it can increase the value of Packet Delivery Ratio (PDR) to 10.39%, decrease End-to-end Delay value up to 68.92%, decrease Overhead Routing value (RO) to 3.24%, and increase the Throughput value to 11.99%.

Keywords: VANETs, AODV, Hello Message, Duplicate RREQ, Alternative Path, Neighboring Node.

KATA PENGANTAR

Puji syukur kepada Allah Yang Maha Esa atas segala karunia dan rahmat-Nya penulis dapat menyelesaikan Tugas Akhir yang berjudul **"Implementasi *Routing Protocol* AODV dengan Penambahan Informasi Dua *Hop* Tetangga pada VANETs".**

Harapan penulis semoga apa yang tertulis di dalam buku Tugas Akhir ini dapat bermanfaat bagi pengembangan ilmu pengetahuan saat ini.

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

1. Allah SWT atas semua rahmat yang diberikan sehingga penulis dapat menyelesaikan Tugas Akhir ini.
2. Bapak Amenan dan Almh. Ibu Sri Rahayu Ningsih selaku kedua orangtua penulis atas segala dukungan berupa motivasi, material, didikan moral, dan doa sehingga penulis dapat mengerjakan Tugas Akhir ini.
3. Andik Ali Musthofa, Anik Yulianti, dan Heru Sofyantoro selaku kakak penulis atas segala doa dan dukungan yang telah diberikan sehingga penulis tetap semangat dalam mengerjakan Tugas Akhir ini.
4. Bapak Dr. Eng. Radityo Anggoro, S.Kom., M.Sc., dan Bapak Ir. F.X. Arunanto M.Sc. selaku dosen pembimbing, atas arahan dan bantuannya dalam pengerjaan Tugas Akhir ini.
5. Bapak dan Ibu Dosen di Departemen Informatika yang telah memberikan ilmu selama penulis berkuliah di Departemen Informatika ITS.
6. Seluruh Staf dan Karyawan Departemen Informatika yang telah memberikan sapaan dan bantuan selama penulis berkuliah di Departemen Informatika.
7. Sahabat dekat penulis (Mutia, Astrid, Lia, Dara, Sirria, Aulia, Vivi, Syavira, Tata, Mala, Putri, dan Asmaul) serta

teman – teman angkatan 2015 dan 2014 yang selama ini sudah setia menemani, membantu dan memotivasi penulis dalam menyelesaikan Tugas Akhir ini.

8. Sahabat NCC dan AJK yang sudah membantu dan menyediakan fasilitas selama penulis mengerjakan Tugas Akhir ini.
9. 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, Mei 2019

Fatimatus Zulfa

DAFTAR ISI

(Halaman ini sengaja dikosongkan)

DAFTAR GAMBAR

DAFTAR TABEL

(Halaman ini sengaja dikosongkan)

BAB I
PENDAHULUAN

1.1 Latar Belakang

Vehicular Ad-hoc Network yang biasanya dikenal sebagai VANETs merupakan bagian dari *Mobile Ad-hoc Network* (MANET). Bedanya, *node-node* dari VANETs memiliki kecepatan yang lebih tinggi dari MANET. Dengan kecepatan mobilitas yang tinggi dari *node-node* VANET topologi berubah dengan cepat dan rute yang dibentuk sering putus[1]. Oleh karena itu, dibutuhkan *routing protocol* yang cepat untuk membuat sebuah rute dan memiliki transmisi yang handal.

Routing protocol dalam VANETs dibedakan menjadi tiga model, yaitu *proactive routing, reactive routing,* dan *hybrid. Proactive routing* adalah protokol yang bekerja dengan cara membentuk tabel *routing* dan melakukan *update* setiap saat pada selang waktu tertentu tanpa memperhatikan beban jaringan, bandwidth dan ukuran jaringan. Sedangkan *reactive routing* adalah mekanisme *routing* yang membentuk tabel *routing* jika ada permintaan pengiriman data atau terjadinya perubahan rute dalam setiap jaringan. Dan *hybrid* adalah mekanisme *routing* yang menggunakan *cluster-cluster* untuk menemukan rute dari setiap *node*[2]. Contoh *proactive routing* adalah *Destination-Sequenced Distance Vector* (DSDV), dan *Optimized Link State Routing* (OLSR) sedangkan contoh *reactive routing* adalah *Adhoc On-Demand Distance Vector* (AODV), dan *Dynamic Source Routing* (DSR), serta contoh *hybrid* adalah *Zone Routing Protocol* (ZRP).

Dewasa ini, VANET menjadi topik hangat para peneliti dan pengembang. Beberapa *routing protocol* yang berperan penting dalam mendapatkan performa jaringan yang lebih baik, banyak dikembangkan saat ini. Untuk menemukan *routing protocol* yang efektif para peneliti memodifikasi *routing protocol* yang diangkat menjadi topik tugas akhir ini yaitu *Improvising*-AODV(I-AODV) yang merupakan pengembangan dari *protocol*

AODV. I-AODV mengembangkan AODV dengan menambahkan informasi dua *hop node* tetangga yang berdekatan. Hal tersebut dapat mengurangi *control message* dan mempercepat proses *route discovery*[3]. Selain itu, juga menambahkan jalur *alternative* pada *node.* Sehingga ketika jalur utama rusak, terdapat jalur *alternative* yang dapat digunakan tanpa melakukan pencarian rute baru yang dapat membebani jaringan dan menambah *delay.* Dengan modifikasi ini, diharapkan kinerja *routing protocol* AODV dapat meningkat pada lingkungan VANETs.

1.2 Rumusan Masalah

Berikut beberapa hal yang menjadi rumusan masalah pada tugas akhir ini:

1. Bagaimana meningkatkan kinerja *protocol* dengan memanfaatkan informasi 2 *hop* tetangga?
2. Bagaimana implementasi *alternative path* pada AODV di lingkungan VANETs?
3. Bagaimana dampak perubahan yang dilakukan terhadap performa AODV?

1.3 Batasan Permasalahan

Batasan masalah pada tugas akhir ini adalah sebagai berikut:

1. Jaringan yang digunakan adalah jaringan *Vehicular Ad hoc Networks* (VANETs).
2. *Routing protocol* yang digunakan merupakan modifikasi dari AODV yaitu *Improvising*-AODV (I-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 pembuatan tugas akhir ini adalah meningkatkan kinerja dengan metode penambahan informasi 2 *hop* tetangga dan *alternative path* pada AODV di lingkungan VANETs.

1.5 Manfaat

Pengerjaan tugas akhir ini dilakukan dengan harapan dapat menjadi pedoman penelitian untuk mengembangkan metode penambahan informasi 2 *hop* tetangga dan *alternative path* pada AODV di lingkungan VANETs di masa yang akan datang.

1.6 Metodologi

Pembuatan Tugas Akhir ini dilakukan dengan menggunakan metodologi sebagai berikut:

1.6.1 Penyusunan Proposal Tugas Akhir

Tahapan awal dari Tugas Akhir ini adalah penyusunan Proposal Tugas Akhir. Proposal Tugas Akhir berisi pendahuluan, deskripsi dan gagasan metode-metode yang dibuat dalam Tugas Akhir ini. Pendahuluan ini terdiri atas hal yang menjadi latar belakang diajukannya Tugas Akhir, rumusan masalah yang diangkat, batasan masalah untuk Tugas Akhir, dan manfaat dari hasil pembuatan Tugas Akhir ini. Selain itu dijabarkan pula tinjauan pustaka yang digunakan sebagai referensi pendukung pembuatan Tugas Akhir. Terdapat pula sub bab jadwal kegiatan yang menjelaskan jadwal pengerjaan Tugas Akhir.

1.6.2 Studi Literatur

Pada tahap ini, dipelajari sejumlah referensi yang diperlukan dalam melakukan implementasi yaitu mengenai

VANETs, AODV, NS2, OpenStreetMap, Java OpenStreetMap (JOSM), SUMO, dan AWK.

1.6.3 Analisis dan Desain Sistem

Pada tahap ini dilakukan analisis dari hasil percobaan modifikasi AODV yang dibuat. Data yang dianalis berasal dari perhitungan *Packet Delivery Ratio (*PDR), *Routing Overhead* (RO), *End-to-End Delay*, dan *Throughput* paket dari *node* ke *node* lainnya. Hal ini bertujuan untuk merumuskan solusi yang tepat untuk konfigurasi AODV yang dimodifikasi dalam lingkungan topologi MANET. Setelah selesai diaplikasikan pada MANET, dilakukan simulasi yang dilakukan pada VANETs dengan bantuan SUMO.

1.6.4 Implementasi Sistem

Implementasi merupakan tahap untuk membangun metode-metode yang sudah diajukan pada proposal Tugas Akhir. Pada tahap ini dilakukan implementasi menggunakan NS-2 sebagai *simulator*, Bahasa C/C++ sebagai bahasa pemrograman, dan SUMO sebagai *tools* untuk uji coba dan mengimplentasikan desain sistem yang sudah dirancang.

1.6.5 Pengujian dan Evaluasi

Pada tahap ini dilakukan pengujian menggunakan SUMO, sebuah *traffic generator* untuk membuat simulasi keadaan topologi yang diujikan. Hasil dari SUMO tersebut akan dijalankan pada NS-2 yang akan menghasilkan *trace file*. *Packet Delivery Ratio* (PDR), *Routing Overhead* (RO), *End-to-End Delay*, dan *Throughput* akan dihitung dari *trace file* tersebut untuk menguji performa AODV yang telah dimodifikasi.

1.6.6 Penyusunan Buku

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

1.7 Sistematika Penulisan 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 dari pembuatan Tugas Akhir.
2. Bab II. Tinjauan 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 VANETs, AODV, NS2, OpenStreetMap, Java OpenStreetMap (JOSM), 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 (*Packet Delivery Ratio*, *Routing Overhead*, *End-to-End Delay*, dan *Throughput*).
4. Bab IV. Implementasi
   Bab ini menjelaskan implementasi yang berbentuk kode sumber dari proses modifikasi protokol AODV, pembuatan simulasi pada NS2, SUMO, dan perhitungan 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.

BAB II
TINJAUAN PUSTAKA

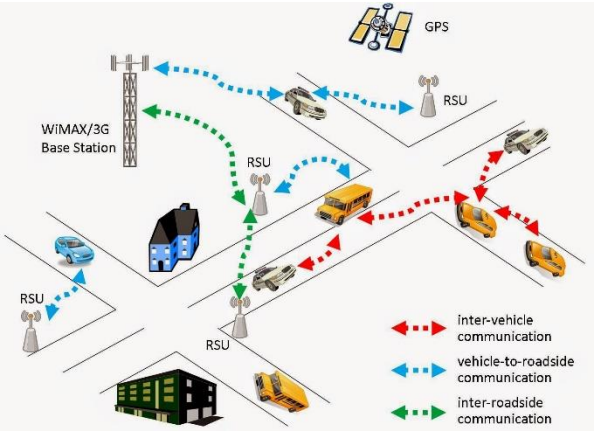
Bab ini berisi pembahasan mengenai teori-teori dasar atau penjelasan dari metode dan *tools* yang digunakan dalam Tugas Akhir.

2.1 VANETs

VANETs adalah kepanjangan dari *Vehicular Ad-hoc Networks* yang merupakan pengembangan dari *Mobile Ad-hoc Network* (MANET) dimana kendaraan bertindak sebagai *node* pada jaringan. VANETs terdiri dari banyak *node* yang juga bertindak sebagai router, baik bagi *node* sumber, *intermediate node* sebagai *forwarding node* maupun *node* tujuan. VANETs memiliki tingkat mobilitas *node* yang lebih tinggi dibandingkan dengan MANET. Tujuan utama dari VANETs yaitu untuk meningkatkan keselamatan pengguna jalan dan kenyamanan penumpang. Hal tersebut tentunya memerlukan implementasi *routing protocol* yang sesuai dengan karakteristiknya di dalam jaringan[4].

Routing sendiri merupakan proses pencarian jalur optimal antara *node* sumber dengan *node* tujuan, untuk mengirimkan pesan secara tepat waktu. Rute antara *node* sumber dan *node* tujuan memungkinkan berisi banyak *hop*. Untuk itu dibutuhkan protocol yang tepat untuk menemukan rute yang efisien. Protokol *routing* pada VANETs memiliki tiga model yaitu protokol *reactive* yang membentuk tabel *routing* hanya saat dibutuhkan, protokol *proactive* yang melakukan pemeliharaan tabel *routing* secara berkala pada waktu tertentu, dan *hybrid* yang menggunakan *cluster-cluster* untuk menemukan rute dari setiap *node*. Pergerakan *node* pada VANETs bisa berubah setiap saat dan terbatas pada rute lalu lintas yang dapat ditentukan dari koordinat peta. Hal ini membuat setiap *node* akan terus memperbarui informasi dalam tabelnya sesuai informasi dari *node* lain. Perubahaan pergerakan pada VANETs menjadi salah satu permasalahan dalam pengiriman paket data sehingga dibutuhkan

informasi jarak antar *node*, kecepatan dan *delay* transmisi [5]. Pada Gambar 2.1 berikut adalah ilustrasi Vanets



Gambar 2.1 Ilustrasi Vanets[6]

Dalam Tugas Akhir ini, penulis akan mengimplementasikan *routing protocol* AODV yang dimodifikasi yaitu I-AODV dan menguji performa protokol tersebut pada lingkungan VANETs.

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

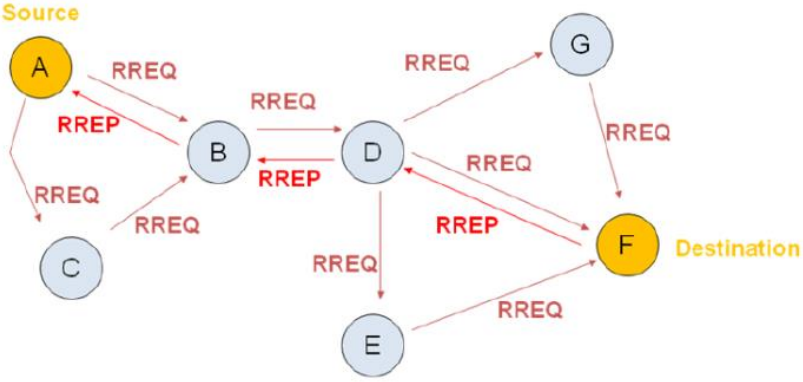
Ad-hoc On demand Distance Vector (AODV) merupakan *distance vector routing protocol* yang termasuk dalam klasifikasi *routing* protokol *reactive*, yang berarti hanya akan membuat rute pada saat dibutuhkan. AODV dikembangkan oleh C. E. Perkins, E.M. Belding-Royer dan S. Das pada RFC 3561[7].

Tabel *routing* pada AODV akan kadaluarsa jika jarang digunakan. Ada dua tahapan dalam AODV yaitu *route discovery* dan r*oute maintenance. Route discovery* terjadi ketika akan dilakukan pengiriman pesan, pada jaringan, namun belum terdapat rute yang benar dari *node* sumber ke *node* tujuan. Pada *route discovery* terdapat dua pesan kontrol yaitu berupa *Route Request* (RREQ) dan *Route Reply* (RREP). Paket RREQ akan di *broadcast* oleh *node* sumber

menuju *node* tujuan yang berisi *source address, hop counter, source* dan *destination, sequence number*, dan *broadcast* ID. Nilai *broadcast* ID akan selalu bertambah satu ketika *node* sumber mengirimkan paket RREQ yang baru dan digunakan untuk mengidentifikasi sebuah paket RREQ. Dan jika sebuah *node* tujuan menerima paket RREQ, *node* tujuan akan mengirimkan paket RREP kembali ke *node* sumber dan menambahkan jalur *node* pada tabel *routing*[2].

Pada *route maintenance* berupa *Route Error* (RERR). Paket RERR merupakan paket yang akan dikirim ke *node* sumber apabila *node* yang ada pada tabel *routing* tidak dapat dilalui kembali untuk mengirim data. Pengiriman paket RERR melalui *node* tetangganya yang mengalami perubahan topologi atau kerusakan sampai ke *node* sumber.

AODV adalah sebuah metode *routing* antar *node* yang memungkinkan *node-node* tersebut untuk melewatkan pesan melalui lingkungannya ke *node* yang tidak dapat dihubungi secara langsung. AODV melakukan ini dengan cara menemukan rute yang bisa dilalui oleh pesan. Selain itu AODV juga memastikan rute ini tidak mengandung perulangan (*loop*), menangani perubahan rute, dan membuat rute baru apabila terjadi *error*[8]. Ilustrasi pencarian rute oleh AODV dapat dilihat pada Gambar 2.2.



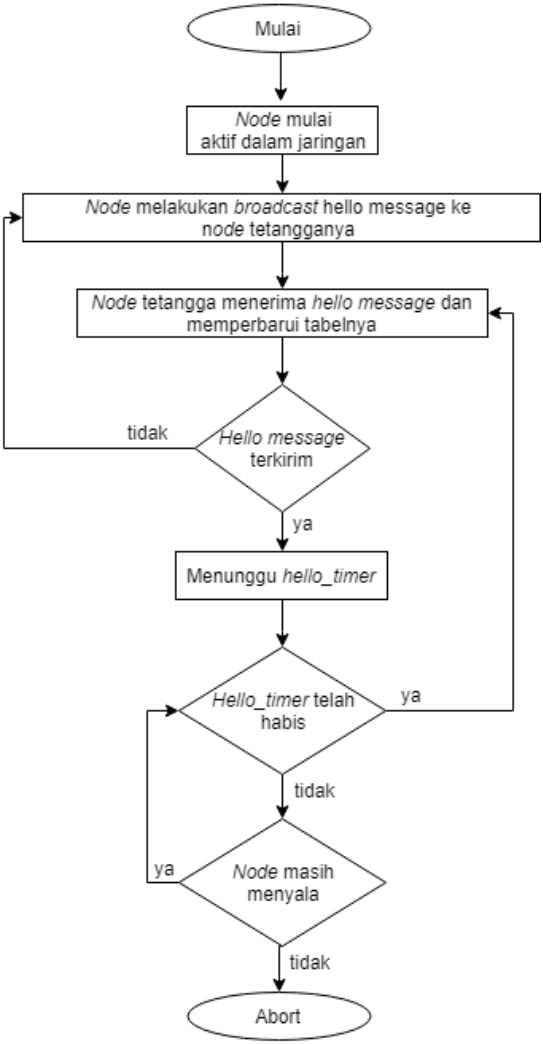
Gambar 2.2 Ilustrasi pencarian rute AODV[9]

Berikut penjelasan lengkap *field* pada *routing table* sebuah node:

- *Destination Address:* berisi alamat dari *node* tujuan.
- *Destination Sequence Number*: *sequence number* dari jalur komunikasi.
- *Next Hop*: alamat *node* yang akan meneruskan paket data.
- *Hop Count*: jumlah *hop* yang harus dilakukan agar paket dapat mencapai *node* tujuan.
- *Lifetime*: waktu dalam milidetik yang diperlukan *node* untuk menerima RREP.
- *Routing Flags*: status jalur. Terdapat tiga tipe status, yaitu *up* (valid), *down* (tidak valid) atau sedang diperbaiki.

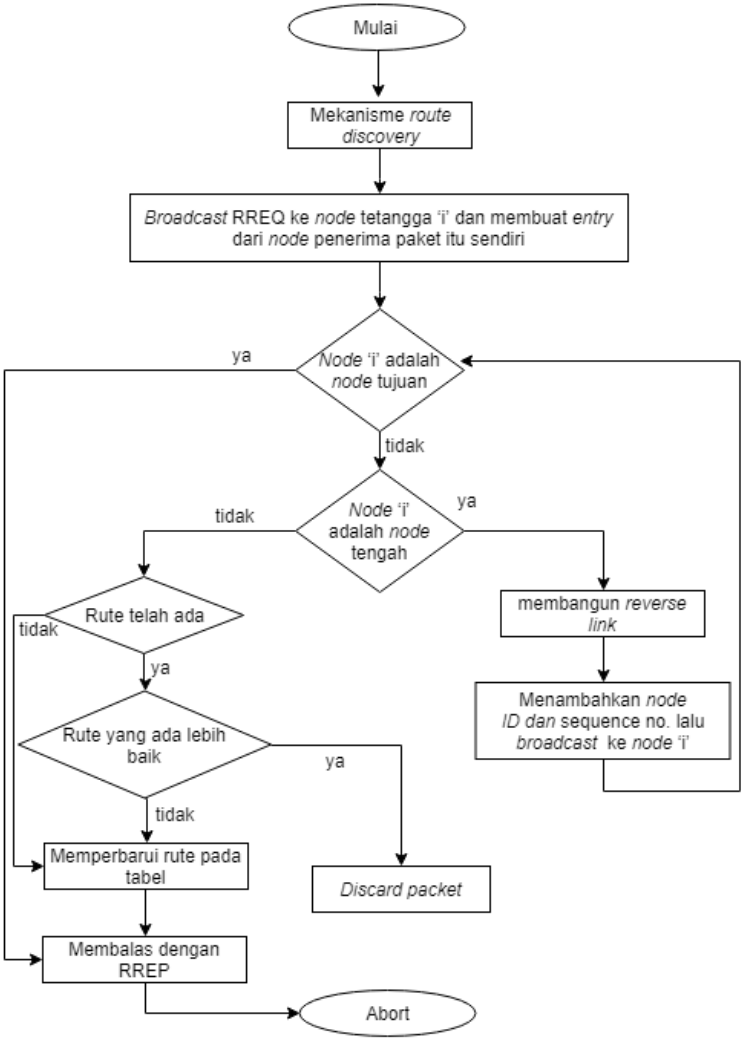
Sebagai contoh proses *route discovery* dalam AODV, ilustrasi pada Gambar 2.2 menggambarkan bagaimana *source node*, yaitu *node* A mencari rute untuk menuju *destination node* yaitu *node* F. *Node* A akan membuat paket RREQ dan melakukan *broadcast* kepada semua *node* tetangganya (*neighbor node*). Jika *destination sequence number* yang terdapat pada paket RREQ sama atau lebih kecil dari yang ada pada *routing table* dan rute menuju *node* tujuan belum ditemukan, maka paket tersebut tidak akan dilanjutkan (*drop*). Jika *destination sequence number* pada RREQ lebih besar dibandingkan dengan yang terdapat pada *routing table*, maka *entry* pada *routing table* akan diperbarui dan paket tersebut akan diteruskan oleh *neighbor node* sekaligus membuat *reverse path* menuju *source node.* Paket RREQ akan diteruskan hingga mencapai *node* F. Kemudian, jika rute menuju *node* F sudah terbentuk di dalam *routing table* dan memiliki *routing flags* "*up*", maka *node* F akan mengirimkan paket RREP melalui rute tersebut menuju *source node* .

Untuk *flow chart* pengiriman *hello message* dapat dilihat pada Gambar 2.3 berikut.



Gambar 2.3 *Flow Chart* Pengiriman *Hello Message* AODV Asli

Sedangkan *flow chart* untuk proses *route discovery* dapat dilihat pada Gambar 2.4 berikut



Gambar 2.4 *Flow Chart* Proses *Route Discovery* AODV Asli

Pada Tugas Akhir ini, penulis menggunakan *routing protocol* AODV yang dimodifikasi yaitu I-AODV dan akan

diimplementasikan pada lingkungan VANETs dengan beberapa skenario.

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

Network Simulator versi kedua atau disebut juga NS-2 adalah sebuah *open-source event-driven* simulator[10]. NS-2 merupakan sebuah perangkat lunak yang dikembangkan oleh lisensi *open-cource*. Lisensi *open-source* tersebut mengizinkan setiap pengguna dapat mengakses kode sumber kompilasi NS-2[11].

NS-2 diciptakan pada tahun 1989 sebagai perangkat lunak untuk mendukung penelitian terhadap jaringan komunikasi[12]. NS-2 terus mendapat revisi dan perbaikan dari berbagai kontributor yang berperan dalam pengembangan perangkat lunak simulator ini. Beberapa kontributor tersebut di antaranya adalah Universitas California dan Universitas Cornell yang mengembangkan REAL network simulator. REAL network simulator pada awalnya diimplementasikan sebagai perangkat lunak untuk mempelajari karakteristik dinamik dari skema kendali aliran dan kongesti (kemacetan) pada packet-switched data network. NS-2 diciptakan berdasarkan pada REAL network simulator. Sejak tahun 1995, Defence Advance Research Project Agency (DAPRA) mendukung pengembangan dari Network Simulator (NS) melalui proyek Virtual InterNetwork Testbed (VINT). Proyek VINT yang didanai oleh DAPRA tersebut bertujuan untuk menciptakan sebuah simulator jaringan untuk mempelajari berbagai protocol yang berbeda untuk jaringan komunikasi. Saat ini, National Science Foundation (NSF) telah bergabung dalam usaha pengembangan perangkat lunak NS-2 ini. Selain itu, para peneliti dan pengembang tetap bekerja untuk menjaga dan meningkatkan kehandalan serta versatilitas perangkat lunak NS-2 ini.

NS-2 dikembangkan pada lingkungan sistem operasi Unix, sehingga kinerja dan proses instalasi NS-2 sangat baik jika digunakan pada sistem operasi tersebut. Namun demikian, NS-2 tetap dapat dijalankan pada system operasi lainnya, yaitu system operasi UNIX-

like (atau Linux), Windows (dengan menggunakan Cygwin, sebuah emulator sistem operasi UNIX pada sistem operasi Windows), dan Mac.

Pada Tugas Akhir ini, NS-2 digunakan untuk melakukan simulasi lingkungan VANETs menggunakan protokol AODV yang sudah dimodifikasi. *Trace file* yang dihasilkan oleh NS-2 juga digunakan untuk mengukur peforma *routing* protokol AODV yang sudah dimodifikasi..

2.3.1 Instalasi

NS-2 membutuhkan beberapa *package* yang harus sudah ter*install* sebelum memulai instalasi NS-2. Untuk meng*install dependency* yang dibutuhkan dapat dilakukan dengan *command* yang ditunjukkan pada Gambar 2.5.

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

Gambar 2.5 Perintah untuk meng*install dependency* NS-2

Setelah meng*install dependency* yang dibutuhkan, ekstrak *package* NS-2 dan ubah baris kode ke-137 pada *file* ls.h di *folder* linkstate menjadi seperti pada Gambar 2.6.

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

Gambar 2.6 Baris kode yang diubah pada file ls.h

Install NS-2 dengan menjalankan perintah ./install pada folder NS-2.

2.3.2 *Trace File*

Trace file merupakan *file* hasil simulasi yang dilakukan oleh NS-2 dan berisikan informasi detail pengiriman paket data. *Trace file*

digunakan untuk menganalisis performa *routing protocol* yang disimulasikan. Detail penjelasan *trace file* ditunjukkan pada Tabel 2.1.

Tabel 2.1 Detail Penjelasan *Trace File* AODV

Kolom ke-	Penjelasan	Isi
1	*Event*	s : *sent* r : *received* f : *forwarded* D : *dropped*
2	*Time*	Waktu terjadinya *event*
3	ID *Node*	_x_ : dari 0 hingga banyak *node* pada topologi
4	*Layer*	AGT : *application* RTR : *routing* LL : *link layer* IFQ : *packet queue* MAC : *MAC* PHY : *physical*
5	*Flag*	--- : Tidak ada
6	*Sequence Number*	Nomor paket
7	*Packet Type*	AODV : paket *routing* AODV cbr : berkas paket CBR (*Constant Bit Rate*) RTS : *Request To Send* yang dihasilkan MAC 802.11 CTS : *Clear To Send* yang dihasilkan MAC 802.11 ACK : MAC ACK ARP : Paket *link layer address resolution protocol*
8	Ukuran	Ukuran paket pada *layer* saat itu
9	Detail MAC	[a b c d] a : perkiraan waktu paket

		b : alamat penerima c : alamat penerima d : IP header
10	*Flag*	-------- : Tidak ada
11	*Detail* IP *source, destination,* dan *nexthop*	[a:b c:d e f] a : IP *source node* b : *port source node* c : IP *destination node* (jika -1 berarti *broadcast*) d : *port destination node* e : IP *header* ttl f : IP *nexthop* (jika 0 berarti *node* 0 atau *broadcast*)

2.4 OpenStreetMap

OpenStreetMap (OSM) adalah sebuah proyek berbasis web untuk membuat peta seluruh dunia yang gratis dan terbuka, dibangun sepenuhnya oleh sukarelawan dengan melakukan survey menggunakan GPS, mendigitasi citra satelit, dan mengumpulkan serta membebaskan data geografis yang tersedia di publik. Melalui *Open Data Commons Open Database License* 1.0, contributor OSM dapat memiliki, memodifikasi, dan membagikan data peta secara 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, innovator, dan banyak pihak lainnya tidak dapat menggunakan data yang tersedia di dalam peta secara bebas. Di sisi lain, baik peta dasar OSM maupun data yang tersedia di dalamnya dapat diunduh secara gratis dan terbuka, untuk kemudia digunakan dan diredistribusikan kembali[13].

Pada Tugas Akhir ini, penulis menggunakan data yang tersedia pada OSM untuk membuat skenario lalu lintas berdasarkan peta daerah di Surabaya. Peta yang diambil lalu digunakan untuk simulasi skenario *rea*l VANETs.

2.5 *Java OpenStreetMap Editor* (JOSM)

JOSM merupakan editor *Java OpenStreetMap* pada desktop yaitu aplikasi untuk menyunting data yang didapatkan dari *OpenStreetMap*[14]. Dalam penggunaannya JOSM tidak memerlukan internet. Hingga 2014, sebagian besar perubahan di *OpenStreetMap* dibuat menggunakan JOSM. Sekitar 80% suntingan OSM dilakukan dengan perangkat lunak ini[15].

Pada Tugas Akhir ini, penulis menggunakan aplikasi ini untuk menyunting dan merapikan peta yang diunduh dari *OpenStreetMap* yaitu dengan menghilangkan dan menyambungkan jalan yang ada. Penyuntingan juga dilakukan dengan menghilangkan gedung – gedung yang ada di peta.

2.6 *Simulation of Urban Mobility* (SUMO)

Simulation of Urban Mobility (SUMO) adalah sebuah paket simulasi lalu lintas yang open source termasuk *net import* dan komponen *demand modeling.* SUMO membantu menyelidiki beberapa topik penelitian, seperti pemilihan rute dan algoritma lampu lalu lintas atau simulasi komunikasi kendaraan[16].

SUMO terdiri dari beberapa *tools* yang dapat membantu pembuatan simulasi lalu lintas pada tahap-tahap yang berbeda. Berikut penjelasan fungsi *tools* yang digunakan dalam pembuatan Tugas Akhir ini:

- *netgenerate*
  netgenerate merupakan *tool* yang berfungsi untuk membuat peta berbentuk seperti *grid*, *spider*, dan bahkan *random network*. Sebelum proses netgenerate, pengguna dapat menentukan kecepatan maksimum jalan dan membuat *traffic light* pada peta. Hasil dari netgenerate ini berupa *file* dengan ekstensi .net.xml. Pada Tugas Akhir ini netgenerate digunakan untuk membuat peta skenario *grid*.
- *netconvert*

netconvert merupakan program CLI yang berfungsi untuk melakukan konversi dari peta seperti *OpenStreetMap* menjadi format *native* SUMO. Pada Tugas Akhir ini penulis menggunakan *netconvert* untuk mengonversi peta dari *OpenStreetMap*.

- randomTrips.py
  Tool dalam SUMO untuk membuat rute acak yang akan dilalui oleh kendaraan dalam simulasi.
- duarouter
  Tool dalam SUMO untuk melakukan perhitungan rute berdasarkan definisi yang diberikan dan memperbaiki kerusakan rute.
- sumo
  Program yang melakukan simulasi lalu lintas berdasarkan data-data yang didapatkan dari netgenerate (skenario *grid*) atau netconvert dari randomTrips.py. Hasil simulasi dapat di-*export* ke sebuah *file* untuk dikonversi menjadi format lain.
- sumo-gui
  GUI untuk melihat simulasi yang dilakukan oleh SUMO secara grafis.
- traceExporter.py
  Tool yang bertujuan untuk mengonversi *output* dari sumo menjadi format yang dapat digunakan pada *simulator* lain. Pada Tugas Akhir ini penulis menggunakan traceExporter.py untuk mengonversi data menjadi format .tcl yang dapat digunakan pada NS-2

Pada Tugas Akhir ini, penulis menggunakan SUMO untuk menghasilkan skenario VANETs, peta area simulasi, dan pergerakan *node* sehingga menyerupai keadaan lalu lintas yang sebenarnya. Untuk setiap skenario VANETs yang dibuat menggunakan SUMO, akan dihasilkan pergerakan *node* yang acak sehingga setiap skenario memiliki pergerakan yang berbeda.

2.7 AWK

AWK adalah bahasa pemrograman yang mencari sekumpulan file untuk pola, dan melakukan perintah tertentu pada *record* atau bidang *record* yang cocok dengan pola[17]. Secara umum Bahasa pemrograman AWK dapat digunakan untuk mengelola database sederhana, membuat *report*, memvalidasi data, menghasilkan indeks dan menampilkan dokumen, serta membuat algoritma yang digunakan untuk mengubah bahasa komputer ke bahasa lainnya. Bahasa pemrograman AWK memiliki fungsi dasar yaitu untuk mencari file per baris (atau unit teks lain) yang berisi pola tertentu. Ketika suatu baris sesuai dengan pola, AWK melakukan aksi yang khusus pada bari tersebut. AWK tetap memproses baris input sedemikian hingga mencapai akhir baris input.

Pada Tugas Akhir ini, AWK digunakan untuk membuat *script* menghitung *Packet Delivery Ratio* (PDR*), Routing Overhead* (RO), *End-to-End Delay* dan *Throughput* dari *trace file* NS2.

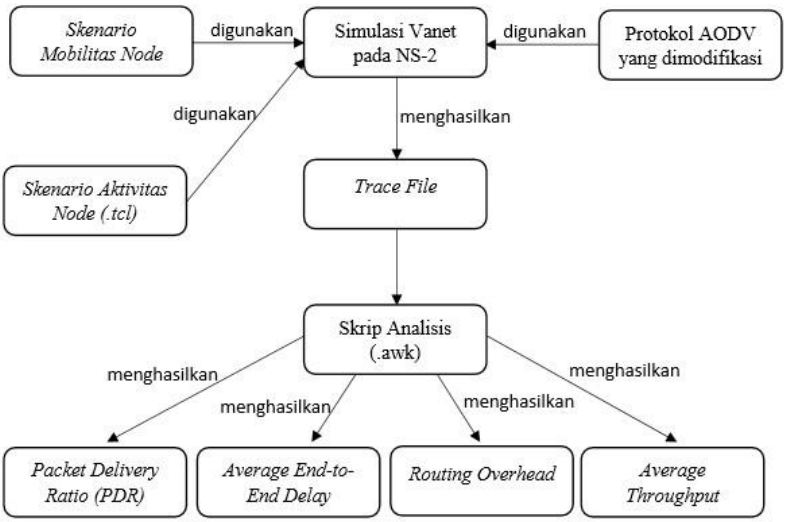
(Halaman ini sengaja dikosongkan)

BAB III
PERANCANGAN

Perancangan merupakan bagian penting dari pembuatan sistem secara teknis sehingga bab ini secara khusus menjelaskan perancangan sistem yang dibuat dalam Tugas Akhir. Berawal dari deskripsi umum sistem hingga perancangan skenario, alur dan implementasinya.

3.1 Deskripsi Umum

Pada Tugas Akhir ini, akan diimplementasikan *routing protocol* AODV dengan memodifikasi pada bagian *hello message* dan proses *route discovery* yang dijalankan pada simulator NS-2. Diagram dari rancangan simulasi dari AODV asli dan AODV modifikasi dapat dilihat pada Gambar 3.1.



Gambar 3.1 Diagram Rancangan Simulasi AODV Modifikasi

Modifikasi *protocol* AODV diawali dengan melakukan modifikasi pada pengiriman *hello message*. Pada *protocol* AODV secara umum, pengiriman *hello message* memungkinkan *node* penerima akan memperbarui tabel *log*-nya hanya 1-*hop*. Namun pada modifikasi *protocol* AODV ini, *hello message* digabungkan dengan informasi *node* tetangga yang berdekatan sehingga memungkinkan *node* penerima akan memperbarui tabel *log*-nya hingga dua segmen yaitu 1-*hop* dan 2-*hop* tetangganya yang aktif. Selanjutnya, modifikasi dilakukan pada proses *route discovery* dengan cara memberikan *flag_RREQ* atau tanda yang akan bertambah apabila sebuah *node* menerima pesan *control* RREQ. *Node* akan meneruskan *broadcast* RREQ apabila *node* penerima memiliki *flag_RREQ* kurang dari 2 dan merupakan *intermediate* node. *Intermediate node* yaitu node perantara antara *node* sumber dan *node* tujuan. Hal tersebut mengakibatkan RREQ yang diterima sebuah *node* menjadi *duplicate*. Sehingga dimanfaatkan sebagai *alternative path* yang berguna apabila rute utama mengalami kerusakan atau terputus. Selain itu, juga dapat mengurangi *control message* dalam jaringan hingga batas tertentu.

Modifikasi yang telah dilakukan akan disimulasikan pada NS-2 dengan peta berbentuk *grid* dan peta *real* pada lingkungan lalu lintas di kota Surabaya. Pembuatan kedua peta tersebut menggunakan bantuan *tools* SUMO. Simulasi tersebut akan memberikan hasil *trace file* yang kemudian dianalisis menggunakan skrip AWK untuk mendapatkan *Packet Delivery Ratio* (PDR), *Routing Overhead* (RO), *End-to-End Delay* (E2E), dan *Throughput*. Analisis tersebut dapat mengukur performa *routing protocol* AODV yang telah dimodifikasi dibandingkan dengan *routing protocol* AODV sebelum dimodifikasi. Analisis ini digunakan untuk mengukur tingkat reliabilitas pengiriman data antara protokol AODV dengan protokol AODV yang dimodifikasi. Daftar istilah yang sering digunakan pada buku Tugas Akhir ini dapat dilihat pada Tabel 3.1.

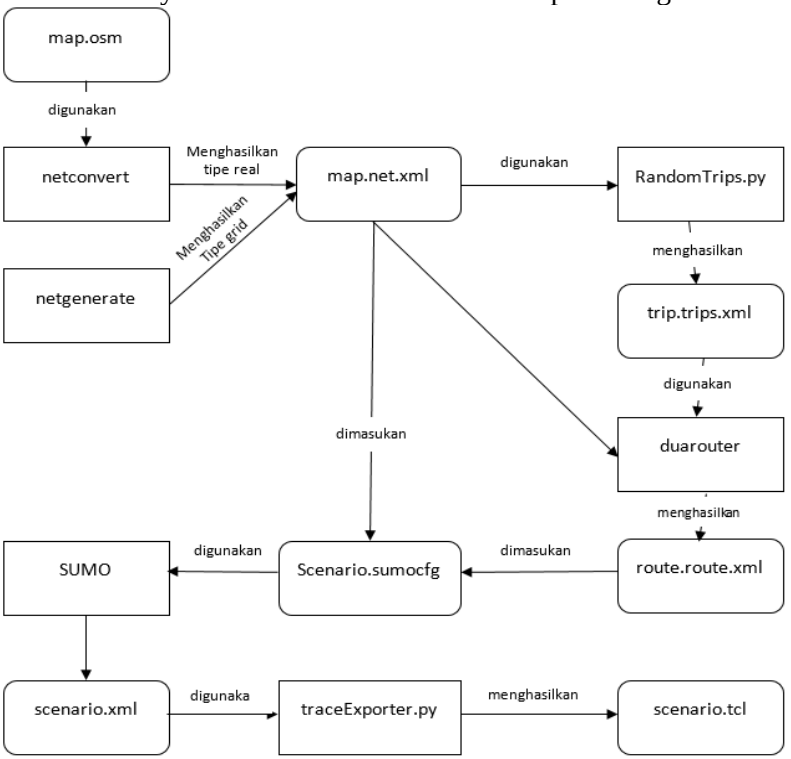
Tabel 3.1 Daftar Istilah

No.	Istilah	Penjelasan
1	AODV	Singkatan dari *Ad hoc On-demand Distance Vector*. Protokol yang digunakan pada Tugas Akhir ini.
2	PDR	*Packet Delivery Ratio*. Salah satu metrik analisis yang diukur. Berupa rasio jumlah pengiriman paket yang terkirim.
3	E2E	*Average End-to-End Delay*. Jeda waktu yang diukur saat paket terkirim.
4	RO	*Routing Overhead*. Jumlah *control packet* yang terkirim
5	*Throughput*	Berbagi informasi node dan membuat jalur rute alternative untuk membantu mengurangi penyiaran *control message* pada jaringan
6	*Hello Message*	Pesan untuk membantu mengetahui tetangga sebuah *node*
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 sudah terbuat.
9	*Flag_RREQ*	Variabel yang disimpan di *node* untuk membatasi penerusan RREQ sebanyak dua kali

3.2 Perancangan Skenario Mobilitas

Perancangan skenario mobilitas dimulai dengan membuat area simulasi, pergerakan *node*, dan implementasi pergerakan *node*. Dalam

Tugas Akhir ini, terdapat dua macam area simulasi yang akan digunakan yaitu peta *grid* dan *real*. Peta *grid* yang dimaksud adalah bentuk jalan berpetak – petak sebagai contoh jalan berpotongan yang sederhana. Peta *grid* digunakan sebagai simulasi awal VANETs karena lebih stabil. Peta *grid* didapatkan dengan menentukan panjang dan jumlah petak area menggunakan SUMO. Sedangkan yang dimaksud peta *real* adalah peta asli / nyata yang digunakan sebagai area simulasi. Peta *real* didapatkan dengan mengambil daerah yang diinginkan sebagai area simulasi menggunakan *OpenStreetMap*. Pada Tugas Akhir ini, peta *real* yang diambil penulis adalah salah satu area di kota Surabaya. Pada Gambar 3.2 berikut alur perancangan skenario



Gambar 3.2 Alur perancangan skenario

3.2.1 Perancangan Skenario *Grid*

Perancangan skenario mobilitas *grid* diawali dengan merancang luas area peta *grid* yang dibutuhkan. Luas area tersebut bisa didapatkan dengan cara menentukan terlebih dahulu jumlah titik persimpangan yang diinginkan, sehingga dari jumlah persimpangan tersebut dapat diketahui berapa banyak peta yang dibutuhkan. Dengan mengetahui jumlah petak yang dibutuhkan, dapat ditentukan panjang tiap petak sehingga mendapatkan luas area yang dibutuhkan yaitu 600 m x 600 m. Dengan 4 titip persimpangan, maka akan didapatkan 9 petak dan panjang tiap peta adalah 200m.

Peta *grid* yang telah ditentukan luasnya tersebut kemudian dibuat dengan menggunakan *tools* SUMO yaitu *netgenerate*. Selain titik persimpangan dan panjang tiap petak *grid*, dibutuhkan juga pengaturan kecepatan kendaraan menggunakan *tools* tersebut. Peta *grid* yang dihasilkan oleh *netgenerate* akan memiliki ekstensi .net.xml. Peta *grid* ini kemudian digunakan untuk membuat pergerakan *node* dengan *tools* SUMO yaitu menggunakan *tools randomTrips* dan *duarouter*.

Skenario mobilitas *grid* dihasilkan dengan menggabungkan *file* peta *grid* dan *file* pergerakan *node* yang telah dibuat. Penggabungan tersebut menghasilkan *file* dengan ekstensi .xml. Selanjutnya, untuk dapat diterapkan pada NS-2, *file* skenario mobilitas *grid* yang berekstensi .xml dikonversi ke dalam bentuk *file* .tcl. Konversi ini dilakukan menggunakan *tool traceExporter*.

3.2.2 Perancangan Skenario *Real*

Perancangan skenario mobilitas *real* diawali dengan memilih area yang akan dijadikan simulasi. Pada Tugas Akhir ini, digunakan peta dari *OpenStreetMap* untuk mengambil area yang dijadikan model simulasi. Setelah memilih area, dilakukan pengunduhan dengan menggunakan fitur *export* yang telah disediakan oleh *OpenStreetMap*. Peta hasil *export* tersebut memiliki ekstensi .osm.

Setelah mendapatkan peta area yang akan dijadikan simulasi, peta tersebut dikonversi ke dalam bentuk *file* dengan ekstensi .net.xml menggunakan *tools* SUMO yaitu *netconvert*. Tahap berikutnya memiliki tahapan yang sama seperti merancang skenario *grid*, yaitu membuat pergerakan *node* menggunakan *randomTrips* dan *duarouter*. Kemudian dilakukan penggabungan *file* peta *real* yang sudah dikonversi ke dalam *file* dengan ekstensi .net.xml dan *file* pergerakan *node* yang sudah dibuat sebelumnya. Hasil dari penggabungan tersebut merupakan *file* skenario berkekstensi .xml. *File* yang dihasilkan tersebut dikonversi ke dalam bentuk *file* dengan ekstensi .tcl agar dapat diterapkan pada NS-2.

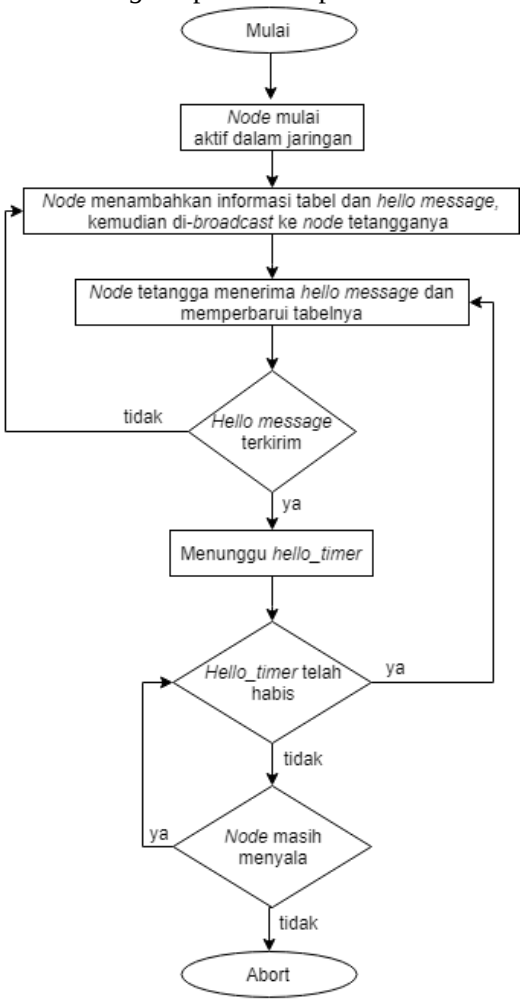
3.3 Perancangan Modifikasi *Routing Protocol* AODV

Protokol AODV yang diajukan pada Tugas Akhir ini merupakan modifikasi dari protokol AODV yang mengubah mekanisme pengiriman *hello message* dan *route discovery*. Pada protokol AODV, pengiriman *hello message* dilakukan oleh *node* dengan cara *broadcast* dengan mengirimkan posisinya kepada *node* yang lain guna mengetahui tetangganya dan memungkinkan *node* penerima memperbarui tabel *log* hanya 1 segmen. Sedangkan pada protokol AODV yang telah dimodifikasi ini pengiriman *hello message* dengan menyertakan informasi *node* tetangga yang berdekatan dan memungkinkan *node* penerima akan memperbarui tabel *log* 1 hingga 2 segmen.

Selanjutnya, dilakukan modifikasi pada proses *route discovery* dengan cara memberikan *flag_RREQ*. *Flag_RREQ* dimulai dengan nilai nol yang akan bertambah apabila sebuah *node* menerima pesan kontrol RREQ. *Node* akan meneruskan RREQ apabila *flag_RREQ* kurang dari 2 dan merupakan *intermediate node. Intermediate node* yaitu *node* perantara antara *node* sumber dan *node* tujuan. Hal tersebut akan mengakibatkan node menerima *duplicate* RREQ yang akan dimanfaatkan sebagai *alternative path.*

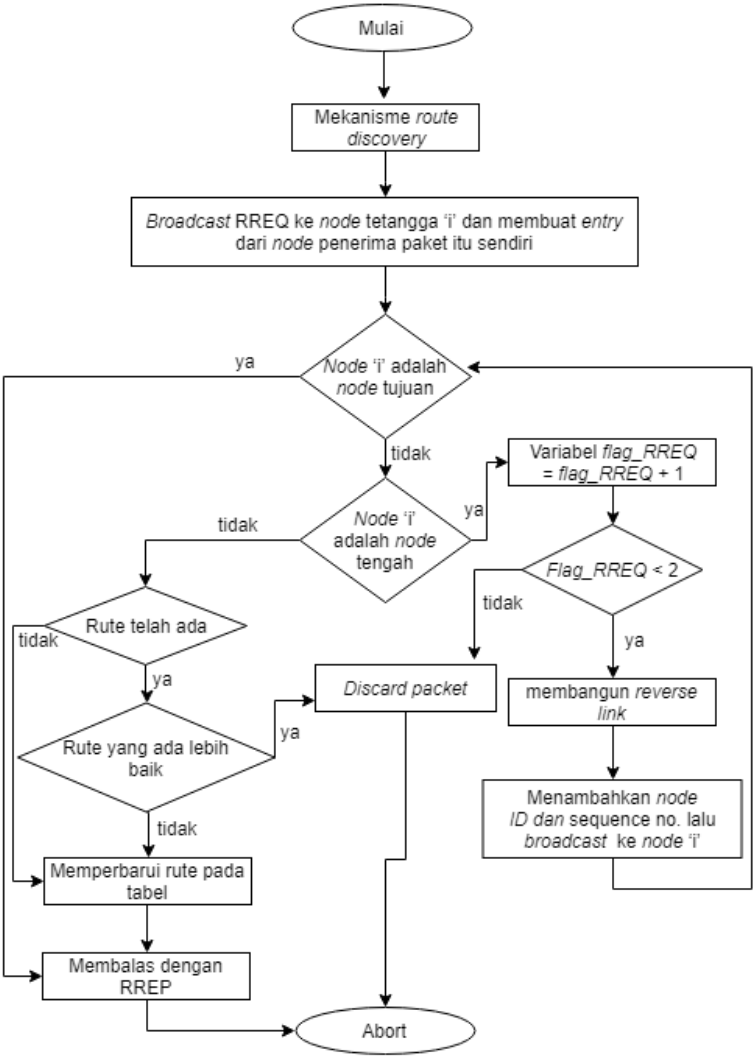
Alternative path atau jalur alteranatif merupakan jalur cadangan yang digunakan ketika jalur utama mengalami kerusakan

atau terputus. Bagi *node* yang menerima RREQ lebih dari dua, maka RREQ tersebut akan di drop dan tidak diteruskan kembali oleh *node* tersebut. Sehingga kemacetan dan tabrakan paket pada jaringan akan lebih rendah dan dapat menaikkan PDR. *Flow chart* modifikasi pada pengiriman *hello message* dapat dilihat pada Gambar 3.3 berikut.



Gambar 3.3 *Flow Chart* Pengiriman *Hello Message* AODV Modifikasi

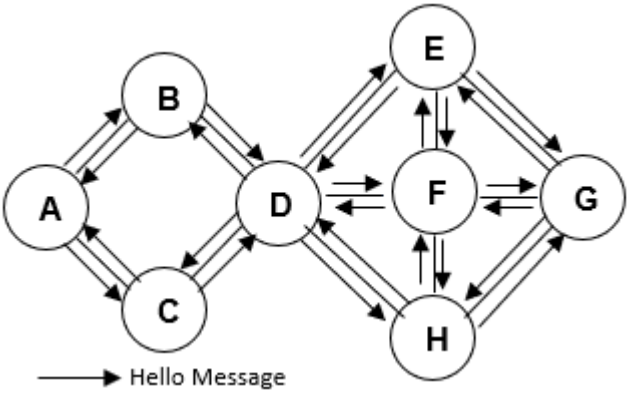
Sedangkan untuk modifikasi pada proses *route discovery* dapat dilihat pada Gambar 3.4 berikut.



Gambar 3.4 *Flow Chart* Proses *Route Discovery* AODV Modifikasi

3.3.1 Perancangan Penggabungan Hello Message dengan Informasi Node yang Berdekatan

Banyak cara untuk dapat menggabungkan *hello message* dengan informasi *node* yang berdekatan. Pada Tugas Akhir ini, penggabungan *hello message* dengan informasi *node* yang berdekatan dengan cara menambahkan *variable* untuk menampung informasi sebuah *node* tetangga dan di-*broadcast* melalui *hello message*. Modifikasi tersebut dilakukan pada fungsi `sendHello()`. Selanjutnya modifikasi dilakukan dengan menambahkan *variable array* untuk menampung *node* tetangga dan informasi *node* yang berdekatan tersebut dengan memodifikasi fungsi `recvHello()`. Fungsi-fungsi yang akan dimodifikasi tersebut terdapat pada file aodv.cc yang terletak di dalam folder ns-2.35/aodv. Dan melakukan deklarasi *variable* pada file aodv_packet.h yang juga terletak di dalam folder ns-2.35/aodv. Berikut Gambar 3.5 merupakan pengiriman *hello message* pada protokol AODV yang akan dimodifikasi



Gambar 3.5 *Flow Hello Message* pada Jaringan

Pseuocode untuk penggabungan *hello message* dan informasi *node* yang berdekatan dapat dilihat pada Gambar 3.6 berikut

```
1. Start
2. Node_i appends log table with HELLO Message
   and broadcst in the network
   [HELLO Message_new = HELLO Message_i + Node_i
   Log]
3. Node_j receives HELLO Message new and
   update log table & RTT (Round-Trip Time)
4. If Node_j already sent HELLO Message, then
5.    Wait until HELLO_TIMER gets over
6. Else
7.    Node_j appends log table with HELLO
   Message and broadcast in the network
   [HELLO Message_new = HELLO Message_j + Node_j
   Log]
8. End If
9. Stop
```

Gambar 3.6 *Pseudocode* Penggabungan *Hello Message* dan Informasi *Node* Terdekat

3.3.2 Perancangan Mekanisme *Flag_RREQ* dan *Broadcast* Node

Modifikasi yang dilakukan pada proses *route discovery*, salah satunya yaitu membatasi *broadcast* RREQ dengan memberikan *flag_RREQ* yang akan bertambah ketika menerima RREQ. RREQ tidak akan diteruskan atau di drop apabila *flag_RREQ* lebih dari dua. Di samping itu, apabila *flag_RREQ* kurang dari dua maka node tersebut akan ditambahkan pada *reverse link*. *Pseducode* untuk mekanisme *flag_RREQ* dapat dilihat pada Gambar 3.7 berikut

```
If (old RREQ)
      If flag_RREQ <2
      Build reverse link
      Rebroadcast
      Increment flag
Else
      Reset flag
```

Gambar 3.7 *Pseudocode Flag*_RREQ

Hal tersebut mengakibatkan *duplicate* RREQ yang dapat dimanfaatkan sebagai *alternative path.* Untuk meneruskan RREQ agar mendapatkan rute berbeda digunakan variable *broadcast_node. Broadcast_node* berisi informasi *list* id tetangga *node* yang meneruskan RREQ. Ketika sebuah *node* pertama kali menerima RREQ maka *node* tersebut akan menyimpan sementara(chace) *broadcast node* yang ada di dalam RREQ. Ketika menerima RREQ, akan dilakukan pengecekan apakah *node* yang meneruskan RREQ ada di list *broadcast_node* atau tidak. Jika iya, RREQ yang memiliki rute tidak lebih baik akan di *drop*. Jika tidak ada, RREQ akan diteruskan dan dibuat *reverse link* yaitu *node* yang dapat meneruskan RREQ. Modifikasi akan dilakukan pada fungsi `recvRequest()` untuk menambahkan *flag_RREQ* dan *broadcast_node.* Sedangkan untuk memperbarui list tetangga dilakukan modifikasi pada fungsi `nb_insert()` dan `nb_delete()`. Fungsi-fungsi tersebut terdapat pada file aodv.cc yang terletak di dalam folder ns-2.35/aodv. *Pseudocode* untuk mekanisme *broadcast_node* dapat dilihat pada Gambar 3.8 berikut

```
1. If (old RREQ)
2.   If (forwarder_RREQ is in
   broadcast_node)
3.     Drop RREQ
4.   Else
5.     Append it's id neighbour to RREQ
   broadcast_node
6.     Build reverse link
7.     Rebroadcast RREQ
8. Else
9.   Clear cache broadcast_node
10.  Cache new broadcast node
```

Gambar 3.8 *Pseudocode* Mekanisme *Broadcast Node*

3.4 Perancangan Simulasi pada NS-2

Simulasi VANETs pada NS-2 dilakukan dengan menggabungkan *file* skenario yang telah dibuat menggunakan SUMO

dan *file* skrip dengan ekstensi .tcl yang berisikan konfigurasi lingkungan simulasi. Pada saat simulasi NS-2 dijalankan, maka *routing protocol* AODV akan menyeleksi *node* mana saja yang berhak melakukan *rebroadcast* paket RREQ.

3.5 Perancangan Metrik Analisis

Berikut ini merupakan parameter – parameter yang akan dianalisis pada Tugas Akhir ini untuk dapat membandingkan performa dari *routing protocol* AODV yang asli dengan AODV yang telah dimodifikasi:

3.5.1 *Packet Delivery Ratio* (PDR)

Packet delivery ratio merupakan perbandingan dari jumlah paket data yang dikirim dengan paket data yang diterima. PDR dapat menunjukkan keberhasilan paket yang dikirimkan. Semakin tinggi PDR artinya semakin berhasil pengiriman paket yang dilakukan Rumus untuk menghitung PDR dapat dilihat pada persamaan 3.1.

$$PDR = \frac{received}{sent} x\ 100\ \% \qquad (3.1)$$

Keterangan:

PDR = *Packet Delivery Ratio*

received = banyak paket data yang diterima

sent = banyak paket data yang dikirimkan

3.5.2 *Average End-to-End Delay* (E2E)

Average End-to-End Delay dihitung berdasarkan rata-rata *delay* antara waktu paket data diterima dan waktu paket dikirimkan

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, maka akan didapatkan rata – rata E2E, yang dapat dihitung dengan persamaan 3.2.

$$E2E = \frac{\sum_{m=1}^{recvnum} CBRRecvTime - CBRSentTime}{recvnum} \quad (3.2)$$

Keterangan:

E2E = *End-to-End Delay*

CBRRecvTime = Waktu *node* asal mengirimkan paket

CBRSentTime = Waktu *node* tujuan menerima paket

recvnum = Jumlah paket yang berhasil diterima

3.5.3 *Routing Overhead* (RO)

Routing Overhead adalah jumlah paket kontrol *routing* yang ditransmisikan per data paket ke *node* tujuan selama simulasi terjadi. *Routing Overhead* didapatkan dengan menjumlahkan semua paket kontrol *routing* yang ditransmisikan, baik itu paket *route request* (RREQ), *route reply* (RREP), maupun *route error* (RERR). Perhitungan *Routing Overhead* dapat dilihat dengan persamaan 3.3.

$$RO = \sum_{m=1}^{sentnum} packet\ sent \quad (3.3)$$

Keterangan:

RO = *Routing Overhead*

Sentnum = Jumlah paket yang dikirim

Packet sent = Jumlah semua paket yang berhasil terkirim

3.5.4 *Throughput*

Throughput merupakan banyaknya *byte* yang diterima dalam selang waktu tertentu dengan satuan *byte* per *second* yang merupakan kondisi data *rate* sebenarnya dalam suatu jaringan. Besarnya selang waktu pengukuran dapat mempengaruhi hasil gambaran perilaku jaringan. Perhitungan *Throughput* dapat dilihat dengan persamaan 3.4

$$T = \frac{Pr}{t}\ x\ ukuran\ paket \qquad (3.4)$$

Keterangan :

Pr = Jumlah paket yang diterima
T = Throughput
T = Total waktu pengamatan

BAB IV
IMPLEMENTASI

Pada bab ini akan dibahas mengenai implementasi dari perancangan yang sudah dilakukan pada bab sebelumnya. Implementasi berupa kode sumber untuk membangun program.

4.1 Implementasi Skenario Mobilitas

Implementasi skenario mobilitas VANETs dibagi menjadi dua, yaitu skenario *grid* yang menggunakan peta jalan berpetak dan skenario *real* yang menggunakan peta hasil pengambilan suatu area di kota Surabaya.

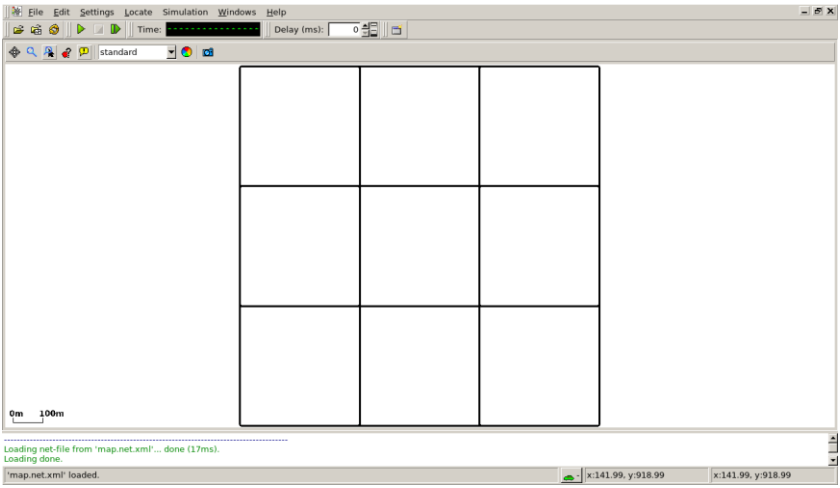
4.1.1 Skenario *Grid*

Dalam mengimplementasikan skenario *grid*, SUMO menyediakan *tools* untuk membuat peta *grid* yaitu *netgenerate*. Pada Tugas Akhir ini, penulis membuat peta *grid* dengan luas 600 m x 600 m yang terdiri dari titik persimpangan antara jalan vertikal dan jalan horizontal sebanyak 4 titik x 4 titik. Dengan jumlah titik persimpangan sebanyak 4 titik tersebut, makat terbentuk 9 buah petak. Sehingga untuk mencapai luas area sebesar 600 m x 600 m dibutuhkan luas per petak sebesar 200 m x 200 m. Berikut perintah *netgenerate* untuk membuat peta tersebut dengan kecepatan *default* kendaraan sebesar 20m/s dapat dilihat pada Gambar 4.1.

```
netgenerate --grid --grid.number=4 --
grid.length=400 --default.speed=20 --
tls.guess=1 --output-file=map.net.xml
```

Gambar 4.1 Perintah *netgenerate*

Setelah itu akan didapat *file* peta berekstensi .xml. Gambar hasil 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 awal dan titik akhir setiap *node* secara *random* menggunakan *tools randomTrips* yang terdapat di SUMO. Perintah penggunaan *tools randomTrips* untuk membuat *node* sebanyak n *node* dengan pergerakannya dapat dilihat pada Gambar 4.3.

```
python $SUMO_HOME/tools/randomTrips.py -n
map.net.xml -e 58 -l --trip-
attributes="departLane=\"best\"
departSpeed=\"max\"
departPos=\"random_free\"" -o trip.trips.xml
```

Gambar 4.3 Perintah *randomTrips*

Selanjutnya, dibuatkan rute yang digunakan kendaraan untuk mencapai tujuan dari *file* hasil sebelumnya menggunakan *tools duarouter*. Perintah penggunaan *tools* duarouter dapat dilihat pada Gambar 4.4.

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

Gambar 4.4 Perintah *duarouter*

Ketika menggunakan *tools duarouter*, SUMO memastikan bahwa jalur untuk *node-node* yang di*generate* tidak akan melenceng dari jalur peta yang sudah di*generate* menggunakan *tools randomTrips*. Selanjutnya untuk menjadikan peta dan pergerakan *node* yang telah di*generate* menjadi sebuah skenario dalam bentuk *file* berekstensi .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="route.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 dan *file* rute pergerakan *node.* Untuk percobaan sebelum dikonversi, *file* .sumocfg dapat dibuka dengan menggunakan *tools* sumo-gui. Kemudian buat *file* skenario dalam bentuk *file* .xml dari sebuah *file* skrip berekstensi .sumocfg menggunakan *tools* SUMO. Perintah untuk menggunakan *tools* SUMO dapat dilihat pada Gambar 4.6.

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

Gambar 4.6 Perintah SUMO untuk membuat skenario .xml

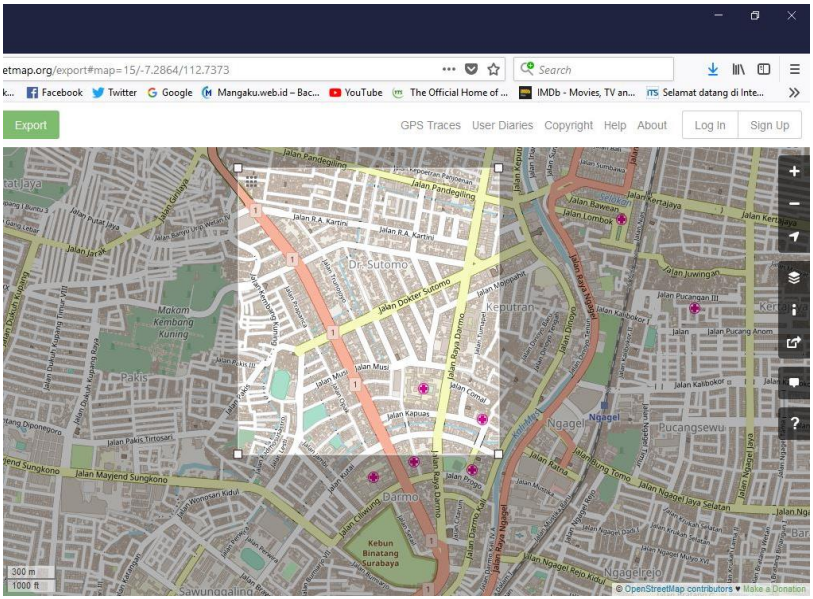
File skenario berekstensi .xml selanjutnya dikonversi ke dalam bentuk *file* berekstensi .tcl agar dapat disimulasikan menggunakan NS-2. *Tools* yang digunakn untuk melakukan konversi ini adalah *traceExporter*. Perintah untuk menggunakan *traceExporter* dapat dilihat pada Gambar 4.7.

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

Gambar 4.7 Perintah *traceExporter*

4.1.2 Skenario *Real*

Dalam mengimplementasikan scenario *real,* langkah pertama adalah menentukan area yang akan dijadikan area simulasi. Pada Tugas Akhir ini penulis mengambil area jalan sekitar Jl. Dr. Soetomo Surabaya. Setelah menentukan area simulasi, ekspor data peta dari *OpenStreetMap* seperti yang ditunjukkan pada Gambar 4.8.



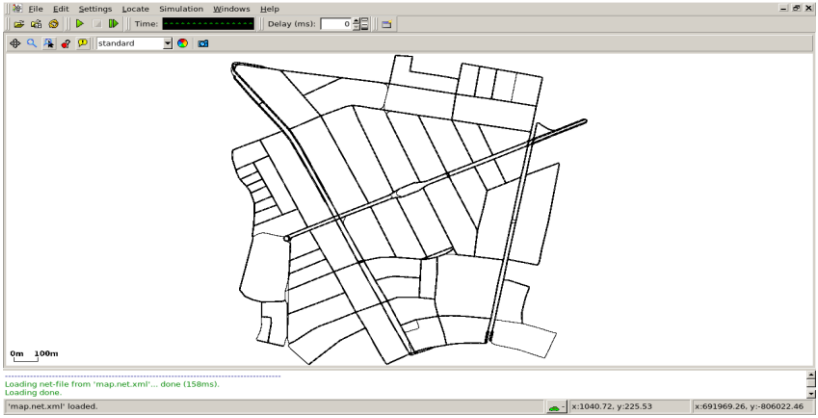
Gambar 4.8 Ekspor Peta dari *OpenStreetMap*

File hasil ekspor dari *OpenStreetMap* tersebut adalah *file* peta dengan ekstensi .osm. Kemudian konversi *file* .osm tersebut menjadi peta dalam bentuk *file* berekstensi .xml menggunakan *tools 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 *tools* sumo-gui seperti yang ditunjukkan pada Gambar 4.10.



Gambar 4.10 Hasil Konversi Peta Real

Langkah selanjutnya sama dengan ketika membuat skenario mobilitas *grid*, yaitu membuat *node* asal dan *node* tujuan menggunakan *tool* randomTrips. Lalu membuat rute *node* untuk sampai ke tujuan menggunakan *tool duarouter*. Kemudian membuat *file* skenario berekstensi .xml menggunakan *tool* SUMO dengan bantuan *file* skrip berekstensi .sumocfg. Selanjutnya dilakukan konversi *file* skenario berekstensi .tcl untuk dapat disimulasikan pada NS-2 menggunakan *tool traceExporter*. Perintah untuk menggunakan *tools* tersebut sama dengan ketika membuat skenario *grid* di atas.

4.2 Implementasi Modifikasi pada *Routing Protocol* AODV untuk Meningkatkan Kinerja Protokol

Pada Tugas Akhir ini, dilakukan modifikasi pada *routing protocol* AODV agar kinerja protokol AODV dapat meningkat. Hal tersebut dilakukan dengan memanfaatkan informasi dua *hop* tetangga dan menambahkan rute alternative. Setiap *node* akan memelihara dua

rute yang berbeda dari *node* sumber ke *node* tujuan dengan memberikan *flag_RREQ* dan menambahkannya pada variable *broadcast node.*

Implementasi modifikasi *routing protocol* AODV ini dibagi menjadi 2 bagian yaitu:

- Implementasi Penggabungan *Hello Message* dengan Informasi *Node* yang Berdekatam
- Implementasi Mekanisme *Flag_RREQ* dan *Broadcast Node*

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 dan sebagainya. Pada Tugas Akhir ini, penulis memodifikasi *file* aodv.cc yang terdapat dalam *folder* ns-2.35/aodv untuk menggabungkan informasi *node* yang berdekatan dengan *hello message*, memberi *flag_RREQ* dan menambahkan *broadcast node* untuk menentukan *node* yang dapat melakukan *rebroadcast* RREQ. Selain itu, penulis juga memodifikasi *file* aodv_packet.h yang ada di dalam *folder* ns-2.35/aodv untuk mendeklarasikan variable *array* untuk menggabungkan *hello message* dan informasi node yang berdekatan. Pada bagian ini penulis akan menjelaskan langkah – langkah dalam mengimplementasikan modifikasi *routing protocol* AODV untuk menggabungkan informasi *node* yang berdekatan dengan *hello message* dan mekanisme *alternative path.*

4.2.1 Implementasi Penggabungan *Hello Message* dengan Informasi *Node* yang Berdekatan

Langkah awal yang dilakukan pada modifikasi AODV ini adalah menggabungkan *hello message* dengan informasi *node* yang berdekatan dengan cara seperti yang dirancang pada subbab 3.3.1. Pertama dilakukan deklarasi *variable array* untuk menyimpan penggabungan antara *hello message* dengan informasi *node* yang berdekatan pada *file* aodv_packet.h. Kode sumbernya dapat dilihat pada Gambar 4.11 berikut

```
nsaddr_t          list_neighbor[302];  //list
neighbor
int               jumlah_tetanggaku;
```

Gambar 4.11 Potongan kode untuk deklarasi *variable array hello message*

Langkah selanjutnya yaitu dengan melakukan penggabungan daftar tetangga yang berdekatan melalui *hello message*, yang dikirim melalui *variable array* yaitu tetangga. Modifikasi dilakukan pada *file* aodv.cc di fungsi `sendHello()` dan kode sumbernya dapat dilihat pada Gambar 4.12 berikut

```
//kirim list tetangga lewat hello message
  AODV_Neighbor *nb = nbhead.lh_first;
  int i = 0;
  for(; nb; nb = nb->nb_link.le_next) {
   rh->list_neighbor[i++] = nb->nb_addr;
  }
  rh->jumlah_tetanggaku = i;
```

Gambar 4.12 Potongan kode untuk pengirman *list* tetangga melalui *hello message*

Setelah itu, dilakukan penambahan informasi tersebut pada *variable list_neighbor* yang dimodifikasi pada fungsi `recvHello(Packet *p)`. Potongan kode sumber yang ditambahkan seperti Gambar 4.13 berikut

```
//memasukkan list tetangganya tetangga ke list
  for (int i = 0; i < rp->jumlah_tetanggaku;
i++)
  {
    //nb_insert(rp->list_neighbor[i]);
    nb = nb_lookup(rp->list_neighbor[i]);
    if(nb == 0) nb_insert(rp->list_neighbor[i]);
  }
```

Gambar 4.13 Potongan kode untuk penggabungan hello message dan informasi node yang berdekatan

4.2.2 Implementasi Mekanisme *Flag_RREQ* dan Broadcast Node

Pada tahap selanjutnya setelah menggabungkan *hello message* dengan informasi *node* yang berdekatan, dilakukan modifikasi pada proses *route discovery*. Modifikasi pada proses ini sesuai dengan yang dirancang pada subbab 3.3.1 adalah sebagai berikut:

1. Deklarasi *variable*

   Variable *flag_RREQ* digunakan sebagai penanda sebuah *node* sudah melakukan *rebroadcast* RREQ berapa kali. *Flag_RREQ* dibuat dalam bentuk *array* dan bertipe *integer*. Deklarasi *flag_RREQ* dilakukan pada *file* aodv.cc. Kode sumbernya dapat dilihat pada Gambar 4.14 berikut

```
int flag_RREQ[302];
std::vector < std::vector<int> >
broadcast_node(302, std::vector<int>(0));
```

Gambar 4.14 Potongan kode untuk deklarasi *variable flag*

2. Pengubahan kode sumber untuk mekanisme *flag_RREQ*

   Setelah melakukan deklarasi *variable flag_RREQ*, maka selanjutnya adalah menambahkan kode sumber jalannya *flag_RREQ* yaitu akan bertambah apabila sebuah *node* menerima *control message* RREQ. Modifikasi dilakukan pada *file* aodv.cc pada fungsi `AODV::recvRequest()` dengan kode sumber pada Gambar 4.15 berikut

```
if (id_lookup(rq->rq_src,rq->rq_bcast_id)){
 if(flag_RREQ[index]<2){
  flag_RREQ[index]++;
 }
 else{
  Packet::free(p);
    return;
```

```
    }
   }
else{
 flag_RREQ[index]=1;
}
```

Gambar 4.15 Potongan kode mekanisme flag_RREQ

3. Penambahan kode sumber untuk mekanisme *broadcast*

Langkah selanjutnya setelah menambahkan kode sumber mekanisme *flag_RREQ* yaitu menambahkan kode untuk melakukan *broadcast* paket kontrol. Modifikasi dilakukan pada *file* aodv.cc pada fungsi `AODV::recvRequest()`. Berikut kode sumber untuk mekanisme *broadcast* paket kontrol dapat dilihat pada Gambar 4.16.

```
if (id_lookup(rq->rq_src, rq->
   rq_bcast_id)) {
 for (unsigned int i = 0; i <
 broadcast_node[index].size(); i++)
 {
 if(broadcast_node[index][i]==ih ->
   saddr()){
  Packet::free(p);
  return;
  }
 }
}
else{
 broadcast_node[index].clear();
 broadcast_node[index].push_back(ih-
   >saddr());
 for (unsigned int i = 0; i <
   list_neighbor[ih->saddr()].size();
   i++){

   broadcast_node[index].push_back(list_ne
   ighbor[ih->saddr()][i]);
```

```
  }
}
```

Gambar 4.16 Potongan kode sumber untuk mekanisme *broadcast* paket kontrol

4. Implementasi *update list neighbor*

   Update list neighbor dilakukan pada saat menambah atau menghapus daftar tetangga sebuah *node.* Penambahan kode dilakukan pada *file* aodv.cc pada fungsi `AODV::nb_insert()` dan `AODV::nb_delete()`. Kode sumber modifikasi dapat dilihat pada Gambar 4.17 berikut

```
AODV::nb_insert(nsaddr_t id) {
 list_neighbor[index].push_back(id);
}
AODV::nb_delete(nsaddr_t id) {
 for (unsigned int i = 0; i <
 list_neighbor[index].size(); i++)
 {
  if(list_neighbor[index][i]==id)
  {
 list_neighbor[index].erase
 (list_neighbor[index].begin()+i);
 break;
  }
 }
}
```

Gambar 4.17 Potongan kode *update neighbor*

4.3 Implementasi Simulasi pada NS-2

Implementasi simulasi VANETs 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 4.18.

```
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)       1500
set opt(y)       1500
set val(ifqlen)  1000
set val(nn)      60
set val(seed)    1.0
set val(adhocRouting) AODV
set val(stop)    200
set val(cp)      "cbr50.txt"
set val(sc)      "scenario1.txt"
```

Gambar 4.18 Implementasi Simulasi NS-2

Pada konfigurasi dilakukan pemanggilan terhadap *file traffic* yang berisikan konfigurasi *node* asal, *node* tujuan, pengiriman paket, dan *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*. Potongan konfigurasi *file traffic* dapat dilihat pada Gambar 4.19.

```
set udp_(0) [new Agent/UDP]
$ns_ attach-agent $node_(58) $udp_(0)
set null_(0) [new Agent/Null]
$ns_ attach-agent $node_(59) $null_(0)
set cbr_(0) [new Application/Traffic/CBR]
$cbr_(0) set packetSize_ 512
$cbr_(0) set interval_ 1
$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"
```

Gambar 4.19 Implementasi Simulasi *File Traffic*

Pada konfigurasi tersebut, ditentukan *node* sumber dan *node* tujuan pengiriman paket. Pengiriman dimulai pada detik ke- 2.55. Implementasi konfigurasi *file traffic* untuk simulasi pada NS-2 dapat dilihat pada lampiran A.5 Kode Konfigurasi *Traffic*

4.4 Implementasi Metrik Analisis

Simulasi yang telah dijalankan oleh NS-2 menghasilkan sebuah *trace file* yang berisikan data mengenai apa saja yang terjadi selama simulasi dalam bentuk *file* berekstensi .tr. Dari data *trace file* tersebut, dapat dilakukan analisis performa *routing protocol* dengan mengukur beberapa metrik. Pada Tugas Akhir ini, metrik yang akan dianalisis adalah PDR, E2E, dan RO, dan *Throughput*.

4.4.1 Implementasi *Packet Delivery Ratio* (PDR)

Pada subbab 2.3.2 telah ditunjukkan contoh struktur data *event* yang dicatat dalam *trace file* oleh NS-2. Kemudian, pada persamaan 3.1 telah dijelaskan bagaimana menghitung PDR. Skrip awk untuk menghitung PDR berdasarkan kedua informasi tersebut dapat dilihat pada lampiran A.6 Kode Skrip AWK *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 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 3.1. *Pseudocode* untuk menghitung PDR dapat dilihat pada Gambar 4.20.

```
sent = 0
received = 0
for i = 1 to the number of rows
     if in a row contains "s" and AGT then
          sent++
     else if in a row contains "r" and AGT then
          received++
     end if
pdr = received / sent
```

Gambar 4.20 *Pseudocode* untuk Menghitung PDR

Contoh perintah pengeksekusian skrip awk untuk menganalisis *trace file* adalah `awk -f pdr.awk tracefile.tr`.

4.4.2 Implementasi *Average End-to-End Delay (E2E)*

Skrip awk untuk menghitung E2E dapat dilihat pada lampiran A.7 Kode Skrip AWK Rata-Rata *End-to-End Delay*.

Dalam perhitungan E2E, langkah yang digunakan untuk mendapatkan E2E 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. *Pseudocode* untuk menghitung rata-rata E2E dapat dilihat pada Gambar 4.21.

```
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]
```

Gambar 4.21 *Pseudocode* untuk Perhitungan Rata-Rata E2E

Contoh perintah pengeksekusian skrip awk untuk menganalisis *trace file* adalah `awk –f e2e.awk tracefile.tr`.

4.4.3 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.

Perhitungan RO telah dijelaskan pada persamaan 3.3. Skrip AWK untuk menghitung RO dapat dilihat pada lampiran A.8 Kode Skrip AWK *Routing Overhead*. *Pseudocode* untuk menghitung RO dapat dilihat pada Gambar 4.22.

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

Gambar 4.22 *Pseudocode* untuk Perhitungan *Routing Overhead*

Contoh perintah pengeksekusian skrip awk untuk menganalisis *trace file* adalah `awk -f ro.awk scenario1.tr`.

4.4.4 Implementasi *Throughput*

Throughput merupakan banyaknya *byte* yang diterima dalam selang waktu tertentu dengan satuan *byte* per *second second* yang merupakan kondisi data *rate* sebenarnya dalam suatu jaringan. Besarnya selang waktu pengukuran dapat mempengaruhi hasil gambaran perilaku jaringan. Perhitungan *Throughput* telah dijelaskan pada persamaan 3.4. Skrip awk untuk menghitung *Throughput* dapat dilihat pada lampiran A.9 Kode Skrip AWK *Throughput*. *Pseudocode* untuk menghitung *Throughput* dapat dilihat pada Gambar 4.23.

```
sum_delay = 0
counter = 0

for i = 1 to the number of rows
     counter++
     if level == AGT and event == s and $7 ==
cbr
      sent ++
      if time < startTime
       startTime = time
     if level == AGT and event == r and $7 ==
cbr
      receive ++
      if time > stopTime
       stopTime = time
      recvSize += pkt_size
throughput = recvSize/(stopTime-
startTime)*(8/1000)
```

Gambar 4.23 *Pseudocode* Perhitungan *Throughput*

Contoh perintah pengeksekusian skrip awk untuk menganalisis *trace file* adalah `awk -f throughput.awk scenario1.tr`.

(Halaman ini sengaja dikosongkan)

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 yang Digunakan

Komponen	**Spesifikasi**
CPU	Intel(R) Core™ i5-7400 CPU @ 3.00GHz x 4
Sistem Operasi	ubuntu 16.04 LTS
Linux Kernel	4.15.0-50-generic
Memori	7.7 GB

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

- SUMO versi 1.2.0 untuk pembuatan skenario mobilitas VANETs.
- JOSM versi 10301 untuk penyuntingan peta *OpenStreetMap*.
- NS-2 versi 2.35 untuk simulasi skenario VANETs.

Parameter lingkungan uji coba yang digunakan pada NS-2 dapat diihat 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 PDR, E2E, RO, dan *Throughput* menggunakan kode yang terdapat pada lampiran A.6 Kode Skrip AWK *Packet Delivery Ratio*, A.7 Kode Skrip AWK Rata-

Rata *End-to-End Delay*, A.8 Kode Skrip AWK *Routing Overhead*, dan A.9 Kode Skrip Awk *Throughput.*

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	600 m x 600 m dan 1500 m x 1500 m
5	Jumlah *Node*	25, 50, 75, 100, 125, 150, 175, 200
6	Radius transmisi	400m
7	Kecepatan maksimum	20 m/s
8	Protokol MAC	IEEE 802.11

5.2 Hasil Uji Coba

5.2.1 Hasil Uji Coba Skenario *Grid*

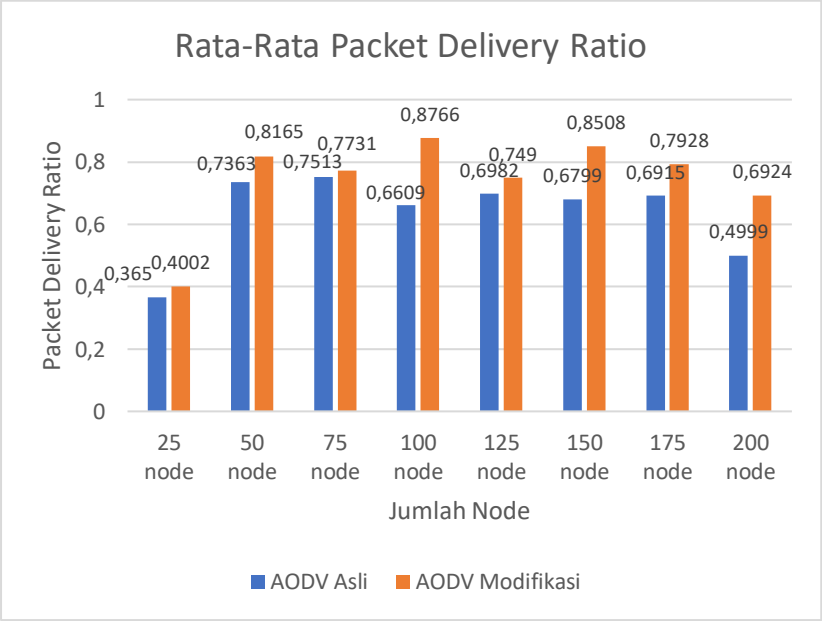
Pengujian pada skenario *grid* digunakan untuk melihat perbandingan PDR, E2E, RO, dan *Throughput* antara *routing protocol* AODV asli dan AODV yang telah dimodifikasi pada proses *route discovery* dan pengiriman paket.

Pengambilan data uji PDR, E2E, RO, dan *Throughput* pada skenario *grid* dilakukan sebanyak 10 kali dengan skenario mobilitas *random* pada peta *grid* dengan luas area 600 m x 600 m dan *node* sebanyak 25 dan 50 untuk lingkungan yang jarang, 75, 100, 125 dan 150 *node* untuk lingkungan yang sedang, dan 175 dan 200 *node* untuk lingkungan yang padat dilakukan pada kecepatan standar yaitu 20 m/s. Hasil analisis dapat dilihat pada Tabel 5.3, Tabel 5.4, Tabel 5.5, dan Tabel 5.6.

Tabel 5.3 Hasil Rata - Rata PDR Skenario *Grid*

Jumlah *Node*	**AODV Asli**	**AODV Modifikasi**
25	0.3650	0.4002
50	0.7363	0.8165
75	0.7513	0.7731
100	0.6609	0.8766
125	0.6982	0.7490
150	0.6799	0.8508
175	0.6915	0.7928
200	0.4999	0.6924

Dari data di atas, dibuat grafik yang merepresentasikan hasil perhitungan PDR yang ditunjukkan pada Gambar 5.1.



Gambar 5.1 Grafik Packet Delivery Ratio Skenario *Grid*

Berdasarkan grafik pada Gambar 5.1 dapat dilihat bahwa *routing protocol* AODV yang telah dimodifikasi mengalami kenaikan yang signifikan pada *packet delivery ratio*. Pada lingkungan yang jarang dengan jumlah 25 *node*, menghasilkan perbedaan selisih PDR sebesar 0.0350, atau naik menjadi sekitar 9.61%, dimana *routing protocol* AODV yang telah dimodifikasi unggul dalam hal PDR tersebut. Pada lingkungan yang jarang dengan jumlah 50 *node*, menghasilkan perbedaan selisih PDR sebesar 0.0802, atau naik menjadi sekitar 10.89%, dimana *routing protocol* AODV yang telah dimodifikasi juga unggul dalam hal PDR tersebut.

Pada lingkungan yang sedang dengan jumlah 75 *node*, menghasilkan perbedaan selisih PDR sebesar 0.0218, atau naik menjadi sekitar 2.89%, dimana *routing protocol* AODV yang telah dimodifikasi unggul dalam hal PDR tersebut dari AODV asli. Pada lingkungan yang sedang dengan jumlah 100 *node*, menghasilkan perbedaan selisih PDR sebesar 0.2178, atau naik menjadi sekitar 32.63%, dimana *routing protocol* AODV yang telah dimodifikasi unggul dalam hal PDR tersebut dari AODV asli. Pada lingkungan yang sedang dengan jumlah 125 *node*, menghasilkan perbedaan selisih PDR sebesar 0.0509, atau naik menjadi sekitar 7.3%, dimana *routing protocol* AODV yang telah dimodifikasi juga unggul dalam hal PDR tersebut dari AODV asli. Pada lingkungan yang sedang dengan jumlah 150 *node*, menghasilkan perbedaan selisih PDR sebesar 0.1709, atau naik menjadi sekitar 25.14%, dimana *routing protocol* AODV yang telah dimodifikasi juga unggul dalam hal PDR tersebut dari AODV asli.

Pada lingkungan yang padat dengan jumlah 175 *node*, menghasilkan perbedaan selisih PDR sebesar 0.1013 atau naik menjadi sekitar 14.65%, dimana *routing protocol* AODV yang telah dimodifikasi unggul dalam hal PDR tersebut. Pada lingkungan yang padat dengan jumlah 200 *node*, menghasilkan perbedaan selisih PDR sebesar 0.1924, atau naik menjadi sekitar 38.49%, dimana *routing protocol* AODV yang telah dimodifikasi juga unggul dalam hal PDR tersebut.

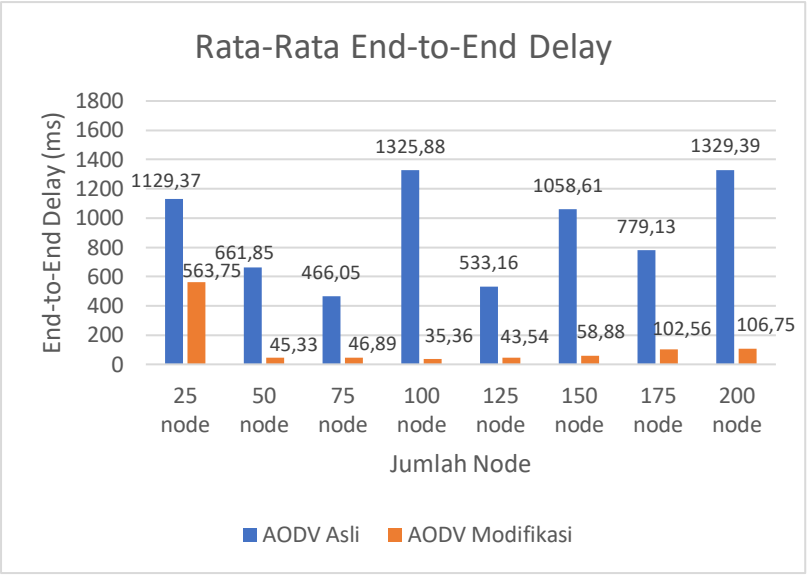
Dapat dilihat rata-rata kenaikan yang terjadi untuk PDR adalah 17.70%. Jika ketiga lingkungan tersebut dirata-rata, dapat dilihat bahwa AODV yang telah dimodifikasi menghasilkan PDR yang lebih bagus daripada AODV asli dengan jumlah selisih PDR yang cukup *signifikan.*

Hasil pengambilan data rata-rata untuk *end-to-end delay* (E2E) pada skenario *grid* dengan jumlah *node* 25, 50, 75, 100, 125, 150, 175, dan 200 dapat dilihat pada Tabel 5.4

Tabel 5.4 Hasil Rata - Rata E2E Skenario Grid

Jumlah *Node*	**AODV Asli(ms)**	**AODV Modifikasi(ms)**
25	1129.37	563.75
50	661.85	45.33
75	466.05	46.89
100	1325.88	35.36
125	533.16	43.54
150	1058.61	58.88
175	779.13	102.56
200	1329.39	106.75

Dari data di atas, dibuat grafik yang merepresentasikan hasil perhitungan rata-rata E2E yang ditunjukkan pada Gambar 5.2.



Gambar 5.2 Grafik *End-to-end Delay* Skenario *Grid*

Berdasarkan grafik pada Gambar 5.2 dapat dilihat bahwa rata-rata *end-to-end delay routing protocol* AODV yang telah dimodifikasi mengalami penurunan yang *signifikan*. Pada lingkungan yang jarang dengan jumlah 25 *node*, terjadi perbedaan selisih *end-to-end delay* sebesar 565.62 ms antara *routing protocol* AODV asli dengan *routing protocol* AODV yang telah dimodifikasi, atau mengalami penurunan sebesar 50.08%, dimana *routing protocol* AODV yang telah dimodifikasi unggul dalam hal *end-to-end delay* tersebut. Pada lingkungan yang jarang dengan jumlah 50 *node*, terjadi perbedaan selisih *end-to-end delay* sebesar 616.52 ms antara *routing protocol* AODV asli dengan *routing protocol* AODV yang telah dimodifikasi atau mengalami penurunan sebesar 93.15%, dimana *routing protocol* AODV yang telah dimodifikasi unggul dalam hal *end-to-end delay* tersebut.

Pada lingkungan sedang dengan jumlah 75 *node*, terjadi perbedaan selisih *end-to-end delay* sebesar 419.16 ms antara *routing protocol* AODV asli dengan *routing protocol* AODV yang telah

dimodifikasi atau mengalami penurunan sebesar 89.94%, dimana *routing protocol* AODV yang telah dimodifikasi juga lebih unggul dalam hal *end-to-end delay* tersebut. Pada lingkungan sedang dengan jumlah 100 *node*, terjadi perbedaan selisih *end-to-end delay* sebesar 1290.52 ms antara *routing protocol* AODV asli dengan *routing protocol* AODV yang telah dimodifikasi atau mengalami penurunan sebesar 97.33%, dimana *routing protocol* AODV yang telah dimodifikasi juga lebih unggul dalam hal *end-to-end delay* tersebut. Pada lingkungan sedang dengan jumlah 125 *node*, terjadi perbedaan selisih *end-to-end delay* sebesar 489.62 ms antara *routing protocol* AODV asli dengan *routing protocol* AODV yang telah dimodifikasi atau mengalami penurunan sebesar 91.83%, dimana *routing protocol* AODV yang telah dimodifikasi juga lebih unggul dalam hal *end-to-end delay* tersebut. Pada lingkungan sedang dengan jumlah 150 *node*, terjadi perbedaan selisih *end-to-end delay* sebesar 999.73 ms antara *routing protocol* AODV asli dengan *routing protocol* AODV yang telah dimodifikasi atau mengalami penurunan sebesar 94.44%, dimana *routing protocol* AODV yang telah dimodifikasi juga lebih unggul dalam hal *end-to-end delay* tersebut.

Pada lingkungan yang padat dengan jumlah 175 *node*, terjadi perbedaan selisih *end-to-end delay* sebesar 676.57 ms antara *routing protocol* AODV asli dengan *routing protocol* AODV yang telah dimodifikasi atau mengalami penurunan sebesar 86.84%, dimana *routing protocol* AODV yang telah dimodifikasi jauh lebih unggul juga dalam hal *end-to-end delay* tersebut. Pada lingkungan yang padat dengan jumlah 200 *node*, terjadi perbedaan selisih *end-to-end delay* sebesar 1222.65 ms antara *routing protocol* AODV asli dengan *routing protocol* AODV yang telah dimodifikasi atau mengalami penurunan sebesar 91.97%, dimana *routing protocol* AODV yang telah dimodifikasi jauh lebih unggul juga dalam hal *end-to-end delay* tersebut.

Dapat dilihat rata-rata penurunan yang terjadi untuk E2E adalah 86.95%. Jika ketiga lingkungan tersebut dirata-rata, *routing protocol* AODV yang telah dimodifikasi selalu lebih unggul dalam hal *end-to-end delay*. Hal ini dikarenakan perbedaan tindakan antara

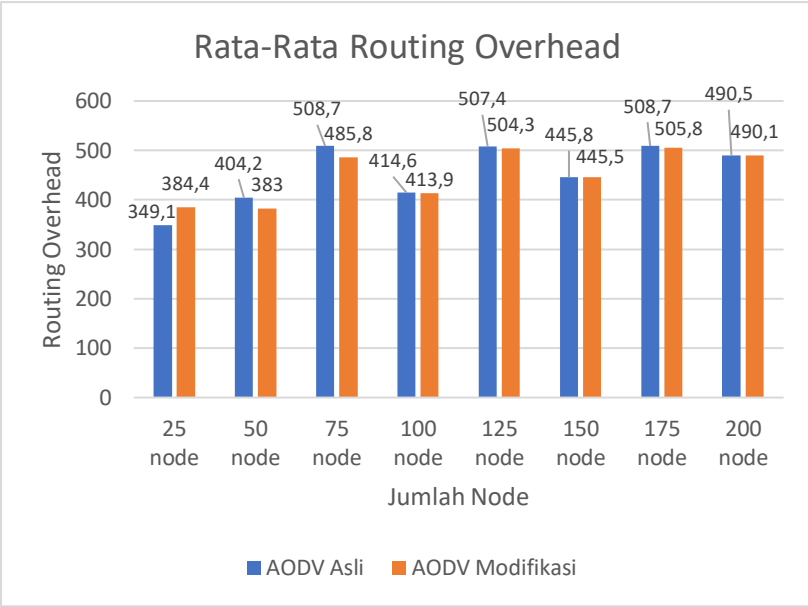
AODV asli dan AODV yang telah dimodifikasi dalam menemukan rute yang rusak. Pada AODV asli akan mencari ulang rute sehingga akan menambah *delay*. Sedangkan pada AODV yang telah dimodifikasi akan menggunakan *alternative path* sehingga tidak melakukan pencarian ulang dan tidak menambah *delay*.

Untuk hasil pengambilan data *routing overhead* pada skenario *grid* 25, 50, 75, 100, 125, 150, 175 dan 200 *node* dapat dilihat pada Tabel 5.5.

Tabel 5.5 Hasil Rata - Rata RO Skenario Grid

Jumlah *Node*	**AODV Asli**	**AODV Modifikasi**
25	349.1	384.4
50	404.2	383
75	508.7	485.8
100	414.6	413.9
125	507.4	504.3
150	445.8	445.5
175	508.7	505.8
200	490.5	490.1

Dari data di atas, dibuat grafik yang merepresentasikan hasil rata-rata RO yang ditunjukkan pada Gambar 5.3.



Gambar 5.3 Grafik *Routing Overhead* Skenario *Grid*

Berdasarkan grafik pada Gambar 5.3 dapat dilihat bahwa rata-rata *routing overhead routing protocol* AODV yang telah dimodifikasi mengalami penurunan yang *fluktuatif*. Pada lingkungan yang jarang dengan jumlah 25 *node*, menghasilkan perbedaan selisih *routing overhead* sebesar 35.3 atau mengalami kenaikan sebesar 10.11%, dimana *routing protocol* AODV asli unggul dalam hal *routing overhead* tersebut dari AODV yang telah dimodifikasi. Pada lingkungan yang jarang dengan jumlah 50 *node*, menghasilkan perbedaan selisih *routing overhead* sebesar 21,1 atau mengalami penurunan sebesar 5.24%, dimana *routing protocol* AODV yang telah dimodifikasi unggul dalam hal *routing overhead* tersebut karena menghasilkan *routing overhead* yang lebih rendah dari *routing protocol* AODV asli.

Pada lingkungan yang sedang dengan jumlah 75 *node*, menghasilkan perbedaan selisih *routing overhead* sebesar 22.9 atau mengalami penurunan sebesar 4.50%, dimana *routing protocol*

AODV yang telah dimodifikasi lebih unggul dalam hal *routing overhead* tersebut dari AODV asli. Pada lingkungan yang sedang dengan jumlah 100 *node*, menghasilkan perbedaan selisih *routing overhead* sebesar 0.7 atau mengalami penurunan sebesar 0.17%, dimana *routing protocol* AODV yang telah dimodifikasi juga lebih unggul dalam hal *routing overhead* tersebut dari AODV asli. Pada lingkungan yang sedang dengan jumlah 125 *node*, menghasilkan perbedaan selisih *routing overhead* sebesar 3.1 atau mengalami penurunan sebesar 0.61%, dimana *routing protocol* AODV yang telah dimodifikasi unggul juga dalam hal *routing overhead* tersebut dari AODV asli. Pada lingkungan yang sedang dengan jumlah 150 *node*, menghasilkan perbedaan selisih *routing overhead* sebesar 0.3 atau mengalami penurunan sebesar 0.07%, dimana *routing protocol* AODV yang telah dimodifikasi unggul juga dalam hal *routing overhead* tersebut dari AODV asli.

Pada lingkungan yang padat dengan jumlah 175 *node*, menghasilkan perbedaan selisih *routing overhead* sebesar 2.9 atau mengalami penurunan sebesar 0.57%, dimana *routing protocol* AODV yang telah dimodifikasi juga lebih unggul daripada AODV asli dalam hal *routing overhead* tersebut. Pada lingkungan yang padat dengan jumlah 200 *node*, menghasilkan perbedaan selisih *routing overhead* sebesar 0.4 atau mengalami penurunan sebesar 0.08%, dimana *routing protocol* AODV yang telah dimodifikasi juga lebih unggul daripada AODV asli dalam hal *routing overhead* tersebut.

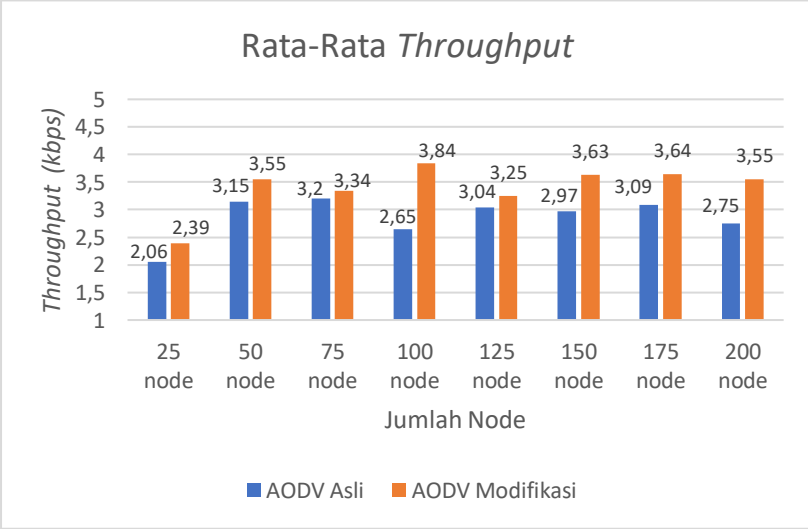
Dapat dilihat rata-rata penurunan yang terjadi untuk *routing overhead* adalah sebesar 0.14%. Jika ketiga lingkungan tersebut dirata-rata, dapat dilihat bahwa AODV yang telah dimodifikasi menghasilkan *routing overhead* yang sedikit lebih rendah dibandingkan dengan AODV asli dikarenakan pada AODV yang telah dimodifikasi memanfaatkan penambahan informasi 2 *hop* tetangga pada *broadcast hello message* dan memanfaatkan *alternative path* untuk mencari rute yang rusak sehingga tidak banyak melakukan *broadcast control message* ulang. Hal tersebut membuktikan bahwa pada AODV yang telah dimodifikasi berhasil mempercepat proses *route discovery.*

Untuk hasil pengambilan data *Throughput* pada skenario *grid* 25, 50, 75, 100, 125, 150, 175, dan 200 *node* dapat dilihat pada Tabel 5.6

Tabel 5.6 Hasil Rata - Rata *Throughput* Skenario Grid

Jumlah *Node*	**AODV Asli(kbps)**	**AODV Modifikasi(kbps)**
25	2.06	2.39
50	3.15	3.55
75	3.2	3.34
100	2.65	3.84
125	3.04	3.25
150	2.97	3.63
175	3.09	3.64
200	2.75	3.55

Dari data di atas, dibuat grafik yang merepresentasikan hasil rata-rata *throughput* yang ditunjukkan pada Gambar 5.4.



Gambar 5.4 Grafik Rata-rata *Throughput* Skenario *Grid*

Berdasarkan grafik pada Gambar 5.4 dapat dilihat bahwa *routing protocol* AODV yang telah dimodifikasi mengalami kenaikan yang signifikan pada rata-rata *throughput*. Pada lingkungan yang jarang dengan jumlah 25 *node*, menghasilkan perbedaan selisih *throughput* sebesar 0.342 kbps atau mengalami kenaikan sebesar 16.64%, dimana *routing protocol* AODV yang telah dimodifikasi unggul dalam hal *throughput* tersebut karena menghasilkan *throughput* yang lebih tinggi dari *routing protocol* AODV asli. Pada lingkungan yang jarang dengan jumlah 50 *node*, menghasilkan perbedaan selisih *throughput* sebesar 0.396 kbps atau mengalami kenaikan sebesar 12.54%, dimana *routing protocol* AODV yang telah dimodifikasi unggul dalam hal *throughput* tersebut karena menghasilkan *throughput* yang lebih tinggi dari *routing protocol* AODV asli.

Pada lingkungan yang sedang dengan jumlah 75 *node*, menghasilkan perbedaan selisih *throughput* sebesar 0.139 kbps atau mengalami kenaikan sebesar 4.34%, dimana *routing protocol* AODV yang telah dimodifikasi unggul dalam hal *throughput* tersebut dari *throughput* AODV asli. Pada lingkungan yang sedang dengan jumlah 100 *node*, menghasilkan perbedaan selisih *throughput* sebesar 1.18 kbps atau mengalami kenaikan sebesar 44.37%, dimana *routing protocol* AODV yang telah dimodifikasi juga unggul dalam hal *throughput* tersebut dari *throughput* AODV asli. Pada lingkungan yang sedang dengan jumlah 125 *node*, menghasilkan perbedaan selisih *throughput* sebesar 0.211 kbps atau mengalami kenaikan sebesar 6.94%, dimana *routing protocol* AODV yang telah dimodifikasi juga unggul dalam hal *throughput* tersebut dari *throughput* AODV asli. Pada lingkungan yang sedang dengan jumlah 150 *node*, menghasilkan perbedaan selisih *throughput* sebesar 0.66 kbps atau mengalami kenaikan sebesar 22.22%, dimana *routing protocol* AODV yang telah dimodifikasi juga unggul dalam hal *throughput* tersebut dari *throughput* AODV asli.

Pada lingkungan yang padat dengan jumlah 175 *node*, menghasilkan perbedaan selisih *throughput* sebesar 0.56 atau

mengalami kenaikan sebesar 17.85%, dimana *routing protocol* AODV yang telah dimodifikasi juga unggul dalam hal *throughput* tersebut. Pada lingkungan yang padat dengan jumlah 200 *node*, menghasilkan perbedaan selisih *throughput* sebesar 0.80 atau mengalami kenaikan sebesar 28.71%, dimana *routing protocol* AODV yang telah dimodifikasi juga unggul dalam hal *throughput* tersebut.

Dapat dilihat rata-rata kenaikan yang terjadi untuk *throughput* adalah sebesar 19.34%. Jika ketiga lingkungan tersebut dirata-rata, dapat dilihat bahwa bahwa AODV yang telah dimodifikasi menghasilkan *throughput* yang lebih tinggi daripada AODV asli dengan jumlah selisih *throughput* yang cukup signifikan. Hal ini dikarenakan pada AODV yang telah dimodifikasi menyimpan informasi *node* tetangga sebanyak 1-*hop* hingga 2-*hop.*

5.2.2 Hasil Uji Coba Skenario *Real*

Pengujian pada skenario *real* digunakan untuk melihat perbandingan PDR, E2E, RO, dan *Througput* antara *routing protocol* AODV asli dan AODV yang telah dimodifikasi pada proses *route discovery* dan pengiriman paket.

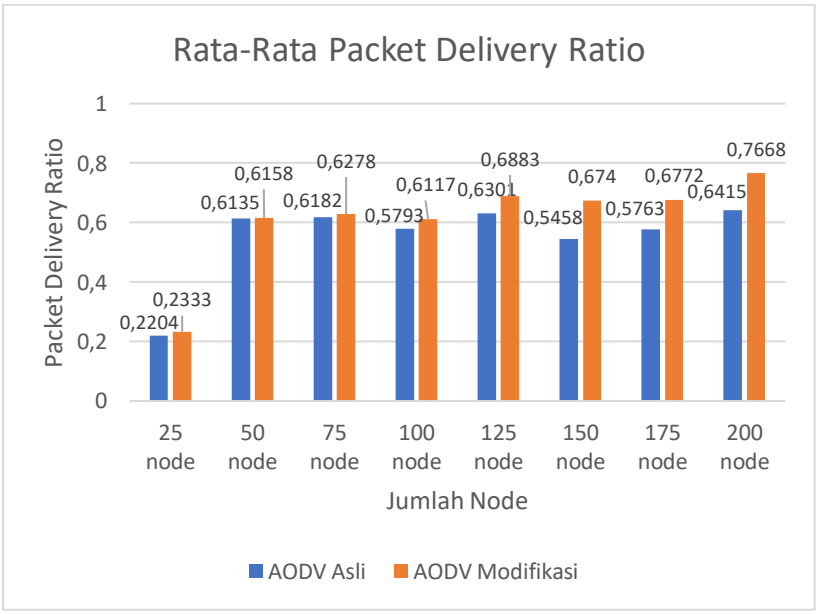
Pengambilan data uji PDR, E2E, RO, dan *Throughput* pada skenario *real* dilakukan sebanyak 10 kali dengan skenario mobilitas *random* pada peta *grid* dengan luas area 1500 m x 1500 m dan *node* sebanyak 25 dan 50 untuk lingkungan yang jarang, 75, 100, 125 dan 150 *node* untuk lingkungan yang sedang, 175 dan 200 *node* untuk lingkungan yang padat dilakukan pada kecepatan standar yaitu 20 m/s. Hasil analisis dapat dilihat pada Tabel 5.7, Tabel 5.8, Tabel 5.9, Tabel 5.10.

Tabel 5.7 Hasil Rata - Rata PDR pada Skenario *Real*

Jumlah *Node*	**AODV Asli**	**AODV Modifikasi**
25	0.2204	0.2333
50	0.6135	0.6158

75	0.6182	0.6278
100	0.5793	0.6117
125	0.6301	0.6883
150	0.5458	0.6740
175	0.5763	0.6772
200	0.6415	0.7668

Dari data di atas, dibuat grafik yang merepresentasikan hasil perhitungan PDR yang ditunjukkan pada Gambar 5.5.



Gambar 5.5 Grafik *Packet Delivery Ratio* Skenario *Real*

Berdasarkan grafik pada Gambar 5.5 dapat dilihat bahwa *routing protocol* AODV yang telah dimodifikasi mengalami kenaikan yang *signifikan* pada *packet delivery ratio*. Pada lingkungan yang jarang dengan jumlah 25 *node*, menghasilkan perbedaan selisih PDR sebesar 0.0129, atau naik menjadi sekitar 5.85% dimana *routing*

protocol AODV yang telah dimodifikasi unggul dalam hal PDR tersebut. Pada lingkungan yang jarang dengan jumlah 50 *node*, menghasilkan perbedaan selisih PDR sebesar 0.0023, atau naik menjadi sekitar 0.37% dimana *routing protocol* AODV yang telah dimodifikasi unggul dalam hal PDR tersebut.

Pada lingkungan yang sedang dengan jumlah 75 *node*, menghasilkan perbedaan selisih PDR sebesar 0.0096, atau naik menjadi sekitar 1.55% dimana *routing protocol* AODV yang telah dimodifikasi unggul dalam hal PDR tersebut dari AODV asli. Pada lingkungan yang sedang dengan jumlah 100 *node*, menghasilkan perbedaan selisih PDR sebesar 0.0324, atau naik menjadi sekitar 5,59% dimana *routing protocol* AODV yang telah dimodifikasi unggul dalam hal PDR tersebut dari AODV asli. Pada lingkungan yang sedang dengan jumlah 125 *node*, menghasilkan perbedaan selisih PDR sebesar 0.0582, atau naik menjadi sekitar 9.24% dimana *routing protocol* AODV yang telah dimodifikasi juga unggul dalam hal PDR tersebut dari AODV asli. Pada lingkungan yang sedang dengan jumlah 150 *node*, menghasilkan perbedaan selisih PDR sebesar 0.1283, atau naik menjadi sekitar 23.51% dimana *routing protocol* AODV yang telah dimodifikasi juga unggul dalam hal PDR tersebut dari AODV asli.

Pada lingkungan yang padat dengan jumlah 175 *node*, menghasilkan perbedaan selisih PDR sebesar 0.1009, atau naik menjadi sekitar 17.51% dimana *routing protocol* AODV yang telah dimodifikasi unggul dalam hal PDR tersebut. Pada lingkungan yang padat dengan jumlah 200 *node*, menghasilkan perbedaan selisih PDR sebesar 0.1253, atau naik menjadi sekitar 19.53% dimana *routing protocol* AODV yang telah dimodifikasi juga unggul dalam hal PDR tersebut.

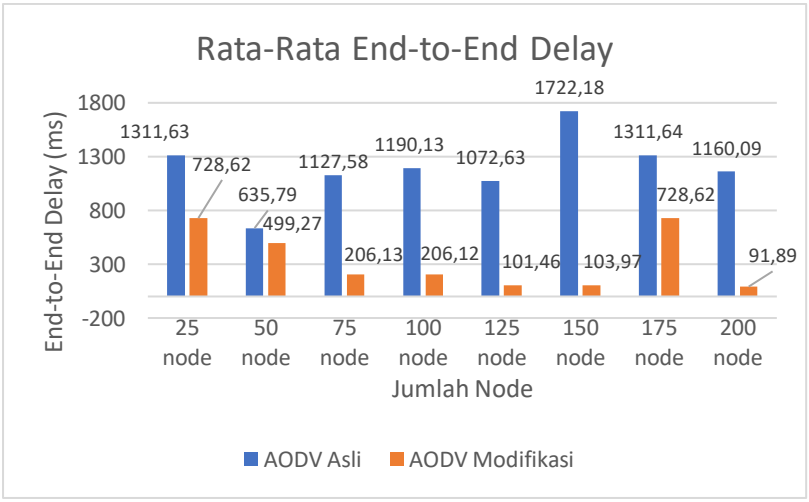
Dapat dilihat rata-rata kenaikan yang terjadi untuk PDR adalah 10.40%. Jika ketiga lingkungan tersebut dirata-rata, dapat dilihat bahwa AODV yang telah dimodifikasi menghasilkan PDR yang lebih bagus daripada AODV asli dengan jumlah selisih PDR yang cukup *signifikan*.

Hasil pengambilan data rata-rata untuk *end-to-end delay* (E2E) pada skenario *real* dengan jumlah *node* 25, 50, 75, 125, 100, 150, 175, dan 200 dapat dilihat pada Tabel 5.8.

Tabel 5.8 Hasil Rata -Rata E2E pada Skenario Real

Jumlah *Node*	**AODV Asli(ms)**	**AODV Modifikasi(ms)**
25	1311.63	728.62
50	635.79	499.27
75	1127.58	206.13
100	1190.13	206.12
125	1072.63	101.46
150	1722.17	103.97
175	1311.64	728.62
200	1160.09	91.89

Dari data di atas, dibuat grafik yang merepresentasikan hasil perhitungan E2E yang ditunjukkan pada Gambar 5.6.



Gambar 5.6 Grafik *End-to-end Delay* pada Skenario *Real*

Berdasarkan grafik pada Gambar 5.6 dapat dilihat bahwa rata-rata *end-to-end delay routing protocol* AODV yang telah dimodifikasi mengalami penurunan yang *signifikan*. Pada lingkungan yang jarang dengan jumlah 25 *node*, terjadi perbedaan selisih *end-to-end delay* sebesar 583.01 ms antara *routing protocol* AODV asli dengan *routing protocol* AODV yang telah dimodifikasi atau mengalami penurunan sebesar 44.45%, dimana *routing protocol* AODV yang telah dimodifikasi unggul dalam hal *end-to-end delay* tersebut. Pada lingkungan yang jarang dengan jumlah 50 *node*, terjadi perbedaan selisih *end-to-end delay* sebesar 136.52 ms antara *routing protocol* AODV asli dengan *routing protocol* AODV yang telah dimodifikasi atau mengalami penurunan sebesar 21.47%, dimana *routing protocol* AODV yang telah dimodifikasi unggul dalam hal *end-to-end delay* tersebut.

Pada lingkungan sedang dengan jumlah 75 *node*, terjadi perbedaan selisih *end-to-end delay* sebesar 921.45 ms antara *routing protocol* AODV asli dengan *routing protocol* AODV yang telah dimodifikasi atau mengalami penurunan sebesar 81.72%, dimana *routing protocol* AODV yang telah dimodifikasi juga lebih unggul dalam hal *end-to-end delay* tersebut. Pada lingkungan sedang dengan jumlah 100 *node*, terjadi perbedaan selisih *end-to-end delay* sebesar 984.01 ms antara *routing protocol* AODV asli dengan *routing protocol* AODV yang telah dimodifikasi atau mengalami penurunan sebesar 82.68%, dimana *routing protocol* AODV yang telah dimodifikasi juga lebih unggul dalam hal *end-to-end delay* tersebut. Pada lingkungan sedang dengan jumlah 125 *node*, terjadi perbedaan selisih *end-to-end delay* sebesar 971.17 ms antara *routing protocol* AODV asli dengan *routing protocol* AODV yang telah dimodifikasi atau mengalami penurunan sebesar 90.54%, dimana *routing protocol* AODV yang telah dimodifikasi juga lebih unggul dalam hal *end-to-end delay* tersebut. Pada lingkungan sedang dengan jumlah 150 *node*, terjadi perbedaan selisih *end-to-end delay* sebesar 1818.19 ms antara *routing protocol* AODV asli dengan *routing protocol* AODV yang telah dimodifikasi atau mengalami penurunan sebesar 94.59%,

dimana *routing protocol* AODV yang telah dimodifikasi juga lebih unggul dalam hal *end-to-end delay* tersebut.

Pada lingkungan yang padat dengan jumlah 175 *node*, terjadi perbedaan selisih *end-to-end delay* sebesar 583.02 ms antara *routing protocol* AODV asli dengan *routing protocol* AODV yang telah dimodifikasi atau mengalami penurunan sebesar 44.45%, dimana *routing protocol* AODV yang telah dimodifikasi lebih unggul dalam hal *end-to-end delay* tersebut. Pada lingkungan yang padat dengan jumlah 200 *node*, terjadi perbedaan selisih *end-to-end delay* sebesar 1068.19 ms antara *routing protocol* AODV asli dengan *routing protocol* AODV yang telah dimodifikasi atau mengalami penurunan sebesar 92.07%, dimana *routing protocol* AODV yang telah dimodifikasi jauh lebih unggul dalam hal *end-to-end delay* tersebut.

Dapat dilihat rata-rata penurunan yang terjadi untuk E2E adalah 68.92%. Jika ketiga lingkungan tersebut dirata-rata, *routing protocol* AODV yang telah dimodifikasi lebih unggul dalam hal *end-to-end delay*. Hal ini dikarenakan perbedaan tindakan antara AODV asli dan AODV yang telah dimodifikasi dalam menemukan rute yang rusak. Pada AODV asli akan mencari ulang rute sehingga akan menambah *delay*. Sedangkan pada AODV yang telah dimodifikasi akan menggunakan *alternative path* sehingga tidak melakukan pencarian ulang dan tidak menambah *delay*.

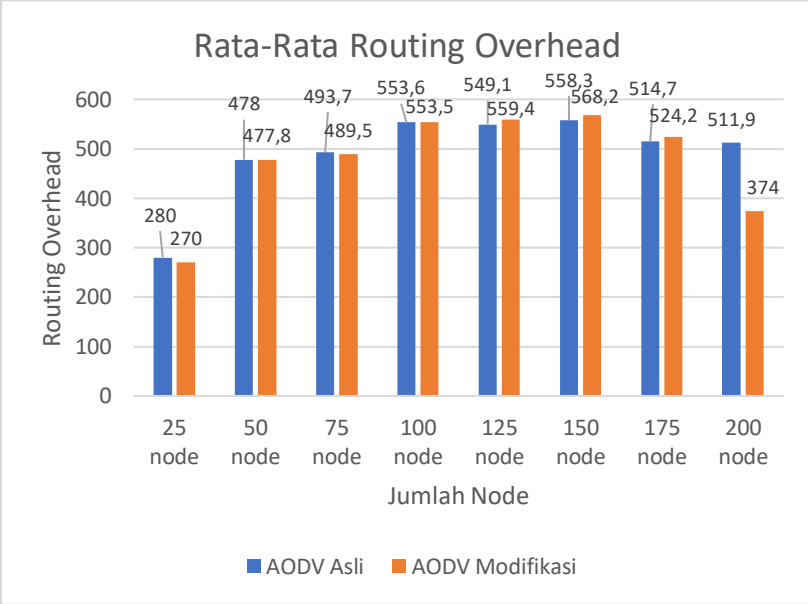
Untuk hasil pengambilan data *routing overhead* pada skenario *real* 25, 50, 75, 100, 125, 150, 175, dan 200 *node* dapat dilihat pada Tabel 5.9.

Tabel 5.9 Hasil Rata - Rata RO Skenario Real

Jumlah *Node*	**AODV Asli**	**AODV Modifikasi**
25	280	270
50	478	477.8
75	493.7	489.5
100	553.6	553.5
125	549.1	559.4
150	558.3	568.2

175	514.7	524.2
200	511.9	374

Dari data di atas, dibuat grafik yang merepresentasikan hasil rata-rata RO yang ditunjukkan pada Gambar 5.7.



Gambar 5.7 Grafik Routing Overhead Skenario Real

Berdasarkan grafik pada Gambar 5.7, dapat dilihat bahwa rata-rata *routing overhead* AODV yang telah dimodifikasi mengalami perubahan yang *fluktuatif*. Pada lingkungan yang jarang dengan jumlah 25 *node*, menghasilkan perbedaan selisih *routing overhead* sebesar 10 atau mengalami penurunan sebesar 3.57%, dimana *routing protocol* AODV yang telah dimodifikasi unggul dalam hal *routing overhead* tersebut karena menghasilkan *routing overhead* yang lebih rendah dari *routing protocol* AODV asli. Pada lingkungan yang jarang dengan jumlah 50 *node*, menghasilkan perbedaan selisih

routing overhead sebesar 0.2 atau mengalami penurunan sebesar 0.04%, dimana *routing protocol* AODV yang telah dimodifikasi unggul dalam hal *routing overhead* tersebut.

Pada lingkungan yang sedang dengan jumlah 75 *node*, menghasilkan perbedaan selisih *routing overhead* sebesar 4.2 atau mengalami penurunan sebesar 0.85%, dimana *routing protocol* AODV yang telah dimodifikasi lebih unggul juga dalam hal *routing overhead* tersebut dari AODV asli. Pada lingkungan yang sedang dengan jumlah 100 *node*, menghasilkan perbedaan selisih *routing overhead* sebesar 0.1 atau mengalami penurunan sebesar 0.02%, dimana *routing protocol* AODV yang telah dimodifikasi lebih unggul juga dalam hal *routing overhead* tersebut dari AODV asli. Pada lingkungan yang sedang dengan jumlah 125 *node*, menghasilkan perbedaan selisih *routing overhead* sebesar 10.3 atau mengalami kenaikan sebesar 1.87%, dimana *routing protocol* AODV asli unggul dalam hal *routing overhead* tersebut dari AODV yang telah dimodifikasi. Pada lingkungan yang sedang dengan jumlah 150 *node*, menghasilkan perbedaan selisih *routing overhead* sebesar 9.9 atau mengalami kenaikan sebesar 1.77%, dimana *routing protocol* AODV asli unggul dalam hal *routing overhead* tersebut dari AODV yang telah dimodifikasi.

Pada lingkungan yang padat dengan jumlah 175 *node*, menghasilkan perbedaan selisih *routing overhead* sebesar 9.5 atau mengalami kenaikan sebesar 1.84%, dimana *routing protocol* AODV yang asli lebih unggul dalam hal *routing overhead* tersebut daripada AODV yang telah dimodifikasi. Pada lingkungan yang padat dengan jumlah 200 *node*, menghasilkan perbedaan selisih *routing overhead* sebesar 137.9 atau mengalami penurunan sebesar 26.93%, dimana *routing protocol* AODV yang telah dimodifikasi lebih unggul dalam hal *routing overhead* tersebut daripada AODV asli.

Dapat dilihat rata-rata penurunan yang terjadi untuk *routing overhead* adalah sebesar 3.24%. Jika ketiga lingkungan tersebut dirata-rata, dapat dilihat bahwa AODV yang telah dimodifikasi menghasilkan *routing overhead* yang sedikit lebih rendah dibandingkan dengan AODV asli dikarenakan pada AODV yang telah

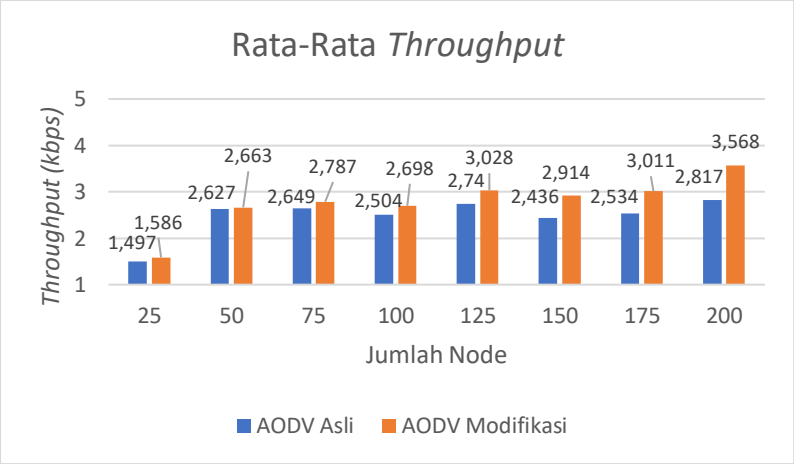
dimodifikasi memanfaatkan *alternative path* untuk mencari rute yang rusak sehingga tidak banyak melakukan *broadcast control message* ulang.

Untuk hasil pengambilan data *Throughput* pada scenario *real* 25, 50, 75, 100, 125, 150, 175, dan 200 *node* dapat dilihat pada Tabel 5.10.

Tabel 5.10 Hasil Rata - Rata *Throughput* pada Skenario Real

Jumlah *Node*	**AODV Asli(kbps)**	**AODV Modifikasi(kbps)**
25	1.497	1.586
50	2.627	2.663
75	2.649	2.787
100	2.504	2.698
125	2.74	3.028
150	2.436	2.914
175	2.534	3.011
200	2.817	3.568

Dari data di atas, dibuat grafik yang merepresentasikan hasil perhitungan *Throughput* yang ditunjukkan pada Gambar 5.8



Gambar 5.8 Grafik *Rata-rata Throughput* Skenario *Real*

Berdasarkan grafik pada Gambar 5.8 dapat dilihat bahwa *routing protocol* AODV yang telah dimodifikasi dan juga *routing protocol* asli mengalami kenaikan *throughput* yang *signifikan*. Pada lingkungan yang jarang dengan jumlah 25 *node*, menghasilkan perbedaan selisih *throughput* sebesar 0.089 kbps atau mengalami kenaikan sebesar 5.95%, dimana *routing protocol* AODV yang telah dimodifikasi unggul dalam hal *throughput* tersebut karena menghasilkan *throughput* yang lebih tinggi dari *routing protocol* AODV asli. Pada lingkungan yang jarang dengan jumlah 50 *node*, menghasilkan perbedaan selisih *throughput* sebesar 0.036 kbps atau mengalami kenaikan sebesar 1.37%, dimana *routing protocol* AODV yang telah dimodifikasi unggul dari *routing protocol* AODV asli dalam hal *throughput* tersebut.

Pada lingkungan yang sedang dengan jumlah 75 *node*, menghasilkan perbedaan selisih *throughput* sebesar 0.038 kbps atau mengalami kenaikan sebesar 1.43%, dimana *routing protocol* AODV yang telah dimodifikasi unggul dalam hal *throughput* tersebut dari *throughput* AODV asli. Pada lingkungan yang sedang dengan jumlah 100 *node*, menghasilkan perbedaan selisih *throughput* sebesar 0.194 kbps atau mengalami kenaikan sebesar 7.75%, dimana *routing protocol* AODV yang telah dimodifikasi juga unggul dalam hal *throughput* tersebut dari *throughput* AODV asli. Pada lingkungan yang sedang dengan jumlah 125 *node*, menghasilkan perbedaan selisih *throughput* sebesar 0.288 kbps atau mengalami kenaikan sebesar 10.51%, dimana *routing protocol* AODV yang telah dimodifikasi juga lebih unggul dalam hal *throughput* tersebut dari *throughput* AODV asli. Pada lingkungan yang sedang dengan jumlah 150 *node*, menghasilkan perbedaan selisih *throughput* sebesar 0.478 kbps atau mengalami kenaikan sebesar 19.62%, dimana *routing protocol* AODV yang telah dimodifikasi juga lebih unggul dalam hal *throughput* tersebut dari *throughput* AODV asli.

Pada lingkungan yang padat dengan jumlah 175 *node*, menghasilkan perbedaan selisih *throughput* sebesar 0.477 kbps atau mengalami kenaikan sebesar 18.82%, dimana *routing protocol*

AODV yang telah dimodifikasi juga unggul dalam hal *throughput* tersebut. Pada lingkungan yang padat dengan jumlah 200 *node*, menghasilkan perbedaan selisih *throughput* sebesar 0.751 kbps atau mengalami kenaikan sebesar 26.66%, dimana *routing protocol* AODV yang telah dimodifikasi juga unggul dalam hal *throughput* tersebut.

Dapat dilihat rata-rata kenaikan yang terjadi untuk *throughput* adalah sebesar 11.51%. Jika ketiga lingkungan tersebut dibandingkan dapat dilihat bahwa AODV yang telah dimodifikasi menghasilkan *throughput* yang lebih tinggi daripada AODV asli dengan jumlah selisih *throughput* yang cukup *signifikan*. Hal ini dikarenakan pada AODV yang telah dimodifikasi menyimpan informasi *node* tetangga lebih 1-*hop* atau 2-*hop* dibandingkan dengan AODV asli.

(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. Peningkatan kinerja protokol dilakukan dengan cara menggabungkan *hello message* dan informasi *node* tetangga yang berdekatan.
2. Implementasi *alternative path* pada AODV dilakukan dengan memberikan *flag_RREQ* untuk pembatasan *broadcast* RREQ.
3. Dampak modifikasi terhadap performa protokol AODV pada skenario grid adalah rata – rata kenaikan *Packet Delivery Ratio* (PDR) sebesar 17.70%, rata-rata penurunan *End-to-end Delay* sebesar 86.95%, rata – rata penurunan *Routing Overhead* (RO) sebesar 0.14%, dan rata-rata kenaikan *Throughput* sebesar 19.34%.
4. Dampak modifikasi terhadap performa protokol AODV pada skenario real adalah rata – rata kenaikan *Packet Delivery Ratio* (PDR) sebesar 10.39%, rata-rata penurunan *End-to-end Delay* sebesar 68.92%, rata – rata penurunan *Routing Overhead* (RO) sebesar 3.24%, dan rata-rata kenaikan *Throughput* sebesar 11.99%.

6.2 Saran

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

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

2. Menambahkan aspek lain untuk memepercepat proses *route discovery* seperti arah, kecepatan, dan energi.

DAFTAR PUSTAKA

[1] Rendra, "analisis perbandingan unjuk kerja protokol routing reaktif (dymo) terhadap protokol routing reaktif (aodv) di jaringan vanet," 2016.

[2] R. N. Aziza, P. C. Siswipraptini, and R. Cahyaningtyas, "Protokol Routing pada VANET: Taksonomi dan Analisis Perbandingan antara DSR, AODV, dan TORA," *J. Ilm. Fifo*, vol. 9, no. 2, pp. 98–109, 2017.

[3] S. Mittal, S. Bisht, K. C. Purohit, and A. Joshi, "Improvising-AODV routing protocol by modifying route discovery mechanism in VANET," in *Advances in Computing, Communication & Automation (ICACCA)(Fall), 2017 3rd International Conference on*, 2017, pp. 1–5.

[4] S. Singh and G. S. Aujla, "A Noble Routing Protocol for Vehicular ad hoc Networks (VANETs) with Less Routing Overheads," *Int. J. Futur. Gener. Commun. Netw.*, vol. 7, no. 5, pp. 23–34, 2014.

[5] R. Brendha and V. S. J. Prakash, "A survey on routing protocols for vehicular Ad Hoc networks," in *2017 4th International Conference on Advanced Computing and Communication Systems (ICACCS)*, 2017, pp. 1–7.

[6] "ilustrasi vanets - Penelusuran Google." [Online]. Available: https://www.google.co.id/search?q=ilustrasi+vanets&safe=strict&source=lnms&tbm=isch&sa=X&ved=0ahUKEwjF25ngxY3hAhXEMY8KHd2YDUQQ_AUIDigB&biw=1366&bih=695#imgrc=ZpvI4eZ_jx_GGM: [Accessed: 19-Mar-2019].

[7] C. Perkins, E. Belding-Royer, and S. Das, "Ad hoc on-demand distance vector (AODV) routing," 2003.

[8] J. Harri, F. Filali, and C. Bonnet, "Mobility models for vehicular ad hoc networks: a survey and taxonomy," *IEEE Commun. Surv. Tutorials*, vol. 11, no. 4, pp. 19–41, 2009.

[9] M. Iqbal, M. Shafiq, J.-G. Choi, H. Attaullah, K. Akram, and X. Wang, "Design and analysis of a novel hybrid wireless mesh network routing protocol," *Int. J. Adapt. Resilient*

Auton. Syst., vol. 5, no. 3, pp. 20–39, 2014.

[10] "The Network Simulator - ns-2." [Online]. Available: https://www.isi.edu/nsnam/ns/. [Accessed: 20-Mar-2019].

[11] A. M. S. Laurent, *Understanding open source and free software licensing: guide to navigating licensing issues in existing & new software*. " O'Reilly Media, Inc.," 2004.

[12] T. Issariyakul and E. Hossain, "Introduction to Network Simulator 2 (NS2)," in *Introduction to Network Simulator NS2*, Springer, 2009, pp. 1–18.

[13] "Tentang OpenStreetMap (OSM) | OpenStreetMap Indonesia." [Online]. Available: https://openstreetmap.id/about/tentang-openstreetmap/. [Accessed: 19-Mar-2019].

[14] P. Jacquet, P. Muhlethaler, T. Clausen, A. Laouiti, A. Qayyum, and L. Viennot, "Optimized link state routing protocol for ad hoc networks," in *Multi Topic Conference, 2001. IEEE INMIC 2001*, 2001, pp. 62–68.

[15] "Editor usage stats - OpenStreetMap Wiki." [Online]. Available: https://wiki.openstreetmap.org/wiki/Editor_usage_stats. [Accessed: 21-Mar-2019].

[16] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz, "SUMO--simulation of urban mobility: an overview," in *Proceedings of SIMUL 2011, The Third International Conference on Advances in System Simulation*, 2011.

[17] A. V Aho, B. W. Kernighan, and P. J. Weinberger, "Awk—a pattern scanning and processing language," *Softw. Pract. Exp.*, vol. 9, no. 4, pp. 267–279, 1979.

LAMPIRAN

A.1 Kode Fungsi recvRequest()

```
void AODV::recvRequest(Packet *p) {

  #ifdef DEBUG
    FILE *fp;
    fp = fopen("debug.txt", "a");
    fprintf(fp, "\n%f  %s function",
CURRENT_TIME, __FUNCTION__);
    fclose(fp);
  #endif

  struct hdr_ip *ih = HDR_IP(p);
  struct hdr_aodv_request *rq =
HDR_AODV_REQUEST(p);
  aodv_rt_entry *rt;
  /*
   * Drop if:
   *      - I'm the source
   *      - I recently heard this request.
   */

 if(rq->rq_dst != index) {
  if(rq->rq_src == index) {
    #ifdef DEBUG
      FILE *fp;
      fp = fopen("debug.txt", "a");
      fprintf(fp, "\n%f  %s: got my own
REQUEST",  CURRENT_TIME, __FUNCTION__);
      fclose(fp);
    #endif // DEBUG
    Packet::free(p);
    return;
  }
```

```
  //I-AODV
  //jika saya pernah denger RREQ sejenis ini
  if (id_lookup(rq->rq_src, rq->rq_bcast_id))
{
    #ifdef DEBUG
        fprintf(stderr, "%s: I recently heard
this request %2.f\n", __FUNCTION__,
CURRENT_TIME);

        fp = fopen("debug.log","a+");
        fprintf(fp, "\n(%2.f) (old) I recently
heard this request and my flag is %d \n",
CURRENT_TIME, flag_RREQ[index]);
        fclose(fp);
      #endif // DEBUG

      for (unsigned int i = 0; i <
broadcast_node[index].size(); i++)
      {
        if(broadcast_node[index][i]==ih-
>saddr()){
          #ifdef DEBUG
            fp = fopen("debug.log","a+");
            fprintf(fp, "\n(%2.f) (nonunique)
drop packet coz node(%d) in broadcast_node
node(%d)\n", CURRENT_TIME, ih->saddr(),
index);
            fclose(fp);
          #endif // DEBUG

          Packet::free(p);
          return;
        }
      }

      if(flag_RREQ[index]<2){
        flag_RREQ[index]++;
      }
```

```
else{
        #ifdef DEBUG
          fp = fopen("debug.log","a+");
          fprintf(fp, "\n(%2.f) (flag) im -
%d- dropping RREQ -%d- coz my flag is %d\n",
CURRENT_TIME, index, ih->saddr(),
flag_RREQ[index]);
          fclose(fp);
        #endif // DEBUG
        Packet::free(p);
        return;
      }
  }
  else{
      #ifdef DEBUG
        fp = fopen("debug.log","a+");
        fprintf(fp, "\n(%2.f) (new) im -%d-
never heard this kind of RREQ from -%d- and
my flag is %d\n", CURRENT_TIME, index, ih-
>saddr(), flag_RREQ[index]);
        fclose(fp);
      #endif // DEBUG
      //reinitiate flag karena rreq baru
      flag_RREQ[index]=1;
      //hapus cache broadcast node
      broadcast_node[index].clear();
      //cache the sender
      broadcast_node[index].push_back(ih-
>saddr());
      for (unsigned int i = 0; i <
list_neighbor[ih->saddr()].size(); i++){
        //cache list_neighbor sender to
broadcast_node

broadcast_node[index].push_back(list_neighbor
[ih->saddr()][i]);
      }
    }
  }
```

```
#ifdef DEBUG
    fp = fopen("debug.log","a+");
    fprintf(fp, "\n(%2.f) (next) im -%d- got
RREQ from -%d- at  with cache
broadcast_node:\n", CURRENT_TIME, index, ih-
>saddr());
    for (unsigned int i = 0; i <
broadcast_node[index].size(); i++)
    {
      fprintf(fp, "::::-%d-\n",
broadcast_node[index][i]);
    }
    fclose(fp);
  #endif // DEBUG

  /*
  * Cache the broadcast ID
  */
  id_insert(rq->rq_src, rq->rq_bcast_id);

  /*
  * We are either going to forward the
REQUEST or generate a
  * REPLY. Before we do anything, we make
sure that the REVERSE
  * route is in the route table.
  */
  aodv_rt_entry *rt0; // rt0 is the reverse
route

  rt0 = rtable.rt_lookup(rq->rq_src);
  if(rt0 == 0) { /* if not in the route table
*/
    // create an entry for the reverse route.
    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)) ) {
```

```
      // If we have a fresher seq no. or
lesser #hops for the
    // same seq no., update the rt entry.
Else don't bother.
    rt_update(rt0, rq->rq_src_seqno, rq-
>rq_hop_count, ih->saddr(), max(rt0-
>rt_expire, (CURRENT_TIME + REV_ROUTE_LIFE))
);
    if (rt0->rt_req_timeout > 0.0) {
      // Reset the soft state and
      // Set expiry time to CURRENT_TIME +
ACTIVE_ROUTE_TIMEOUT
      // This is because route is used in the
forward direction,
      // but only sources get benefited by
this change
      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;
    }
    /* Find out whether any buffered packet
can benefit from the
    * reverse route.
    * May need some change in the following
code - Mahesh 09/11/99
    */
    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);
      }
    }
  }
  // End for putting reverse route in rt
table
rt = rtable.rt_lookup(rq->rq_dst);
```

```
  // First check if I am the destination ..

  if(rq->rq_dst == index) {

  #ifdef DEBUG
    FILE *fp;
    fp = fopen("debug.txt", "a");
    fprintf(fp, "\n%f  %s function",
CURRENT_TIME, __FUNCTION__);
    fprintf(fp, "\n%f  node %d - %s:
destination sending reply",  CURRENT_TIME,
index, __FUNCTION__);
    fclose(fp);
  #endif // DEBUG

  // Just to be safe, I use the max. Somebody
may have
  // incremented the dst seqno.
  seqno = max(seqno, rq->rq_dst_seqno)+1;
  if (seqno%2) seqno++;
    sendReply(rq->rq_src,           // IP
Destination
              1,                    // Hop
Count
              index,                // Dest
IP Address
              seqno,                // Dest
Sequence Num
              MY_ROUTE_TIMEOUT,     //
Lifetime
              rq->rq_timestamp);    //
timestamp
    Packet::free(p);
  }

 // I am not the destination, but I may have
a fresh enough route.
  else if (rt && (rt->rt_hops != INFINITY2)
&& (rt->rt_seqno >= rq->rq_dst_seqno) ) {
//assert (rt->rt_flags == RTF_UP);
    assert(rq->rq_dst == rt->rt_dst);
```

```
//assert ((rt->rt_seqno%2) == 0);     // is
the seqno even?
    sendReply(rq->rq_src,
              rt->rt_hops + 1,
              rq->rq_dst,
              rt->rt_seqno,
              (u_int32_t) (rt->rt_expire -
CURRENT_TIME),
                    //rt->rt_expire -
CURRENT_TIME,
              rq->rq_timestamp);
    // Insert nexthops to RREQ source and
RREQ destination in the
    // precursor lists of destination and
source respectively
    rt->pc_insert(rt0->rt_nexthop); //
nexthop to RREQ source
    rt0->pc_insert(rt->rt_nexthop); //
nexthop to RREQ destination

    #ifdef RREQ_GRAT_RREP

    sendReply(rq->rq_dst,
              rq->rq_hop_count,
              rq->rq_src,
              rq->rq_src_seqno,
              (u_int32_t) (rt->rt_expire -
CURRENT_TIME),
              //rt->rt_expire - CURRENT_TIME,
              rq->rq_timestamp);
    #endif

    // TODO: send grat RREP to dst if G flag
set in RREQ using rq->rq_src_seqno, rq-
>rq_hop_counT
    // DONE: Included gratuitous replies to
be sent as per IETF aodv draft specification.
As of now, G flag has not been dynamically
used and is always set or reset in aodv-
packet.h --- Anant Utgikar, 09/16/02.
```

```
Packet::free(p);
  }
  /*
  * Can't reply. So forward the  Route
Request
  */
  else {
    ih->saddr() = index;
    ih->daddr() = IP_BROADCAST;
    rq->rq_hop_count += 1;
    // Maximum sequence number seen en route
    if (rt) rq->rq_dst_seqno = max(rt-
>rt_seqno, rq->rq_dst_seqno);
    forward((aodv_rt_entry*) 0, p, DELAY);
  }
}
```

A.2 Kode Fungsi nb_insert()

```
void AODV::nb_insert(nsaddr_t id) {
  AODV_Neighbor *nb = new AODV_Neighbor(id);


  list_neighbor[index].push_back(id);
//tambah neighbor

  #ifdef DEBUG
    FILE *nbmylist;
    nbmylist = fopen("nbmylist.txt", "a");
    fprintf(nbmylist, "\n ------------------
-----\n");
    for (unsigned int i = 0; i < 3; i++)
    {
      fprintf(nbmylist,"node %d --",i);
      for (unsigned int j = 0; j <
list_neighbor[i].size(); j++)
      {
        fprintf(nbmylist, " %d",
list_neighbor[i][j]);
      }
      fprintf(nbmylist,"\n");
    }
    fprintf(nbmylist,"at %2.f\n",
CURRENT_TIME);
    fclose(nbmylist);
  #endif

 assert(nb);
 nb->nb_expire = CURRENT_TIME +
                 (1.5 * ALLOWED_HELLO_LOSS *
HELLO_INTERVAL);
 LIST_INSERT_HEAD(&nbhead, nb, nb_link);
 seqno += 2;             // set of neighbors
changed
 assert ((seqno%2) == 0);
}
```

A.3 Kode Fungsi nb_delete()

```
void AODV::nb_delete(nsaddr_t id) {

  #ifdef DEBUG
    FILE *fp = fopen("trace.log","a+");
    fprintf(fp, "AODV::nb_delete\n");
    fclose(fp);
  #endif

AODV_Neighbor *nb = nbhead.lh_first;

  for (unsigned int i = 0; i <
list_neighbor[index].size(); i++) //hapus
neighbor
  {
    if(list_neighbor[index][i]==id)
    {
      list_neighbor[index].erase
(list_neighbor[index].begin()+i);
      break;
    }
  }

#ifdef DEBUG
  FILE *nbmylist;
  nbmylist = fopen("nbmylist.txt", "a");
  fprintf(nbmylist, "\n ----------------------
---\n");
  for (unsigned int i = 0; i < 3; i++)
  {
    fprintf(nbmylist,"node %d --",i);
    for (unsigned int j = 0; j <
list_neighbor[i].size(); j++)
    {
      fprintf(nbmylist, " %d",
list_neighbor[i][j]);
    }
    fprintf(nbmylist,"\n");
```

```
}
  fprintf(nbmylist,"at %2.f\n",
CURRENT_TIME);
  fclose(nbmylist);
#endif

 log_link_del(id);
 seqno += 2;     // Set of neighbors changed
 assert ((seqno%2) == 0);

 for(; nb; nb = nb->nb_link.le_next) {
   if(nb->nb_addr == id) {
     LIST_REMOVE(nb,nb_link);
     delete nb;
     break;
   }
 }

 handle_link_failure(id);
}
```

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(ifq)    Queue/DropTail/PriQueue;
set val(ll)     LL;
set val(ant)    Antenna/OmniAntenna;
set opt(x)      1500;
set opt(y)      1500;
set val(ifqlen) 1000;
set val(nn)     50;
set val(seed)   1.0;
set val(adhocRouting)   AODV;
set val(stop)   200;
set val(cp)     "cbr.tcl";
set val(sc)     "scenario.tcl";

set ns_         [new Simulator]

# setup topography object

set topo   [new Topography]

# create trace object for ns and nam

set tracefd     [open scenario1.tr w]
set namtrace    [open scenario1.nam w]

$ns_ trace-all $tracefd
$ns_ namtrace-all-wireless $namtrace
$opt(x) $opt(y)
```

```
# Create God
set god_ [create-god $val(nn)]

#global node setting
$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)
\
                  -topoInstance $topo \
                  -agentTrace ON \
                  -routerTrace ON \
                  -macTrace ON \
                  -movementTrace ON \

# 802.11p default parameters
Phy/WirelessPhy set  RXThresh_ 5.57189e-
11 ; #400m
Phy/WirelessPhy set  CSThresh_ 5.57189e-
11 ; #400m

#  Create the specified number of nodes
[$val(nn)] and "attach" them
#  to the channel.
for {set i 0} {$i < $val(nn)} {incr i} {
           set node_($i) [$ns_ node]
           $node_($i) random-motion 0 ;#
disable random motion
}
```

```
# Define node movement model
puts "Loading connection pattern..."
source $val(cp)

# Define traffic model
puts "Loading scenario file..."
source $val(sc)

# Define node initial position in nam

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

    # 20 defines the node size in nam,
must adjust it according to your scenario
    # The function must be called after
mobility model is defined

    $ns_ initial_node_pos $node_($i) 20
}

# Tell nodes when the simulation ends
for {set i 0} {$i < $val(nn) } {incr i} {
    $ns_ at $val(stop).0 "$node_($i)
reset";
}

#$ns_ at  $val(stop)  "stop"
$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_(58) $udp_(0)
set null_(0) [new Agent/Null]
$ns_ attach-agent $node_(59) $null_(0)
set cbr_(0) [new Application/Traffic/CBR]
$cbr_(0) set packetSize_ 512
$cbr_(0) set interval_ 1
$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"
```

A.6 Kode Skrip AWK *Packet Delivery Ratio*

```
BEGIN {
        sendLine = 0;
        recvLine = 0;
        fowardLine = 0;
}

$0 ~/^s.* AGT/ {
        sendLine ++ ;
}

$0 ~/^r.* AGT/ {
        recvLine ++ ;
}

$0 ~/^f.* RTR/ {
        fowardLine ++ ;
}

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

A.7 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.8 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.9 Kode Skrip AWK *Throughput*

```
BEGIN {
      recvdSize = 0
      startTime = 1
      stopTime = 0
      sent=0
      receive=0
}

{
      event = $1
      time = $2
      node_id = $3
      pkt_size = $8
      level = $4

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

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

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

BIODATA PENULIS

FATIMATUS ZULFA, lahir di Madiun, 11 April 1997. Penulis adalah anak bungsu dari empat bersaudara. Penulis telah menempuh Pendidikan formal mulai dari SDN Pilangkenceng 1, SMPN 1 Pilangkenceng, SMAN 1 Mejayan dan terakhir sebagai mahasiswa Departemen Informatika Institut Teknologi Sepuluh Nopember dengan rumpun mata kuliah Arsistektur Jaringan Komputer (2015-2019).

Selama perkuliahan, penulis aktif dalam organisasi kemahasiswaan, antara lain sebagai Staff *Event Organizer* FTIf Festival 2016, Staff Departemen Kesejeahteraan Mahasiswa pada Himpunan Mahasiswa Teknik Computer-Informatika (HMTC) ITS 2016-2017, Staff Departemen OSR BEM FTIf ITS 2016-2017, Staff Webkes Schematics 2016 dan 2017, *Steering Commite* FTIf Festival 2017, Ketua Kemuslimahan An-Nisa pada KMI Departemen Informatika 2017-2018 dan Staff Ahli OSR BEM FTIf ITS 2017-2018. Selain itu, penulis juga berperan sebagai Administrator Laboratorium Net Centric Computing (NCC) Departemen Informatika ITS. Selama kuliah di Departemen Informatika ITS, penulis mengambil bidang minat Arsitektur Jaringan Komputer (AJK). Penulis dapat dihubungi melalui surel fatimatuszulfa@gmail.com.