TUGAS AKHIR – KI141502

PEMBATASAN *FORWARDING NODE* YANG ADAPTIF PADA *ROUTE DISCOVERY PROCESS AD-HOC ON DEMAND DISTANCE VECTOR* (AODV) BERDASARKAN LEVEL KONEKTIVITAS NODE TETANGGA DI VANETS

RIZKY FENALDO MAULANA
NRP 05111440000040

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

Dosen Pembimbing II
Ir. Muchammad Husni, M.Kom.

Departemen Informatika
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya 2018

(Halaman ini sengaja dikosongkan)

TUGAS AKHIR – KI141502

PEMBATASAN *FORWARDING NODE* YANG ADAPTIF PADA *ROUTE DISCOVERY PROCESS AD-HOC ON DEMAND DISTANCE VECTOR* (AODV) BERDASARKAN LEVEL KONEKTIVITAS NODE TETANGGA DI VANETS

RIZKY FENALDO MAULANA
NRP 05111440000040

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

Dosen Pembimbing II
Ir. Muchammad Husni, M.Kom.

Departemen Informatika
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya 2018

(Halaman ini sengaja dikosongkan)

UNDERGRADUATE THESES – KI141502

LIMITATIONS OF ADAPTIVE FORWARDING NODE FOR ROUTE DISCOVERY PROCESS IN AD-HOC ON DEMAND DISTANCE VECTOR (AODV) BASED ON NEIGHBOUR NODE CONNECTIVITY LEVEL IN VANETS

RIZKY FENALDO MAULANA
NRP 05111440000040

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

Second Advisor
Ir. Muchammad Husni, M.Kom.

Department of Informatics
Faculty of Information Technology and Communication
Sepuluh Nopember Institute of Technology
Surabaya 2018

(Halaman ini sengaja dikosongkan)

LEMBAR PENGESAHAN

PEMBATASAN *FORWARDING NODE* YANG ADAPTIF PADA *ROUTE DISCOVERY PROCESS AD-HOC ON DEMAND DISTANCE VECTOR* (AODV) BERDASARKAN LEVEL KONEKTIVITAS NODE TETANGGA DI VANETS

TUGAS AKHIR

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

Oleh:
RIZKY FENALDO MAUALANA
NRP: 05111440000040

Disetujui oleh Pembimbing Tugas Akhir:

1. Dr.Eng. Radityo Anggoro, S.Kom., M.Sc. .....
   (NIP. 198410162008121002) (Pembimbing 1)

2. Ir. Muchammad Husni, M.Kom. .....
   (NIP. 196002211984031001) (Pembimbing 2)

SURABAYA
JUNI, 2018

(Halaman ini sengaja dikosongkan)

PEMBATASAN *FORWARDING NODE* YANG ADAPTIF PADA *ROUTE DISCOVERY PROCESS AD-HOC ON DEMAND DISTANCE VECTOR* (AODV) BERDASARKAN LEVEL KONEKTIVITAS NODE TETANGGA DI VANETS

Nama Mahasiswa	**: Rizky Fenaldo Maulana**
NRP	**: 05111440000040**
Departemen	**: Informatika FTIK-ITS**
Dosen Pembimbing 1	**: Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.**
Dosen Pembimbing 2	**: Ir. Muchammad Husni, M.Kom.**

Abstrak

Vehicular Ad hoc Networks (VANETs) merupakan pengembangan dari *Mobile Ad hoc Network* (MANET) dimana *node* memiliki karakteristik dengan mobilitas yang sangat tinggi dan terbatas pada pola pergerakannya. Ada banyak *routing protocol* yang dapat diimplementasikan pada VANETs, salah satunya adalah *Ad hoc On demand Distance Vector* (AODV).

AODV merupakan salah satu *routing protocol* yang termasuk dalam klasifikasi *reactive routing protocol*, sebuah protokol yang hanya akan membuat rute ketika *node* sumber membutuhkannya. AODV memiliki dua fase, yaitu *route discovery* dan *route maintenance. Route discovery* digunakan untuk meminta dan meneruskan informasi rute yang terdiri dari proses pengiriman *Route Request* (RREQ) dan *Route Reply* (RREP), sedangkan *route maintenance* digunakan untuk mengetahui informasi adanya kesalahan pada rute. Pada fase ini terdapat proses pengiriman *Route Error* (RERR).

Pada Tugas Akhir ini akan dilakukan pembatasan pada proses *forwarding node* yang adaptif pada *route discovery* berdasarkan level konektivitas *node* tetangga, yaitu dengan cara mengeliminasi jumlah *forwarding node* yang bertugas untuk mengirim ulang *(re-broadcast)* RREQ dengan batas *Threshold* yang dihitung tiap

interval waktu, interval waktu yang digunakan adalah 5 detik, 10 detik dan 15 detik. Hal ini dilakukan agar dapat meningkatkan kinerja protokol AODV untuk mencari rute yang stabil dengan cara memodifikasi beberapa bagian dari mekanisme pengiriman paket RREQ. Dari hasil uji coba, AODV yang dimodifikasi pada skenario grid berhasil meningkatkan nilai *Packet Delivery Ratio* (PDR) hingga 2.61% dan penurunan nilai *Routing Overhead* (RO) hingga 3,41% sedangkan pada skenario real berhasil meningkatkan nilai *Packet Delivery Ratio* (PDR) hingga 3.92% dan penurunan nilai *Routing Overhead* (RO) hingga 0,86%.

Kata kunci: VANETs, AODV, NS2, SUMO, Forwarding Node yang adatif, Node Tetangga

LIMITATIONS OF ADAPTIVE FORWARDING NODE FOR ROUTE DISCOVERY IN AD-HOC ON DEMAND DISTANCE VECTOR (AODV) BASED ON NEIGHBOUR NODE CONNECTIVITY LEVEL IN VANETS

Student's Name	**: Rizky Fenaldo Maulana**
Student's ID	**: 05111440000040**
Department	**: Informatics – FTIK ITS**
First Advisor	**: Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.**
Second Advisor	**: Ir. Muchammad Husni, M.Kom.**

Abstract

VANETs are an improvement of MANET which have high mobility node characteristic and limited movement pattern. There are many routing protocols that can be implemented on VANETs and one of them is AODV.

AODV is an example of reactive routing protocol classification, a protocol that will only create a route when the source node needs it. AODV have 2 phase which are route discovery and route maintenance. Route discovery is used for requesting and forwarding a route information that consist of Route Request (RREQ) and Route Reply (RREP), meanwhile route maintenance that consist of Route Error (RERR) is used for finding out an error information in route.

In this final project, there will be limitation on adaptive forwarding node on AODV routing protocol based on neighbour node connectivity level by eliminating number of one-hop node that eligible rebroadcast a RREQ with a Threshold which has been calculated every interval. The interval used are 5 seconds 10 seconds and 15 seconds. This is done to improve the performance of the AODV routing protocol to find a stable route by modifying some parts of the RREQ packet delivery mechanism. The

evaluation shows that, in the grid scenario the value of Packet Delivery Ratio (PDR) has increased by 2,61%, the value of Routing Overhead (RO) has decreased by 3,41% and in the real scenario the value of Packet Delivery Ratio (PDR) has increased by 3,92%, the value of Routing Overhead (RO) has decreased by 0,86%

Keyword: VANETs, AODV, NS2, SUMO, Adaptive Forwarding Node, Neighbor Node

KATA PENGANTAR

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

Puji syukur kepada Allah Yang Maha Esa atas segala karunia dan rahmat-Nya penulis dapat menyelesaikan Tugas Akhir yang berjudul **"Pembatasan *Forwarding Node* yang Adaptif pada *Route Discovery Process Ad-hoc On demand Distance Vector* (AODV) berdasarkan Level Konektivitas Node Tetangga di 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 Slamet Efendy Bhaskara dan Ibu Indah Isnaini selaku kedua orangtua penulis atas segala dukungan berupa motivasi serta doa sehingga penulis dapat mengerjakan Tugas Akhir ini.
3. Arifian Remianto, Kristiana dan Almh. Dian Septiayu Fendini selaku kakak penulis atas segala 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. Muchammad Husni, M.Kom. selaku dosen pembimbing, atas arahan dan bantuannya dalam pengerjaan Tugas Akhir ini.
5. Sahabat Warkop X Jojoran, Sahabat Korek Api dan teman – teman angkatan 2014 yang selama ini sudah membantu penulis dalam menyelesaikan Tugas Akhir ini

6. Teman teman HMTC optimasi, HMTC Inspirasi dan HMTC Kreasi yang sudah memberikan kesibukan lebih untuk penulis semasa kuliah di Informatika ITS.
7. TC 15, TC 16 dan TC 17 yang sudah membuat penulis cukup pusing, sedih dan senang akan tingkah lakunya.
8. Bapak Dwi Kristianto S.T, M.Kom selaku pembina AIS ITS dan PhibiSeo, serta teman teman AIS ITS dan PhibiSeo yang sudah membuat penulis sadar akan artinya kerja keras dan *pressure* dalam perkuliahan dan pekerjaan.
9. Teman – teman IKAMISAYA 2014 dan SIMPANSE yang selama ini sudah menyemangati Penulis dalam menjalankan kuliah di ITS.
10. Sahabat AJK yang sudah membantu dan menyediakan fasilitas selama penulis mengerjakan Tugas Akhir ini.
11. 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, Juni 2018

Rizky Fenaldo Maulana

DAFTAR ISI

(Halaman ini sengaja dikosongkan)

DAFTAR GAMBAR

DAFTAR TABEL

(Halaman ini sengaja dikosongkan)

BAB I
PENDAHULUAN

1.1 Latar Belakang

Kemajuan terbaru dalam perangkat keras, perangkat lunak dan teknologi komunikasi memungkinkan adanya desain dan implementasi berbagai macam jaringan yang diimplementasikan di berbagai jenis lingkungan. Salah satu jaringan yang bertumbuh pesat dan mendapat perhatian cukup besar adalah *Vehicular Ad-Hoc Networks*(VANETs). *Vehicle Ad hoc Networks* (VANETs) merupakan perkembangan dari *Mobile Ad hoc Network* (MANET) dimana setiap node memiliki mobilitas yang tinggi yang menyebabkan perubahan dalam topologi jaringan sehingga kesulitan dalam mengaturnya. Topologi yang dinamis ini merupakan perbedaan mendasar antara VANET dan MANET. Tingkat mobilitas yang tinggi pada setiap node juga dapat menyebabkan adanya rute terputus dikarenakan node keluar dari jangkauan sinyal transmisi [1].

Routing protocol dalam VANET dibedakan menjadi dua model, yaitu *proactive* dan *reactive routing*. *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 merupakan mekanisme *routing* yang membentuk tabel *routing* jika ada permintaan pengiriman data atau terjadinya perubahan rute dalam setiap jaringan. Contoh proactive routing protocol adalah *Destination-Sequenced Distance Vector* (DSDV), dan *Optimized Link State Routing* protocol (OLSR) sedangkan contoh reactive routing protocol adalah *Adhoc On-Demand Distance Vector* (AODV), dan *Dynamic Source Routing* (DSR).

Pencarian rute menjadi suatu mekanisme yang penting untuk mendukung mobilitas di VANET. Pemilihan rute yang stabil saat proses pencarian rute sangat diperlukan untuk memperpanjang

waktu penggunaan rute. Salah satu cara dapat dilakukan adalah dengan memilih rute yang memiliki kemungkinan kecil terputus [2].

Pada Tugas Akhir ini diusulkan suatu mekanisme *routing discovery* yang bersifat adaptif pada *reactive routing* AODV untuk memperoleh rute yang paling stabil berdasarkan level konektivitas *node* tetangga pada VANETs. Hasil akhir yang diharapkan adalah mengetahui perbandingan kinerja antara AODV dan AODV yang telah dimodifikasi diukur berdasarkan performansi Packet Delivery Ratio (PDR), Routing Overhead, End-to-End Delay, dan Forwarded Route Request (RREQ F).

1.2 Rumusan Masalah

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

1. Bagaimana membatasi jumlah *forwarding node* secara adaptif dalam proses *rebroadcast Route Request* (RREQ) di lingkungan yang dinamis?
2. Bagaimana dampak pembatasan *forwarding node* terhadap performa protokol AODV secara keseluruhan di lingkungan yang dinamis?
3. Berapakah interval terbaik yang digunakan dalam membatasi jumlah *forwarding node* secara adaptif dalam proses *rebroadcast Route Request* (RREQ) di lingkungan yang dinamis?

1.3 Batasan Permasalahan

Batasan masalah pada Tugas Akhir ini adalah sebagai berikut:

1. Jaringan yang digunakan adalah VANETs.
2. Routing protocol yang diujicobakan yaitu DSR.

3. Simulasi pengujian jaringan menggunakan *Network Simulator 2* (NS-2).
4. Pembuatan skenario uji coba menggunakan *Simulation of Urban Mobility* (SUMO).

1.4 Tujuan

Tujuan dari Tugas Akhir ini adalah sebagai berikut:

1. Mereduksi jumlah *forwarding node* yang bertanggung jawab untuk *rebroadcast* RREQ *message*.
2. Mengurangi jumlah *control packet* yang di*broadcast* dalam jaringan.

1.5 Manfaat

Manfaat yang diperoleh dari pengerjaan Tugas Akhir ini adalah dapat memberikan informasi tentang dampak dari pembatasan *forwarding node* yang adaptif berdasarkan level konektivitas node tetangga terhadap kinerja *routing protocol* AODV di lingkungan VANETs.

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, manfaat dan tujuan 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 studi literatur ini, akan dipelajari sejumlah referensi yang diperlukan dalam pengerjaan Tugas Akhir ini yaitu mengenai *Vehicular Ad Hoc Networks* (VANETs), *routing protocol* DSR, *Network Simulator* 2 (NS-2), OpenStreetMap, Java OpenStreetMap (JOSM), *Simulation of Urban Mobility* (SUMO), dan AWK.

1.6.3 Implementasi Sistem

Implementasi merupakan tahap untuk mengimplementasikan metode-metode yang sudah diajukan pada proposal Tugas Akhir. Untuk membangun algoritma yang telah dirancang sebelumnya, implementasi dilakukan dengan menggunakan NS-2 sebagai *Network Simulator*, bahasa C/C++ sebagai bahasa pemrograman, SUMO, dan JOSM sebagai perangkat lunak untuk uji coba mengimplementasikan metode yang sudah diajukan.

1.6.4 Pengujian dan Evaluasi

Pengujian dilakukan dengan VANETs *simulator generator* dan *traffic generator* yaitu SUMO untuk membuat simulasi keadaan topologi untuk diujikan. Kemudian simulasi yang dibuat pada SUMO tersebut dijalankan pada NS-2 *Network Simulator* dan akan menghasilkan *trace file*. Dari *trace file* tersebut akan dihitung *packet delivery ratio* (PDR), *end-to-end delay* (E2E), *routing overhead* (RO), dan *forwarded route request* (RREQ F) untuk mengetahui performa *routing protocol* yang telah dimodifikasi.

1.6.5 Penyusunan Buku

Pada tahap ini disusun buku sebagai dokumentasi dari pelaksanaan Tugas Akhir yang mencakup seluruh konsep, teori, implementasi, serta hasil yang telah dikerjakan

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, DSR, NS-2, OpenStreetMap, Java OpenStreetMap Editor, SUMO, dan AWK.
3. Bab III. Perancangan
   Bab ini berisi pembahasan mengenai perancangan skenario mobilitas *grid* dan *real*, perancangan simulasi pada NS-2, perancangan modifikasi DSR, serta perancangan metrik analisis (*Packet Delivery Ratio, End-to-End Delay, Routing Overhead*, dan *Forwarded Route Request*)
4. Bab IV. Implementasi
   Bab ini menjelaskan implementasi yang berbentuk kode sumber dari proses modifikasi protokol DSR, pembuatan peta untuk skenario *grid* dan skenario *real* menggunakan OpenStreetMap dan SUMO, melakukan simulasi menggunakan NS-2, dan perhitungan metrik analisis.
5. Bab V. Pengujian dan Evaluasi
   Bab ini berisikan hasil uji coba dan evaluasi dari implementasi modifikasi pada *routing protocol* DSR yang telah dilakukan

untuk menyelesaikan masalah yang dibahas pada Tugas Akhir. Pengujian dilakukan dengan skenario yang di*generate* oleh SUMO dan dijalankan di NS-2 untuk mendapatkan data uji PDR, E2E, RO, dan RREQ F yang nantinya akan dianalisis menggunakan skrip awk dan dilakukan perbandingan performa *routing protocol* DSR asli dengan *routing protocol* DSR yang telah dimodifikasi.

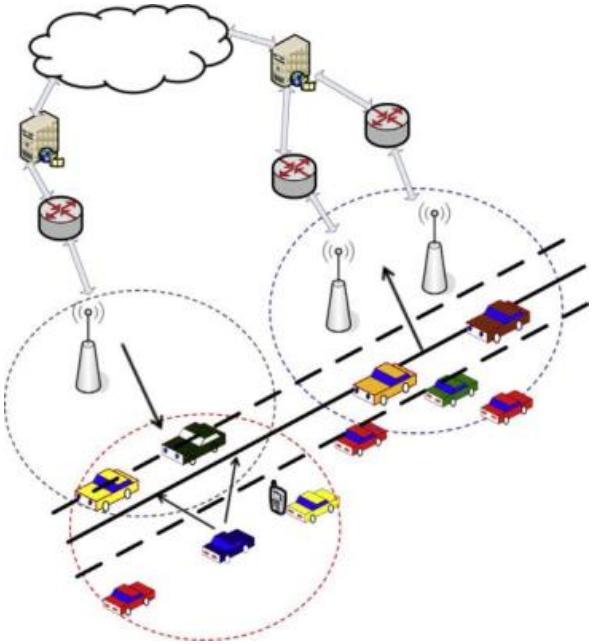
6. Bab VI. Kesimpulan dan Saran
   Bab ini merupakan bab yang menyampaikan kesimpulan dari hasil uji coba yang dilakukan, masalah-masalah yang dialami pada proses pengerjaan Tugas Akhir, dan saran untuk pengembangan solusi ke depannya.
7. Daftar Pustaka
   Bab ini berisi daftar pustaka yang dijadikan literatur dalam Tugas Akhir.
8. Lampiran
   Dalam lampiran terdapat 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. Penjelasan ini bertujuan untuk memberikan gambaran secara umum terhadap program yang dibuat dan berguna sebagai penunjang dalam pengembangan riset yang berkaitan.

2.1 VANETs

Vehicular Ad-hoc Networks (VANETs) merupakan pengembangan dari *Mobile Ad-hoc Network* (MANET) dimana pengembangannya difokuskan pada kendaraan (*vehicle*) yang dapat saling berkomunikasi maupun mengirimkan data. VANETs adalah sebuah teknologi baru yang memadukan kemampuan komunikasi nirkabel kendaraan menjadi sebuah jaringan yang bebas infrastuktur serta memiliki karakteristik mobilitas yang sangat tinggi dan terbatas pada pola pergerakannya. *Node* dalam jaringan dianggap sebagai *router* yang bebas bergerak dan bebas menentukan baik menjadi *client* maupun menjadi *router*. Protokol *routing* pada VANETs memiliki dua model yaitu protokol *reactive routing* yang membentuk tabel *routing* hanya saat dibutuhkan dan protokol *proactive routing* yang melakukan pemeliharaan tabel *routing* secara berkala pada waktu tertentu. 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 [3]. Ilustrasi VANETs dapat dilihat pada Gambar 2.1



Gambar 2.1 Ilustrasi VANETs [3]

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

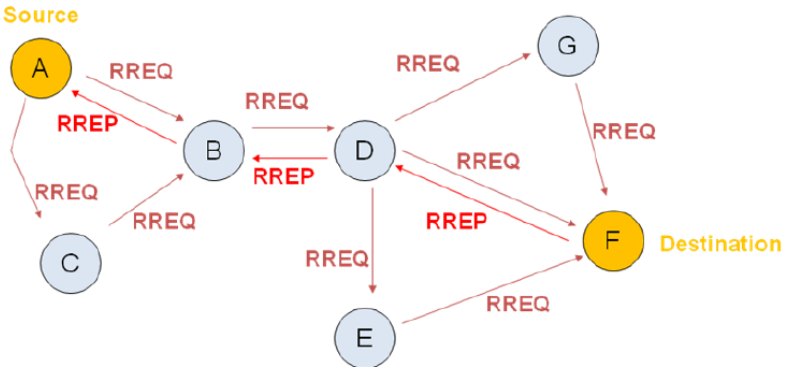
2.2 *Ad-hoc On demand Distance Vevtor* (AODV)

Ad-hoc On demand Distance Vector (AODV) adalah salah satu routing protokol yang termasuk dalam klasifikasi reactive routing protocol. Sebuah protokol yang hanya membuat sebuah rute saat dibutuhkan. AODV dikembangkan oleh C. E. Perkins, E.M. Belding-Royer dan S. Das pada RFC 3561.

Ciri utama dari AODV adalah menjaga timer-based state pada setiap node sesuai dengan penggunaan tabel routing. Tabel routing akan kadaluarsa jika jarang digunakan. Ada dua tahapan dalam

AODV yaitu route discovery dan route maintenance. Route discovery memiliki dua pesan yaitu berupa Route Request (RREQ) dan Route Reply (RREP). Sedangkan Route maintenance berupa Route Error (RERR).

AODV adalah sebuah metode *routing* pesan 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* [4]. Ilustrasi pencarian rute oleh AODV dapat dilihat pada Gambar 2.2



Gambar 2.2 Ilustrasi pencarian rute routing protocol AODV [4]

Pada setiap *node* yang menggunakan protokol AODV pasti memiliki sebuah *routing table* dengan *field* sebagai berikut:

- *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 *node* .

Pada Tugas Akhir ini, penulis menggunakan *routing protocol* AODV yang akan diimplementasikan pada lingkungan VANETs dengan beberapa skenario.

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

Network Simulator (NS) adalah suatu *interpreter* yang berorientasi objek, dan *discrete event-driven* yang dikembangkan oleh University of California Berkeley dan USC ISI sebagai bagian dari projek *Virtual INternet Testbed* (VINT). NS yang banyak dikenal dengan NS-2 (versi 2) menjadi salah satu *tool* yang sangat berguna untuk menunjukkan simulasi jaringan melibatkan *Local Area Network* (LAN), *Wide Area Network* (WAN), tapi fungsi dari *tool* ini telah berkembang selama beberapa tahun belakangan untuk

memasukkan jaringan nirkabel (*wireless*) dan juga jaringan *ad hoc* [5].

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 2.3

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

Gambar 2.3 Perintah untuk menginstall dependency NS-2

Setelah menginstall dependency yang dibutuhkan, ekstrak *package* NS-2 dan ubah baris kode ke-137 pada file ls.h di folder linkstate menjadi seperti pada Gambar 2.4. Lalu *Install* NS-2 dengan menjalankan perintah ./install pada folder NS-2.

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

Gambar 2.4 *Baris kode yang diubah pada file ls.h*

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 dunia yang gratis dan terbuka, dibangun sepenuhnya oleh sukarelawan dengan melakukan survei menggunakan GPS, mendigitalisasi citra satelit dan mengumpulan serta membebaskan data geografis yang tersedia di publik. Melalui *Open Data Commons Open Database License 1.0,* kontributor 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, *inovator*, dan banyak pihak lainnya tidak dapat menggunakan data yang tersedia di dalam peta tersebut secara luas. Di sisi lain, baik peta dasar OSM maupun data yang tersedia di dalamnya dapat diunduh secara gratis dan terbuka, untuk kemudian digunakan untuk diredistribusikan kembali.

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

baik itu pengembangan ekonomi, tata kota, kontinjensi bencana, maupun untuk berbagai tujuan lainnya [6].

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)

Java OpenStreetMap Editor (JOSM) adalah aplikasi untuk menyunting data yang didapatkan dari OpenStreetMap [7]

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) merupakan paket simulasi lalu lintas yang bersifat *open-source* dimana pengembangannya dimulai pada tahun 2001. Dan semenjak itu SUMO telah berubah menjadi sebuah simulasi lalu lintas dengan kelengkapan fitur dan pemodelannya termasuk kemampuan jalannya jaringan untuk membaca *format* yang berbeda.

SUMO juga memungkinkan untuk mendefinisikan kendaraan dengan sifat tertentu seperti panjang kendaraan, kecepatan maksimum, percepatan dan perlambatannya. SUMO juga menyediakan pilihan bagi pengguna menentukan rute acak untuk kendaraan. Ada juga pilihan yang tersedia untuk model sistem transportasi umum, dimana setiap kendaraan datang dan berangkat sesuai dengan jadwal [8].

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 digunakan untuk melakukan *text processing* dan ekstraksi data [9]. AWK merupakan sebuah program filter untuk teks, seperti halnya perintah grep pada terminal linux. AWK dapat digunakan untuk mencari bentuk / model dalam sebuah berkas teks ke dalam bentuk teks lain. AWK dapat juga digunakan untuk melakukan proses aritmatika seperti yang dilakukan oleh perintah expr. AWK sama halnya seperti bahasa shell atau C yang memiliki karakteristik yaitu sebagai *tool* yang cocok untuk *jobs* juga sebagai pelengkap untuk *filter* standar.

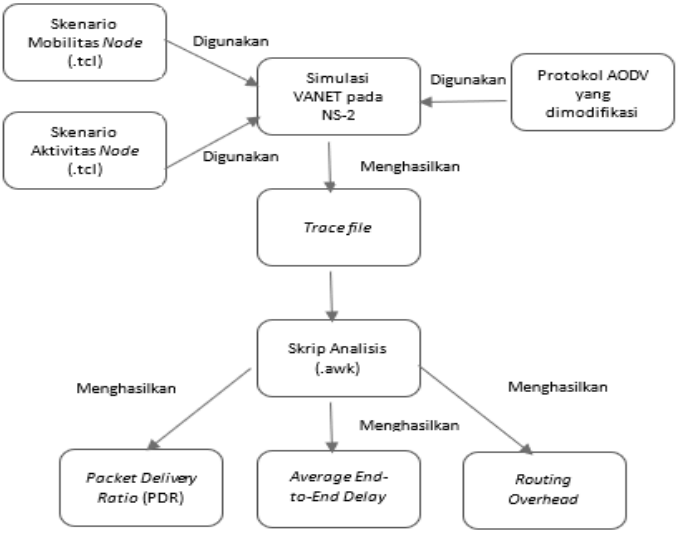
Pada Tugas Akhir ini, AWK digunakan untuk membuat script menghitung *Packet Delivery Ratio* (PDR), *Routing Overhead* (RO), *Forwarded Route Request* (RREQ F), dan *End-to-End Delay* dari *trace file* NS2.

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 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 akan diawali dengan pencarian jumlah tetangga setiap *node* perantara (*one-hop node)*. Jumlah tetangga tiap *node* akan didapatkan dengan menggunakan HELLO *messages* yang terdapat pada AODV. Setelah jumlah tetangga setiap *node* didapatkan, maka modifikasi dilanjutkan untuk melakukan perhitungan nilai *threshold* yang dilakukan setiap *i* interval waktu dan penyeleksian *forwarding node* yang bertugas untuk *rebroadcast* paket *Route Request* (RREQ) yang akan diteruskan. Hal tersebut dilakukan dengan cara menyeleksi *node* – *node* mana saja yang mempunyai jumlah tetangga lebih dari nilai *threshold* yang sudah ditentukan. Jika *node* tersebut mempunyai jumlah tetangga lebih dari *threshold*, maka *node* tersebut akan meneruskan paket RREQ, namun jika sebaliknya, maka *node* tersebut tidak akan meneruskan paket RREQ atau paket akan di-*drop*.
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), dan *Forwarded Route Request* (RREQ F), dan *End-to-End Delay* (E2E). 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.

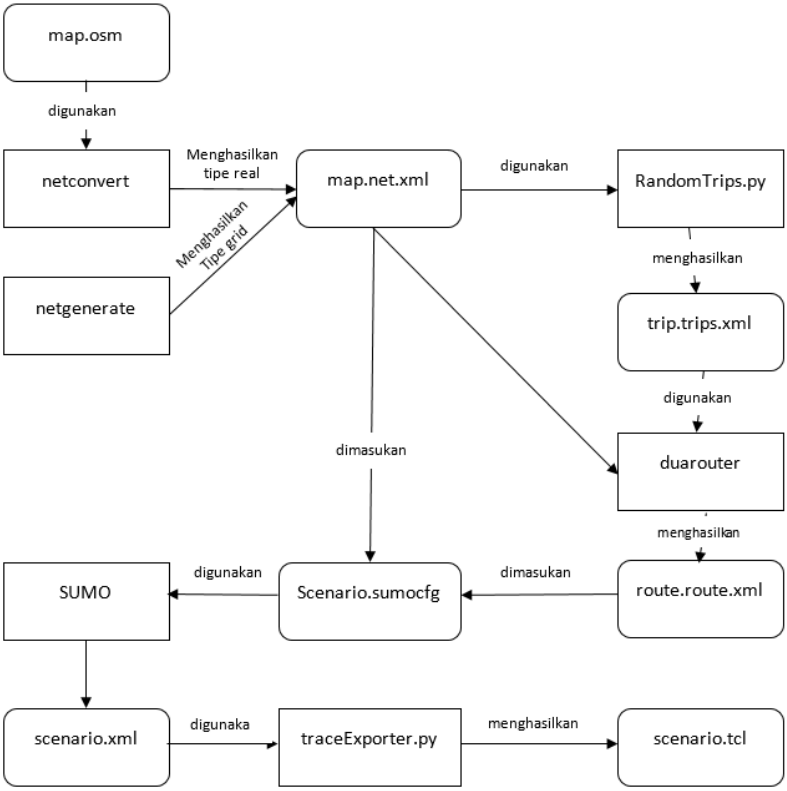
Tabel 3.1 Daftar Istilah

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

No.	Istilah	Penjelasan
		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	RREQ	*Route Request*. Paket *request* pada AODV yang dikirim untuk mendapatkan rute
6	RREP	*Route Reply*. Paket *reply* pada AODV yang dikirim ke *node* sumber melalui rute yang sudah terbuat.
7	*Threshold*	Batas nilai yang dijadikan sebagai acuan

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.



Gambar 3.2 Alur peracangan 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

1300 m x 1300 m. Dengan 4 titip persimpangan, maka akan didapatkan 9 petak dan panjang tiap peta adalah 400m.

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 *route discovery* pada protokol tersebut. Pada protokol AODV, mekanisme pencarian *node* untuk pengiriman ulang (*rebroadcast*) paket RREQ langsung dikirim begitu saja, maka pada AODV yang dimodifikasi ini akan ada proses penyeleksian *node*. Seleksi *node* dilakukan dengan cara *node* sumber mengetahui jumlah tetangga yang dimiliki *node* perantara (*one-hop node*). Setelah mengetahui jumlah tetangga yang dimiliki setiap node maka terdapat fungsi yang berjalan setiap *i* interval yang akan menghitung *threshold* berdasarkan modus jumlah tetangga tiap node.

Selanjutnya, dilakukan perbandingan menggunakan *threshold* yang didapatkan dari fungsi modifikasi untuk pengiriman RREQ. Apabila jumlah tetangga *one-hop node* tersebut lebih besar atau sama dengan dengan *threshold* yang sudah ditentukan, maka pengiriman RREQ akan dilanjutkan. Namun sebaliknya, apabila jumlahnya kurang dari *threshold*, maka paket akan di*drop*. Jika sudah mencapai *node* tujuan, *node* akan mengembalikannya dengan paket RREP. Rute yang dilalui paket RREP adalah rute dengan pembatasan *forwarding node* yang sudah dilakukan sebelumnya. Rute tersebut tidak melakukan *rebroadcast* RREQ ke semua *node* dan paket RREQ hanya melewati *node* – *node* terpilih untuk sampai ke *node* tujuan. Karena beberapa paket RREQ di-drop, maka kemacetan dan tabrakan paket pada jaringan akan lebih rendah sehingga bisa menaikkan PDR.

3.3.1 Perancangan Penghitungan Jumlah *Node* Tetangga untuk Setiap *Node*

Terdapat beberapa cara untuk dapat mengetahui jumlah tetangga yang ada di sekitar *node*. Pada Tugas Akhir ini,

penghitungan jumlah tetangga dilakukan dengan memanfaatkan HELLO *messages* yang ada pada protokol AODV. HELLO *packets* akan mengirimkan HELLO *messages* secara terus menerus untuk memberikan informasi kepada *node* tersebut mengenai tetangga yang ada di sekitarnya. Setiap *node* yang menerima HELLO *messages* dari *node* lainnya, sudah dipastikan menjadi tetangga *node* tersebut. Dengan begitu, jumlah tetangga dapat dihitung dari setiap *node* yang mengirimkan HELLO *messages*. Modifikasi akan dilakukan dengan menambahkan counter jumlah tetangga tiap node pada fungsi nb_insert() yang digunakan untuk menambah node tetangga dan nb_delete() yang digunakan untuk mengurangi node tetangga pada file aodv.cc yang terletak di dalam *folder* ns-2.35/aodv.

3.3.2 Perancangan Perhitungan *threshold*

Threshold akan ditentukan melalui perhitungan nilai modus dari jumlah tetangga tiap node. Perhitungan threshold dilakukan setiap *i* interval waktu. Interval waktu yang digunakan adalah 5 detik, 10 detik dan 15 detik. Jumlah tetangga tiap node disimpan dalam bentuk *array* sehingga diperlukan proses *sorting array* terlebih dahulu. Setelah array di sorting, maka akan dicari nilai yang paling sering keluar atau disebut nilai modus. Nilai modus yang didapatkan akan digunakan sebagai *threshold*. *Pseudocode* untuk perhitungan *threshold* dapat dilihat pada Gambar 3.3.

```
for i=0 to node_count do
 for j=i+1 to node_count do
  if neighbour_count[i]>neighbour_count[j]
  then
   tmp=neighbour_count[i]
   neighbour_count[i]=neighbour_count[j]
   neighbour_count[j]=tmp
  endif
 endfor
endfor
for i=0 to node_count do
```

```
 for j=0 to node_count do
  if neighbour_count[i]==neighbour_count[j]
  then
   count_mode[i}++
  endif
 endfor
endfor

for i=0 to node_count
 if count_mode[i]>threshold then
  threshold=count_mode[i]
 endif
endfor
```

Gambar 3.3 *Pseudocode* perancangan perhitungan *Threshold*

3.3.3 Perancangan Pemilihan *Forwarding Node*

Setelah melakukan penghitungan jumlah *node* tetangga, akan dilakukan pemilihan *forwarding node*, yaitu *node* yang berhak untuk melakukan *rebroadcast* paket RREQ. Penentuan *forwarding node* dilakukan dengan cara membandingkan jumlah *node* tetangga dengan *threshold* atau batas nilai yang sudah ditentukan. Apabila jumlah tetangga yang dimiliki *node* tersebut melebihi *threshold*, maka *node* tersebut berhak melakukan *rebroadcast*, jika sebaliknya maka paket akan di-*drop*. Langkah tersebut dilakukan untuk mengurangi jumlah paket RREQ yang dikirim sehingga dapat mengurangi kemacetan yang terjadi pada jaringan. *Pseudocode* untuk pemilihan *forwarding node* dapat dilihat pada Gambar 3.4.

```
if count < threshold
      drop RREQ packet
end if
for i=0 to node_count do
 for j=i+1 to node_count do
  if neighbour_count[i]>neighbour_count[j]
```

```
  then
   tmp=neighbour_count[i]
   neighbour_count[i]=neighbour_count[j]
   neighbour_count[j]=tmp
  endif
 endfor
```

Gambar 3.4 *Pseudocode* Pemilihan *Forwarding 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.

Kode yang diubah diantaranya adalah pencarian jumlah *node* tetangga pada *file* node.cc dan perbandingan dengan *threshold* pada *file* aodv.cc. 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 Rata-rata *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.

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

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.

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

3.5.4 *Forwarded Route Request* (RREQ F)

Forwarded Route Request adalah jumlah paket kontrol *route request* yang di*forward* per data paket ke *node* tujuan selama simulasi terjadi. RREQ F didapatkan dengan menjumlahkan semua paket control *routing* yang ditransmisikan khususnya *route request* bagian *forwarding* (RREQ F). Perhitungan RREQ F dapat dilihat dengan persamaan 3.4

$$Forwarded\ Route\ Request = \sum_{n=1}^{rreqsent} packet\ sent \quad (3.4)$$

(Halaman ini sengaja dikosongkan)

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.

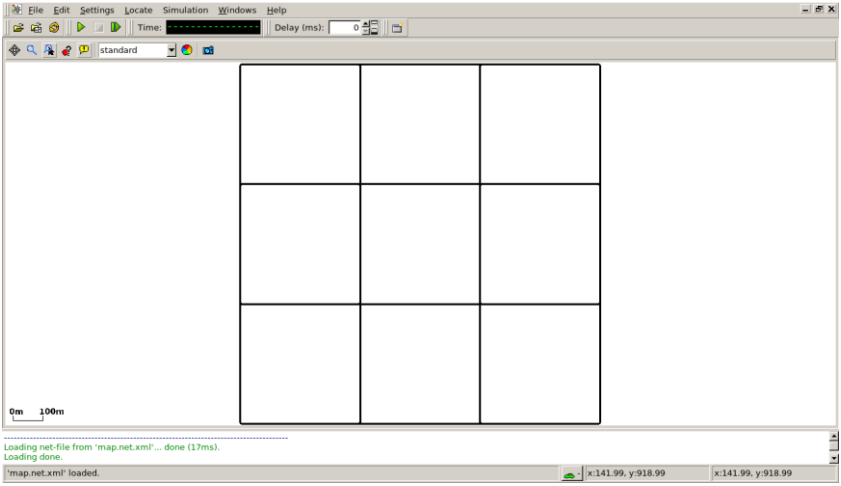
4.1.1 Skenario *Grid*

Dalam mengimplementasikan skenario *grid*, SUMO menyediakan *tools* untuk membuat peta *grid* yaitu netgenerate. Pada Tugas Akhir ini, penulis membuat peta *grid* dengan luas 1300 m x 1300 m yang terdiri dari titik persimpangan antara jalan vertikal dan jalan horisontal 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 1300 m x 1300 m dibutuhkan luas per petak sebesar 400 m x 400 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* seacara 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="routes.rou.xml"/>
   </input>
   <time>
      <begin value="0"/>
      <end value="200"/>
   </time>
</configuration>
```

Gambar 4.5 FIle Skrip .sumocfg

File .sumocfg disimpan dalam direktori yang sama dengan *file* peta 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
```

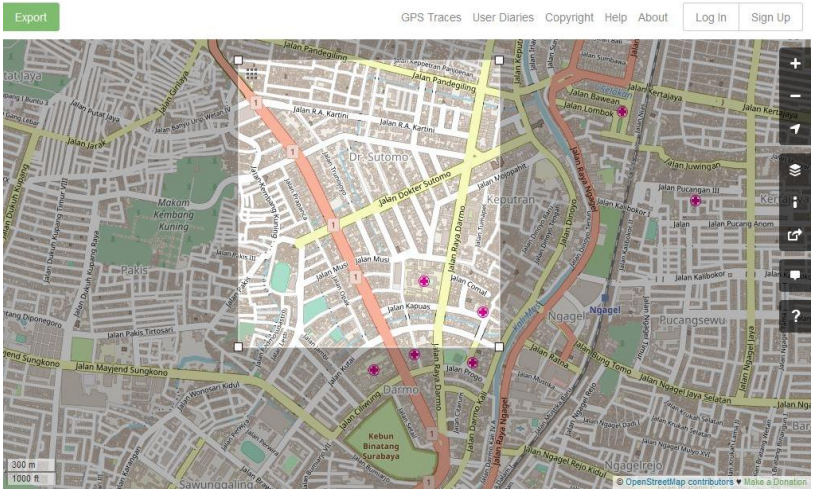
Gambar 4.7 *Perintah traceExporter*

Pada file scenario.tcl yang dihasilkan dari perintash diatas akan di *random* posisi dan pergerakan node. Oleh karena itu, terdapat beberapa node yang memiliki posisi awal diluar rute peta grid, sehingga menyebabkan node tidak akan bergerak atau statis.

4.1.2 Skenario *Real*

Dalam mengimplementasikan skenario *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.



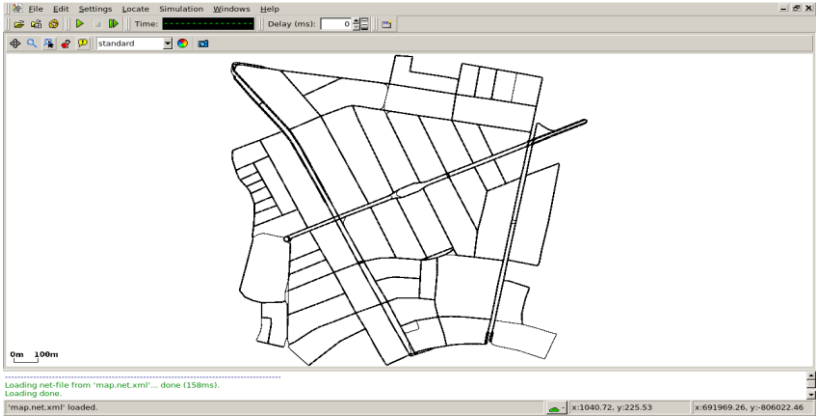
Gambar 4.8 *Eksport Peta dari OpenStreetMap*

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 Menentukan *Forwarding Node*

Pada Tugas Akhir ini dilakukan modifikasi pada *routing protocol* AODV agar dapat mengurangi jumlah *forwarding node*, yaitu *node* yang bertugas untuk melakukan *rebroadcast* paket RREQ. Hal tersebut dilakukan dengan cara memilih *forwarding node* berdasarkan jumlah tetangga *node* tersebut dengan *threshold* yang dihitung berdasarkan fungsi yang berjalan setiap *i* interval yang membuat *threshold* bersifat adaptif, sehingga dapat dilihat peningkatan performa pada *routing* AODV yang telah dimodifikasi.

Implementasi modifikasi routing protocol AODV ini dibagi menjadi 3 bagian yaitu:

- Implementasi Penghitungan Jumlah *Node* Tetangga
- Implementasi Penghitungan *Threshold*
- Implementasi Pemilihan *Forwarding 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 menghitung jumlah *node* tetangga, menghitung *threshold* dan menentukan forwarding node dan *file* aodv.h yang ada di dalam *folder* ns-2.35/aodv untuk mendaftarkan fungsi dan *timer* baru. Pada bagian ini penulis akan menjelaskan langkah – langkah dalam mengimplementasikan modifikasi *routing protocol* AODV untuk mengurangi jumlah *forwarding node* yang melakukan *rebroadcast.*

4.2.1 Implementasi Menghitung Jumlah *Node* Tetangga

Langkah awal yang dilakukan untuk menghitung jumlah tetangga seperti yang telah dirancang pada subbab 3.3.1 adalah dengan cara menangkap pesan HELLO *messages* dari *node* yang mengirimkannya.

Node yang mengirimkan HELLO *messages* sudah dapat dipastikan berada di sekitar *node* tersebut dan berada di *range* transmisi dari *node* yang sedang dieksekusi. Secara *default*, AODV menginformasikan siapa saja *node* tetangga yang ada di sekitar *node* tersebut. Terdapat fungsi nb_insert dan nb_delete yang digunakan untuk menambah atau menghapus node tetangga yang mendekat atau menjauh. Pada tugas akhir ini, penulis memanfaatkan kedua fungsi tersebut dengan tambahan modifikasi counter setiap penambahan atau pengurangan node. Kedua fungsi tersebut terdapat pada kode sumber aodv.cc yang terdapat dalam folder ns2.3.5/aodv. Untuk potongan

kode tersebut bisa dilihat pada Gambar 4.11. Kode selengkapnya dapat dilihat di lampiran A1.

```
void AODV::nb_insert(nsaddr_t id){
  count_neighbour[index]+=1;
  AODV_Neighbor *nb = new
AODV_Neighbor(id);
  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);
}

void AODV::nb_delete(nsaddr_t id){
  count_neighbour[index]-=1;
  AODV_Neighbor *nb = nbhead.lh_first;
  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);
}
```

Gambar 4.11 Potongan modifikasi kode fungsi nb_insert() dan nb_delete()

4.2.2 Implementasi Perhitungan *Threshold*

Pada tahap selanjutnya setelah dapat mengetahui jumlah tetangga, langkah yang harus dilakukan selanjutnya adalah menghitung threshold secara adaptif untuk digunakan sebagai pembatasan *forwarding node* untuk melanjutkan paket RREQ.

Penghitungan ini dilakukan pada fungsi countThreshold() yang terletak pada kode sumber aodv.cc yang terletak pada ns2.35/aodv. Jumlah tetangga yang sudah ditemukan, disimpan dalam variabel *count_neighbour*. Jumlah tetangga kemudian akan dicari nilai modus. nilai modus dari jumlah tetangga akan dibuat sebagai acuan untuk Threshold. Fungsi ini berjalan setiap interval yang ditentukan. Interval yang digunakan dalam uji coba adalah 5 detik, 10 detik dan 15 detik. Potongan kode untuk proses seleksi *forwarding node* dapat dilihat pada Gambar 4.12

```
for(int i=0;i<nodes_count;i++){
 for(int j=(i+1);j<nodes_count;j++){

if(count_neighbour[i]>count_neighbour[j]){
     int tmp;
     tmp=count_neighbour[i];

count_neighbour[i]=count_neighbour[j];
     count_neighbour[j]=tmp;
    }
  }
}
for(int i=0;i<nodes_count;i++){
 count_mode[i]=0;
  for(int j=0;j<nodes_count;j++){

if(count_neighbour[i]==count_neighbour[j])
{
      count_mode[i]++;
```

```
      }
   }
}
for(int i=0;i<nodes_count;i++){
  if(count_mode[i]>th ){
    th=count_mode[i];
  }
}
```

Gambar 4.12 Potongan kode perhitungan *Threshold*

4.2.3 Implementasi Pemilihan *Forwarding Node*

Pada tahap selanjutnya setelah dapat mengetahui jumlah tetangga dan *Threshold* , langkah yang harus dilakukan selanjutnya adalah pemilihan *forwarding node* untuk melanjutkan paket RREQ.

Penghitungan ini dilakukan pada fungsi recvRequest() yang terletak pada kode sumber aodv.cc yang terletak pada ns2.35/aodv.. Apabila jumlah tetangga kurang dari *threshold* yang ditentukan dan bukan *node* sumber, maka paket RREQ akan di*drop.* Potongan kode untuk proses seleksi *forwarding node* dapat dilihat pada Gambar 4.13

```
if (count < threshold && rq->rq_dst !=
index){
     Packet::free(p);
     return;
}
```

Gambar 4.13 Potongan kode penyeleksian *Forwarding Node*

Dengan proses penyeleksian *node* mana saja yang dapat melanjutkan paket RREQ sudah dapat mengurangi jumlah paket *routing overhead* untuk paket RREQ. Kode implementasi ini diletakkan pada lampiran A.3 Kode Fungsi nb_remove().

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 4.14.

```
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)      "cbr60.txt"
set val(sc)      "scenario1.txt"
```

Gambar 4.14 Implementasi simulasi NS-2

Pada konfigurasi dilakukan pemanggilan terhadap *file traffic* yang berisikan konfigurasi *node* asal, *node* tujuan, pengiriman paket, serta *file* skenario yang berisi pergerakan *node* yang telah di*generate* oleh SUMO. Kode implementasi pada NS-2 dapat dilihat pada lampiran A.4 Kode Skenario NS-2.

Konfigurasi untuk *file traffic* bisa dilakukan dengan membuat *file* berekstensi .txt 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 4.15.

```
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 4.15 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 Forwarded Route Request (RREQ F).

4.4.1 Implementasi *Packet Delivery Ratio*

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 4.16.

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

Gambar 4.16 *Pseudocode* untuk menghitung PDR

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

4.4.2 Implementasi Rata-Rata *End-to-End Delay*

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 4.17.

```
sum_delay = 0
counter = 0

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

Gambar 4.17 *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*

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 4.18.

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

Gambar 4.18 *Pseudocode* untuk perhitungan *Routing Overhead*

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

4.4.4 Implementasi *Forwarded Route Request*

Forwarded Route Request (RREQ F) adalah jumlah paket control route request yang diteruskan per-data paket ke node tujuan selama simulasi terjadi. Dengan begitu, untuk mendapatkan RREQ F yang perlu dilakukan adalah menjumlahkan tiap paket dengan filter event sent pada kolom pertama, event layer RTR pada kolom ke-4, dan event layer route request pada kolom-25. Penyaringan juga dilakukan pada kolom ke-3 yang menunjukan node id. Selama node

bukan node sumber, maka akan terus dilakukan penjumlahan baris yang terdapat pada file dengan ekstensi .tr. Perhitungan RREQ F telah dijelaskan pada persamaan 3.4. Skrip awk untuk menghitung RREQ F dapat dilihat pada lampiran A.9 Kode Skrip AWK Forwarded Route Request . *Pseudocode* untuk menghitung RREQ F dapat dilihat pada Gambar 4.19.

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

Gambar 4.19 *Pseudocode* perhitungan *Forwarded Route Request*

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

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) Celeron (R) 1007U 1.50 GHz
Sistem Operasi	Xubuntu 16.04 LTS
Linux Kernel	Linux kernel 4.4
Memori	2.0 GB

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

- SUMO versi 0.25.0 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 X. 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 RREQ F 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 *Forwarded Route Request*.

Tabel 5.2 Lingkungan uji coba

No.	Parameter	Spesifikasi
1	Network simulator	NS-2.35
2	Routing protocol	AODV
3	Waktu simulasi	200 detik
4	Area simulasi	1500 m x 1500 m
5	Jumlah *Node*	150, 200, 300
6	Radius transmisi	400m
7	Kecepatan maksimum	20 m/s
8	Protokol MAC	IEEE 802.11p
9	Model Propagasi	*Two-ray ground*

5.2 Hasil Uji Coba

5.2.1 Hasil Uji Coba Skenario *Grid*

Pengujian pada skenario *grid* digunakan untuk melihat perbandingan PDR, RO, RREQ F, dan E2E antara *routing protocol* AODV asli dan AODV yang telah dimodifikasi dalam pemilihan *node* yang dapat menerima paket *route request*.

Pengambilan data uji PDR, RO, RREQ F, dan E2E pada skenario *grid* dilakukan sebanyak 10 kali dengan skenario mobilitas *random* pada peta *grid* dengan luas area 1300 m x 1300 m dan *node* sebanyak 150 untuk lingkungan yang jarang, 200 *node* untuk lingkungan yang sedang, dan 300 *node* untuk lingkungan yang padat dilakukan pada kecepatan standar yaitu 20 m/s. Untuk uji coba setiap lingkungan menggunakan interval yang berbeda-beda untuk mencari nilai interval yang terbaik dari hasil skenario. Interval waktu yang digunakan adalah 5 detik, 10 detik dan 15 detik. Hasil simulasi 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**			**Perbedaan**		
		5s	**10s**	**15s**	**5s**	**10s**	**15s**
150	0,867	0,898	0,880	0,884	0,031	0,013	0,017
200	0,796	0,888	0,799	0,852	0,092	0,003	0,056
300	0,867	0,900	0,814	0,867	0,033	0,054	0,001

Tabel 5.4 Hasil rata rata E2E skenario *grid*

Jumlah *Node*	**AODV Asli**	**AODV Modifikasi**			**Perbedaan**		
		5s	**10s**	**15s**	**5s**	**10s**	**15s**
150	0,867	0,898	0,880	0,884	0,031	0,013	0,017
200	0,796	0,888	0,799	0,852	0,092	0,003	0,056
300	0,867	0,900	0,814	0,867	0,033	0,054	0,001

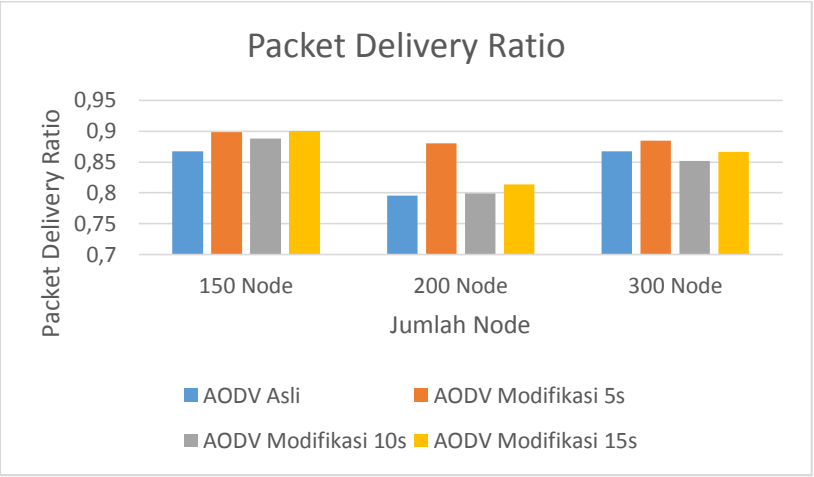
Tabel 5.5 Hasil rata rata RO skenario *grid*

Jumlah *Node*	**AODV Asli**	**AODV Modifikasi**			**Perbedaan**		
		5s	**10s**	**15s**	**5s**	**10s**	**15s**
150	69988	67905	68914	68952	2083	1074	1036
200	49228	46552	48440	48061	2676	789	1168
300	66981	65219	64684	64979	1762	2297	2002

Tabel 5.6 Hasil rata rata RREQ F seknario *grid*

Jumlah *Node*	**AODV Asli**	**AODV Modifikasi**			**Perbedaan**		
		5s	**10s**	**15s**	**5s**	**10s**	**15s**
150	9179	7040	7978	8013	2139	1201	1167
200	8624	6032	7753	7446	2592	872	1178
300	7107	5313	4808	5122	1794	2299	1985

5.2.1.1 Analisa *Packet Delivery Ratio* (PDR)



Gambar 5.1 Grafik PDR skenario *grid*

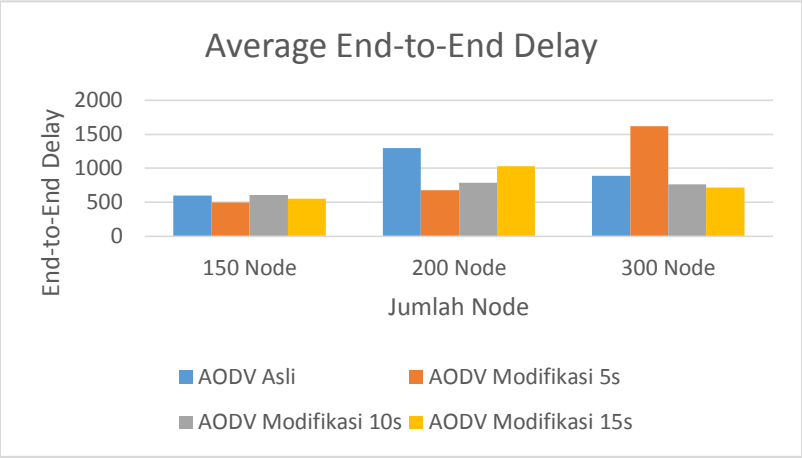
Berdasarkan grafik pada Gambar 5.1, dapat dilihat bahwa *routing protocol* AODV asli dan AODV yang telah dimodifikasi baik dengan interval 5 detik, 10 detik dan 15 detik perubahan yang fluktuaktif. Pada lingkungan yang jarang dengan *node* berjumlah 150 dan dibandingkan dengan hasil PDR AODV asli, AODV modifikasi dengan interval 5 detik menghasilkan perbedaan selisih PDR sebesar 0,03113, dimana terjadi kenaikan sebesar 3,5 %, sedangkan pada AODV modifikasi dengan interval 10 detik menghasilkan perbedaan selisih PDR sebesar 0,01298, dimana terjadi kenaikan sebesar 1,5 % dan pada AODV modifikasi dengan interval 15 detik menghasilkan perbedaan selisih PDR sebesar 0,01714, dimana terjadi kenaikan sebesar 1,97 %. Berdasarkan hal tersebut, routing protocol AODV modifikasi dengan interval 5 detik berhasil mengungguli *routing protocol* AODV asli, *routing protocol* ADOV modifikasi dengan interval 10 detik *dan routing protocol* AODV dengan interval 15 detik.

Pada lingkungan yang sedang dengan *node* berjumlah 200 dan dibandingkan dengan hasil PDR AODV asli, AODV modifikasi dengan interval 5 detik menghasilkan perbedaan selisih PDR sebesar 0,0921, dimana terjadi kenaikan sebesar 11,5 %, sedangkan pada AODV modifikasi dengan interval 10 detik menghasilkan perbedaan selisih PDR sebesar 0,00303, dimana terjadi penurunan sebesar 0.4 % dan pada AODV modifikasi dengan interval 15 detik menghasilkan perbedaan selisih PDR sebesar 0,05576, dimana terjadi penurunan sebesar 7 %. Berdasarkan hal tersebut, routing protocol AODV modifikasi dengan interval 5 detik berhasil mengungguli *routing protocol* AODV asli, *routing protocol* ADOV modifikasi dengan interval 10 detik *dan routing protocol* AODV dengan interval 15 detik.

Pada lingkungan yang padat dengan *node* berjumlah 300 dan dibandingkan dengan hasil PDR AODV asli, AODV modifikasi dengan interval 5 detik menghasilkan perbedaan selisih PDR sebesar 0,03289, dimana terjadi kenaikan sebesar 3,79 %, sedangkan pada AODV modifikasi dengan interval 10 detik menghasilkan perbedaan selisih PDR sebesar 0,05388, dimana terjadi penurunan sebesar 6,2 % dan pada AODV modifikasi dengan interval 15 detik menghasilkan perbedaan selisih PDR sebesar 0,00055, dimana terjadi penurunan sebesar 0,06 %. Berdasarkan hal tersebut, routing protocol AODV modifikasi dengan interval 5 detik berhasil mengungguli *routing protocol* AODV asli, *routing protocol* ADOV modifikasi dengan interval 10 detik *dan routing protocol* AODV dengan interval 15 detik.

Berdasarkan hasil simulasi pada ketiga lingkungan tersebut, dapat dilihat bahwa dengan menggunakan AODV modifikasi dengan interval 5 detik menghasilkan PDR yang lebih baik daripada menggunakan AODV asli, AODV modifikasi dengan interval 10 detik dan AODV modifikasi dengan interval 15 detik. Nilai rata kenaikan PDR pada skenario grid dengan menggunakan AODV modifikikasi dengan interval 5 detik adalah sebesar 6,26%.

5.2.1.2 Analisa *End-to-End Delay* (E2E)



Gambar 5.2 Grafik E2E skenario *grid*

Berdasarkan grafik pada Gambar 5.2, dapat dilihat bahwa rata-rata E2E antara *routing protocol* AODV asli dan AODV yang telah dimodifikasi dengan interval 5 detik, 10 detik dan 15 detik mengalami perubahan yang fluktuatif. Pada lingkungan yang jarang dengan jumlah 150 *node* dan dibandingkan dengan AODV asli, AODV modifikasi dengan interval 5 detik menghasilkan perbedaan selilish *end-to-end delay* sebesar 102,39 ms dimana AODV asli unggul dalam hal ini, sedangkan pada AODV modifikasi dengan interval 10 detik menghasilkan perbedaan selisih *end-to-end delay* sebesar 5,8132 ms dimana AODV modifikasi dengan interval 10 detik unggul dalam hal ini dan pada AODV modifikasi dengan interval 15 detik menghasilkan perbedaan selilish *end-to-end delay* sebesar 47,83 ms dimana AODV asli unggul dalam hal ini.

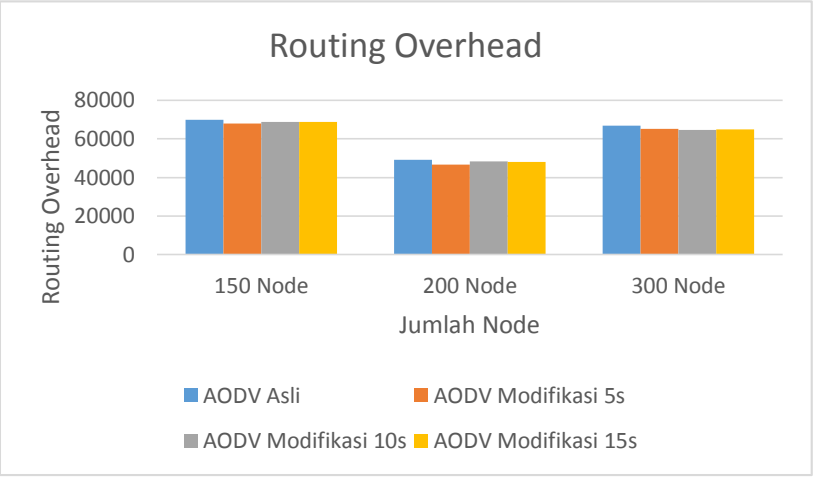
Pada lingkungan yang sedang dengan jumlah 200 *node* dan dibandingkan dengan AODV asli, AODV modifikasi dengan interval 5 detik menghasilkan perbedaan selilish *end-to-end delay* sebesar 621,9476 ms dimana AODV asli unggul dalam hal ini, sedangkan

pada AODV modifikasi dengan interval 10 detik menghasilkan perbedaan selisih *end-to-end delay* sebesar 506,3746 ms dimana AODV asli unggul dalam hal ini dan pada AODV modifikasi dengan interval 15 detik menghasilkan perbedaan selilish *end-to-end delay* sebesar 263,1221 ms dimana AODV asli unggul dalam hal ini.

Pada lingkungan yang padat dengan jumlah 300 *node* dan dibandingkan dengan AODV asli, AODV modifikasi dengan interval 5 detik menghasilkan perbedaan selilish *end-to-end delay* sebesar 621,9476 ms dimana AODV asli unggul dalam hal ini, sedangkan pada AODV modifikasi dengan interval 10 detik menghasilkan perbedaan selisih *end-to-end delay* sebesar 506,3746 ms dimana AODV asli unggul dalam hal ini dan pada AODV modifikasi dengan interval 15 detik menghasilkan perbedaan selilish *end-to-end delay* sebesar 263,1221 ms dimana AODV asli unggul dalam hal ini.

Jika ketiga lingkungan tersebut dibandingkan, memang pada lingkungan yang jarang, sedang, maupun padat tidak selalu lebih unggul dalam hal E2E. Routing protocol AODV tidak selalu lebih unggul daripada *routing protocol* AODV yang telah dimodifikasi kecuali pada AODV modifikasi dengan interval 5 detik pada lingkungan padat dengan jumlah 300 node. Hasil rata – rata E2E tidak dapat dianalisis karena terjadi fluktuasi dan tidak stabil. Hal ini dikarenakan waktu *delay* tergantung dari rata – rata waktu paket yang terkirim. Semakin banyak paket yang terkirim, maka semakin beragam *delay*nya.

5.2.1.3 Analisa *Routing Overhead* (RO)



Gambar 5.3 Grafik *Routing Overhead* skenario *grid*

Berdasarkan grafik pada Gambar 5.3, dapat dilihat bahwa *routing protocol* AODV yang telah dimodifikasi dengan interval 5 detik, 10 detik dan 15 detik dan juga *routing protocol* AODV asli mengalami perubahan yang fluktuatif. Pada lingkungan yang jarang dengan jumlah 150 *node* dan dibandingkan dengan AODV asli, AODV modifikasi dengan interval 5 detik menghasilkan perbedaan selisih RO sebesar 2083, dimana terjadi penurunan RO sebesar 2,98%, sedangkan pada AODV modifikasi dengan interval 10 detik menghasilkan perbedaan selisih RO sebesar 1073,8, dimana terjadi penurunan RO sebesar 1,53 % dan pada AODV modifikasi dengan interval 15 detik menghasilkan perbedaan selisih RO sebesar 1036, dimana terjadi penurunan RO sebesar 1,48%. Berdasarkan hasil tersebut, maka *routing protocol* AODV yang telah dimodifikasi dengan interval 5 detik unggul dalam hal RO tersebut karena menghasilkan RO yang lebih rendah dari AODV asli.

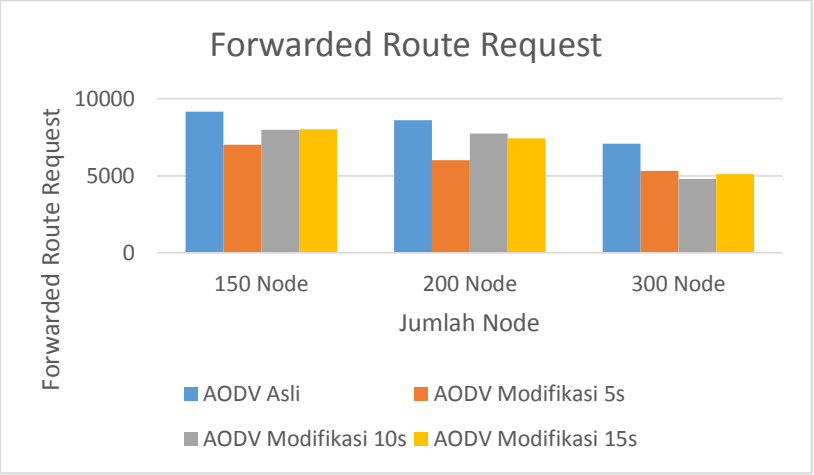
Pada lingkungan yang sedang dengan jumlah 200 *node* dan dibandingkan dengan AODV asli, AODV modifikasi dengan interval 5 detik menghasilkan perbedaan selisih RO sebesar 2675,7, dimana terjadi penurunan RO sebesar 5,43%, sedangkan pada AODV modifikasi dengan interval 10 detik menghasilkan perbedaan selisih RO sebesar 788,5, dimana terjadi penurunan RO sebesar 1,6 % dan pada AODV modifikasi dengan interval 15 detik menghasilkan perbedaan selisih RO sebesar 1167,5, dimana terjadi penurunan RO sebesar 2,37%. Berdasarkan hasil tersebut, maka *routing protocol* AODV yang telah dimodifikasi dengan interval 5 detik unggul dalam hal RO tersebut karena menghasilkan RO yang lebih rendah dari AODV asli.

Pada lingkungan yang padat dengan jumlah 300 *node* dan dibandingkan dengan AODV asli, AODV modifikasi dengan interval 5 detik menghasilkan perbedaan selisih RO sebesar 1761,8, dimana terjadi penurunan RO sebesar 2,63%, sedangkan pada AODV modifikasi dengan interval 10 detik menghasilkan perbedaan selisih RO sebesar 2296,6, dimana terjadi penurunan RO sebesar 3,42 % dan pada AODV modifikasi dengan interval 15 detik menghasilkan perbedaan selisih RO sebesar 2001,9, dimana terjadi penurunan RO sebesar 2,99%. Berdasarkan hasil tersebut, maka *routing protocol* AODV yang telah dimodifikasi dengan interval 10 detik unggul dalam hal RO tersebut karena menghasilkan RO yang lebih rendah dari AODV asli.

Jika ketiga lingkungan tersebut dibandingkan dapat dilihat bahwa dengan jumlah *node* yang lebih sedikit atau pada lingkungan dengan *node* yang jarang, menghasilkan RO yang lebih sedikit daripada di lingkungan dengan jumlah *node* yang sedang maupun yang padat baik untuk *routing protocol* AODV asli maupun *routing protocol* AODV yang telah dimodifikasi. AODV modifikasi dengan interval 5 detik mengungguli AODV asli, AODV modifikasi dengan interval 10 detik dan AODV modifikasi dengan interval 15 detik dengan nilai rata – rata penurunan RO pada AODV yang dimodifikasi adalah sebesar 3,68%. Dapat dilihat pula bahwa AODV yang telah

dimodifikasi dengan interval 5 detik menghasilkan RO yang lebih bagus atau dalam hal ini lebih rendah daripada AODV asli.

5.2.1.4 Analisa *Forwarded Route Request* (RREQ F)



Gambar 5.4 Grafik *Forwarded Route Request* skenario *grid*

Berdasarkan grafik pada Gambar 5.4, dapat dilihat bahwa *routing protocol* AODV yang telah dimodifikasi dan juga *routing protocol* asli mengalami perubahan *forwarded route request* (RREQ F) yang signifikan. Pada lingkungan yang jarang dengan jumlah 150 *node* dan dibandingkan dengan AODV asli, AODV modifikasi dengan interval 5 detik menghasilkan perbedaan selisih *forwarded route request* sebesar 2139, dimana terjadi penurunan RREQ F sebesar 23,3 %, sedangkan AODV modifikasi dengan 10 detik menghasilkan perbedaan selisih *forwarded route request* sebesar 1201,4, dimana terjadi penurunan RREQ F sebesar 13,09 % dan pada AODV modifikasi dengan interval 15 detik menghasilkan perbedaan selisih *forwarded route request* sebesar 1166, dimana terjadi penurunan RREQ F sebesar 12,71 %. Dari hasil tersebut *routing protocol* AODV yang telah dimodifikas dengan interval 5 detik

unggul dalam hal RREQ F tersebut karena menghasilkan RREQ F yang lebih rendah dari *routing protocol* AODV asli, AODV modifikasi dengan interval 10 detik dan AODV modifikasi dengan interval 15 detik.

Pada lingkungan yang sedang dengan jumlah 200 *node* dan dibandingkan dengan AODV asli, AODV modifikasi dengan interval 5 detik menghasilkan perbedaan selisih *forwarded route request* sebesar 2592,2, dimana terjadi penurunan RREQ F sebesar 30,05 %, sedangkan AODV modifikasi dengan 10 detik menghasilkan perbedaan selisih *forwarded route request* sebesar 871,6, dimana terjadi penurunan RREQ F sebesar 10,10 % dan pada AODV modifikasi dengan interval 15 detik menghasilkan perbedaan selisih *forwarded route request* sebesar 1178,1, dimana terjadi penurunan RREQ F sebesar 13,66 %. Dari hasil tersebut *routing protocol* AODV yang telah dimodifikasi dengan interval 5 detik unggul dalam hal RREQ F tersebut karena menghasilkan RREQ F yang lebih rendah dari *routing protocol* AODV asli, AODV modifikasi dengan interval 10 detik dan AODV modifikasi dengan interval 15 detik.

Pada lingkungan yang padat dengan jumlah 300 *node* dan dibandingkan dengan AODV asli, AODV modifikasi dengan interval 5 detik menghasilkan perbedaan selisih *forwarded route request* sebesar 1793,7, dimana terjadi penurunan RREQ F sebesar 25,24 %, sedangkan AODV modifikasi dengan 10 detik menghasilkan perbedaan selisih *forwarded route request* sebesar 2292,2, dimana terjadi penurunan RREQ F sebesar 32,35 % dan pada AODV modifikasi dengan interval 15 detik menghasilkan perbedaan selisih *forwarded route request* sebesar 1984,7, dimana terjadi penurunan RREQ F sebesar 27,92 %. Dari hasil tersebut *routing protocol* AODV yang telah dimodifikas dengan interval 10 detik unggul dalam hal RREQ F tersebut karena menghasilkan RREQ F yang lebih rendah dari *routing protocol* AODV asli, AODV modifikasi dengan interval 5 detik dan AODV modifikasi dengan interval 15 detik.

Jika ketiga lingkungan tersebut dibandingkan, dapat dilihat bahwa dengan jumlah *node* yang sedikit atau lingkungan jarang menghasilkan RREQ F yang lebih bagus atau lebih sedikit daripada

di lingkungan dengan jumlah *node* yang sedang maupun yang padat baik untuk *routing protocol* AODV asli maupun *routing protocol* AODV yang telah dimodifikasi. AODV modifikasi dengan interval 5 detik mengungguli AODV asli, AODV modifikasi dengan interval 10 detik dan AODV modifikasi dengan interval 15 detik dengan nilai rata – rata penurunan yang terjadi pada RREQ F adalah sebesar 26,19 %. Dapat dilihat pula bahwa AODV yang telah dimodifikasi dengan beberapa interval menghasilkan RREQ F yang lebih bagus atau dalam hal ini lebih rendah daripada RREQ F asli dengan jumlah selisih RREQ F yang cukup signifikan.

5.2.2 Hasil Uji Coba Skenario *Real*

Pengujian pada skenario *grid* digunakan untuk melihat perbandingan PDR, RO, RREQ F, dan E2E antara *routing protocol* AODV asli dan AODV yang telah dimodifikasi dalam pemilihan *node* yang dapat menerima paket *route request*.

Pengambilan data uji PDR, RO, RREQ F, dan E2E pada skenario *grid* dilakukan sebanyak 10 kali dengan skenario mobilitas *random* pada peta *grid* dengan luas area 1300 m x 1300 m dan *node* sebanyak 150 untuk lingkungan yang jarang, 200 *node* untuk lingkungan yang sedang, dan 300 *node* untuk lingkungan yang padat dilakukan pada kecepatan standar yaitu 20 m/s. Untuk uji coba setiap lingkungan menggunakan interval yang berbeda-beda untuk mencari nilai interval yang terbaik dari hasil skenario. Interval waktu yang digunakan adalah 5 detik, 10 detik dan 15 detik. Hasil analisis dapat dilihat pada Tabel 5.7, Tabel 5.8, Tabel 5.9 dan Tabel 5.10.

Tabel 5.7 Hasil rata rata perhitungan PDR pada skenario *real*

Jumlah	**AODV**	**AODV Modifikasi**			**Perbedaan**		
Node	**Asli**	**5s**	**10s**	**15s**	**5s**	**10s**	**15s**
150	0,818	0,873	0,885	0,858	0,055	0,068	0,040
200	0,886	0,857	0,868	0,899	0,029	0,017	0,013
300	0,700	0,794	0,641	0,830	0,094	0,059	0,129

Tabel 5.8 Hasil rata rata perhitungan E2E pada skenario *real*

Jumlah *Node*	**AODV Asli**	**AODV Modifikasi**			**Perbedaan**		
		5s	**10s**	**15s**	**5s**	**10s**	**15s**
150	487	354	982	326	133	494	161
200	815	885	883	701	70	68	114
300	483	359	814	2771	125	330	2288

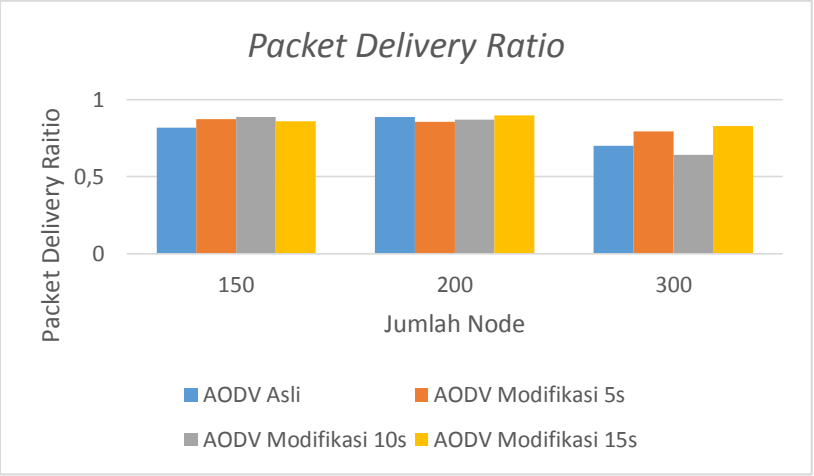
Tabel 5.9 Hasil rata rata perhitungan RO pada skenario *real*

Jumlah *Node*	**AODV Asli**	**AODV Modifikasi**			**Perbedaan**		
		5s	**10s**	**15s**	**5s**	**10s**	**15s**
150	40119	39209	39676	38987	909	442	1132
200	46920	47923	48273	48409	1004	1353	1490
300	61050	61612	61841	63237	562	791	2187

Tabel 5.10 Hasil rata rata perhitungan RREQ F pada skenario *real*

Jumlah *Node*	**AODV Asli**	**AODV Modifikasi**			**Perbedaan**		
		5s	**10s**	**15s**	**5s**	**10s**	**15s**
150	2541	1674	2105	1506	867	436	1035
200	6511	7422	7743	7872	911	1232	1362
300	1327	1820	2084	3472	493	757	2145

5.2.2.1 Analisa *Packet Delivery Ratio* (PDR)



Gambar 5.5 Grafik rata rata PDR skenario *real*

Berdasarkan grafik pada Gambar 5.5, *routing protocol* AODV asli dan AODV yang telah dimodifikasi baik dengan interval 5 detik, 10 detik dan 15 detik perubahan yang fluktuaktif. Pada lingkungan yang jarang dengan *node* berjumlah 150 dan dibandingkan dengan hasil PDR AODV asli, AODV modifikasi dengan interval 5 detik menghasilkan perbedaan selisih PDR sebesar 0,05542, dimana terjadi kenaikan sebesar 6,77 %, sedangkan pada AODV modifikasi dengan interval 10 detik menghasilkan perbedaan selisih PDR sebesar 0,06766, dimana terjadi kenaikan sebesar 8,27 % dan pada AODV modifikasi dengan interval 15 detik menghasilkan perbedaan selisih PDR sebesar 0,04048, dimana terjadi kenaikan sebesar 4,95 %. Berdasarkan hal tersebut, routing protocol AODV modifikasi dengan interval 10 detik berhasil mengungguli *routing protocol* AODV asli, *routing protocol* ADOV modifikasi dengan interval 5 detik *dan routing protocol* AODV dengan interval 15 detik.

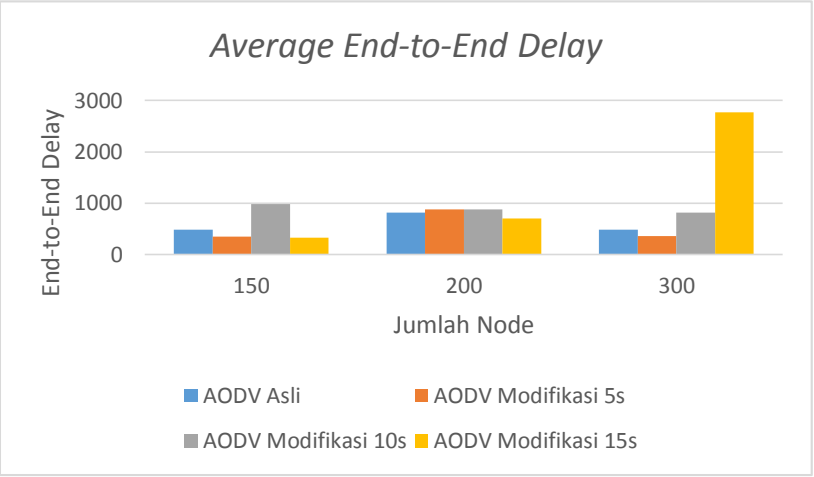
Pada lingkungan yang sedang dengan *node* berjumlah 200 dan dibandingkan dengan hasil PDR AODV asli, AODV modifikasi

dengan interval 5 detik menghasilkan perbedaan selisih PDR sebesar 0,02918, dimana terjadi penurunan sebesar 13,4 %, sedangkan pada AODV modifikasi dengan interval 10 detik menghasilkan perbedaan selisih PDR sebesar 0,01744, dimana terjadi penurunan sebesar 3,29 % dan pada AODV modifikasi dengan interval 15 detik menghasilkan perbedaan selisih PDR sebesar 0,0131, dimana terjadi kenaikan sebesar 1,48 %. Berdasarkan hal tersebut, routing protocol AODV modifikasi dengan interval 15 detik berhasil mengungguli *routing protocol* AODV asli, *routing protocol* ADOV modifikasi dengan interval 5 detik *dan routing protocol* AODV dengan interval 10 detik.

Pada lingkungan yang padat dengan *node* berjumlah 300 dan dibandingkan dengan hasil PDR AODV asli, AODV modifikasi dengan interval 5 detik menghasilkan perbedaan selisih PDR sebesar 0,09386, dimana terjadi kenaikan sebesar 13,4 %, sedangkan pada AODV modifikasi dengan interval 10 detik menghasilkan perbedaan selisih PDR sebesar 0,05922, dimana terjadi penurunan sebesar 8,45 % dan pada AODV modifikasi dengan interval 15 detik menghasilkan perbedaan selisih PDR sebesar 0,12944, dimana terjadi kenaikan sebesar 0,06 %. Berdasarkan hal tersebut, routing protocol AODV modifikasi dengan interval 15 detik berhasil mengungguli *routing protocol* AODV asli, *routing protocol* ADOV modifikasi dengan interval 5 detik *dan routing protocol* AODV dengan interval 10 detik.

Berdasarkan hasil simulasi pada ketiga lingkungan tersebut, dapat dilihat bahwa dengan menggunakan AODV modifikasi dengan interval 15 detik menghasilkan PDR yang lebih baik daripada menggunakan AODV asli, AODV modifikasi dengan interval 5 detik dan AODV modifikasi dengan interval 10 detik. Nilai rata kenaikan PDR pada skenario grid dengan menggunakan AODV modifikikasi dengan interval 15 detik adalah sebesar 8,3 %.

5.2.2.2 Analisa *End-to-End Delay* (E2E)



Gambar 5.6 Grafik E2E pada skenario *real*

Berdasarkan grafik pada Gambar 5.6, dapat dilihat bahwa rata-rata E2E antara *routing protocol* AODV asli dan AODV yang telah dimodifikasi dengan interval 5 detik, 10 detik dan 15 detik mengalami perubahan yang fluktuatif. Pada lingkungan yang jarang dengan jumlah 150 *node* dan dibandingkan dengan AODV asli, AODV modifikasi dengan interval 5 detik menghasilkan perbedaan selilish *end-to-end delay* sebesar 132,843 ms dimana AODV asli unggul dalam hal ini, sedangkan pada AODV modifikasi dengan interval 10 detik menghasilkan perbedaan selisih *end-to-end delay* sebesar 494,3304 ms dimana AODV modifikasi dengan interval 10 detik unggul dalam hal ini dan pada AODV modifikasi dengan interval 15 detik menghasilkan perbedaan selilish *end-to-end delay* sebesar 161,183 ms dimana AODV asli unggul dalam hal ini.

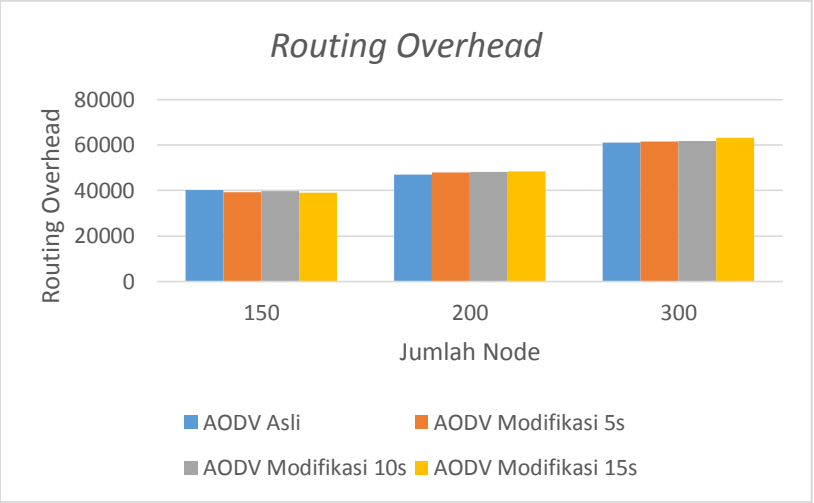
Pada lingkungan yang sedang dengan jumlah 200 *node* dan dibandingkan dengan AODV asli, AODV modifikasi dengan interval 5 detik menghasilkan perbedaan selilish *end-to-end delay* sebesar

70,0932 ms dimana AODV asli unggul dalam hal ini, sedangkan pada AODV modifikasi dengan interval 10 detik menghasilkan perbedaan selisih *end-to-end delay* sebesar 68,1022 ms dimana AODV asli unggul dalam hal ini dan pada AODV modifikasi dengan interval 15 detik menghasilkan perbedaan selilish *end-to-end delay* sebesar 113,8398 ms dimana AODV modifikasi dengan interval 15 detik dalam hal ini.

Pada lingkungan yang padat dengan jumlah 300 *node* dan dibandingkan dengan AODV asli, AODV modifikasi dengan interval 5 detik menghasilkan perbedaan selilish *end-to-end delay* sebesar 124,64852 ms dimana AODV modifikasi dengan interval 5 detik unggul dalam hal ini, sedangkan pada AODV modifikasi dengan interval 10 detik menghasilkan perbedaan selisih *end-to-end delay* sebesar 330,43026 ms dimana AODV asli unggul dalam hal ini dan pada AODV modifikasi dengan interval 15 detik menghasilkan perbedaan selilish *end-to-end delay* sebesar 2288,02 ms dimana AODV asli unggul dalam hal ini.

Jika ketiga lingkungan tersebut dibandingkan, memang pada lingkungan yang jarang, sedang, maupun padat tidak selalu lebih unggul dalam hal E2E. Routing protocol AODV tidak selalu lebih unggul daripada *routing protocol* AODV yang telah dimodifikasi. Hasil rata – rata E2E tidak dapat dianalisis karena terjadi fluktuasi dan tidak stabil. Hal ini dikarenakan waktu *delay* tergantung dari rata – rata waktu paket yang terkirim. Semakin banyak paket yang terkirim, maka semakin beragam *delay*nya.

5.2.2.3 Analisa *Routing Overhead* (RO)



Gambar 5.7 Grafik rata rata RO skenario *real*

Berdakan pada grafik Gambar 5.7, dapat dilihat bahwa *routing protocol* AODV yang telah dimodifikasi dengan interval 5 detik, 10 detik dan 15 detik dan juga *routing protocol* AODV asli mengalami perubahan yang fluktuatif. Pada lingkungan yang jarang dengan jumlah 150 *node* dan dibandingkan dengan AODV asli, AODV modifikasi dengan interval 5 detik menghasilkan perbedaan selisih RO sebesar 909,4, dimana terjadi penurunan RO sebesar 2,26%, sedangkan pada AODV modifikasi dengan interval 10 detik menghasilkan perbedaan selisih RO sebesar 442,4, dimana terjadi penurunan RO sebesar 1,1 % dan pada AODV modifikasi dengan interval 15 detik menghasilkan perbedaan selisih RO sebesar 1132, dimana terjadi penurunan RO sebesar 2,82%. Berdasarkan hasil tersebut, maka *routing protocol* AODV yang telah dimodifikasi dengan interval 15 detik unggul dalam hal RO tersebut karena menghasilkan RO yang lebih rendah dari AODV asli.

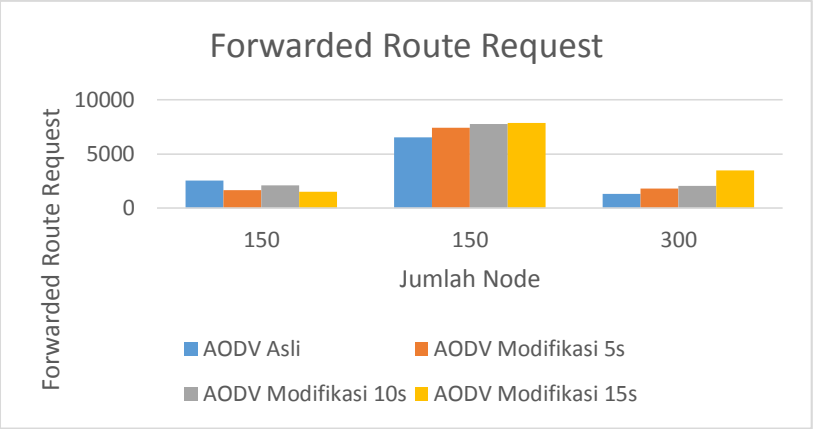
Pada lingkungan yang sedang dengan jumlah 200 *node* dan dibandingkan dengan AODV asli, AODV modifikasi dengan interval 5 detik menghasilkan perbedaan selisih RO sebesar 1003,8, dimana terjadi kenaikan RO sebesar 2,14%, sedangkan pada AODV modifikasi dengan interval 10 detik menghasilkan perbedaan selisih RO sebesar 1353,4, dimana terjadi kenaikan RO sebesar 2,88 % dan pada AODV modifikasi dengan interval 15 detik menghasilkan perbedaan selisih RO sebesar 1489,6, dimana terjadi kenaikan RO sebesar 3,17%. Berdasarkan hasil tersebut, maka *routing protocol* AODV yang telah dimodifikasi tidak ada yang mengungguli AODV asli tersebut karena menghasilkan RO yang lebih tinggi dari pada AODV asli.

Pada lingkungan yang padat dengan jumlah 300 *node* dan dibandingkan dengan AODV asli, AODV modifikasi dengan interval 5 detik menghasilkan perbedaan selisih RO sebesar 562,2, dimana terjadi kenaikan RO sebesar 0,92%, sedangkan pada AODV modifikasi dengan interval 10 detik menghasilkan perbedaan selisih RO sebesar 791, dimana terjadi kenaikan RO sebesar 1,29 % dan pada AODV modifikasi dengan interval 15 detik menghasilkan perbedaan selisih RO sebesar 2186,8, dimana terjadi kenaikan RO sebesar 3,58%. Berdasarkan hasil tersebut, maka *routing protocol* AODV yang telah dimodifikasi tidak ada yang mengungguli AODV asli tersebut karena menghasilkan RO yang lebih tinggi dari pada AODV asli.

Jika ketiga lingkungan tersebut dibandingkan dapat dilihat bahwa dengan jumlah *node* yang lebih sedikit atau pada lingkungan dengan *node* yang jarang, menghasilkan RO yang lebih sedikit daripada di lingkungan dengan jumlah *node* yang sedang maupun yang padat baik untuk *routing protocol* AODV asli maupun *routing protocol* AODV yang telah dimodifikasi. AODV yang dimodifikasi dengan beberapa interval hanya mengungguli AODV asli pada lingkungan jarang, yakni AODV modifikasi dengan interval 15 detik dengan nilai penurunan RO pada AODV yang dimodifikasi adalah sebesar 2,82%. Dapat dilihat pula bahwa AODV yang telah dimodifikasi dengan interval 5 detik menghasilkan RO yang lebih

bagus atau dalam hal ini lebih rendah daripada AODV asli. Sedangkan AODV modifikasi dengan beberapa interval menghasilkan RO yang lebih besar daripada AODV asli pada lingkungan sedang dan padat.

5.2.2.4 Analisa *Forwarded Route Request* (RREQ F)



Gambar 5.8 Grafik rata rata RREQ F skenario *real*

Berdasarkan grafik pada Gambar 5.8, dapat dilihat bahwa *routing protocol* AODV yang telah dimodifikasi dan juga *routing protocol* asli mengalami perubahan *forwarded route request* (RREQ F) yang signifikan. Pada lingkungan yang jarang dengan jumlah 150 *node* dan dibandingkan dengan AODV asli, AODV modifikasi dengan interval 5 detik menghasilkan perbedaan selisih *forwarded route request* sebesar 866,6, dimana terjadi penurunan RREQ F sebesar 34,11 %, sedangkan AODV modifikasi dengan 10 detik menghasilkan perbedaan selisih *forwarded route request* sebesar 436, dimana terjadi penurunan RREQ F sebesar 17,16 % dan pada AODV modifikasi dengan interval 15 detik menghasilkan perbedaan selisih *forwarded route request* sebesar 1034,8, dimana terjadi penurunan RREQ F sebesar 40,72 %. Dari hasil tersebut *routing protocol* AODV yang telah dimodifikas dengan interval 15 detik unggul dalam hal RREQ F tersebut karena menghasilkan RREQ F yang lebih rendah

dari *routing protocol* AODV asli, AODV modifikasi dengan interval 5 detik dan AODV modifikasi dengan interval 10 detik.

Pada lingkungan yang sedang dengan jumlah 200 *node* dan dibandingkan dengan AODV asli, AODV modifikasi dengan interval 5 detik menghasilkan perbedaan selisih *forwarded route request* sebesar 911, dimana terjadi kenaikan RREQ F sebesar 14 %, sedangkan AODV modifikasi dengan 10 detik menghasilkan perbedaan selisih *forwarded route request* sebesar 1231,8, dimana terjadi penurunan RREQ F sebesar 18,92 % dan pada AODV modifikasi dengan interval 15 detik menghasilkan perbedaan selisih *forwarded route request* sebesar 1361,6, dimana terjadi penurunan RREQ F sebesar 20,91 %. Dari hasil tersebut *routing protocol* AODV yang telah dimodifikasi dengan beberapa interval tidak ada yang unggul dalam hal RREQ F tersebut karena menghasilkan RREQ F yang lebih tinggi dari *routing protocol* AODV asli.

Pada lingkungan yang padat dengan jumlah 300 *node* dan dibandingkan dengan AODV asli, AODV modifikasi dengan interval 5 detik menghasilkan perbedaan selisih *forwarded route request* sebesar 492,6, dimana terjadi kenaikan RREQ F sebesar 37,11 %, sedangkan AODV modifikasi dengan 10 detik menghasilkan perbedaan selisih *forwarded route request* sebesar 757, dimana terjadi kenaikan RREQ F sebesar 57,04 % dan pada AODV modifikasi dengan interval 15 detik menghasilkan perbedaan selisih *forwarded route request* sebesar 2145,2, dimana terjadi kenaikan RREQ F sebesar 161,63 %. Dari hasil tersebut *routing protocol* AODV yang telah dimodifikasi dengan beberapa interval tidak ada yang unggul dalam hal RREQ F tersebut karena menghasilkan RREQ F yang lebih tinggi dari *routing protocol* AODV asli.

Jika ketiga lingkungan tersebut dibandingkan, dapat dilihat bahwa dengan jumlah *node* yang sedikit atau lingkungan jarang menghasilkan RREQ F yang lebih bagus atau lebih sedikit daripada di lingkungan dengan jumlah *node* yang sedang maupun yang padat baik untuk *routing protocol* AODV asli maupun *routing protocol* AODV yang telah dimodifikasi. AODV yang dimodifikasi dengan beberapa interval hanya mengungguli AODV asli pada lingkungan

jarang, yakni AODV modifikasi dengan interval 15 detik mengungguli AODV asli, AODV modifikasi dengan interval 5 detik dan AODV modifikasi dengan interval 10 detik dengan nilai penurunan yang terjadi pada RREQ F adalah sebesar 40.72 %. Sedangkan AODV modifikasi dengan beberapa interval menghasilkan RREQ F yang lebih besar daripada AODV asli pada lingkungan sedang dan padat.

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 dari hasil uji coba dan evaluasi Tugas Akhir ini adalah sebagai berikut:

1. AODV yang dimodifikasi sudah berhasil membatasi jumlah *forwarding node* yang bertugas untuk *rebroadcast* paket RREQ.
2. Dampak pembatasan *forwarding node* terhadap performa protokol AODV pada skenario *grid* adalah rata – rata kenaikan *Packet Delivery Ratio* (PDR) sebesar 2,61% dan rata – rata penurunan *Routing Overhead* (RO) sebesar 3,41%.
3. Dampak pembatasan *forwarding node* terhadap performa protokol AODV pada skenario *real* adalah rata – *rata* kenaikan *Packet Delivery Ratio* (PDR) sebesar 3,92% dan rata – rata penurunan *Routing Overhead* (RO) sebesar 0,86%.
4. Interval terbaik yang digunakan dalam membatasi jumlah *forwarding node* secara adaptif dalam proses *rebroadcast* paket RREQ pada skenario *grid* adalah 5 detik dengan hasil rata – rata kenaikan *Packet Delivery Ratio* (PDR) sebesar 6,26% dan rata – rata penurunan *Routing Overhead* (RO) sebesar 2,68%.
5. Interval terbaik yang digunakan dalam membatasi jumlah *forwarding node* secara adaptif dalam proses *rebroadcast* paket RREQ pada skenario *real* adalah 5 detik dengan hasil rata – rata kenaikan *Packet Delivery Ratio* (PDR) sebesar 2,25% dan rata – rata kenaikan *Routing Overhead* (RO) sebesar 0,26%.

6.2 Saran

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

1. Lebih banyak uji coba yang dilakukan untuk mendapatkan hasil yang lebih akurat.
2. Mencari metode lain dalam perhitungan threshold sehingga membuat hasil kian variatif untuk di analisis.
3. Menambahkan aspek lain untuk melakukan pembatasan *forwarding node* yang meneruskan paket RREQ seperti arah, kecepatan, dan energi.

DAFTAR PUSTAKA

[1] "VANET - Vehicle Ad hoc Network," [Online]. Available: http://comp.ist.utl.pt/~rmr/WSN/CaseStudies2007-no/WSN_Transportation/. [Diakses 15 Mei 2018].

[2] J. Harri, F. Filali dan C. Bonnet, "Mobility Models for Vehicular Ad Hoc Network: A Survey and Taxonomy," IEEE, Florida, 2009.

[3] R. Brendha dan V. S. J. Prakash, "A Survey on Routing Protocols for Vehicular Ad hoc Networks," IEEE, Coimbatore, 2017.

[4] R. F. Sari dan A. Syarif, "Analisis Kinerja Protokol Routing Ad Hoc On-Demand Distance Vector (AODV) pada Jaringan Ad Hoc," 22 October 2010.

[5] P. Meeneghan dan D. Delaney, "P. Meeneghan dan D. Delaney, "An Introduction to NS Nam and OTcl scripting," April 2004.

[6] "openstreetmap," [Online]. Available: https://www.openstreetmap.org/. [Diakses 20 May 2018].

[7] "JOSM," [Online]. Available: https://josm.openstreetmap.de/. [Diakses 20 May 2018].

[8] D. Krajzewics, J. Erdmann, M. Behrisch dan L. Bieker, "Recent Development and Application of SUMO," *International Journal On Advances in Systems and Measurements,* p. 128, December 2012.

[9] "AWK," [Online]. Available: http://tldp.org/LDP/abs/html/awk.html. [Diakses 20 May 2018].

(Halaman ini sengaja dikosongkan)

LAMPIRAN

A.1 Kode Fungsi CountThreshold()

```
void CountThreshold::handle(Event *)
{
  double now =
Scheduler::instance().clock(); // get the
time
  double interval = 5.0;

  int run = now / interval;
  if (masuk[run]==0) masuk[run]=0;

  if(now > 0.000000 && masuk[run]==0)
  {
    masuk[run]=100;
    for(int i=0;i<nodes_count;i++)
    {
     for(int j=(i+1);j<nodes_count;j++)
      {

if(count_neighbour[i]>count_neighbour[j])
        {
         int tmp;
         tmp=count_neighbour[i];

count_neighbour[i]=count_neighbour[j];
         count_neighbour[j]=tmp;
        }
      }
    }
    for(int i=0;i<nodes_count;i++)
    {
     count_mode[i]=0;
      for(int j=0;j<nodes_count;j++)
```

```
      {

if(count_neighbour[i]==count_neighbour[j])
          {
           count_mode[i]++;
          }
      }
    }
    for(int i=0;i<nodes_count;i++)
    {
      if(count_mode[i]>th )
      {
        th=count_mode[i];
      }
    }


    old_th = th;
  }


  Scheduler::instance().schedule(this,
&intr, interval);
}
```

A.2 Kode Fungsi nb_insert()

```
void AODV::nb_insert(nsaddr_t id)
{
  count_neighbour[index]+=1;
  AODV_Neighbor *nb = new AODV_Neighbor(id);

  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_remove()

```
void AODV::nb_delete(nsaddr_t id)
{
  count_neighbour[index]-=1;

  AODV_Neighbor *nb = nbhead.lh_first;

  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)      1300;
set opt(y)      1300;
set val(ifqlen) 1000;
set val(nn)     60;
set val(seed)   1.0;
set val(adhocRouting)   AODV;
set val(stop)   200;
set val(cp)      "cbr1.txt";
set val(sc)      "scen1modif.txt";

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 Deliver 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 Forwarded Route Request

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

(Halaman ini sengaja dikosongkan)

BIODATA PENULIS

Rizky Fenaldo Maulana, lahir di Denpasar, 14 Juli 1997. Penulis adalah anak keempat dari empat bersaudara. Penulis menempuh pendidikan sekolah dasar di SDN Dinoyo 2 Malang lalu melanjutkan pendidikan sekolah menengah pertama di SMPN 4 Malang dan penulis menempuh pendidikan menengah atas di SMA Negeri 1 Malang. Selanjutnya penulis melanjutkan pendidikan sarjana di Departemen Informatika, Fakultas Teknologi Informasi dan Komunikasi, Institut Teknologi Sepuluh Nopember Surabaya. Selama kuliah, penulis aktif dalam berbagai organisasi baik tingkat jurusan maupun universitas.

Dalam menyelesaikan pendidikan S1, penulis mengambil bidang minat Arsitektur dan Jaringan Komputer (AJK). Sebagai mahasiswa, penulis berperan aktif dalam beberapa organisasi kampus seperti staf Dalam Negeri Himpunan Mahasiswa Teknik-Computer (HMTC) ITS, staf ahli Himpunan Mahasiswa Teknik-Computer (HMTC) ITS , staf Kementerian Pemuda dan Kebangsaan BEM ITS dan Gerigi 2015. Selain itu, penulis juga menjadi staf REEVA SCHEMATICS 2015 dan koordinator 1 REEVA pada acara SCHEMATICS 2016. Penulis pernah melakukan kerja praktik di PT. Telekomunikasi Indonesia Juni – Agustus 2017 dan membuat aplikasi berbasis web Monitoring FO Migrations. Penulis dapat dihubungi melalui nomor *handphone*: 081230777099 atau *email*: rizkyfenaldoo@gmail.com.