TUGAS AKHIR – KI141502

MODIFIKASI PEMILIHAN *FORWARDING NODE* PADA *AD-HOC ON-DEMAND DISTANCE VECTOR* (AODV) BERDASARKAN TINGKAT KEPADATAN *NEIGHBORING NODE* DAN *DYNAMIC CONNECTIVITY FACTOR* (DCF) DI VANETS

TRASTIAN SATRIA WIBOWO
NRP 05111440000016

Dosen Pembimbing I
Prof. Ir. Supeno Djanali, M.Sc., Ph.D.

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

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

(Halaman ini sengaja dikosongkan)

TUGAS AKHIR – KI141502

MODIFIKASI PEMILIHAN *FORWARDING NODE* PADA *AD-HOC ON-DEMAND DISTANCE VECTOR* (AODV) BERDASARKAN TINGKAT KEPADATAN *NEIGHBORING NODE* DAN *DYNAMIC CONNECTIVITY FACTOR* (DCF) DI VANETS

TRASTIAN SATRIA WIBOWO
NRP 05111440000016

Dosen Pembimbing I
Prof. Ir. Supeno Djanali, M.Sc., Ph.D.

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

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

(Halaman ini sengaja dikosongkan)

UNDERGRADUATE THESES – KI141502

MODIFICATION OF FORWARDING NODE SELECTION FOR AD-HOC ON-DEMAND DISTANCE VECTOR (AODV) BASED ON NEIGHBORING NODE DENSITY AND DYNAMIC CONNECTIVITY FACTOR (DCF) IN VANETS

TRASTIAN SATRIA WIBOWO
NRP 05111440000016

First Advisor
Prof. Ir. Supeno Djanali, M.Sc., Ph.D.

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

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

(Halaman ini sengaja dikosongkan)

LEMBAR PENGESAHAN

MODIFIKASI PEMILIHAN *FORWARDING NODE* PADA *AD-HOC ON-DEMAND DISTANCE VECTOR* (AODV) BERDASARKAN TINGKAT KEPADATAN *NEIGHBORING NODE* DAN *DYNAMIC CONNECTIVITY FACTOR* (DCF) 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:
TRASTIAN SATRIA WIBOWO
NRP: 05111440000016

Disetujui oleh Pembimbing Tugas Akhir:

1. Prof. Ir. Supeno Djanali, M.Sc., Ph.D. .....
   (NIP. 194806191973011001) (Pembimbing 1)

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

SURABAYA
JULI, 2018

(Halaman ini sengaja dikosongkan)

MODIFIKASI PEMILIHAN *FORWARDING NODE* PADA *AD-HOC ON-DEMAND DISTANCE VECTOR* (AODV) BERDASARKAN TINGKAT KEPADATAN *NEIGHBORING NODE* DAN *DYNAMIC CONNECTIVITY FACTOR* (DCF) DI VANETS

Nama Mahasiswa	**: Trastian Satria Wibowo**
NRP	**: 05111440000016**
Departemen	**: Informatika FTIK-ITS**
Dosen Pembimbing 1	**: Prof. Ir. Supeno Djanali, M.Sc, Ph.D.**
Dosen Pembimbing 2	**: Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.**

Abstrak

Vehicular Ad hoc Networks (VANETs) merupakan pengembangan dari *Mobile Ad hoc Networks* (MANETs), dimana *node* memiliki karakteristik dengan mobilitas yang 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 *routing protocol* yang hanya akan membuat rute ketika ada paket yang ingin dikirim.

Modifikasi akan dilakukan pada proses pengiriman paket *route request* (RREQ), yaitu dengan cara mengeliminasi jumlah *neighbor node* yang bertugas mengirim ulang (*rebroadcast*) paket RREQ. Hal ini dilakukan dengan cara melihat jumlah *node* tetangga dari tiap *node* tersebut, lalu jumlah *node* tersebut akan diolah dengan menjadi sebuah variabel *Forwarding Probability* (FP) yang dihitung dengan persamaan *Dynamic Connectivity Factor* (DCF). Jika paket RREQ sampai pada *node* tujuan, maka *node* tujuan akan mengirim paket *route reply* (RREP) ke *node* asal. Lalu rute untuk pengiriman paket akan terbentuk. Modifikasi yang dilakukan akan menghasilkan *routing overhead* dan *forwarded*

route request yang lebih kecil daripada *routing protocol* AODV yang asli.

Pada tugas akhir ini, performa pada *routing protocol* AODV-DCF menghasilkan performa yang lebih bagus. Dibuktikan dengan skenario *real* yang menghasilkan peningkatan rata-rata *packet delivery ratio* sebesar 7,35%, rata-rata penurunan *routing overhead* sebesar 3,44%, dan juga rata-rata penurunan *forwarded route request* sebesar 57,53%.

Kata kunci: AODV, *Dynamic Connectivity Factor, Forwarding Node, Forwarding Probability,* VANETs

MODIFICATION OF FORWARDING *NODE* SELECTION FOR AD-HOC ON-DEMAND DISTANCE VECTOR (AODV) BASED ON NEIGHBORING *NODE* DENSITY AND DYNAMIC CONNECTIVITY FACTOR (DCF) IN VANETS

Student's Name	**:**	**Trastian Satria Wibowo**
Student's ID	**:**	**05111440000016**
Department	**:**	**Informatika FTIK-ITS**
First Advisor	**:**	**Prof. Ir. Supeno Djanali, M.Sc, Ph.D.**
Second Advisor	**:**	**Dr.Eng. Radityo Anggoro, S.Kom, M.Sc.**

Abstract

Vehicular Ad hoc Networks (VANETs) are development of Mobile Ad hoc Networks (MANETs), where the node was characterized by high mobility and limited in its movement pattern. There are many routing protocol on VANETs, one of them is Ad-Hoc On-Demand Distance Vector (AODV). AODV is classified to reactive routing protocol. A routing protocol that only make the route when there are pakets to be sent.

In this thesis a solution have been made by modifying the delivery of route request (RREQ) process, which is eliminating the number of neighboring node that assigned to rebroadcast RREQ packet. By looking at the number of neighbor node of each node, and then the number of neighbor nodes processed to become a Forwarding Probability (FP) by Dynamic Connectivity Factor (DCF) Method. If the RREQ packet has reached the destination node, the destination node will send back a route reply (RREP) packet to the source node and the route will be formed on cache.

The performance of the modified protocol has a better result than the original routing protocol AODV. It is proven that in real scenario, there is an enhancement of average packet delivery ratio by 7.35%*, reduction of average routing overhead by* 3.44%*, and reduction of average forwarded route request by* 57.53%*.*

Keyword: **AODV*****, Dynamic Connectivity Factor, Forwarding Node, Forwarding Probability,*** **VANETs**

KATA PENGANTAR

بِسۡمِ ٱللَّهِ ٱلرَّحۡمَٰنِ ٱلرَّحِيمِ

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

***"Modifikasi Pemilihan Forwarding Node Pada Ad-Hoc On-Demand Distance Vector (AODV) Berdasarkan Tingkat Kepadatan Neighboring Node Dan Dynamic Connectivity Factor (DCF) Di VANETs*"**

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

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

1. Allah SWT.
2. Keluarga penulis (Mama, Bapak, Pakdhe Badrun, Budhe Karmi, Mbak Dewi, Via, Mas Bima, Maryam, Mba Ti, Noval, Mas Rio, Ikan-ikan Koi, dan keluarga penulis yang lain) yang selalu memberikan dukungan baik berupa doa, moral, dan material yang tak terhingga kepada penulis, sehingga penulis dapat menyelesaikan Tugas Akhir ini.
3. Bapak Dr.Eng. Radityo Anggoro, S.Kom., M.Sc. dan Bapak Prof. Ir. Supeno Djanali, M.Sc, Ph.D. selaku dosen pembimbing penulis yang telah membimbing, memberikan nasihat, dan memotivasi penulis sehingga penulis dapat menyelesaikan Tugas Akhir ini.
4. Bapak Dr.Eng. Darlis Herumurti, S.Kom., M.Kom. selaku kepala Departemen Informatika ITS.

5. Teman-teman dari Warkop x Jojoran (Afif, Faris, Aldo, Nezar, Sani, Botak, Lian, Buyung, Paul, Pentol, Oing, Dyo, Hilman, Fikry, Hamka, Faiq, Glleen, Cimeng, Hari, Hakim, Upil, Rage, Fito, Anandi, Amik, Bajikas, Sekbay, Penyok, Kevin, Riefqy, Akhyar, Nanda, Dito, Rian, Nanda, dan Petrus) yang selalu memberikan semangat, selalu memberikan hiburan kepada penulis, teman-teman yang sering diajak nongkrong, teman-teman yang bisa diajak untuk bertukar pikiran dan pendapat, dan juga menjadi keluarga baru penulis saat berkuliah di Departemen Informatika ITS.
6. Teman-teman dari keluarga besar laboratorium ALPRO, laboratorium AJK, yang telah menemani, memberi semangat, memotivasi, memberikan doa, serta memberikan hiburan di kala penulis sedang jenuh saat pengerjaan Tugas Akhir ini.
7. Teman-teman penulis di angkatan 2014 (Vivi, Kania, Bebet, Nay, Raras, Pina, Anindita, Nafia, Rani, Sabila, dan teman-teman 2014 yang lain) yang telah menemani dan berjuang bersama penulis selama berkuliah di Departemen Informatika ITS.
8. Teman-teman kelas SPAGET1 (Anggun, Dwine, Dina, Mira, Gilang, Cokel, Helmy, Tiara, Konci, Aris, Mayang, Farida, Andre, Dicky, Rani, Ridha, Ben, Rossi) yang telah menemani dan mau direpoti untuk datang bahkan ikut menyelenggarakan acara saat penulis sedang kangen dengan kenangan masa SMA.
9. Teman-teman LABKOM SMALA (Barikh, Mas Sat, Mbah, Tony, Helmy, Simo, Alpin, Apip, Tika, Nosa, Ayam, Bayu, Sakaw) yang telah menghibur saya dengan bermain DotA 2 disaat penulis sedang jenuh saat pengerjaan Tugas Akhir ini.
10. Sahabat BANANA (Riyan, Manda, Tanti, Putri, Rena, Cici, Vida, Karti) yang telah menemani saat penulis sedang suntuk dan berada di Kota Mojokerto.

11. Yang terakhir untuk orang-orang yang tidak dapat disebutkan oleh penulis, dan yang sudah membaca buku Tugas Akhir ini.

Penulis telah berusaha sebaik-baiknya dalam menyusun tugas akhir ini, namun penulis mohon maaf apabila terdapat kekurangan, kesalahan maupun kelalaian yang telah penulis lakukan. Kritik dan saran yang membangun dapat disampaikan sebagai bahan perbaikan selanjutnya. Semoga untuk semua orang dan terutama yang sudah membaca buku ini, dimanapun, kapanpun, dan bagaimanapun kondisinya penulis doakan selalu bahagia. Aamiin.

Surabaya, 07 Juni 2018

Trastian Satria Wibowo

(Halaman ini sengaja dikosongkan)

DAFTAR ISI

Abstrak ..... **vii**
Abstract ..... **ix**
KATA PENGANTAR ..... **xi**
DAFTAR ISI ..... **xv**
DAFTAR GAMBAR ..... **xix**
DAFTAR TABEL ..... **xxi**
BAB I PENDAHULUAN **1**
1.1 Latar Belakang 1
1.2 Rumusan Masalah 2
1.3 Batasan Permasalahan 3
1.4 Tujuan 3
1.5 Manfaat 3
1.6 Metodologi 3
1.6.1 Penyusunan Proposal Tugas Akhir 4
1.6.2 Studi Literatur 4
1.6.3 Implementasi Sistem 4
1.6.4 Pengujian dan Evaluasi 5
1.6.5 Penyusunan Buku 5
1.7 Sistematika Penulisan Laporan 5
BAB II TINJAUAN PUSTAKA **7**
2.1 VANETs 7
2.2 *Ad-hoc On-demand Distance Vector (AODV)* 8
2.3 *Network Simulator-2* (NS-2) 10
2.3.1 Instalasi 11
2.3.2 *Trace File* 11
2.4 OpenStreetMap 13
2.5 Java OpenStreetMap Editor (JOSM) 13
2.6 Simulation of Urban Mobility (SUMO) 14
2.7 AWK 14
2.8 *Dynamic Connectivity Factor* (DCF) 15
BAB III PERANCANGAN **21**
3.1 Deskripsi Umum 21
3.2 Perancangan Skenario Mobilitas 23

3.2.1 Perancangan Skenario Grid.....24
3.2.2 Perancangan Skenario Real.....25
3.3 Analisis dan Perancangan Modifikasi *Routing Protocol* AODV 26
3.3.1 Perancangan Penghitungan Jumlah *Node* Tetangga dan Membuat Daftar *Node* Tetangga untuk Setiap *Node* 26
3.3.2 Perancangan Penghitungan *Forwarding Probability* (FP) 27
3.3.3 Perancangan Pemilihan *Forwarding Node* 28
3.4 Perancangan Simulasi pada NS-2.....28
3.5 Perancangan Metrik Analisis.....29
3.5.1 *Packet Delivery Ratio* (PDR).....29
3.5.2 Rata-rata *End-to-End Delay* (E2E) 30
3.5.3 *Routing Overhead* (RO).....30
3.5.4 *Forwarded Route Request* (RREQ F) 30
BAB IV IMPLEMENTASI.....31
4.1 Implementasi Skenario Mobilitas.....31
4.1.1 Skenario *Grid*.....31
4.1.2 Skenario *Real* 34
4.2 Implementasi Modifikasi pada *Routing Protocol* AODV untuk Memilih *Forwarding Node* 36
4.2.1 Implementasi Menghitung Jumlah *Node* Tetangga dan Membuat Daftar Tetangga 37
4.2.2 Implementasi Menghitung *Forwarding Probability* 40
4.2.3 Implementasi Pemilihan *Forwarding Node*.....42
4.3 Implementasi Simulasi pada NS-2 43
4.4 Implementasi Metrik Analisis 44
4.4.1 Implementasi *Packet Delivery Ratio*.....45
4.4.2 Implementasi Rata-Rata *End-to-End Delay*.....46
4.4.3 Implementasi *Routing Overhead* 47
4.4.4 Implementasi *Forwarded Route Request*.....47
BAB V UJI COBA DAN EVALUASI.....49
5.1 Lingkungan Uji Coba 49

5.2 Hasil Uji Coba 50
5.2.1 Hasil Uji Coba Skenario *Grid* 50
5.2.2 Hasil Uji Coba Skenario *Real* 57
BAB VI KESIMPULAN DAN SARAN 65
6.1 Kesimpulan 65
6.2 Saran 65
DAFTAR PUSTAKA 67
LAMPIRAN 69
A.1 Kode Fungsi AODV::recvRequest() 69
A.2 Kode Fungsi nb_insert() 77
A.3 Kode Fungsi nb_delete() 78
A.4 Kode Skenario NS-2 79
A.5 Kode Konfigurasi *Traffic* 82
A.6 Kode Skrip AWK *Packet Delivery Ratio* 83
A.7 Kode Skrip AWK Rata-Rata *End-to-End Delay* 84
A.8 Kode Skrip AWK *Routing Overhead* 85
A.9 Kode Skrip AWK *Forwarded Route Request* 86
BIODATA PENULIS 87

(Halaman ini sengaja dikosongkan)

DAFTAR GAMBAR

Gambar 2.1 Ilustrasi VANETs [1] 8
Gambar 2.2 Ilustrasi pencarian rute routing protocol AODV [11] 9
Gambar 2.3 Grafik Rata-rata Jumlah Tetangga dari Percobaan vs DCF 16
Gambar 2.4 Flowchart dari AODV-DCF 19
Gambar 3.1 Diagram Rancangan Simulasi AODV Modifikasi . 21
Gambar 3.2 Alur Perancangan Skenario 24
Gambar 3.3 Pseudocode Penghitungan Forwarding Probability (FP) 27
Gambar 3.4 Pseudocode pemilihan Forwarding Node 28
Gambar 4.1 Hasil Generate Peta Grid 32
Gambar 4.2 Perintah randomTrips 32
Gambar 4.3 File Skrip .sumocfg 33
Gambar 4.4 Perintah traceExporter 34
Gambar 4.5 Ekspor Peta dari OpenStreetMap 35
Gambar 4.6 Hasil Konversi Peta Real 36
Gambar 4.7 Potongan modifikasi Kode Fungsi `nb_insert()` dan `nb_delete()` 39
Gambar 4.8 Potongan modifikasi Kode Fungsi recvRequest() .. 41
Gambar 4.9 Potongan Kode Untuk Pemilihan Forwarding Node 42
Gambar 4.10 Implementasi Simulasi NS-2 43
Gambar 4.11 Konfigurasi Lingkungan Simulasi 44
Gambar 4.12 Pseudocode untuk Menghitung PDR 45
Gambar 4.13 Pseudocode untuk Perhitungan Rata-Rata End-to-End Delay 46
Gambar 4.14 Pseudocode untuk Perhitungan Routing Overhead 47
Gambar 4.15 Pseudocode untuk Perhitungan Forwarded Route Request 48
Gambar 5.1 Grafik Rata-Rata Packet Delivery Ratio pada Skenario Grid 51

Gambar 5.2 Grafik Rata-Rata End-to-End Delay pada Skenario Grid.....52
Gambar 5.3 Grafik Rata-Rata Routing Overhead pada Skenario Grid.....54
Gambar 5.4 Grafik Rata-Rata Forwarded route request pada Skenario Grid.....56
Gambar 5.5 Grafik Rata-Rata Packet Delivery Ratio pada Skenario Real.....58
Gambar 5.6 Grafik End-to-End Delay pada Skenario Real 60
Gambar 5.7 Grafik Routing Overhead pada Skenario Real 61
Gambar 5.8 Grafik Rata-Rata Forwarded route request pada Skenario Real.....63

DAFTAR TABEL

Tabel 2.1 Detail Penjelasan Trace File AODV.....12
Tabel 2.2 Tabel Rata-rata Jumlah Tetangga [7] 15
Tabel 2.3 Tabel Nilai Tetap DCF [7].....17
Tabel 3.1 Daftar Istilah.....22
Tabel 3.2 Parameter Lingkungan Simulasi dengan Skenario.....28
Tabel 5.1 Spesifikasi Perangkat yang Digunakan 49

(Halaman ini sengaja dikosongkan)

BAB I
PENDAHULUAN

1.1 Latar Belakang

Teknologi telah mengalami kemajuan yang sangat pesat pada zaman ini. Semua bidang kehidupan sudah terjamah teknologi, termasuk pada bidang transportasi. *Intelligent Transport System (ITS)* merupakan upaya untuk membuat sistem transportasi menjadi lebih baik. ITS dapat menyediakan berbagai kemudahan dan keamanan untuk pengguna alat transportasi, seperti informasi titik kemacetan, mengirim pesan darurat, menemukan rute terbaik, dll. *Vehicular Ad-Hoc Networks* (VANETs) merupakan salah satu tipe *Mobile Ad-Hoc Networks* (MANETs). VANETs merupakan komponen utama dalam pengembangan ITS. Tingkat keberhasilan ITS ini ditentukan oleh seberapa baik *routing protocol* pada VANETs. Karakteristik yang ada pada VANETs, seperti tingkat konektivitas yang tidak stabil, tingginya tingkat pergerakan *node* dan kecepatan *node* sehingga terjadi banyak perubahan topologi, membuat pergerakan informasi semakin sulit dan memerlukan *routing protocol* yang efisien.

Routing protocol yang ada pada lingkungan VANETs terbagi dalam dua model, yaitu *proactive* dan *reactive routing*. *Proactive routing* merupakan protokol yang bekerja dengan menggunakan tabel *routing* dan tabel ini akan diperbarui secara terus menerus dengan rentang waktu yang pendek tanpa memperhatikan faktor lain yang dapat membebani jaringan tersebut, namun stabilitas rute dapat terjaga dengan baik. Sedangkan *reactive routing* merupakan mekanisme *routing* yang juga menggunakan tabel *routing* namun tabel *routing* akan diperbarui hanya jika terdapat permintaan pengiriman data atau terjadi kerusakan jalur *routing*. 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 terbaik merupakan mekanisme utama untuk mendukung terciptanya mobilitas pada VANETs. Pemilihan rute yang stabil saat pencarian rute sangat diperlukan untuk memperpanjang waktu penggunaan rute sehingga jaringan menjadi lebih efisien. Salah satu cara pemilihan rute yang dapat dilakukan adalah dengan memilih rute berdasarkan potensi terputusnya rute. Semakin kecil potensi suatu rute untuk terputus semakin baik rute tersebut untuk dipilih, berlaku juga sebaliknya.

Pada Tugas Akhir ini diusulkan diusulkan suatu mekanisme *routing discovery* yang bersifat adaptif pada *reactive routing* AODV untuk memperoleh rute yang paling stabil berdasarkan perhitungan probabilitas dari tingkat kepadatan *node* tetangga pada VANETs. Hasil akhir yang diharapkan adalah mengetahui perbandingan kinerja antara AODV dan AODV-DCF diukur berdasarkan performansi *Packet Delivery Ratio* (PDR), *Routing Overhead*, dan *Delivery Delay*. Adapun bagian yang akan dimodifikasi dalam *routing protocol* ini yaitu melakukan reduksi jumlah paket RREQ yang *redundant* dengan cara membuat probabilitas dari jumlah *neighbor* dan sebuah variabel dinamis *Dynamic Connectivity Factor* (DCF).

1.2 Rumusan Masalah

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

1. Bagaimana cara memilih dan membatasi jumlah *forwarding node* dalam proses *rebroadcast route request* (RREQ)?
2. Bagaimana dampak dari pemilihan dan pembatasan jumlah *forwarding node* terhadap performa *routing protocol* AODV secara keseluruhan?

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

1.4 Tujuan

Berikut beberapa hal yang menjadi tujuan pada Tugas Akhir ini:

1. Mereduksi jumlah *neighbor node* yang bertanggungjawab untuk *rebroadcast* RREQ *packet* sehingga dapat mengurangi jumlah *control packet* yang di*broadcast* dalam jaringan pada *routing protocol* AODV.
2. Mengetahui dampak yang terjadi setelah dilakukan pembatasan *forwarding node* terhadap performa *routing protocol* AODV secara keseluruhan.

1.5 Manfaat

Manfaat yang dapat diambil dengan dibuatnya Tugas Akhir ini dapat memberikan informasi tentang dampak dari pemilihan dan pembatasan *forwarding node* berdasarkan tingkat kepadatan *neighbor node* dan Dynamic Connectivity Factor (DCF) 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 dalam pembuatan Tugas Akhir. Terdapat pula sub bab jadwal kegiatan yang menjelaskan jadwal pengerjaan yang menjadi target selesainya 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* AODV, *Network Simulator* 2 (NS-2), OpenStreetMap, Java OpenStreetMap (JOSM), *Simulation of Urban Mobility* (SUMO), AWK dan Dynamic Connectivity Factor (DCF).

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, AODV, 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 AODV, 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 AODV, 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* AODV 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* AODV dengan *routing protocol* AODV-DCF.
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.

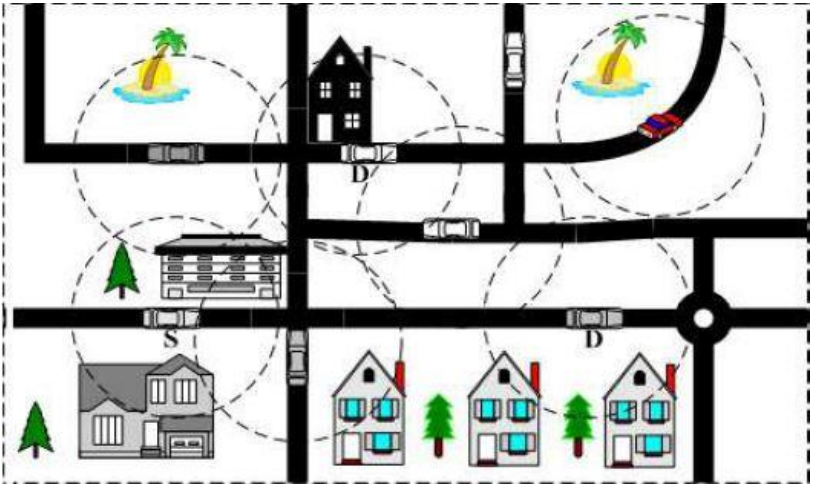
BAB II
TINJAUAN PUSTAKA

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

Sebuah jaringan terorganisir yang dibentuk dengan menghubungkan kendaraan dan RSU (*Roadside Unit*) disebut *Vehicular Ad Hoc Network* (VANET), dan RSU lebih lanjut terhubung ke jaringan *backbone* berkecepatan tinggi melalui koneksi jaringan. Kepentingan peningkatan baru-baru ini telah diajukan pada aplikasi melalui V2V (*Vehicle to Vehicle*) dan V2I (*Vehicle to Infrastructure*) komunikasi, bertujuan untuk meningkatkan keselamatan mengemudi dan manajemen lalu lintas sementara menyediakan *driver* dan penumpang dengan akses Internet. Dalam VANETs, RSUs dapat memberikan bantuan dalam menemukan fasilitas seperti restoran dan pompa bensin, dan mem*broadcast* pesan yang terkait seperti (maksimum kurva kecepatan) pemberitahuan untuk memberikan pengendara informasi. Sebagai contoh, sebuah kendaraan dapat berkomunikasi dengan lampu lalu lintas cahaya melalui V2I komunikasi, dan lampu lalu lintas dapat menunjukkan ke kendaraan ketika keadaan lampu ke kuning atau merah. Ini dapat berfungsi sebagai tanda pemberitahuan kepada pengemudi, dan akan sangat membantu para pengendara ketika mereka sedang berkendara selama kondisi cuaca musim dingin atau di daerah asing. Hal ini dapat mengurangi terjadinya kecelakaan. Melalui komunikasi V2V, pengendara bisa mendapatkan informasi yang lebih baik dan mengambil tindakan awal untuk menanggapi situasi yang abnormal. Untuk mencapai hal ini, suatu OBU (*On Board Unit*) secara teratur menyiarkan pesan yang terkait dengan informasi dari posisi pengendara, waktu saat ini, arah mengemudi, kecepatan, status rem,

sudut kemudi, lampu sen, percepatan / perlambatan, kondisi lalu lintas [1]. Ilustrasi VANETs dapat dilihat pada Gambar 2.1.



Gambar 2.1 Ilustrasi VANETs [3]

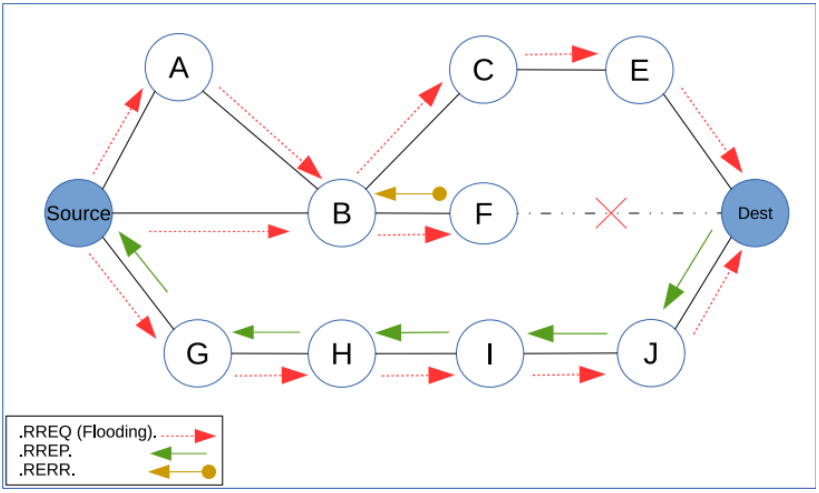
2.2 *Ad-hoc On-demand Distance Vector (AODV)*

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 [7]. Ilustrasi pencarian rute oleh AODV dapat dilihat pada Gambar 2.2 Ilustrasi pencarian rute routing protocol AODV [11]



Gambar 2.2 Ilustrasi pencarian rute routing protocol AODV [11]

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*, mencari rute untuk menuju *Destination Node*. Source *Node* 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 *Destination Node*. Kemudian, jika rute menuju *Destination Node* sudah terbentuk di dalam routing table dan memiliki *routing flags* "*up*", maka *Destination Node* 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 (Versi 2), yang banyak dikenal dengan NS-2, adalah sebuah alat simulasi berbasis aktivitas yang berguna dalam mempelajari sifat dinamis jaringan komunikasi. Simulasi fungsi jaringan kabel, nirkabel, dan protokol dapat dilakukan dengan menggunakan NS-2. NS-2 menggunakan dua bahasa utama yaitu Bahasa C++ dan *Object-oriented Tool Command Language* (OTCL). Di NS-2 mendefinisikan mekanisme internal (*backend*) dari objek simulasi, dan OTCL mendefinisikan lingkungan simulasi eksternal (*frontend*) untuk perakitan dan konfigurasi objek. Setelah simulasi, NS-2 memberikan output simulasi baik dalam bentuk file NAM atau *trace file* [2].

Pada Tugas Akhir ini, NS-2 digunakan untuk melakukan simulasi lingkungan VANETs menggunakan *routing protocol* AODV yang sudah dimodifikasi. *Trace file* yang dihasilkan oleh NS-2 juga digunakan sebagai informasi untuk mengukur performa *routing protocol* 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 perintah berikut:

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

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

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

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

2.3.2 *Trace File*

Trace file merupakan *file* hasil simulasi yang dilakukan oleh NS-2 dan berisikan informasi detail pengiriman paket data. *Trace file* digunakan untuk menganalisis performa *routing protocol* yang disimulasikan. Detail penjelasan *trace file* ditunjukkan pada Tabel 2.1.

Tabel 2.1 Detail Penjelasan *Trace File* AODV

Kolom ke-	Penjelasan	Isi
1	Event	s: *sent* r: *received* f: *forwarded* D: *dropped*
2	*Time*	Waktu terjadinya *event*
3	ID *Node*	_x_: dari 0 hingga banyak *node* pada topologi
4	*Layer*	AGT: *application* RTR: *routing* LL: *link layer* IFQ: *packet queue* MAC: MAC PHY: *physical*
5	*Flag*	---: Tidak ada
6	*Sequence Number*	Nomor paket
7	Tipe Paket	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

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

2.4 OpenStreetMap

OpenStreetMap merupakan proyek kolaboratif untuk membuat sebuah peta dunia yang dapat dengan bebas disunting oleh siapapun. OpenStreetMap digunakan sebagai data peta pada berbagai *website*, aplikasi *mobile*, dan *hardware device*. OpenStreetMap dibangun oleh komunitas pembuat peta yang berkontribusi dan mengelola data mengenai jalan raya, kafe, stasiun kereta api, dan sebagainya di seluruh dunia [3].

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 *real* VANETs.

2.5 Java OpenStreetMap Editor (JOSM)

Java OpenStreetMap Editor (JOSM) adalah aplikasi yang dikembangkan oleh Immanuel Scholz. Aplikasi JOSM untuk menyunting data yang didapatkan dari OpenStreetMap. Aplikasi

JOSM dapat diunduh pada halaman https://josm.openstreetmap.de. Penulis menggunakan aplikasi ini untuk menyunting dan merapikan peta yang diunduh dari OpenStreetMap [4].

Pada Tugas Akhir ini, penulis menggunakan data yang telah diambil dari OSM lalu disunting. Map yang disunting yaitu menghilangkan dan juga menyambungkan jalan yang ada, dan menghilangkan gedung-gedung yang ada di map.

2.6 Simulation of Urban Mobility (SUMO)

Simulation of Urban Mobility atau disingkat SUMO merupakan simulasi lalu lintas jalan raya yang *open source*, *microscopic*, dan *multi-modal*. SUMO mensimulasikan bagaimana permintaan lalu lintas yang terdiri dari beberapa kendaraan berjalan pada jalan raya yang telah ditentukan. Simulasi SUMO dapat menunjukkan beberapa topik manajemen lalu lintas dalam skala besar. SUMO murni *microscopic*, yang artinya setiap kendaraan dimodelkan secara eksplisit, memiliki rutenya sendiri, dan bergerak secara individu dalam jaringan [5].

Pada Tugas Akhir ini, penulis menggunakan SUMO untuk men*generate* skenario VANETs, peta area simulasi, dan pergerakan *node* sehingga menyerupai keadaan sebenarnya lalu lintas yang ingin disimulasikan. Untuk setiap *generate* skenario VANETs menggunakan SUMO akan menghasilkan pergerakan *node* yang acak sehingga tidak akan sama setiap *generate* menggunakan SUMO.

2.7 AWK

AWK merupakan sebuah bahasa pemrograman yang didesain untuk *text-processing* dan biasanya digunakan sebagai alat ekstrasi data dan pelaporan. AWK bersifat *data-driven* yang berisikan kumpulan perintah yang akan dijalankan pada data tekstural baik secara langsung pada *file* atau digunakan sebagai bagian dari *pipeline* [6].

2.8 *Dynamic Connectivity Factor* (DCF)

Dynamic Connectivity Factor (DCF) adalah sebuah persamaan baru yang dibentuk berdasarkan jumlah tetangga yang ada pada sebuah jaringan. Pada dasarnya, persamaan baru DCF ini dibentuk dari percobaan yang dilakukan sebanyak tiga puluh percobaan untuk setiap titik rata-rata jumlah tetangga pada sumbu Y, dengan kepadatan *node* yang bervariasi mulai dari 50 hingga 300 *node*.

Jumlah total semua tetangga untuk semua *node* di tiap jaringan akan dihitung, dan setelah percobaan telah dilakukan, maka akan dihitung rata-rata jumlah tetangganya. Sebagai contoh, rata-rata jumlah tetangga pada jaringan dengan 50 *node* adalah 5,732 dan untuk jumlah tetangga pada jaringan dengan kepadatan 300 *node* adalah 12,827. Hasil lengkap semua percobaan yang telah dilakukan terdapat pada Tabel 2.2.

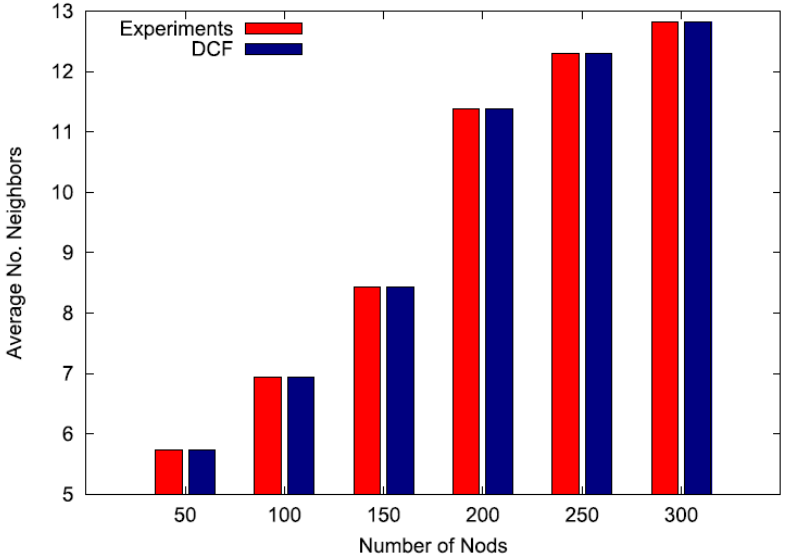
Tabel 2.2 Tabel Rata-rata Jumlah Tetangga [7]

No.	Jumlah *Node*	Rata-rata Jumlah Tetangga
1	50	5,732
2	100	6,939
3	150	8,426
4	200	11,378
5	250	12,291
6	300	12,827

Setelah mendapatkan rata-rata jumlah tetangga, akan dibentuk sebuah grafik yang membandingkan jumlah total *node* tetangga yang ada pada jaringan antara data jumlah rata-rata tetangga yang didapatkan melalui percobaan dan data jumlah rata-rata tetangga yang didapatkan melalui persamaan DCF. Grafik perbandingan kedua data ini dapat dilihat pada Gambar 2.3.

Dari tren yang didapat dari perbandingan kedua data maka data-data tersebut digunakan dalam pembentukan persamaan DCF yang dapat merepresentasikan hubungan yang sama antara rata-rata jumlah tetangga dan jumlah total semua *node* yang ada pada suatu jaringan. Agar persamaan DCF yang terbentuk dapat

merepresentasikan hubungan antara rata-rata jumlah tetangga dan jumlah total semua *node*, maka pada persamaan DCF membutuhkan variabel-variabel baru. Variabel-variabel baru ini merupakan konstanta statis yang dibutuhkan dalam persamaan DCF. Variabel konstanta statis ini dapat dilihat pada Tabel 2.3. [7]



Gambar 2.3 Grafik Rata-rata Jumlah Tetangga dari Percobaan vs DCF

Dynamic Connectivity Factor (DCF) merupakan sebuah ukuran konektivitas yang berfungsi sebagai acuan dalam penentuan sebuah RREQ akan diteruskan pada seluruh tetangganya atau akan didrop. Perhitungan DCF ini dibentuk berdasarkan dari rata-rata jumlah tetangga pada sebuah jaringan. Sehingga rata-rata jumlah tetangga merupakan faktor penting dalam penentuan apakah sebuah RREQ packet akan diteruskan atau di*drop*.

Pada DCF terdapat beberapa variabel untuk menghitung nilai probabilitas yang menjadi acuan untuk pemilihan *forwarding node* pada Tugas Akhir ini. DCF memiliki beberapa nilai tetap yang

digunakan dalam perhitungan probabilitas, nilai tersebut ada pada Tabel 2.3.

Tabel 2.3 Tabel Nilai Tetap DCF [7]

Variabel	Nilai Tetap
a	8.796112
b	37.29479
c	7.229256
d	11.26162
m	5.094965

DCF membutuhkan beberapa variabel untuk mendapatkan *Forwarding Probability* (FP), yaitu NB(n_i), DCF(n_i), DAF(n_i), Ra(n_i). Variabel NB(n_i) dapat dihitung menggunakan Persamaan 2.1.

$$NB(n_i) = 1 + (\frac{N(n_i)}{c})^b \tag{2.1}$$

dimana N(n_i) adalah jumlah *node* tetangga sebuah *node* yang menerima RREQ, sedangkan b dan c merupakan nilai tetap yang ada pada tabel 2.1.

Representasi dari jumlah total *node* dapat dinotasikan dengan DCF dan didapatkan dengan Persamaan 2.2.

$$DCF(n_i) = d + \frac{a-d}{[NB(n_i)]^m} \tag{2.2}$$

dimana a,d, dan m merupakan nilai tetap yang juga ada pada tabel 2.1.

Dynamic connectivity-Aware Factor (DAF) merupakan variabel baru yang berfungsi untuk mereduksi jumlah RREQ paket yang redundan. DAF dapat dihitung dengan Persamaan 2.3.

$$DAF(n_i) = \frac{DCF(n_i)}{N(n_i)} \tag{2.3}$$

DAF digunakan sebagai acuan posisi sebuah *node* apakah *node* tersebut berada pada area yang padat atau tidak.

Additional Coverage Ratio (Ra(n_i)) adalah rasio dari jumlah *node* yang tidak terjangkau oleh *node* yang menerima *rebroadcast* dibanding dengan jumlah total tetangga *node* tersebut. Ra(n_i) dapat dihitung menggunakan Persamaan 2.4.

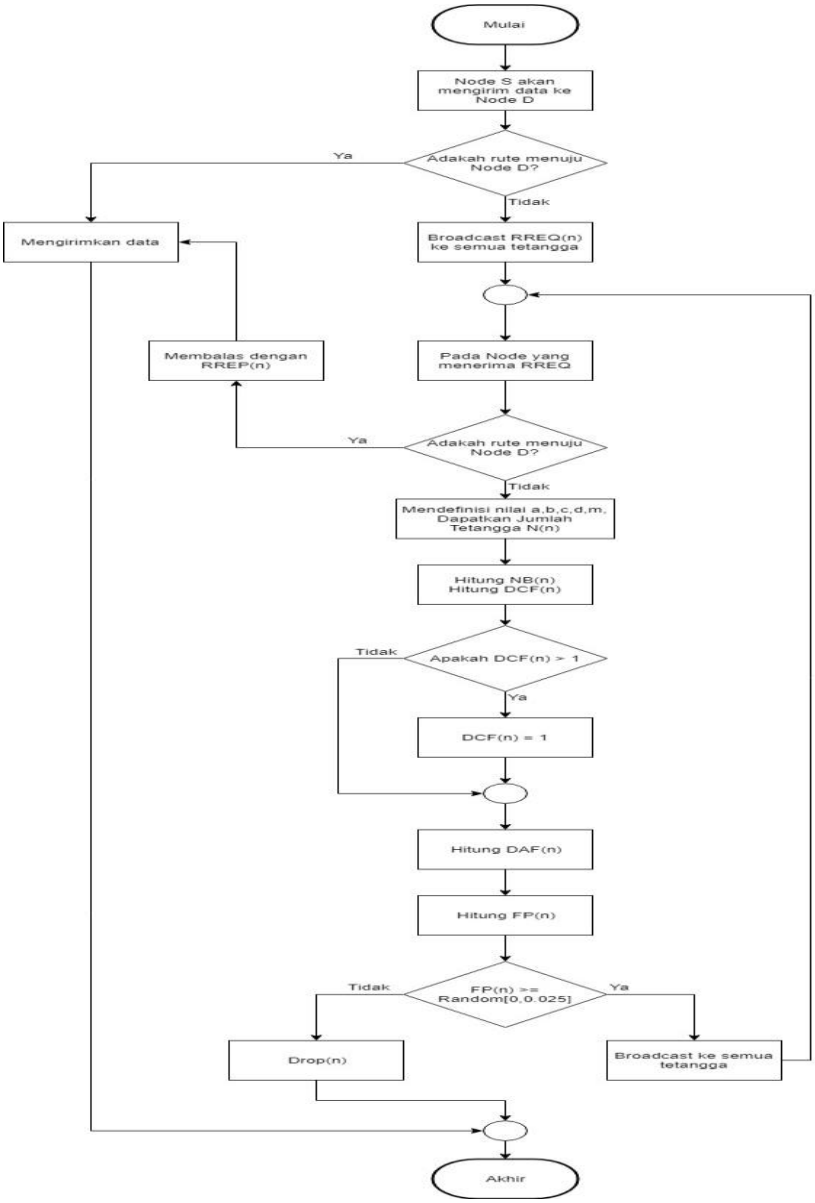
$$Ra(n_i) = \frac{|U(n_i)|}{|N(n_i)|} \tag{2.4}$$

FP adalah probabilitas yang digunakan suatu *node* untuk menentukan sebuah RREQ paket akan di *rebroadcast* atau tidak. Selanjutnya FP dapat dihitung menggunakan Persamaan 2.5.

$$FP(n_i) = DAF(n_i).Ra(n_i) \tag{2.5}$$

Perhitungan DCF dimulai saat suatu *node* yang menerima paket RREQ dari sebuah *node* sumber untuk diteruskan hingga *node* tujuan tidak memiliki rute untuk sampai ke *node* tujuan. Karena *node* yang menerima paket RREQ tidak berhak membalas dengan paket RREP maka *node* tersebut menghitung FP yang dapat menyatakan apakah *node* tersebut berhak untuk me*rebroadcast* paket RREQ yang diterima.

Pada perhitungan FP, terdapat algoritma yang dimulai dengan mendefinisikan nilai konstanta statis dan menghitung jumlah tetangga yang dimiliki *node* tersebut. Lalu jumlah tetangga yang didapat, akan menjadi masukan pada persamaan NB. Jika nilai NB telah dikalkulasi, selanjutnya akan dihitung nilai DCF dengan menggunakan NB. Nilai DCF yang didapat, akan dievaluasi terlebih dahulu. Jika nilai DCF hasil perhitungan bernilai lebih dari 1, maka nilai DCF akan ditetapkan menjadi 1. Jika tidak, maka nilai DCF tetap pada nilai hasil perhitungan. Setelah nilai DCF dapat ditentukan, tahap selanjutnya adalah menghitung nilai DAF. Selanjutnya, maka nilai FP telah dapat dikalkulasi untuk mendapatkan keputusan yang menentukan seberapa pantas *node* tersebut me*rebroadcast* paket RREQ yang diterimanya. Untuk mendapatkan keputusan yang adil, maka nilai FP akan dibandingkan dengan nilai *random* antara 0 hingga 1. Algoritma dari DCF dapat dilihat pada Gambar 2.4.



Gambar 2.4 Flowchart dari AODV-DCF

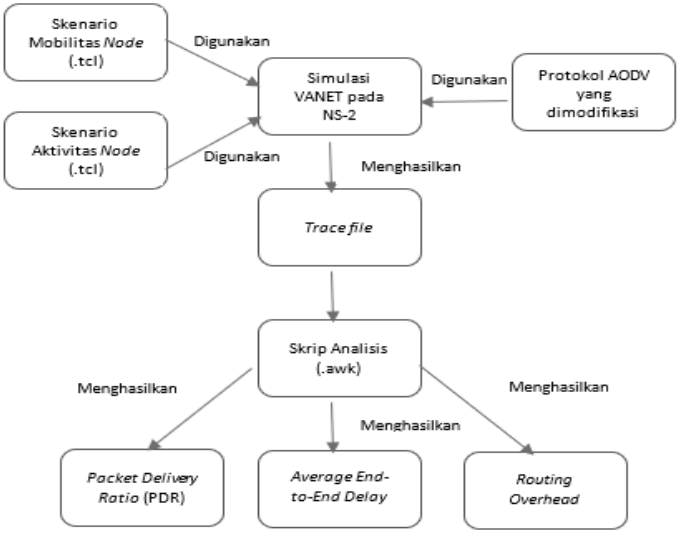
(Halaman ini sengaja dikosongkan)

BAB III
PERANCANGAN

Bab ini membahas mengenai perancangan implementasi sistem yang dibuat pada Tugas Akhir. Bagian yang akan dijelaskan pada bab ini berawal dari deskripsi umum, perancangan skenario, hingga alur dan implementasinya.

3.1 Deskripsi Umum

Pada Tugas Akhir ini akan diimplementasikan routing protocol AODV dengan memodifikasi pada bagian proses route discovery dengan menambahkan proses seleksi pemilihan forwarding *node* yang berhak melakukan *rebroadcast* paket route request (RREQ) yang akan diteruskan. Simulasi dijalankan pada simulator NS-2. Diagram dari rancangan simulasi dari AODV dan AODV modifikasi dapat dilihat pada Gambar 3.1.



Gambar 3.1 Diagram Rancangan Simulasi AODV Modifikasi

Proses seleksi pemilihan forwarding *node* dimulai dengan menghitung jumlah *node* tetangga yang berada dalam *range* transmisi sinyal pada *node* yang menerima RREQ packet. 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 probabilitas *Dynamic Connectivity Factor* (DCF) dengan rumus yang telah ditentukan. Selanjutnya nilai probabilitas yang telah dikalkulasi akan digunakan dalam penentuan apakah *node* tersebut terbilang efisien untuk mem*broadcast* RREQ. Proses penentuan akan dilakukan dengan membandingkan nilai probabilitas dengan bilangan real *random* antara 0-0.025.

Modifikasi *routing protocol* AODV ini akan disimulasikan menggunakan NS-2 dengan skenario pergerakan *node* yang dibuat menggunakan *tools* SUMO. Simulasi yang dilakukan menghasilkan *trace file*, yang kemudian dianalisis menggunakan AWK untuk mendapatkan *packet delivery ratio (PDR), average end-to-end delay, routing overhead* dan *forwarded route request* (RREQ F). Analisis tersebut dapat mengukur performa *routing protocol* AODV-DCF dibandingkan dengan *routing protocol* AODV. 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	*Ad-hoc On-demand Distance Vector*.
2	PDR	*Packet Delivery Ratio*
3	E2E	*Average End-to-End Delay*
4	RO	*Routing Overhead*
5	RREQ	*Route Request*
6	RREP	*Route Reply*
7	*DCF*	*Dynamic Connectivity Factor*
8	*FP*	*Forwarding Probability*
9	AODV-DCF	*Ad-hoc On-demand Distance Vector-Dynamic Connectivity Factor*

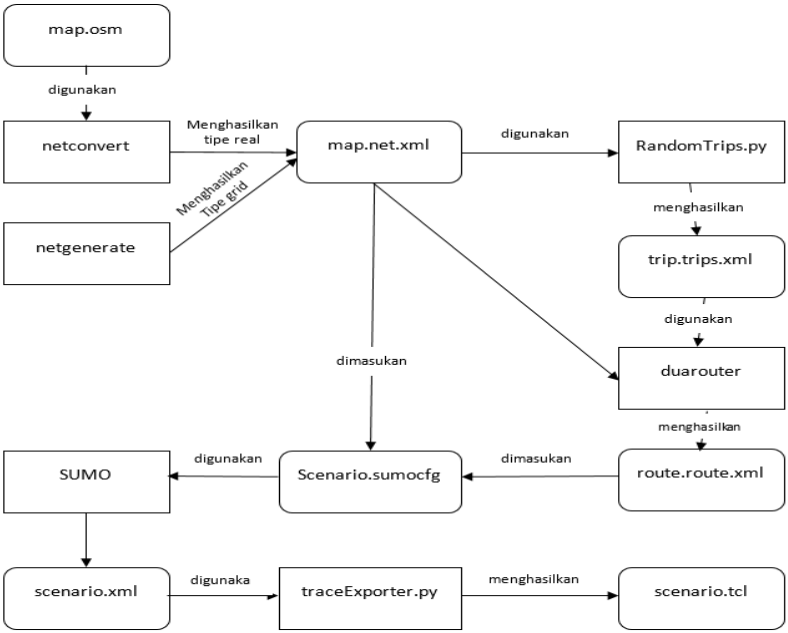
3.2 Perancangan Skenario Mobilitas

Perancangan skenario mobilitas pada Tugas Akhir ini mencakup pada perancangan area peta, pergerakan *node*, dan implementasi pergerakan *node*. Pembuatan peta skenario pada Tugas Akhir ini terbagi menjadi dua, yaitu peta berbentuk *grid* dan peta berbentuk *real*. Peta dengan bentuk *grid* adalah peta dengan jalan berpetak-petak sebagai contoh jalan yang sederhana. Peta *grid* dipilih sebagai simulasi awal karena bentuknya yang lebih stabil dan seimbang. Peta *grid* dapat di*generate* dengan menentukan panjang dan jumlah petak area menggunakan beberapa *tools* yang terdapat pada SUMO. Sedangkan untuk peta *real* digunakan peta asli sebagai sebagai bahan acuan untuk area simulasi. Pada Tugas Akhir ini peta *real* didapatkan dengan bantuan OpenStreetMap dan mengambil salah satu area yang ada di Surabaya.

Peta dengan bentuk *grid* adalah peta dengan jalan berbentuk petak-petak seperti lantai rumah. Representasi jalan atau lajur pada peta dengan bentuk berpetak-petak adalah garis yang membentuk petak tersebut. Maka dari itu, pergerakan *node* akan sangat terbatas dengan garis-garis yang membentuk petak tersebut. Pergerakan *node* juga tidak terlalu variatif, yaitu hanya bergerak kanan kiri atau atas bawah. Dengan terbatasnya ruang gerak dari *node*, maka peta dengan bentuk *grid* ini menjadi lebih stabil dan seimbang.

Peta dengan bentuk *real* adalah peta hasil konversi dari peta asli dengan bantuan OpenStreetMap sebagai acuan peta dunia nyata. Representasi jalan atau lajur pada peta dengan bentuk real merupakan jalan yang sesungguhnya dari sebuah area peta yang telah dikonversi tersebut. Dengan bervariasinya bentuk jalur, maka pergerakan *node* juga semakin bervariatif. *Node* dapat bergerak ke segala arah sesuai dengan jalan yang telah dikonversi, contoh jika pada hasil konversi peta terdapat jalan yang melintang dari ujung satu ke ujung lainnya atau membentuk diagonal, *node* juga dapat bergerak secara diagonal. Maka dari itu peta *real* dapat dikatakan memiliki ruang gerak yang

banyak, sehingga peta dengan bentuk *real* ini kurang stabil dan seimbang jika dibandingkan dengan peta dengan bentuk *grid*.



Gambar 3.2 *Alur Perancangan Skenario*

3.2.1 Perancangan Skenario Grid

Perancangan skenario mobilitas *grid* diawali dengan merancang luas area peta *grid* yang dibutuhkan untuk simulasi. Luas area tersebut bisa didapatkan dengan cara menentukan terlebih dahulu jumlah titik persimpangan yang diinginkan pada peta *grid*, sehingga dari jumlah persimpangan tersebut dapat diketahui berapa banyak petak yang dibutuhkan. Dengan mengetahui jumlah petak yang dibutuhkan, dapat digunakan untuk menentukan panjang tiap petak sehingga mendapatkan luas area yang dibutuhkan adalah 1300 m x 1300 m. Dengan 4 titik persimpangan, maka akan didapatkan 9 petak.

Dengan begitu panjang tiap petak untuk mendapatkan luas area 1300 m x 1300 m adalah 400 m.

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

Skenario mobilitas *grid* dihasilkan dengan menggabungkan *file* peta *grid* dan *file* pergerakan *node* yang telah di*generate*, yang akan menghasilkan *file* dengan ekstensi .xml. Selanjutnya, untuk dapat menerapkannya pada NS-2 *file* skenario mobilitas *grid* yang berekstensi .xml dikonversi ke dalam bentuk *file* .tcl. Konversi ini dilakukan menggunakan *tools* traceExporter.

3.2.2 Perancangan Skenario Real

Perancangan skenario mobilitas *real* diawali dengan memilih area yang akan dijadikan simulasi. Pada Tugas Akhir ini, penulis menggunakan peta dari bantuan OpenStreetMap untuk mengambil area yang dijadikan model simulasi. Pada OpenStreetMap bisa memilih area dengan cara mengisi koordinat yang dibutuhkan untuk area yang spesifik ataupun dapat menggunakan *tools* yang telah tersedia di OpenStreetMap dengan cara menggunakan opsi memilih peta secara manual. Setelah memilih area, unduh dengan menggunakan fitur *export* yang telah disediakan oleh OpenStreetMap. Peta hasil *export* tersebut memiliki ekstensi .osm.

Untuk proses selanjutnya sebenarnya sama seperti merancang skenario mobilitas *grid*, hanya saja terdapat perbedaan penggunaan *tools* untuk mengkonversi peta dari OpenStreetMap tersebut. Setelah mendapatkan peta area yang 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 mobilitas *grid*, yaitu membuat pergerakan *node* menggunakan randomTrips dan duarouter. Kemudian gabungkan *file* peta *real* yang sudah dikonversi ke dalam *file* dengan ekstensi .net.xml dan *file* pergerakan *node* yang sudah di*generate*. Hasil dari penggabungan tersebut merupakan *file* skenario berekstensi .xml. *File* yang dihasilkan tersebut dikonversi ke dalam bentuk *file* .tcl agar dapat digunakan untuk simulasi pada NS-2.

3.3 Analisis dan Perancangan Modifikasi *Routing Protocol* AODV

Modifikasi Routing Protocol AODV yang diajukan pada Tugas Akhir ini membutuhkan satu variabel tambahan yaitu jumlah tetangga yang mendapatkan *broadcast* RREQ. Variabel ini lalu digunakan dalam perhitungan probabilitas suatu *node* dapat mem*broadcast* ulang (*rebroadcast*) pesan tersebut. Pada protokol AODV terdapat mekanisme *hello message* yang dapat untuk mengenali tetangga yang berada pada *range* transmisi suatu *node*. Mekanisme ini dapat dimanfaatkan untuk memenuhi kebutuhan modifikasi pada Tugas Akhir ini. Bersamaan dengan berjalannya hello message, perhitungan jumlah tetangga dapat dilakukan pada saat ini juga. Sehingga ketika terdapat suatu RREQ, informasi jumlah tetangga tiap *node* sudah ada dan dapat dilanjutkan untuk perhitungan selanjutnya.

3.3.1 Perancangan Penghitungan Jumlah *Node* Tetangga dan Membuat Daftar *Node* Tetangga untuk Setiap *Node*

Untuk mendapatkan jumlah tetangga pada *node* yang menerima RREQ, Tugas Akhir ini memanfaatkan mekanisme *hello messages* yang telah ada pada protokol AODV. *Hello messages* dilakukan pada tiap saat simulasi akan dimulai dan akan terus menerus dilakukan untuk saling bertukar informasi untuk setiap *node* yang berada dalam satu lingkup transmisi. Sehingga perhitungan

jumlah *node* tetangga dapat dilakukan dengan cara menambahkan *counter* jumlah tetangga tiap *node*. Variabel *counter* akan menggunakan tipe data *int* dengan bentuk *array* 2 dimensi. Variabel *counter* ini akan disisipkan pada fungsi nb_insert() untuk menambah jumlah *node* tetangga dan nb_delete() untuk mengurangi jumlah *node* tetangga.

Untuk dapat membuat daftar tetangga pada setiap *node*, akan dibuat sebuah variabel baru dengan bentuk *vector* multidimensi. Pembuatan *vector* akan memanfaatkan *library* standar yang telah ada pada C++. Variabel daftar tetangga ini juga akan disisipkan pada fungsi nb_insert() bersamaan dengan bertambahnya jumlah *node* tetangga dan nb_delete() pada saat mengurangi *node* tetangga.

3.3.2 Perancangan Penghitungan *Forwarding Probability* (FP)

Pada AODV-DCF, perhitungan FP dapat dilakukan dengan dimulai dengan mengolah nilai jumlah total tetangga suatu *node* yang menerima paket RREQ yang telah didapatkan sebelumnya. Jumlah

```
Get N(ni).
Calculate NB(ni).
Calculate DCF(ni).
If DCF(ni) > 1 then
     DCF(ni) = 1
End
Calculate DAF(ni).
Calculate FP(ni).
```

Gambar 3.3 *Pseudocode Penghitungan Forwarding Probability (FP)*

total tetangga ini akan menjadi masukan dalam persamaan DCF. Dari perhitungan DCF, keluaran yang akan didapatkan adalah nilai probabilitas FP yang selanjutnya akan digunakan pada tahap berikutnya. Tahap perhitungan DCF dapat dilihat pada *pseudocode* pada Gambar 3.3.

3.3.3 Perancangan Pemilihan *Forwarding Node*

Pada AODV-DCF, pemilihan *node* yang menggunakan *Forwarding Probability* (FP) yang telah dihitung sebelumnya. FP akan dibandingkan dengan bilangan real random antara 0 hingga 0.025. Jika FP lebih besar atau sama dengan bilangan real *random*, maka *node* tersebut akan mem*broadcast* RREQ ke semua tetangga. Jika FP lebih kecil daripada bilangan real *random*, RREQ akan di*drop* pada *node* tersebut. Untuk pemilihan *forwarding node* dapat dilihat pada perintah *pseudocode* pada Gambar 3.4.

```
if (FP >= Random[0,0.025])
then
     Forward(n) to all neighbors
else
     Drop(n)
```

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 tcl yang berisikan konfigurasi lingkungan simulasi. Konfigurasi lingkungan simulasi VANETs pada NS-2 dapat dilihat pada Tabel 3.2.

Tabel 3.2 Parameter Lingkungan Simulasi dengan Skenario

No.	**Parameter**	**Spesifikasi**
1.	*Network Simulator*	NS-2, 2.35
2.	*Routing Protocol*	AODV

3.	Waktu Simulasi	200, 400, 500
4.	Area Simulasi	1300 m x 1300 m, 1500 m x 1500 m
5.	Banyak Kendaran	60, 150, 300
6.	Agen Pengirim	*Constant Bit Rate* (CBR)
7.	*Protocol* MAC	IEEE 802.11
8.	Propagasi sinyal	*Two-ray ground*
9.	*Source/Destination*	Statis
10.	Tipe Kanal	*Wireless Channel*

3.5 Perancangan Metrik Analisis

Berikut ini merupakan parameter-parameter yang akan digunakan untuk analisis pada Tugas Akhir ini untuk dapat membandingkan performa dari *routing protocol* AODV yang telah dilakukan modifikasi dan *routing protocol* AODV yang asli:

3.5.1 *Packet Delivery Ratio* (PDR)

Packet Delivery Ratio (PDR) merupakan perbandingan antara jumlah paket yang diterima dengan jumlah paket yang dikirimkan. PDR dihitung dengan Persamaan 3.1.

$$PDR = \frac{received}{sent} x\ 100\ \% \tag{3.1}$$

PDR dapat menunjukkan keberhasilan paket yang dikirimkan. Semakin tinggi PDR, artinya semakin berhasil pengiriman paket yang dilakukan.

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

Rata-rata *End to End Delay* merupakan rata-rata dari *delay* atau waktu yang dibutuhkan tiap paket untuk sampai ke *node* tujuan dalam satuan detik. *Delay* tiap paket didapatkan dari rentang waktu antara *node* asal mengirimkan paket (CBRSentTime) dan *node* tujuan menerima paket (CBRRecvTime). Dari *delay* tiap paket tersebut semua dijumlahkan dan dibagi dengan jumlah paket yang berhasil diterima (recvnum), maka akan didapatkan rata-rata *end to end delay*, yang dapat dihitung dengan persamaan 3.2.

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

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.

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

3.5.4 *Forwarded Route Request* (RREQ F)

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

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

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

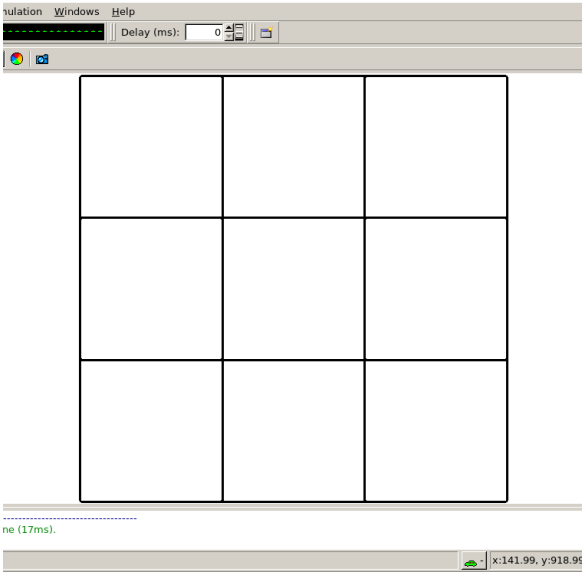
Implementasi skenario mobilitas VANETs dibagi menjadi dua, yaitu skenario *grid* yang menggunakan peta jalan berpetak dan skenario *real* yang menggunakan peta hasil pengambilan suatu area di kota Surabaya.

4.1.1 Skenario *Grid*

Dalam mengimplementasikan skenario *grid*, SMO 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 vertical dan jalan horizontal sebanyak 4 titik x 4 titik. Dengan jumlah titik persimpangan sebanyak 4 titik tersebut, maka 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. Perintah netgenerate untuk membuat peta tersebut dengan kecepatan *default* kendaraan sebesar 20 m/s dapat dilihat pada perintah berikut:

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

Setelah itu akan didapat file map berekstensi .xml. Gambar hasil peta yang telah dibuat dengan netgenerate dapat dilihat pada Gambar 4.1.



Gambar 4.1 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 4.2.

```
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 4.2 Perintah randomTrips

Selanjutnya dibuatkan rute yang digunakan kendaraan untuk mencapai tujuan dari *file* hasil sebelumnya menggunakan *tools* duarouter. Secara *default* algoritma yang digunakan untuk membuat rute ini adalah algoritma djikstra. Penggunaan *tools* duarouter dapat dilihat pada perintah berikut:

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

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 4.3.

```
<?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 4.3 *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. Untuk menggunakan *tools* SUMO dapat dilihat pada perintah berikut:

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

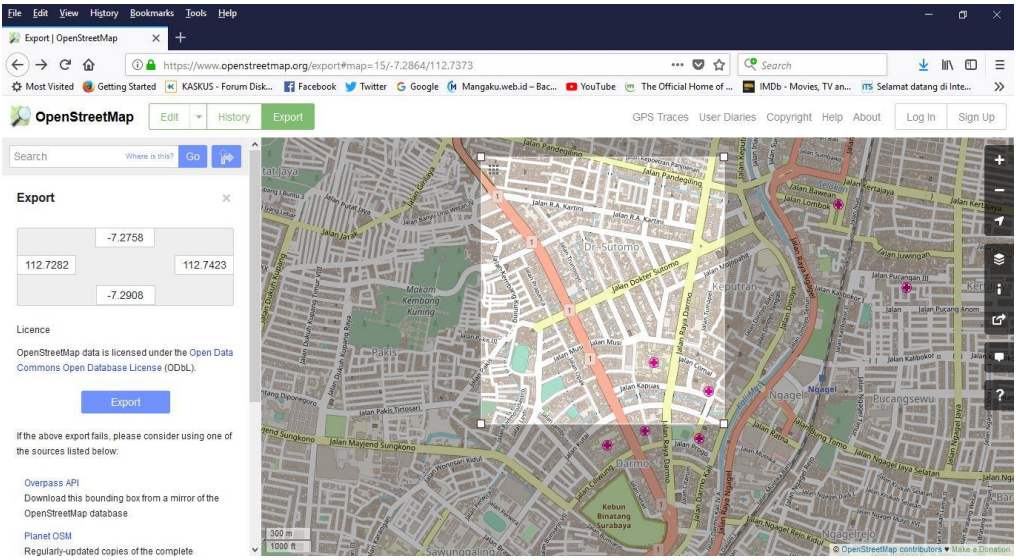
File skenario berekstensi .xml selanjutnya dikonversi ke dalam bentuk *file* berekstensi .tcl agar dapat disimulasikan menggunakan NS-2. *Tools* yang digunakn untuk melakukan konversi ini adalah traceExporter. Perintah untuk menggunakan traceExporter dapat dilihat pada Gambar 4.4.

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

Gambar 4.4 Perintah traceExporter

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 4.5.

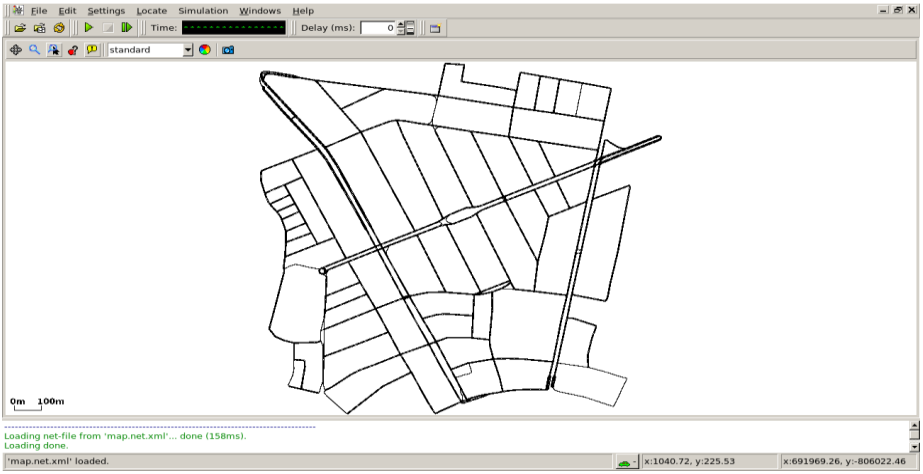


Gambar 4.5 Ekspor Peta dari OpenStreetMap

File hasil ekspor dari OpenStreetMap tersebut adalah *file* peta dengan ekstensi .osm. Kemudian konversi *file* .osm tersebut menjadi peta dalam bentuk *file* berekstensi .xml menggunakan *tools* netconvert dari SUMO. Perintah untuk menggunakan netconvert dapat dilihat pada perintah berikut:

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

Hasil konversi peta dari *file* berekstensi .osm menjadi *file* berekstensi .xml dapat dilihat menggunakan *tools* sumo-gui seperti yang ditunjukkan pada Gambar 4.6.



Gambar 4.6 Hasil Konversi Peta *Real*

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

4.2 Implementasi Modifikasi pada *Routing Protocol* AODV untuk Memilih *Forwarding Node*

Routing protocol AODV merupakan salah satu *routing protocol* yang umum digunakan dalam simulasi VANETs. Tetapi, *routing protocol* ini belum optimal dalam melakukan pengiriman

paket RREQ ketika *node* ingin me*rebroadcast* sehingga terjadi routing overhead yang dapat membebani jaringan tersebut.

Pada Tugas Akhir ini dilakukan modifikasi pada *routing protocol* AODV agar dapat mengurangi jumlah *node* yang melakukan *rebroadcast* dengan memilih *forwarding node* berdasarkan jumlah tetangga dan melakukan perhitungan probabilitas untuk setiap *node* yang menerima paket RREQ. Sehingga pada Tugas Akhir ini bisa dilihat performa pada *routing protocol* AODV dapat meningkat.

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

- Implementasi Menghitung Jumlah *Node* Tetangga dan Membuat Daftar Tetangga
- Implementasi Menghitung Forwarding Probability
- Implementasi Pemilihan *Forwarding Node*

Kode implementasi dari *routing protocol* AODV pada NS-2 versi 2.35 berada pada direktori ns-2.35/aodv. Di dalam direktori tersebut terdapat beberapa *file* diantaranya seperti aodv.cc, aodv.h dan sebagainya. Pada Tugas Akhir ini penulis hanya memodifikasi *file* aodv.cc untuk dapat menghitung jumlah *node* tetangga, menghitung FP, dan menentukan forwarding *node*. Pada bagian ini penulis akan menjelaskan langkah-langkah dalam mengimplemetnasikan modifikasi *routing protocol* AODV untuk mengurangi jumlah *node* yang melakukan *rebroadcast*.

4.2.1 Implementasi Menghitung Jumlah *Node* Tetangga dan Membuat Daftar 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 dan menerima hello messages dapat dipastikan bahwa *node* tersebut berada pada dalam jangkauan *range* transmisi *node* yang sedang dieksekusi.

Pada mekanisme AODV, setiap *node* akan saling bertukar informasi siapa saja yang menjadi *node* tetangga. 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* dan dapat mendapatkan indeks *node* untuk membuat daftar tetangga pada 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 4.7. Kode selengkapnya dapat dilihat di lampiran A1.

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

AODV_Neighbor *nb = new AODV_Neighbor(id);

if(nb_count[index]==NULL)
  {
    nb_count[index]=0;
    nb_count[index]++;
    nb_list[index].push_back(id);
  }
else {
    nb_count[index]++;
    nb_list[index].push_back(id);
  }
assert(nb);
nb->nb_expire = CURRENT_TIME +
                (1.5 * ALLOWED_HELLO_LOSS *
HELLO_INTERVAL);
LIST_INSERT_HEAD(&nbhead, nb, nb_link);
seqno += 2;
assert ((seqno%2) == 0);
}
```

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

AODV_Neighbor *nb = nbhead.lh_first;

for (unsigned int i = 1; i <=
nb_list[index].size()-1; i++)
  {
    if(nb_list[index][i]==id)
    {
      nb_list[index].erase
(nb_list[index].begin()+i);
      nb_count[index]--;
      break;
    }
  }

 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 4.7 Potongan modifikasi Kode Fungsi `nb_insert()` dan `nb_delete()`

4.2.2 Implementasi Menghitung *Forwarding Probability*

Setelah mendapatkan jumlah *node* tetangga yang telah disimpan dalam sebuah *counter* berbentuk array nb_count[id_int], tahap selanjutnya adalah menggunakan nilai tersebut pada persamaan DCF. Perhitungan *Forwarding Probability* (FP) dimulai pada saat suatu *node* menerima paket RREQ, maka *node* tersebut akan mulai mengolah data jumlah tetangga dengan menggunakan persamaan DCF.

Perhitungan FP ini dilakukan pada fungsi recvRequest() yang ada pada kode sumber aodv.cc yang terletak di ns2.35/aodv. Setiap *node* akan mulai menghitung nilai FP pada saat *node* tersebut telah menerima paket RREQ. Variabel jumlah tetangga nb_count[id_int] merupakan masukan pada algoritma perhitungan FP. Nilai nb_count akan digunakan pada persamaan NB. Selanjutnya nilai NB akan digunakan sebagai masukan dalam perhitungan persamaan DCF. Apabila nilai hasil persamaan DCF lebih dari satu maka nilai DCF akan menjadi satu. Setelah mendapat nilai DCF, maka dilakukan perhitungan persamaan DAF. Untuk mendapatkan Ra, dibutuhkan jumlah tetangga yang tidak dijangkau oleh dua *node* yang mengirim dan menerima paket RREQ. Sehingga untuk mendapatkan jumlah *node* yang tidak terjangkau, dapat dilakukan dengan mengiris daftar tetangga yang telah didapatkan sebelumnya. Dengan mengiris nb_list[id] dari kedua *node* yang salaing berhubungan, maka jumlah *node* yang tidak terjangkau dapat dihitung. Setelah mendapatkan nilai jumlah tetangga yang tidak terjangkau, maka nilai tersebut digunakan pada persamaan Ra. Jika nilai Ra sudah didapat, maka FP dapat dihitung dengan perkalikan DAF dan Ra.

Pada perancangan yang telah direncanakan pada bab sebelumnya, telah diubah menjadi kode C++ yang disisipkan pada file aodv.cc yang ada pada ns2.35/aodv. Implementasi kode terdapat sedikit perbedaan pada penamaan variabel. Hal ini dilakukan untuk *error checking* atau *debug* terhadap hasil tiap persamaan yang diimplementasikan pada kode. Potongan kode untuk perhitungan FP dapat dilihat pada Gambar 4.8.

```
double now = Scheduler::instance().clock();
double a=8.796112, b=37.29479, c=7.229256,
d=11.26162, m=5.094965;
double preNB = (double)nb_count[index] / c;
double NB=1+(pow(preNB,b));
double ad = a-d;
double nbm = (pow(NB,m));
double preDCF = ad/nbm;
double DCF=d+preDCF;

if(DCF > 1.0) DCF=1.0;

double DAF = DCF/nb_count[index];

int irisan = 0;
for (unsigned int i = 1; i <=
nb_list[index].size()-1; i++)
    {
      for (unsigned int j = 1; j <=
nb_list[ih->saddr()].size()-1; j++)
      {
        if(nb_list[index][i]==nb_list[ih-
>saddr()][j]) irisan++;
      }
    }

int nb_uncovered = std::abs(nb_count[index]-
irisan-1);

double Ra= (double)nb_uncovered/(double)
std::abs(nb_count[index]);
  double FP=DAF*Ra;
```

Gambar 4.8 Potongan modifikasi Kode Fungsi recvRequest()

4.2.3 Implementasi Pemilihan *Forwarding Node*

Tahap selanjutnya setelah mendapatkan nilai FP dari perhitungan yang sebelumnya telah dilakukan adalah memasuki tahap memilih keputusan suatu *node* dapat melakukan *rebroadcast* paket RREQ yang telah diterima atau tidak. Pemilihan ini berdasar pada nilai FP yang telah dihitung sebelumnya.

Algoritma pemilihan *node* ini telah diimplementasikan sesuai dengan perencanaan sebelumnya. Implementasi dilakukan pada fungsi recvRequest() pada kode sumber aodv.cc yang terletak pada ns2.35/aodv. Keputusan pemilihan *node* berdasar pada nilai FP dan dibandingkan dengan nilai real *random* yang berkisar antara 0 hingga 0.025. Jika nilai FP lebih besar atau sama dengan nilai real *random*, maka paket RREQ yang diterima akan di*forward*. Begitu juga dengan sebaliknya, jika nilai FP ternyata lebih kecil daripada nilai real *random*, maka paket RREQ yang diterima *node* tersebut akan

```
int r = rand()%250;

double random_num = r/10000.0;

if(FP>=random_num)
  {
    ih->saddr() = index;
    ih->daddr() = IP_BROADCAST;
    rq->rq_hop_count += 1;
    // Maximum sequence number seen en route
    if (rt) rq->rq_dst_seqno = max(rt-
>rt_seqno, rq->rq_dst_seqno);
    forward((aodv_rt_entry*) 0, p, DELAY);
  }
  else {
    Packet::free(p);  }
```

Gambar 4.9 Potongan Kode Untuk Pemilihan Forwarding Node

langsung di*drop* pada saat itu juga. Potongan kode untuk pemilihan Forwarding Node dapat dilihat pada Gambar 4.9.

Dengan proses penyeleksian *node* mana saja yang dapat menerima paket RREQ sudah dapat mengurangi jumlah paket *routing overhead* dan mengurangi jumlah *forwarded route request*. Dengan begitu akan mengurangi *cost* untuk pengiriman paket.

4.3 Implementasi Simulasi pada NS-2

Implementasi simulasi VANETs diawali dengan pendeskripsian lingkungan simulasi pada sebuah *file* berekstensi .tcl. *File* ini berisikan konfigurasi yang dibutuhkan untuk setiap *node* dan langkah-langkah yang dilakukan selama simulasi. Konfigurasi ini sangat menentukan berhasil tidaknya suatu simulasi. Jika konfigurasi yang digunakan tidak sesuai dengan pengaturan yang seharusnya, simulasi akan gagal untuk dijalankan. Maka dari itu, konfigurasi harus sesuai dengan standar yang ada. Potongan konfigurasi lingkungan simulasi dapat dilihat pada Gambar 4.10.

```
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 4.10 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 4.11.

```
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 4.11 Konfigurasi Lingkungan Simulasi

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 keluar sebuah *trace file* yang berisikan data mengenai apa saja yang terjadi selama simulasi dalam bentuk *plain text* 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 *Packet Delivery Ratio* (PDR), Rata-rata *End-to-End Delay* (E2E), dan *Routing Overhead* (RO).

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. Dengan begitu, melakukan perhitungan PDR melalui bantuan skrip awk. Skrip awk untuk menghitung PDR berdasarkan kedua informasi tersebut dapat dilihat pada lampiran A.6 Kode Skrip AWK *Packet Delivery Ratio*.

Dalam perhitungan PDR, kata kunci yang perlu diperhatikan dari *trace file* adalah layer AGT, karena kata kunci tersebut menunjukkan *event* yang bersangkutan dengan paket komunikasi data. Kemudian hitung jumlah paket yang dikirimkan dan paket yang diterima dengan menggunakan karakter pada kolom pertama sebagai *filter*, karena kolom pertama yang menunjukkan *event* yang terjadi dari sebuah paket. Setelah itu nilai PDR dapat dihitung dengan persamaan yang telah dijelaskan. *Pseudocode* untuk menghitung PDR dapat dilihat pada Gambar 4.12.

```
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 4.12 *Pseudocode* untuk Menghitung PDR

Perhitungan PDR pada Tugas Akhir ini dimulai dari sesi kedua karena implementasi Tugas Akhir ini diimplementasikan pada

sesi kedua. Contoh perintah pengeksekusian skrip awk untuk menganalisis *trace file* adalah awk -f pdr.awk scenario1.tr.

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

Perhitungan E2E telah dijelaskan melalui persamaan. 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 sayarat 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 *end-to-end delay* dapat dilihat pada Gambar 4.13.

```
sum_delay = 0
counter = 0
for i = 1 to the number of rows
     counter++
     if layer == AGT and event == s then
      start_time[packet_id] = time
     else if layer == AGT and event == r then
      end_time[packet_id] = time
     end if
     delay[packet_id] = end_time[packet_id] -
start_time[packet_id]
     sum_delay += delay[packet_id]
e2e = sum_delay / counter
```

Gambar 4.13 *Pseudocode* untuk Perhitungan Rata-Rata *End-to-End Delay*

Pada Tugas Akhir ini, perhitungan *end-to-end delay* juga dilakukan saat sesi kedua seperti perhitungan *packet delivery ratio*. Contoh perintah pengeksekusian skrip awk untuk menganalisis *trace file* adalah awk –f e2e.awk scenario1.tr.

4.4.3 Implementasi *Routing Overhead*

Perhitungan *routing overhead* telah dijelaskan pada persamaan 3.3. Skrip awk untuk menghitung *routing overhead* dapat dilihat pada lampiran A.8 Kode Skrip AWK *Routing Overhead*.

Seperti yang telah dijelaskan sebelumnya, *routing overhead* merupakan jumlah dari paket kontrol *routing* baik itu paket RREQ, paket RREP, maupun paket RERR. Dengan begitu, untuk mendapatkan *route request* yang perlu dilakukan adalah menjumlahkan tiap paket dengan *filter event sent* pada kolom pertama dan *event layer* RTR pada kolom ke-4. *Pseudocode* untuk menghitung *routing overhead* dapat dilihat pada Gambar 4.14.

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

Gambar 4.14 *Pseudocode* untuk Perhitungan *Routing Overhead*

Pada Tugas Akhir ini, perhitungan *routing overhead* juga dilakukan pada sesi kedua seperti perhitungan *packet delivery ratio* dan *end-to-end delay*. Contoh perintah pengeksekusian skrip awk untuk menganalisis *trace file* adalah awk -f ro.awk scenario1.tr.

4.4.4 Implementasi *Forwarded Route Request*

Perhitungan *Forwarded Route Request* telah dijelaskan pada persamaan 3.4. Skrip awk untuk menghitung *forwarded route request*

dapat dilihat pada lampiran A.9 Kode Skrip AWK *Forwarded Route Request*.

Seperti yang telah dijelaskan sebelumnya, *forwarded route request* adalah jumlah paket kontrol *route request* yang di*forward* per data paket ke *node* tujuan selama simulasi terjadi. Dengan begitu, untuk mendapatkan *forwarded route request* yang perlu dilakukan adalah menjumlahkan tiap paket dengan *filter event forwarded* pada kolom pertama, *event layer* RTR pada kolom ke-4 dan *event layer route request* pada kolom-19. *Pseudocode* untuk menghitung *forwarded route request* dapat dilihat pada Gambar 4.15.

```
rreqf = 0
for i = 1 to the number of rows
     if in a row contains "f" and RTR and route
request then
        rreqf++
     end if
```

Gambar 4.15 *Pseudocode* untuk Perhitungan *Forwarded Route Request*

Pada Tugas Akhir ini, perhitungan *forwarded route request* juga dilakukan pada sesi kedua seperti perhitungan *packet delivery ratio*, *end-to-end delay*, dan *routing overhead*. 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) Core (TM) i5-4200M CPU @ 2.50GHz
Sistem Operasi	Linux Mint 18 Cinnamon
Linux Kernel	Linux kernel 4.4
Memori	8.0 GB
Penyimpanan	80 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

.

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 *packet delivery ratio*, rata-rata *end-to-end delay*, *routing overhead*, dan *forwarded route request* menggunakan kode pada masing-masing 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*.

5.2 Hasil Uji Coba

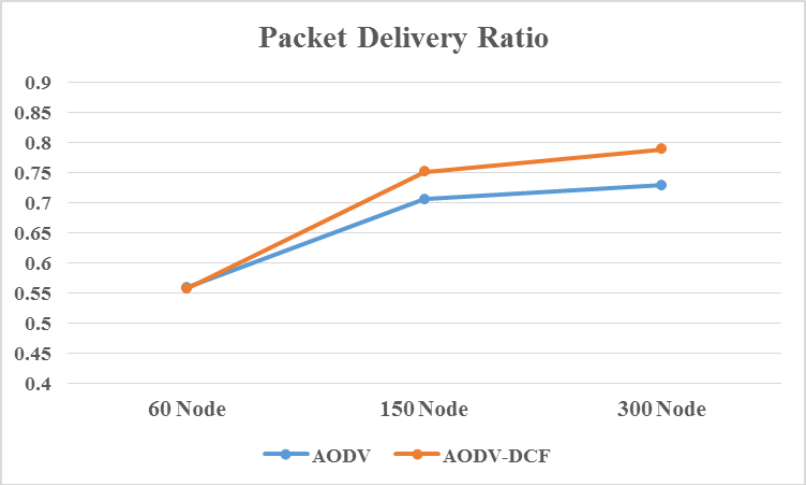
Hasil uji coba modifikasi protokol AODV pada lingkungan VANETs berbasis skenario grid dan skenario real yang telah dilakukan pada Tugas Akhir ini dapat dilihat sebagai berikut:

5.2.1 Hasil Uji Coba Skenario *Grid*

Pengujian pada skenario *grid* digunakan untuk melihat perbandingan *packet delivery ratio*, rata-rata *end-to-end delay*, *routing overhead*, dan *forwarded route request* antara *routing protocol* AODV dan *routing protocol* AODV-DCF dalam pemilihan *node* yang dapat menerima paket *route request*. Pengujian dilakukan sesuai dengan perancangan *parameter* lingkungan simulasi yang dapat dilihat pada Tabel 3.2.

Pengambilan data uji PDR, rata-rata *end-to-end delay*, *routing overhead*, dan *forwarded route request* dengan luas area 1300 m x 1300 m dan *node* sebanyak 60 untuk lingkungan yang jarang, 150 untuk lingkungan yang sedang, dan 300 untuk lingkungan yang padat dilakukan pada kecepatan standar yaitu 20 m/s. Untuk uji coba setiap lingkungan terdapat 10 skenario yang berbeda dan dilakukan sebanyak minimal 5 kali, lalu data yang diambil merupakan hasil rata-rata dari kelima data untuk setiap skenario. Hal ini dilakukan agar mendapat hasil yang akurat pada setiap skenario karena pada teknik pengambilan keputusan suatu *node* menjadi *forwarding node* yang akan melakukan *rebroadcast*, terdapat nilai random yang tidak akan sama pada setiap uji coba yang dilakukan walau dengan skenario yang sama.

Hasil pengambilan data *packet delivery ratio* pada skenario *grid* 60, 150, dan 300 *node* dengan menggunakan AODV dan AODV-DCF dapat dilihat pada Gambar 5.1.



Gambar 5.1 Grafik Rata-Rata *Packet Delivery Ratio* pada Skenario *Grid*

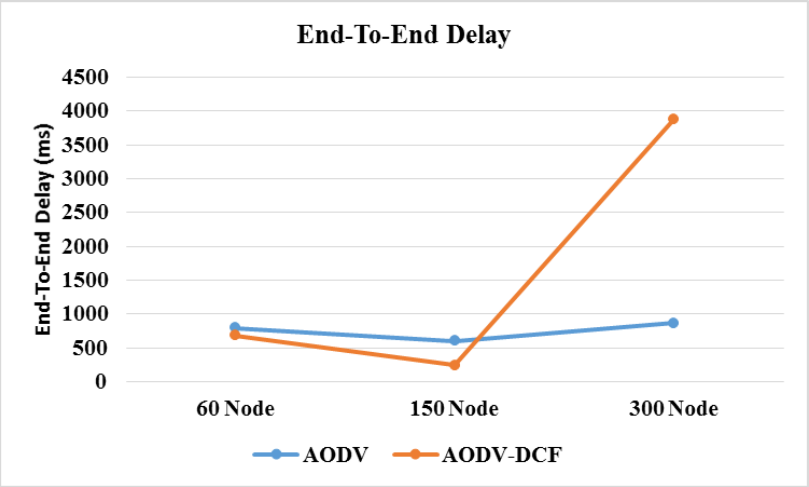
Berdasarkan grafik pada Gambar 5.1 dapat dilihat bahwa *routing protocol* AODV-DCF dan juga *routing protocol* AODV mengalami kenaikan *packet delivery ratio* yang cukup signifikan pada lingkungan yang padat. Pada lingkungan yang jarang dengan jumlah 60 *node*, tidak menghasilkan selisih PDR atau memiliki nilai PDR yang sama antara *routing protocol* AODV-DCF dan *routing protocol* AODV. Pada lingkungan yang sedang dengan jumlah 150 *node*, menghasilkan perbedaan selisih PDR sebesar 0,045, atau naik menjadi sekitar 6,43% dimana *routing protocol* AODV-DCF unggul dalam hal PDR tersebut dari AODV. Pada lingkungan yang padat dengan jumlah 300 *node*, menghasilkan perbedaan selisih PDR sebesar 0,06, atau naik menjadi sekitar 8,31% dimana *routing protocol* AODV-DCF juga unggul dalam hal PDR tersebut.

Dapat dilihat rata-rata kenaikan yang terjadi untuk PDR adalah 4,91%. Jika ketiga lingkungan tersebut dibandingkan dapat dilihat bahwa dengan jumlah *node* yang lebih banyak atau pada lingkungan dengan *node* yang padat, menghasilkan PDR yang lebih bagus daripada di lingkungan dengan jumlah *node* yang jarang

maupun yang sedang baik untuk *routing protocol* AODV maupun *routing protocol* AODV-DCF. Dapat dilihat pula bahwa AODV-DCF menghasilkan PDR yang lebih bagus daripada AODV dengan jumlah selisih PDR yang cukup signifikan.

Pada grafik Gambar 5.1 terlihat bahwa hasil uji coba untuk rata-rata PDR terdapat tren naik yang menunjukan bahwa rasio keberhasilan sebuah paket sampai hingga ke tujuan semakin baik. Tren kenaikan PDR terjadi seiring dengan bertambah padatnya *node* karena semakin padat lingkungan semakin stabil sebuah rute yang terbentuk, sehingga kemungkinan sebuah rute akan terputus semakin kecil. Dan pada AODV-DCF terdapat kenaikan PDR yang menunjukan bahwa pemilihan rute menjadi lebih baik.

Hasil pengambilan data rata-rata untuk *end-to-end delay* (E2E) pada skenario *grid* dengan jumlah *node* 60, 150, dan 300 dapat dilihat pada Gambar 5.2.



Gambar 5.2 Grafik Rata-Rata *End-to-End Delay* pada Skenario *Grid*

Berdasarkan grafik pada Gambar 5.2 dapat dilihat bahwa rata-rata *end-to-end delay* antara *routing protocol* AODV dan AODV-DCF mengalami perubahan yang fluktuatif. Pada lingkungan yang

jarang dengan jumlah 60 *node*, terjadi perbedaan selisih *end-to-end delay* sebesar 111,8 ms antara *routing protocol* AODV dengan *routing protocol* AODV-DCF atau mengalami penurunan sebesar 14,14%, dimana *routing protocol* AODV-DCF unggul dalam hal *end-to-end delay* tersebut. Sedangkan pada lingkungan sedang dengan jumlah 150 *node*, terjadi perbedaan selisih *end-to-end delay* sebesar 356,98 ms antara *routing protocol* AODV dengan *routing protocol* AODV-DCF atau mengalami penurunan sebesar 59,61%, dimana *routing protocol* AODV-DCF lebih unggul dalam hal *end-to-end delay* tersebut. Pada lingkungan yang padat dengan jumlah 300 *node*, terjadi perbedaan selisih *end-to-end delay* sebesar 3010,36 ms antara *routing protocol* AODV dengan *routing protocol* AODV-DCF atau mengalami kenaikan sebesar 347,23%, dimana *routing protocol* AODV jauh lebih unggul dalam hal *end-to-end delay* tersebut.

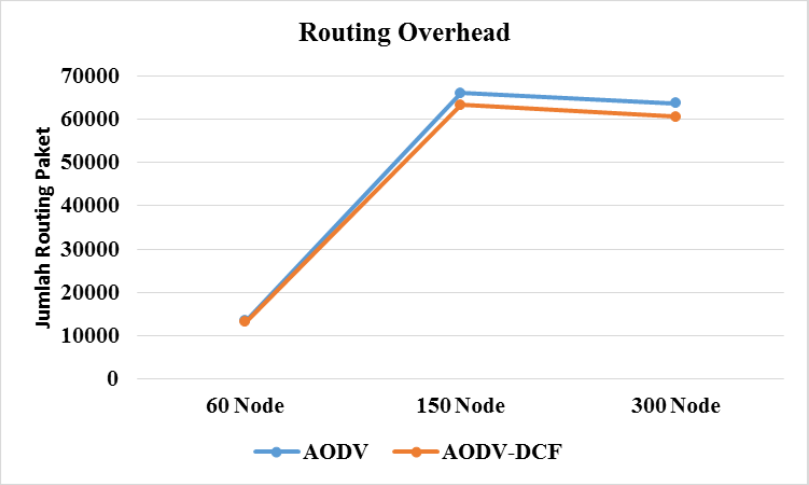
Jika ketiga lingkungan tersebut dibandingkan, memang pada lingkungan yang jarang, sedang, maupun padat *routing protocol* AODV-DCF tidak selalu lebih unggul dalam hal *end-to-end delay*. Juga, *routing protocol* AODV tidak selalu lebih unggul. Tetapi memang terlihat bahwa pada lingkungan jarang dan sedang, *routing protocol* AODV-DCF lebih unggul dalam hal *end-to-end delay* daripada routing protocol AODV. Akan tetapi dari grafik yang terlihat memang tidak bisa dibaca tren dari grafik *end-to-end delay* karena grafik tersebut menunjukkan perubahan yang fluktuatif.

Grafik *end-to-end delay* yang fluktuatif memang tidak bisa dianalisis dikarenakan delay tergantung dari waktu rata-rata paket yang terkirim sampai *node* tujuan. Jika paket yang terkirim sampai *node* tujuan semakin banyak, maka semakin beragam pula jumlah delay tersebut. Jumlah paket yang diterima semakin banyak dan juga jika semakin banyak pula node tersebut pada suatu lingkungan, maka rute yang terbentuk juga akan semakin banyak. Salah satu hal yang dapat mempengaruhi fluktuasi *end-to-end delay* adalah posisi dari *node* yang ada pada simulasi. Posisi *node* yang tidak tepat pada jalur yang *random* juga berpengaruh dalam terbentuknya rute.

Pada grafik *end-to-end delay* juga tidak dapat terlihat tren naik ataukah tren turun yang dihasilkan saat pengambilan data. Hal

ini terjadi karena *end-to-end delay routing protocol* AODV-DCF pada lingkungan padat atau lingkungan dengan jumlah node 300, mengalami kenaikan yang cukup besar jika dibandingkan dengan *routing protocol* AODV. Jika dilihat pada teknik pengambilan data yang melibatkan 10 skenario, kenaikan yang cukup besar hanya terjadi pada 2 skenario yang menyebabkan rata-rata *end-to-end delay* menjadi sangat besar. Pada 2 skenario ini keputusan pemilihan *forwarding node* dapat diasumsikan memilih rute yang panjang. Dimana semakin panjang rute yang terbentuk, semakin besar pula nilai *end-to-end delay* yang dihasilkan pada skenario tersebut.

Untuk hasil pengambilan data *routing overhead* pada skenario *grid* 60 *node*, 150 *node*, dan 300 *node* dapat dilihat pada Gambar 5.3.



Gambar 5.3 Grafik Rata-Rata *Routing Overhead* pada Skenario *Grid*

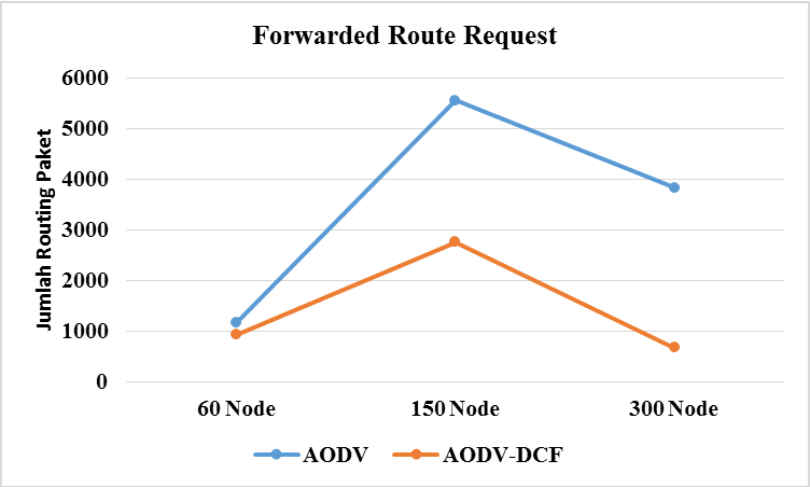
Berdasarkan grafik pada Gambar 5.3 dapat dilihat bahwa *routing protocol* AODV-DCF dan juga *routing protocol* AODV mengalami penurunan yang cukup signifikan pada *routing overhead*. Pada lingkungan yang jarang dengan jumlah 60 *node*, menghasilkan perbedaan selisih *routing overhead* sebesar 228,48 atau mengalami

penurunan sebesar 1,70%, dimana *routing protocol* AODV-DCF unggul dalam hal *routing overhead* tersebut karena menghasilkan *routing overhead* yang lebih rendah dari *routing* AODV. Pada lingkungan yang sedang dengan jumlah 150 *node*, menghasilkan perbedaan selisih *routing overhead* sebesar 2772,2 atau mengalami penurunan sebeasr 4,20%, dimana *routing protocol* AODV-DCF juga unggul dalam hal *routing overhead* tersebut dari AODV. Pada lingkungan yang padat dengan jumlah 300 *node*, menghasilkan perbedaan selisih *routing overhead* sebesar 3178,4 atau mengalami penurunan sebesar 4,99%, dimana *routing protocol* AODV-DCF juga unggul dalam hal *routing overhead* tersebut.

Dapat dilihat rata-rata penurunan yang terjadi untuk *routing overhead* adalah sebesar 3,63%. Jika ketiga lingkungan tersebut dibandingkan berdasarkan selisih RO, dapat dilihat bahwa terdapat tren tingkat penurunan RO. Dapat dilihat pula bahwa AODV-DCF menghasilkan RO yang lebih bagus atau dalam hal ini lebih rendah daripada AODV dengan jumlah selisih RO yang cukup signifikan. Namun, terdapat fluktuatif pada jumlah RO yang dihasilkan. Pada lingkungan dengan *node* yang sedang, menghasilkan jumlah *routing overhead* yang lebih banyak atau lebih besar daripada di lingkungan dengan jumlah *node* yang padat, baik untuk *routing protocol* AODV maupun *routing protocol* AODV-DCF.

Dapat dilihat pada grafik rata-rata RO pada kedua *routing protocol*, tren yang dihasilkan untuk rata-rata RO tidak dapat ditentukan. Hal ini terjadi karena adanya sedikit penurunan jumlah RO dari lingkungan sedang menuju jumlah RO pada lingkungan padat. Salah satu hal yang dapat mempengaruhi hal ini adalah karena terjadi banyaknya paket RERR pada lingkungan yang sedang. Sehingga pada lingkungan yang padat, jumlah RO akan menjadi lebih sedikit karena rute yang tercipta lebih stabil.

Untuk hasil pengambilan data *forwarded route request* (RREQ F) pada skenario *grid* 60 *node*, 150 *node*, dan 300 *node* dapat dilihat pada Gambar 5.4.



Gambar 5.4 Grafik Rata-Rata *Forwarded route request* pada Skenario *Grid*

Berdasarkan grafik pada Gambar 5.4 dapat dilihat bahwa *routing protocol* AODV-DCF dan juga *routing protocol* asli mengalami penurunan jumlah *forwarded route request* (RREQ F). Pada lingkungan yang jarang dengan jumlah 60 *node*, menghasilkan perbedaan selisih *forwarded route request* sebesar 232,4 atau mengalami penurunan sebesar 19.95%, dimana *routing protocol* AODV-DCF unggul dalam hal *forwarded route request* tersebut karena menghasilkan *forwarded route request* yang lebih rendah dari *routing protocol* AODV. Pada lingkungan yang sedang dengan jumlah 150 *node*, menghasilkan perbedaan selisih *forwarded route request* sebesar 2804,2 atau mengalami penurunan sebesar 50,44%, dimana *routing protocol* AODV-DCF juga unggul dalam hal *forwarded route request* tersebut dari *forwarded route request* AODV. Pada lingkungan yang padat dengan jumlah 300 *node*, menghasilkan perbedaan selisih *forwarded route request* sebesar 3156,7 atau mengalami penurunan sebesar 82,35%, dimana *routing protocol* AODV-DCF juga unggul dalam hal *forwarded route request* tersebut.

Dapat dilihat rata-rata penurunan yang terjadi untuk *forwarded route request* adalah sebesar 50.91%. Jika ketiga lingkungan tersebut dibandingkan dapat dilihat bahwa dengan jumlah *node* yang padat, menghasilkan selisih *forwarded route request* yang lebih bagus daripada di lingkungan dengan jumlah *node* yang jarang maupun yang jarang baik untuk *routing protocol* AODV maupun *routing protocol* AODV-DCF. Dapat dilihat pula bahwa AODV-DCF menghasilkan *forwarded route request* yang lebih bagus atau dalam hal ini lebih rendah daripada AODV dengan jumlah selisih *forwarded route request* yang cukup signifikan.

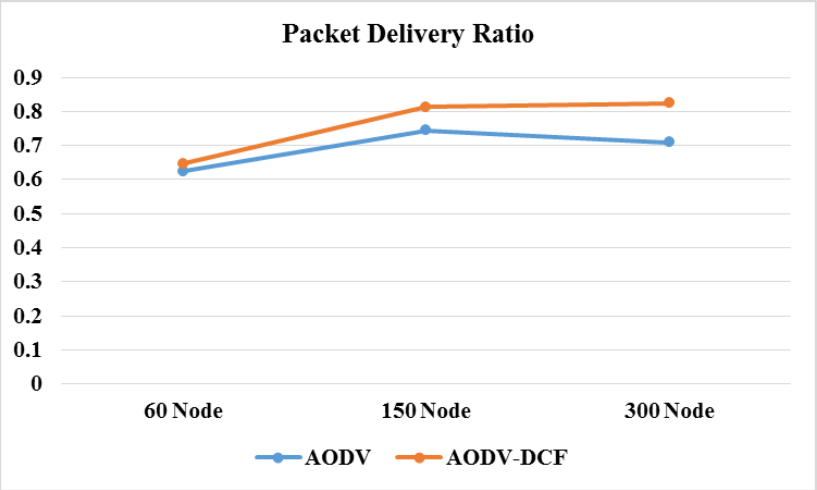
Pada grafik rata-rata *forwarded route request* tidak terdapat tren naik atau turun. Tren yang terjadi cenderung fluktuatif yang disebabkan turunnya jumlah F-RREQ dari lingkungan sedang ke lingkungan padat. RREQ merupakan sebuah *control packet* yang merupakan bagian dari proses *route discovery* atau saat terjadi kerusakan pada sebuah rute yang telah terbentuk sebelumnya. Salah satu hal yang menjadi penyebab turunnya jumlah F-RREQ pada lingkungan padat adalah stabilitas rute yang terbentuk pada saat *route discovery*. Stabilitas rute yang baik menyebabkan *link breakage* atau kerusakan rute menjadi jarang terjadi, sehingga mekanisme *route discovery* tidak perlu dilakukan berulang-ulang.

5.2.2 Hasil Uji Coba Skenario *Real*

Pengujian pada skenario *real* digunakan untuk melihat perbandingan *packet delivery ratio*, rata-rata *end-to-end delay*, *routing overhead*, dan *forwarded route request* antara *routing protocol* AODV dan *routing protocol* AODV-DCF dalam pemilihan *node* yang dapat menerima paket *route request*. Pengujian dilakukan sesuai dengan perancangan *parameter* lingkungan simulasi yang dapat dilihat pada Tabel 3.2.

Pengambilan data uji PDR, rata-rata *end-to-end delay*, *routing overhead*, dan *forwarded route request* dengan luas area 1500 m x 1500 m dan *node* sebanyak 60 untuk lingkungan yang jarang, 150 untuk lingkungan yang sedang, dan 300 untuk lingkungan yang. Seperti pada uji coba skenario grid, uji coba setiap lingkungan

terdapat 10 skenario yang berbeda dan dilakukan sebanyak minimal 5 kali, lalu data yang diambil merupakan hasil rata-rata dari kelima data untuk setiap skenario. Hal ini dilakukan agar mendapat hasil yang akurat pada setiap skenario karena pada teknik pengambilan keputusan suatu *node* menjadi *forwarding node* yang akan melakukan *rebroadcast*, terdapat nilai random yang tidak akan sama pada setiap uji coba yang dilakukan walau dengan skenario yang sama. Hasil pengambilan data *packet delivery ratio* (PDR) pada skenario *real* 60, 150, dan 300 *node* dengan menggunakan AODV masing dapat dilihat pada Gambar 5.5.



Gambar 5.5 Grafik Rata-Rata *Packet Delivery Ratio* pada Skenario *Real*

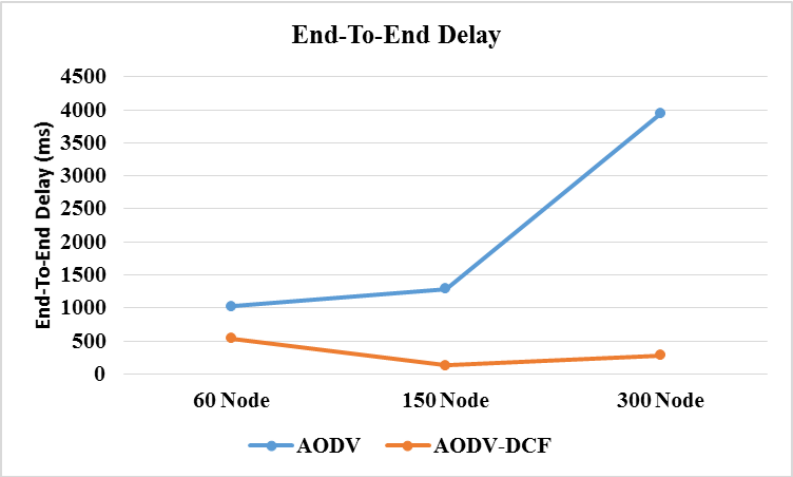
Berdasarkan grafik pada Gambar 5.5 dapat dilihat bahwa *routing protocol* AODV-DCF dan juga *routing protocol* AODV mengalami kenaikan yang signifikan pada *packet delivery ratio*. Pada lingkungan yang jarang dengan jumlah 60 *node*, menghasilkan perbedaan selisih PDR sebesar 0,022 atau mengalami kenaikan sebesar 3.51%, dimana *routing protocol* AODV-DCF unggul dalam hal PDR tersebut. Pada lingkungan yang sedang dengan jumlah 150 *node*, menghasilkan perbedaan selisih PDR sebesar 0.07 atau

mengalami kenaikan sebesar 9.39%, dimana *routing protocol* AODV-DCF juga unggul dalam hal PDR tersebut dari AODV. Pada lingkungan yang padat dengan jumlah 300 *node*, menghasilkan perbedaan selisih PDR sebesar 0.11, atau mengalami kenaikan sebesar 16.20%, dimana *routing protocol* AODV-DCF juga unggul dalam hal PDR tersebut.

Dapat dilihat rata-rata kenaikan yang terjadi untuk PDR adalah sebesar 10,52%. Jika ketiga lingkungan tersebut dibandingkan dapat dilihat bahwa dengan jumlah *node* yang lebih banyak atau pada lingkungan dengan *node* yang padat, menghasilkan PDR yang lebih bagus daripada di lingkungan dengan jumlah *node* yang sedang maupun yang jarang pada *routing protocol* AODV-DCF. Namun pada *routing protocol* AODV, dengan jumlah node yang padat terdapat penurunan PDR. Salah satu hal yang mempengaruhi hal ini adalah posisi *node* yang kurang sesuai pada jalur yang ada. Karena pada skenario real, jalur yang tercipta merupakan hasil konversi dari peta asli. Jika dilihat berdasarkan tren tingkat selisih kenaikan PDR, maka pada *routing protocol* AODV-DCF menghasilkan PDR yang lebih bagus daripada *routing protocol* AODV dengan jumlah selisih PDR yang cukup signifikan.

Pada grafik rata-rata PDR hasil uji coba pada skenario *real* terlihat bahwa hasil uji coba untuk rata-rata PDR terdapat tren naik yang menunjukan bahwa rasio keberhasilan sebuah paket sampai hingga ke tujuan semakin baik. Tren kenaikan PDR terjadi seiring dengan bertambah padatnya *node* karena semakin padat lingkungan semakin stabil sebuah rute yang terbentuk, sehingga kemungkinan sebuah rute akan terputus semakin kecil. Dan pada AODV-DCF terdapat kenaikan PDR yang menunjukan bahwa pemilihan rute menjadi lebih baik.

Hasil pengambilan data rata-rata untuk *end-to-end delay* (E2E) pada skenario *real* dengan jumlah *node* 60, 150, dan 300 dapat dilihat pada Gambar 5.6



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

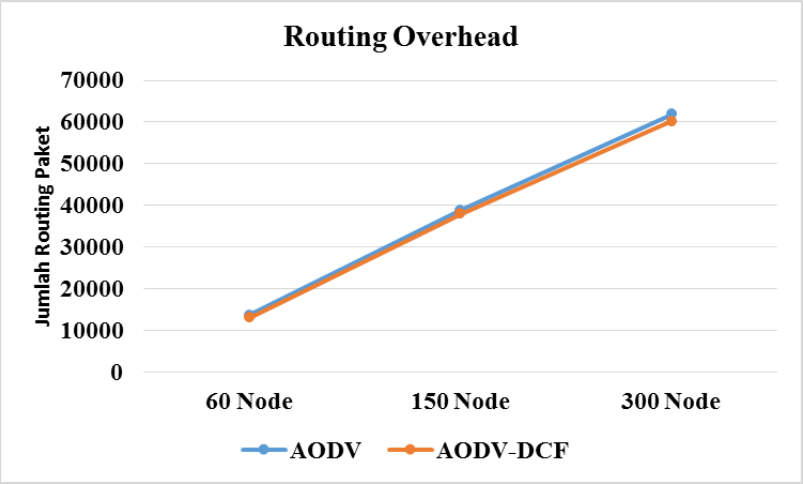
Berdasarkan grafik pada Gambar 5.6 dapat dilihat bahwa rata-rata *end-to-end delay* antara *routing protocol* AODV mengalami perubahan yang fluktuatif tetapi pada *routing protocol* AODV-DCF mengalami penurunan yang signifikan. Pada lingkungan yang jarang dengan jumlah 60 *node*, terjadi perbedaan selisih *end-to-end delay* sebesar 489,58 ms antara *routing protocol* AODV dengan *routing protocol* AODV-DCF atau mengalami penurunan sebesar 47,52%, dimana *routing protocol* AODV-DCF lebih unggul dalam hal *end-to-end delay* tersebut. Sedangkan pada lingkungan sedang dengan jumlah 150 *node*, terjadi perbedaan selisih *end-to-end delay* sebesar 1164,54 ms antara *routing protocol* AODV dengan *routing protocol* AODV-DCF atau mengalami penurunan sebesar 89,99%, dimana *routing protocol* AODV-DCF lebih unggul dalam hal *end-to-end delay* tersebut. Pada lingkungan yang padat dengan jumlah 300 *node*, terjadi perbedaan selisih *end-to-end delay* sebesar 3656,30 ms antara *routing protocol* AODV dengan *routing protocol* AODV-DCF atau mengalami penurunan sebesar 92,79%, dimana *routing protocol* AODV-DCF jauh lebih unggul dalam hal *end-to-end delay* tersebut.

Dapat dilihat rata-rata penurunan yang terjadi untuk *end-to-end delay* adalah sebesar 76,76%. Jika ketiga lingkungan tersebut

dibandingkan dapat dilihat bahwa dengan jumlah *node* yang lebih banyak atau pada lingkungan dengan *node* yang padat, menghasilkan selisih *end-to-end delay* yang lebih bagus daripada di lingkungan dengan jumlah *node* yang sedang maupun yang jarang baik untuk routing protocol AODV maupun routing protocol AODV-DCF. Dapat dilihat pula bahwa AODV-DCF menghasilkan *end-to-end delay* yang lebih bagus daripada AODV dengan jumlah selisih *end-to-end delay* yang cukup signifikan.

Jika dilihat pada grafik *end-to-end delay* pada *routing protocol* AODV-DCF terjadi fluktuatif pada lingkungan padat, dimana terjadi kenaikan yang cukup kecil. Salah satu faktor yang mempengaruhi terjadinya hal semacam ini adalah rute yang terbentuk mengarah pada jalur yang memiliki *node* yang lebih banyak atau lebih jauh dari *node* tujuan. Terjadi pembentukan rute yang sedikit memutar untuk sampai ke *node* tujuan. Sehingga rata-rata *end-to-end delay* menjadi lebih besar karena banyaknya *node* yang dilewati dan tidak melewati rute yang terbaik, melainkan terbentuk rute melebar.

Untuk hasil pengambilan data *routing overhead* pada skenario *grid* dengan variasi jumlah node 60 *node*, 150 *node*, dan 300 *node* dapat dilihat pada Gambar 5.7.



Gambar 5.7 Grafik *Routing Overhead* pada Skenario *Real*

Berdasarkan grafik pada Gambar 5.7 dapat dilihat bahwa *routing protocol* AODV-DCF dan juga *routing protocol* AODV mengalami kenaikan yang signifikan pada *routing overhead*. Pada lingkungan yang jarang dengan jumlah 60 *node*, menghasilkan perbedaan selisih *routing overhead* sebesar 614,52 atau mengalami penurunan sebesar 4,51%, dimana *routing protocol* AODV-DCF unggul dalam hal *routing overhead* tersebut karena menghasilkan *routing overhead* yang lebih rendah dari *routing* AODV. Pada lingkungan yang sedang dengan jumlah 150 *node*, menghasilkan perbedaan selisih *routing overhead* sebesar 1016,64 atau mengalami penurunan sebesar 2.61%, dimana *routing protocol* AODV-DCF juga unggul dalam hal *routing overhead* tersebut dari AODV. Pada lingkungan yang padat dengan jumlah 300 *node*, menghasilkan perbedaan selisih *routing overhead* sebesar 1611,8 atau mengalami penurunan sebesar 2,61%, dimana *routing protocol* AODV-DCF juga unggul dalam hal *routing overhead* tersebut.

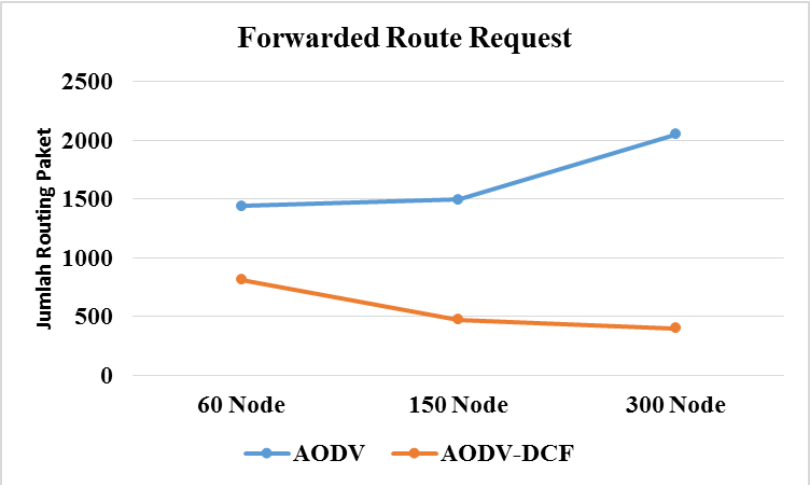
Dapat dilihat rata-rata penurunan yang terjadi untuk *routing overhead* adalah sebesar 3,24%. Jika ketiga lingkungan tersebut dibandingkan dapat dilihat bahwa dengan jumlah *node* yang lebih banyak atau pada lingkungan dengan *node* yang padat, menghasilkan tingkat penurunan *routing overhead* yang sama dengan di lingkungan dengan jumlah *node* yang sedang untuk *routing protocol* AODV maupun *routing protocol* AODV-DCF.

Jika dilihat dari tingkat penurunan *routing overhead*, terjadi tren penurunan pada tingkat penurunan *routing overhead*. Pada lingkungan jarang atau 60 *node*, tingkat penurunan *routing overhead* lebih besar dari tingkat penurunan *routing overhead* pada lingkungan sedang dan padat. Namun secara keseluruhan dapat dilihat bahwa *routing protocol* AODV-DCF menghasilkan *routing overhead* yang lebih bagus atau dalam hal ini lebih rendah daripada AODV walau dengan jumlah selisih *routing overhead* yang tidak terlalu jauh.

Dapat dilihat pada grafik rata-rata RO pada kedua *routing protocol*, tren yang dihasilkan untuk rata-rata RO adalah tren naik. Hal ini terjadi karena semakin banyaknya jumlah *node* maka semakin banyak pula *control packet* yang ada pada jaringan. Jika dibandingkan

dengan skenario *grid*, dapat dikatakan tingkat stabilitas rute yang terbentuk antara simulasi dengan lingkungan sedang dan lingkungan padat memiliki tingkat yang sama. Sehingga tidak terjadi anomali yang ada pada hasil uji coba dengan skenario *grid*.

Untuk hasil pengambilan data *forwarded route request* (RREQ F) pada skenario *grid* 60 *node*, 150 *node*, dan 300 *node* dapat dilihat pada Gambar 5.8.



Gambar 5.8 Grafik Rata-Rata *Forwarded route request* pada Skenario *Real*

Berdasarkan grafik pada Gambar 5.8 dapat dilihat bahwa *routing protocol* AODV-DCF dan juga *routing protocol* asli mengalami perubahan *forwarded route request* (RREQ F) yang signifikan. Pada lingkungan yang jarang dengan jumlah 60 *node*, menghasilkan perbedaan selisih *forwarded route request* sebesar 628,36 atau mengalami penurunan sebesar 43,60%, dimana *routing protocol* AODV-DCF unggul dalam hal *forwarded route request* tersebut karena menghasilkan *forwarded route request* yang lebih rendah dari *routing protocol* AODV. Pada lingkungan yang sedang

dengan jumlah 150 *node*, menghasilkan perbedaan selisih *forwarded route request* sebesar 1026,78 atau mengalami penurunan sebesar 68,47%, dimana *routing protocol* AODV-DCF juga unggul dalam hal *forwarded route request* tersebut dari *forwarded route request* AODV. Pada lingkungan yang padat dengan jumlah 300 *node*, menghasilkan perbedaan selisih *forwarded route request* sebesar 1648,8 atau mengalami penurunan sebesar 80,39%, dimana *routing protocol* AODV-DCF juga unggul dalam hal *forwarded route request* tersebut.

Dapat dilihat rata-rata penurunan yang terjadi untuk *forwarded route request* adalah sebesar 64,16%. Jika ketiga lingkungan tersebut dibandingkan dapat dilihat bahwa dengan jumlah *node* yang padat, menghasilkan *forwarded route request* yang lebih bagus atau lebih sedikit daripada di lingkungan dengan jumlah *node* yang sedang maupun yang jarang baik untuk *routing protocol* AODV maupun *routing protocol* AODV-DCF. Dapat dilihat pula bahwa AODV-DCF menghasilkan *forwarded route request* yang lebih bagus atau dalam hal ini lebih rendah daripada AODV dengan jumlah selisih *forwarded route request* yang cukup signifikan.

Pada grafik rata-rata *forwarded route request* terdapat tren turun pada grafik AODV-DCF. Penurunan ini terjadi karena rute yang terbentuk dengan menggunakan *routing protocol* AODV-DCF dapat dikatakan memiliki stabilitas rute yang baik, sehingga mekanisme *route maintenance* ataupun *route discovery* tidak dilakukan sebanyak yang dilakukan pada *routing protocol* AODV. Maka dari itu, jumlah *routing packet* yang ada pada jaringan, khususnya untuk F-RREQ mengalami penurunan seiring dengan bertambah padatnya *node* pada lingkungan.

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. Jumlah forwarding *node* yang melakukan *rebroadcast* dapat direduksi dengan menambahkan perhitungan jumlah tetangga pada *routing protocol* AODV dan membatasi *node* mana saja yang bisa me*rebroadcast* paket RREQ dengan menggunakan perhitungan Forwarding Probability (FP) yang berdasarkan pada metode Dynamic Connectivity Factor (DCF).
2. Dampak pembatasan *forwarding node* terhadap performa *routing protocol* AODV secara keseluruhan untuk skenario *real* menghasilkan peningkatan rata-rata *packet delivery ratio* sebesar 7,35%, rata-rata penurunan *routing overhead* sebesar 3,44%, dan juga rata-rata penurunan *forwarded route request* sebesar 57,53%.

6.2 Saran

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

1. Untuk mendapatkan hasil uji coba yang lebih baik, dapat dilakukan uji coba yang lebih banyak, misal membuat skenario real dengan peta yang lebih luas dan dilakukan lebih dari 10 kali percobaan untuk tiap skenario.
2. Nilai random yang ada pada pemilihan forwarding *node* yang melakukan *rebroadcast* dapat diganti dengan metode

pemilihan yang lebih adil, misal dengan mengubah nilai random menjadi nilai statis.

3. Kedepannya bisa mengimplementasikan dengan menambahkan aspek lain yang dijadikan untuk pembatasan *forwarding node* seperti energi, kecepatan, arah, dan lain-lain.

DAFTAR PUSTAKA

[1] L. Khan, N. Ayub dan A. Saeed, “Anycast Based Routing in Vehicular Adhoc Networks (VANETS) using Vanetmobisim,” COMSATS IIT Abbottabad, Pakistan, 2009.

[2] N. M. Mittal dan S. Choudhary, “Comparative Study of Simulators for Vehicular Ad-hoc Networks (VANETs),” IJETAE, Haryana, India, 2014.

[3] “OpenStreetMap,” OpenStreetMap, [Online]. Available: https://www.openstreetmap.org/about. [Diakses 6 December 2017].

[4] “JOSM,” JOSM, [Online]. Available: http://wiki.openstreetmap.org/wiki/JOSM. [Diakses 6 December 2017].

[5] “SUMO,” SUMO, [Online]. Available: http://www.sumo.dlr.de/userdoc/Sumo_at_a_Glance.html. [Diakses 6 December 2017].

[6] A. V. Aho, B. W. Kerninghan dan P. J. Weinberger, The AWK Programming Language, Boston: Addison-Wesley, 1988.

[7] A. Mohamed E. Ejmaa, S. Subramaniam, Z. Ahmad Zukarnain dan Z. Mohd Hanapi, “Neighbor-Based Dynamic Connectivity Factor Routing Protocol for Mobile Ad Hoc Network,” *IEEE,* vol. 4, p. 12, 2016.

[8] K. Majumder dan S. K. Sarkar, “Performance Analysis of AODV and DSR Routing Protocols in Hybrid Network Scenario,” IEEE, Gujarat, India, 2009.

[9] M. Tamilarasi, S. S. V. R, U. M. Haputhantiri, C. Somathilaka, N. R. Babu, S. Chandramathi dan T. G. Palanivelu, “Scalability Improved DSR Protocol for MANETs,” IEEE, Puducherry, 2007.

[10] S. Y. Dong, “Optimization of OLSR Routing Protocol in UAV ad Hoc Network,” IEEE, Sichuan, 2016.

[11] S. N. Ferdous dan M. S. Hossain, “Randomized Energy-Based AODV Protocol For Wireless Ad-Hoc Network,” IEEE, Dhaka, 2016.

[12] X. Ming Zhang, E. Bo Wang, J. Jing Xia dan D. Keun Sung, “A Neighbor Coverage-Based Probabilistic Rebroadcast for Reducing Routing Overhead in Mobile Ad Hoc Networks,” *IEEE*, vol. 12, p. 10, 2013.

LAMPIRAN

A.1 Kode Fungsi AODV::recvRequest()

```
void
AODV::recvRequest(Packet *p) {
struct hdr_aodv_request *rq =
HDR_AODV_REQUEST(p);
struct hdr_ip *ih = HDR_IP(p);
aodv_rt_entry *rt;
aodv_rt_entry *rt0;

  /*
   * Drop if:
   *      - I'm the source
   *      - I recently heard this request.
   */

  if(rq->rq_src == index) {
#ifdef DEBUG
    fprintf(stderr, "%s: got my own
REQUEST\n", __FUNCTION__);
#endif // DEBUG
    Packet::free(p);
    return;
  }

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

#ifdef DEBUG
   fprintf(stderr, "%s: discarding
request\n", __FUNCTION__);
#endif // DEBUG
   Packet::free(p);
   return;
 }
```

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

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

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

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

   if ( (rq->rq_src_seqno > rt0->rt_seqno )
||
      ((rq->rq_src_seqno == rt0->rt_seqno)
&&
       (rq->rq_hop_count < rt0->rt_hops)) )
{
   // If we have a fresher seq no. or
lesser #hops for the
   // same seq no., update the rt entry.
Else don't bother.
rt_update(rt0, rq->rq_src_seqno, rq-
>rq_hop_count, ih->saddr(),
             max(rt0->rt_expire,
(CURRENT_TIME + REV_ROUTE_LIFE)) );
     if (rt0->rt_req_timeout > 0.0) {
```

```
      // Reset the soft state and
     // Set expiry time to CURRENT_TIME +
ACTIVE_ROUTE_TIMEOUT
     // This is because route is used in
the forward direction,
     // but only sources get benefited by
this change
       rt0->rt_req_cnt = 0;
       rt0->rt_req_timeout = 0.0;
       rt0->rt_req_last_ttl = rq-
>rq_hop_count;
       rt0->rt_expire = CURRENT_TIME +
ACTIVE_ROUTE_TIMEOUT;
     }

     /* Find out whether any buffered
packet can benefit from the
      * reverse route.
      * May need some change in the
following code - Mahesh 09/11/99
      */
     assert (rt0->rt_flags == RTF_UP);
     Packet *buffered_pkt;
     while ((buffered_pkt =
rqueue.deque(rt0->rt_dst))) {
       if (rt0 && (rt0->rt_flags ==
RTF_UP)) {
      assert(rt0->rt_hops != INFINITY2);
         forward(rt0, buffered_pkt,
NO_DELAY);
       }
     }
   }
   // End for putting reverse route in rt
table
```

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

 // First check if I am the destination ..

 if(rq->rq_dst == index) {

#ifdef DEBUG
   fprintf(stderr, "%d - %s: destination
sending reply\n",
                    index, __FUNCTION__);
#endif // DEBUG


   // Just to be safe, I use the max.
Somebody may have
   // incremented the dst seqno.
   seqno = max(seqno, rq->rq_dst_seqno)+1;
   if (seqno%2) seqno++;

   sendReply(rq->rq_src,           // IP
Destination
             1,                    // Hop
Count
             index,                // Dest
IP Address
             seqno,                // Dest
Sequence Num
             MY_ROUTE_TIMEOUT,     //
Lifetime
             rq->rq_timestamp);    //
timestamp

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

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

   //assert (rt->rt_flags == RTF_UP);
   assert(rq->rq_dst == rt->rt_dst);
   //assert ((rt->rt_seqno%2) == 0);  // is
the seqno even?
   sendReply(rq->rq_src,
             rt->rt_hops + 1,
             rq->rq_dst,
             rt->rt_seqno,
          (u_int32_t) (rt->rt_expire -
CURRENT_TIME),

   // Insert nexthops to RREQ source and
RREQ destination in the precursor lists of
destination and source respectively
   rt->pc_insert(rt0->rt_nexthop); //
nexthop to RREQ source
   rt0->pc_insert(rt->rt_nexthop); //
nexthop to RREQ destination

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

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

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

#ifdef RREQ_GRAT_RREP
   sendReply(rq->rq_dst,
             rq->rq_hop_count,
             rq->rq_src,
             rq->rq_src_seqno,
           (u_int32_t) (rt->rt_expire -
CURRENT_TIME),
          //             rt->rt_expire -
CURRENT_TIME,
             rq->rq_timestamp);
#endif
Packet::free(p); }
```

```
else {
   //initial value DCF CALCULATIOn
   double now =
Scheduler::instance().clock();
   double
a=8.796112,b=37.29479,c=7.229256,d=11.26162
,m=5.094965;
   double preNB = (double)nb_count[index] /
c;
   double NB=1+(pow(preNB,b));
   double ad = a-d;
   double nbm = (pow(NB,m));
   double preDCF = ad/nbm;
   double DCF=d+preDCF;
   if(DCF > 1.0) DCF=1.0;
   double DAF = DCF/nb_count[index];

    int irisan = 0;
    for (unsigned int i = 1; i <=
nb_list[index].size()-1; i++)
    {
      for (unsigned int j = 1; j <=
nb_list[ih->saddr()].size()-1; j++)
      {
        if(nb_list[index][i]==nb_list[ih-
>saddr()][j]) irisan++;
      }
    }

  int nb_uncovered =
std::abs(nb_count[index]-irisan-1);

  double Ra= (double)nb_uncovered/(double)
std::abs(nb_count[index]);
  double FP=DAF*Ra;
```

```
int r = rand()%250; //this produces numbers
between 0 - 2000

  double random_num = r/10000.0;

  if(FP>=random_num)
  {
    ih->saddr() = index;
    ih->daddr() = IP_BROADCAST;
    rq->rq_hop_count += 1;
    // Maximum sequence number seen en
route
    if (rt) rq->rq_dst_seqno = max(rt-
>rt_seqno, rq->rq_dst_seqno);
    forward((aodv_rt_entry*) 0, p, DELAY);
  }
  else {
    Packet::free(p);
  }

 }

}
```

A.2 Kode Fungsi nb_insert()

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


AODV_Neighbor *nb = new AODV_Neighbor(id);

  if(nb_count[index]==NULL)
  {
    nb_count[index]=0;
    nb_count[index]++;
    nb_list[index].push_back(id);
  }
  else {
    nb_count[index]++;
    nb_list[index].push_back(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_delete()

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


AODV_Neighbor *nb = nbhead.lh_first;

  for (unsigned int i = 1; i <=
nb_list[index].size()-1; i++)
  {
    if(nb_list[index][i]==id)
    {
      nb_list[index].erase
(nb_list[index].begin()+i);
      nb_count[index]--;
      break;
    }
  }

 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)    CMUPriQueue;
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"
$ns_ at 100.0000000000000000 "$cbr_(0) stop"
```

A.6 Kode Skrip AWK *Packet Delivery Ratio*

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

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

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

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

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

A.7 Kode Skrip AWK Rata-Rata *End-to-End Delay*

```
BEGIN{
      sum_delay = 0;
      count = 0;
}
{
      if ($2 >= 101) {

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

            if($4 == "AGT" && $1 == "s") {
                  start_time[$6] = $2;
            }

            else if(($7 == "cbr") && ($1 ==
"r")) {
                  end_time[$6] = $2;
            }

            else if($1 == "D" && $7 == "cbr")
{
                  end_time[$6] = -1;
            }
      }
}
END {

    for(i=0; i<=seqno; i++) {
            if(end_time[i] > 0) {
                  delay[i] = end_time[i] -
start_time[i];
                  count++;
        }
        else {
            delay[i] = -1;
        }
    }
```

```
    for(i=0; i<=seqno; i++) {
        if(delay[i] > 0) {
           n_to_n_delay = n_to_n_delay +
delay[i];
        }
    }
     n_to_n_delay = n_to_n_delay/count;
     printf "End-to-End Delay \t= "
n_to_n_delay * 1000 " ms \n";
}
```

A.8 Kode Skrip AWK *Routing Overhead*

```
BEGIN {
     rt_pkts = 0;
}
{
     if ($2 >= 101) {
           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 {
      rreqf = 0;
}
{
      if ($2 >= 101) {
            if (($1 == "f") && ($4 ==
"RTR") && ($7 == "AODV") ($19 == "[1")) {

                  rreqf++;
      }
}
END {
      printf "Route Request F \t= %d \n",
rreqf;
}
```

BIODATA PENULIS

Trastian Satria Wibowo lahir di Kebumen pada 06 September 1995. Penulis menempuh pendidikan formal TK Pembina (2000-2002), SDN Gedongan 1 Mojokerto (2002-2008), SMPN 1 Mojokerto (2008-2011), SMAN 5 Surabaya (2011-2014), dan melanjutkan studi S1 di Departemen Informatika ITS (2014-2018). Bidang studi yang diambil oleh penulis pada saat berkuliah di Departemen Informatika ITS adalah Arsitektur Jaringan Komputer (AJK). Penulis aktif dalam organisasi Himpunnan Mahasiswa Teknik Computer-Informatika (2015-2017). Penulis juga aktif dalam kegiatan kepanitian seperti SCHEMATICS 2015 dan SCHEMATICS 2016 divisi Perlengkapan dan Transportasi (PERKAPTRANS). Penulis pernah kerja praktik di PT. PERTAMINA (PERSERO) MOR V Surabaya periode Juli – Agustus 2017. Penulis dapat dihubungi melalui nomor *handphone* 085851574848 atau di email strastian@gmail.com.