
TUGAS AKHIR – KI141502

# MODIFIKASI PEMILIHAN *FORWARDING NODE* PADA *DYNAMIC SOURCE ROUTING* (DSR) BERDASARKAN TINGKAT KESTABILAN *NEIGHBORING NODE* DI VANETS

**GLLEEN ALLAN M**
**NRP 5114100171**

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

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

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

*(Halaman ini sengaja dikosongkan)*

TUGAS AKHIR – KI141502

# MODIFIKASI PEMILIHAN *FORWARDING NODE* PADA *DYNAMIC SOURCE ROUTING* (DSR) BERDASARKAN TINGKAT KESTABILAN *NEIGHBORING NODE* DI VANETS

**GLLEEN ALLAN M**
**NRP 5114100171**

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

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

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

*(Halaman ini sengaja dikosongkan)*


UNDERGRADUATE THESES – KI141502

# MODIFICATION OF FORWARDING NODE SELECTION FOR DYNAMIC SOURCE ROUTING (DSR) BASED ON NEIGHBORING NODE STABILITY LEVEL IN VANETS

**GLLEEN ALLAN M**
**NRP 5114100171**

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

**Second Advisor**
**Ir. Muchammad Husni, M.Kom.**

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

*(Halaman ini sengaja dikosongkan)*

## LEMBAR PENGESAHAN

**MODIFIKASI PEMILIHAN *FORWARDING NODE* PADA *DYNAMIC SOURCE ROUTING* (DSR) BERDASARKAN TINGKAT KESTABILAN *NEIGHBORING NODE* 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:
**GLLEEN ALLAN M**
**NRP: 5114100171**

Disetujui oleh Pembimbing Tugas Akhir:

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

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

**SURABAYA**
**JANUARI, 2018**



v

*(Halaman ini sengaja dikosongkan)*

**MODIFIKASI PEMILIHAN *FORWARDING NODE* PADA *DYNAMIC SOURCE ROUTING* (DSR) BERDASARKAN TINGKAT KESTABILAN *NEIGHBORING NODE* DI VANETS**


| | |
|---|---|
| **Nama Mahasiswa** | **: Glleen Allan M** |
| **NRP** | **: 5114100171** |
| **Departemen** | **: Informatika FTIK-ITS** |
| **Dosen Pembimbing 1** | **: Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.** |
| **Dosen Pembimbing 2** | **: Ir. Muchammad Husni, M.Kom.** |


## Abstrak


*Vehicular Ad hoc Networks* (VANETs) merupakan pengembangan dari *Mobile Ad hoc 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 *Dynamic Source Routing* (DSR). DSR 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 jika *node* tersebut memiliki jumlah *node* tetangga lebih dari jumlah *threshold*, *node* tersebut menjadi *forwarding node* dan *node* tersebut yang bisa melakukan proses *rebroadcast*. 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* DSR yang asli.

Pada tugas akhir ini, performa pada *routing protocol* DSR yang telah dimodifikasi menghasilkan performa yang lebih bagus. Dibuktikan dengan skenario *real* yang menghasilkan peningkatan rata-rata *packet delivery ratio* sebesar 5.41%, rata-rata penurunan *routing overhead* sebesar 50.72%, dan juga rata-rata penurunan *forwarded route request* sebesar 16.9%.

**Kata kunci: DSR, *Forwarding Node, Threshold*, VANETs**

**MODIFICATION OF FORWARDING NODE SELECTION FOR DYNAMIC SOURCE ROUTING (DSR) BASED ON NEIGHBORING NODE STABILITY LEVEL IN VANETS**


| | |
|---|---|
| **Student's Name** | **: Glleen Allan M** |
| **Student's ID** | **: 5114100171** |
| **Department** | **: Informatika FTIK-ITS** |
| **First Advisor** | **: Dr.Eng. Radityo Anggoro, S.Kom, M.Sc.** |
| **Second Advisor** | **: Ir. Muchammad Husni, M.Kom.** |


## 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 Dynamic Source Routing (DSR). DSR 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 if the number of neighbor nodes more than threshold, the node becomes a forwarding node and the node will rebroadcast the packet. 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 DSR. It is proven that in real scenario, there is an enhancement of average packet delivery ratio by 5.41%, reduction of average routing overhead by 50.72%, and reduction of average forwarded route request by 16.9%.*



***Keyword: DSR, Forwarding Node, Threshold, VANETs***

*(Halaman ini sengaja dikosongkan)*

# KATA PENGANTAR

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

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

**"Modifikasi Pemilihan *Forwarding Node* Pada *Dynamic Source Routing* (DSR) Berdasarkan Tingkat Kestabilan *Neighboring Node* 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, Papa, Clara, Bu Tun, Pak Nan, Pungky, Mbak Eva, Mbak Supik, Pak Yanto, Rissa, Mbak Al, Pak No, 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 Ir. Muchammad Husni, M.Kom. 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, Tras, Cimeng, Hari, Hakim, Upil, Rage, Fito, Anandi, Amik, Bajikas, Sekbay, Penyok, Kevin, Riefqy, Akhyar, Nanda, Dito, Rian, Nanda, Fathur, 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 AJK (Syukron, Fatih, Thoni, Didin, Fuad, Awan, Satriya, Daus, Nahda, Hana, Raldo, Khawari, dan Router), laboratorium DTK (mas Mujib, mas Andre, mas Iqbal, mas Icang, mas Hendro, mas Reza, mas Pipis, mas Adim, mas Ipul, mas Ridwan, mas Naufal, mas Ivan, dan mas Wahyu), dan laboratorium NCC (Zulfa, Mila, Rafiar, Hania, Yoga, Risma, Sisil, Hero, Hendri, Dely, Faizal, Ubut, dan Zayn) yang telah menemani, memberi semangat, memotivasi, memberikan doa, serta memberikan hiburan di kala penulis sedang jenuh saat pengerjaan Tugas Akhir ini.
7. Sahabat-sahabat penulis di angkatan 2014 (Vivi, Kania, Bebet, Nay, Raras, Tiara, Pina, Anindita, Nafia dan sahabat 2014 yang lain) yang telah menemani dan berjuang bersama penulis selama berkuliah di Departemen Informatika ITS.
8. Teman-teman 21 cabang Surabaya yang telah berjuang bersama penulis selama berkuliah di Surabaya.
9. Sahabat penulis (Asyraf, Ditya, Ranti, Tika, Mimi, Yudha, dan sahabat penulis lain yang tidak dapat disebutkan satu per satu) yang selalu membantu, menghibur, menjadi tempat bertukar ilmu, serta berjuang bersama-sama dengan penulis.
10. 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, 09 Januari 2018

Glleen Allan M

*(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 ..... 3
1.3 Batasan Permasalahan ..... 3
1.4 Tujuan ..... 3
1.5 Manfaat ..... 4
1.6 Metodologi ..... 4
1.6.1 Penyusunan Proposal Tugas Akhir ..... 4
1.6.2 Studi Literatur ..... 4
1.6.3 Implementasi Sistem ..... 5
1.6.4 Pengujian dan Evaluasi ..... 5
1.6.5 Penyusunan Buku ..... 5
1.7 Sistematika Penulisan Laporan ..... 5
**BAB II TINJAUAN PUSTAKA** ..... **9**
2.1 VANETs ..... 9
2.2 *Dynamic Source Routing* (DSR) ..... 10
2.3 *Network Simulator-2* (NS-2) ..... 12
2.3.1 Instalasi ..... 13
2.3.2 *Trace File* ..... 13
2.4 OpenStreetMap ..... 15
2.5 Java OpenStreetMap Editor (JOSM) ..... 15
2.6 Simulation of Urban Mobility (SUMO) ..... 16
2.7 AWK ..... 16
**BAB III PERANCANGAN** ..... **17**
3.1 Deskripsi Umum ..... 17
3.2 Perancangan Skenario Mobilitas ..... 20
3.2.1 Perancangan Skenario Grid ..... 20

3.2.2 Perancangan Skenario Real....22
3.3 Analisis dan Perancangan Modifikasi *Routing Protocol* DSR....23
3.3.1 Perancangan Penghitungan Jumlah *Node* Tetangga untuk Setiap *Node*....24
3.3.2 Perancangan Pemilihan *Forwarding Node*....24
3.4 Perancangan Simulasi pada NS-2....25
3.5 Perancangan Metrik Analisis....26
3.5.1 *Packet Delivery Ratio* (PDR)....26
3.5.2 Rata-rata *End-to-End Delay* (E2E)....26
3.5.3 *Routing Overhead* (RO)....27
3.5.4 *Forwarded Route Request* (RREQ F)....27
**BAB IV IMPLEMENTASI....29**
4.1 Implementasi Skenario Mobilitas....29
4.1.1 Skenario *Grid*....29
4.1.2 Skenario *Real*....32
4.2 Implementasi Modifikasi pada *Routing Protocol* DSR untuk Memilih *Forwarding Node*....34
4.2.1 Implementasi Menghitung Jumlah *Node* Tetangga....35
4.2.2 Implementasi Pemilihan *Forwarding Node*....38
4.3 Implementasi Simulasi pada NS-2 dan *Traffic* Pengiriman Dua Sesi....39
4.4 Implementasi Metrik Analisis....42
4.4.1 Implementasi *Packet Delivery Ratio*....42
4.4.2 Implementasi Rata-Rata *End-to-End Delay*....43
4.4.3 Implementasi *Routing Overhead*....44
4.4.4 Implementasi *Forwarded Route Request*....45
**BAB V UJI COBA DAN EVALUASI....47**
5.1 Lingkungan Uji Coba....47
5.2 Hasil Uji Coba....48
5.2.1 Hasil Pra Uji Coba Penentuan *Threshold*....48
5.2.2 Hasil Uji Coba Skenario *Grid*....50
5.2.3 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 DSRAgent::recv() ..... 69
A.2 Kode Skenario NS-2 ..... 74
A.3 Kode Konfigurasi *Traffic* ..... 77
A.4 Kode Skrip AWK *Packet Delivery Ratio* ..... 78
A.5 Kode Skrip AWK Rata-Rata *End-to-End Delay* ..... 79
A.6 Kode Skrip AWK *Routing Overhead* ..... 80
A.7 Kode Skrip AWK *Forwarded Route Request* ..... 81
**BIODATA PENULIS ..... 83**

*(Halaman ini sengaja dikosongkan)*

# DAFTAR GAMBAR

**Gambar 2.1** Ilustrasi VANETs [3] ..... 10
**Gambar 2.2** Mekanisme RREQ pada DSR [5] ..... 11
**Gambar 2.3** Mekanisme RREP pada DSR [5] ..... 12
**Gambar 3.1** Diagram Rancangan Simulasi dengan Routing Protocol DSR Asli ..... 18
**Gambar 3.2** Diagram Rancangan Simulasi dengan Routing Protocol DSR yang telah dimodifikasi ..... 19
**Gambar 3.3** Alur Pembuatan Skenario Mobilitas Grid ..... 21
**Gambar 3.4** Alur Pembuatan Skenario Mobilitas Real ..... 23
**Gambar 3.5** Pseudocode Penghitungan Jumlah Node ..... 24
**Gambar 4.1** Hasil Generate Peta Grid ..... 30
**Gambar 4.2** Perintah randomTrips ..... 30
**Gambar 4.3** File Skrip .sumocfg ..... 31
**Gambar 4.4** Perintah traceExporter ..... 32
**Gambar 4.5** Ekspor Peta dari OpenStreetMap ..... 33
**Gambar 4.6** Hasil Konversi Peta Real ..... 34
**Gambar 4.7** Potongan Kode untuk Deklarasi Variabel ..... 36
**Gambar 4.8** Potongan Kode untuk Menghitung Tetangga ..... 37
**Gambar 4.9** Potongan Kode Untuk Sesi Pertama dan Sesi Kedua ..... 39
**Gambar 4.10** Konfigurasi Lingkungan Simulasi ..... 40
**Gambar 4.11** Konfigurasi Sesi Pertama Pada File Traffic ..... 41
**Gambar 4.12** Konfigurasi Sesi Kedua Pada File Traffic ..... 41
**Gambar 4.13** Pseudocode untuk Menghitung PDR ..... 43
**Gambar 4.14** Pseudocode untuk Perhitungan Rata-Rata End-to-End Delay ..... 44
**Gambar 4.15** Pseudocode untuk Perhitungan Routing Overhead ..... 45
**Gambar 4.16** Pseudocode untuk Perhitungan Forwarded Route Request ..... 46
**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 .....59
**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 DSR ..... 14
**Tabel 3.1** Daftar Istilah ..... 19
**Tabel 3.2** Parameter Lingkungan Simulasi dengan Skenario ..... 25
**Tabel 5.1** Spesifikasi Perangkat yang Digunakan ..... 47
**Tabel 5.2** Hasil Pra Uji Coba Lingkungan Jarang (60 Node) ..... 48
**Tabel 5.3** Hasil Pra Uji Coba Lingkungan Sedang (150 Node) .. 49
**Tabel 5.4** Hasil Pra Uji Coba Lingkungan Padat (300 Node) ..... 49

*(Halaman ini sengaja dikosongkan)*

# BAB I
# PENDAHULUAN

## 1.1 Latar Belakang

Informasi merupakan suatu hal yang sangat mudah didapat dewasa ini. Setiap detiknya seseorang dapat mengetahui apa yang terjadi di belahan dunia lain. Didukung dengan perkembangan dunia internet yang sangat pesat, informasi menjadi semakin mudah untuk didapat. Teknologi internet saat ini juga digunakan sebagai suatu pemecahan suatu masalah. Contohnya masalah di jalan raya. Beberapa contoh masalah di jalan antara lain yaitu penentuan rute untuk sampai ke tujuan lebih cepat, terjadi kecelakaan, dan terjadinya kemacetan dikarenakan waktu *traffic light* disamaratakan pada suatu persimpangan. Proses penentuan rute perjalanan berhubungan dengan rute pengiriman data informasi dalam jaringan internet. Proses penentuan rute ini disebut dengan *routing*. Untuk dapat mendapatkan rute yang cepat diperlukan jalan alternatif yang banyak. Jalan alternatif disini berhubungan dengan infrastruktur yang dimana akan memakan banyak biaya untuk pembangunannya. Untuk menanggulangi hal tersebut dapat memanfaatkan teknologi jaringan *Ad-Hoc* yang mana mendasari pembuatan *Vehicular Ad-Hoc Networks* (VANETs).

VANETs merupakan pengembangan dari *Mobile Ad-Hoc Networks* (MANETs). Implementasi dari MANETs telah menciptakan beberapa *routing protocol*. Berdasarkan perlakuannya terhadap rute, *routing protocol* dalam MANETs dibedakan menjadi dua, yaitu *routing protocol* bersifat proaktif dan *routing protocol* bersifat reaktif. *Routing protocol* proaktif menggunakan basis data untuk menyimpan *node* dan rute yang berada dalam jaringan, sedangkan *routing protocol* reaktif tidak menggunakan basis data untuk membentuk sebuah rute.

Contoh untuk *routing protocol* reaktif adalah *Ad hoc On demand Distance Vector* (AODV). *Hello packets* digunakan untuk menentukan informasi konektivitas *node* tetangga. Secara *default*, informasi keonektivitas tersebut didapatkan di AODV oleh mekanisme *link layer detection*. Kegagalan penerimaan *hello message* dapat diartikan *node* tidak berada pada jangkauan komunikasi. Mendapatkan lebih banyak *hello packet* dari *node* ke i untuk *node* ke i+1 dapat diartikan *node* ke i akan bertahan lebih lama pada jangkauan transmisi *node* ke i+1. Sehingga, *node* tersebut stabil untuk dilakukannya komunikasi. Setelah interval ke c, *node* ke i+1 akan membagi jumlah total *hello packet* yang diterima oleh c. Nilai ini kemudian akan menjadi *node* pada *range factor* dan akan menjadi angka antara 0 dan d. *Node-node* pada MANETs sangat tidak stabil. Sehingga, *node* pada *range factor* ini akan menentukan, *node* mana yang akan dikirimkan RREQ, sehingga mengurangi pengiriman paket RREQ yang tidak perlu. Karena beberapa paket RREQ di-*drop*, maka kemacetan jaringan akan lebih rendah [1].

Contoh untuk *routing protocol* proaktif adalah *Optimized Link State Routing Protocol* (OLSR). OLSR biasanya digunakan pada lingkungan MANETs berdasarkan kebutuhan MANETs, optimasi tersebut berdasarkan pada *link state* (LS) *protocol* tradisional, kuncinya berada pada *multi-point relay*. Hanya beberapa *node* yang terpilih sebagai *node* perantara untuk mengontrol paket, jadi untuk mengurangi *control packet* yang terlalu banyak. Karena *node* tersebut hanya memilih beberapa *node* tetangga yang akan menjadi *node* perantara, dan hanya *node* perantara yang dapat meneruskan *route request* (RREQ), *node* tetangga sisanya hanya menerima informasi dari paket tanpa meneruskannya [2].

Pada Tugas Akhir ini akan mengimplementasikan *routing protocol Dynamic Source Routing* (DSR) yang menggabungkan cara kerja dari *routing protocol* AODV yang dimodifikasi pada

bagian *hello packets* dan juga implementasi *routing protocol* OLSR. Adapun bagian yang akan dimodifikasi dalam *routing protocol* ini yaitu melakukan eliminasi jumlah *forwarding node* yang bertugas *rebroadcast*.

## 1.2 Rumusan Masalah

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

1. Bagaimana membatasi jumlah *forwarding node* dalam proses *rebroadcast route request* (RREQ)?
2. Bagaimana dampak pembatasan jumlah *forwarding node* terhadap performa *routing protocol* DSR 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 DSR.
3. Simulasi pengujian jaringan menggunakan *Network Simulator 2* (NS-2).
4. Pembuatan skenario uji coba menggunakan *Simulation of Urban Mobility* (SUMO).

## 1.4 Tujuan

Tujuan dari Tugas Akhir ini adalah untuk mereduksi jumlah *neighbor node* yang bertanggungjawab untuk *rebroadcast* RREQ *packet* sehingga dapat mengurangi jumlah *control packet* yang di*broadcast* dalam jaringan pada *routing protocol* DSR.

## 1.5 Manfaat

Dengan dibuatnya Tugas Akhir ini dapat memberikan informasi tentang dampak dari pembatasan *forwarding node* berdasarkan level stabilitas *neighbor node* terhadap kinerja *routing protocol* DSR di lingkungan VANETs.

## 1.6 Metodologi

Pembuatan Tugas Akhir ini dilakukan dengan menggunakan metodologi sebagai berikut:

### 1.6.1 Penyusunan Proposal Tugas Akhir

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

### 1.6.2 Studi Literatur

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

### 1.6.3 Implementasi Sistem

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

### 1.6.4 Pengujian dan Evaluasi

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

### 1.6.5 Penyusunan Buku

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

## 1.7 Sistematika Penulisan Laporan

Sistematika penulisan laporan Tugas Akhir adalah sebagai berikut:

1. Bab I. Pendahuluan

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

2. Bab II. Tinjauan Pustaka
   Bab ini berisi kajian teori atau penjelasan dari metode, algoritma, *library*, dan *tools* yang digunakan dalam penyusunan Tugas Akhir ini. Bab ini berisi tentang penjelasan singkat mengenai VANETs, DSR, NS-2, OpenStreetMap, Java OpenStreetMap Editor, SUMO, dan AWK.
3. Bab III. Perancangan
   Bab ini berisi pembahasan mengenai perancangan skenario mobilitas *grid* dan *real*, perancangan simulasi pada NS-2, perancangan modifikasi DSR, serta perancangan metrik analisis (*Packet Delivery Ratio, End-to-End Delay, Routing Overhead*, dan *Forwarded Route Request*)
4. Bab IV. Implementasi
   Bab ini menjelaskan implementasi yang berbentuk kode sumber dari proses modifikasi protokol DSR, pembuatan peta untuk skenario *grid* dan skenario *real* menggunakan OpenStreetMap dan SUMO, melakukan simulasi menggunakan NS-2, dan perhitungan metrik analisis.
5. Bab V. Pengujian dan Evaluasi
   Bab ini berisikan hasil uji coba dan evaluasi dari implementasi modifikasi pada *routing protocol* DSR yang telah dilakukan untuk menyelesaikan masalah yang dibahas pada Tugas Akhir. Pengujian dilakukan dengan skenario yang di*generate* oleh SUMO dan dijalankan di NS-2 untuk mendapatkan data uji PDR, E2E, RO, dan RREQ F yang nantinya akan dianalisis menggunakan skrip awk dan dilakukan perbandingan performa *routing protocol* DSR asli dengan *routing protocol* DSR yang telah dimodifikasi.
6. Bab VI. Kesimpulan dan Saran
   Bab ini merupakan bab yang menyampaikan kesimpulan dari hasil uji coba yang dilakukan, masalah-masalah yang dialami

pada proses pengerjaan Tugas Akhir, dan saran untuk pengembangan solusi ke depannya.

7. Daftar Pustaka
   Bab ini berisi daftar pustaka yang dijadikan literatur dalam Tugas Akhir.
8. Lampiran
   Dalam lampiran terdapat kode sumber program secara keseluruhan.

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

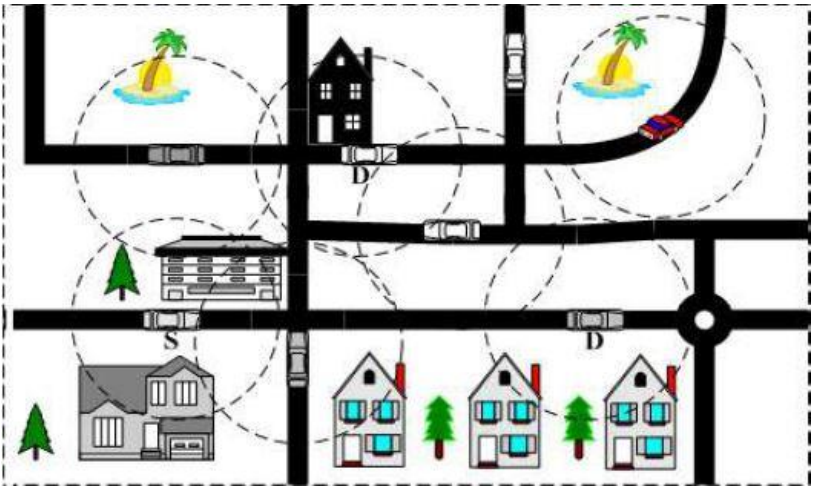
# BAB II
# TINJAUAN PUSTAKA

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 membroadcast 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 [3]. Ilustrasi VANETs dapat dilihat pada Gambar 2.1.

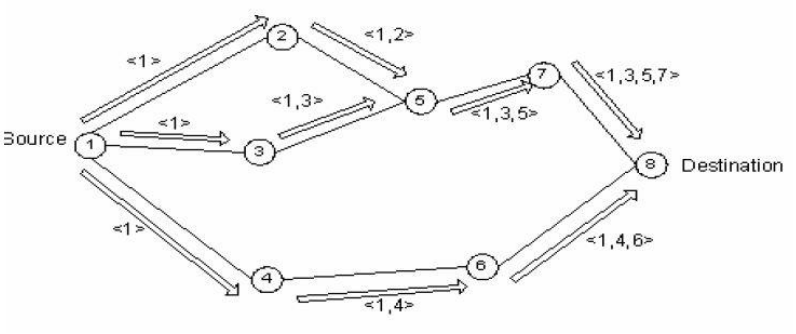



**Gambar 2.1** Ilustrasi VANETs [3]

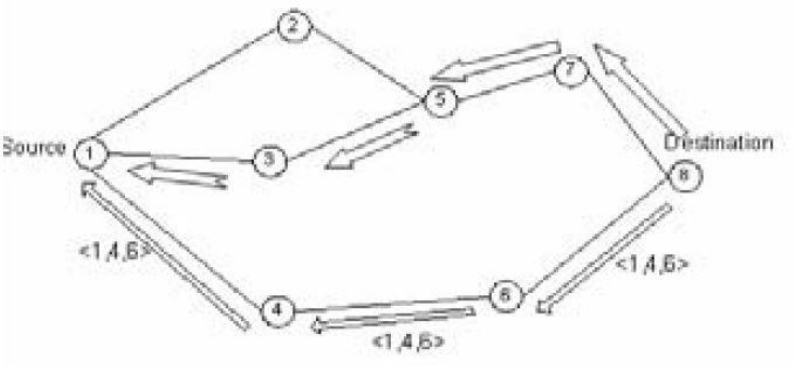
## 2.2 *Dynamic Source Routing* (DSR)

DSR adalah *routing protocol* yang sederhana dan efisien dirancang khusus untuk jaringan *ad hoc* nirkabel *multi-hop*. DSR tidak memerlukan jaringan infrastruktur yang sudah ada. *Nodes* jaringan bekerja sama untuk meneruskan paket untuk yang lainnya untuk memungkinkan komunikasi melalui beberapa '*hop*' antar *nodes* yang tidak secara langsung berada dalam jangkauan transmisi nirkabel satu sama lain. Sebagai *nodes* dalam jaringan, bergerak atau bergabung atau meninggalkan jaringan dan sebagai kondisi transmisi nirkabel seperti sumber gangguan, semua *routing* ditentukan dan dipelihara secara otomatis oleh *routing protocol* DSR [4].

Ketika *node* asal ingin mengirim paket ke *node* tujuan, jika tidak ditemukan rute menuju ke *node* tujuan, maka *node* asal menginisiasi mekanisme *route discovery* dengan cara mem*broadcast route request* (RREQ) ke tetangga. Jika tetangga mendapatkan *request* yang sama dari sumber yang berbeda atau alamatnya yang terdaftar pada RREQ, maka paket tersebut di *drop*. Sebaliknya jika itu adalah *node* tujuan, maka *node* tujuan tersebut mengirim *route reply* (RREP) kembali ke *node* asal. Ketika kondisi tersebut tidak terpenuhi, *node* yang menerima menambahkan alamat diri sendiri pada RREQ. RREP akan menelusuri rute yang sudah tersimpan sampai ke *node* asal [5]. Untuk ilustrasi RREQ dan RREP pada DSR bisa dilihat masing-masing pada Gambar 2.2 dan Gambar 2.3.




**Gambar 2.2** Mekanisme RREQ pada DSR [5]




**Gambar 2.3** Mekanisme RREP pada DSR [5]

Pada Tugas Akhir ini, penulis menggunakan DSR sebagai *routing protocol* yang akan dimodifikasi. Penulis akan membandingkan performa *routing protocol* DSR asli dengan *routing protocol* DSR yang dimodifikasi.

## 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* [6].

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

digunakan sebagai informasi untuk mengukur performa *routing protocol* DSR 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* DSR

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

| Kolom ke- | Penjelasan | Isi |
|---|---|---|
| 10 | *Flag* | ------: Tidak ada |
| 11 | Detail IP *source*, *destination*, dan *nexthop* | [a:b c:d e f]<br>a: IP *source node*<br>b: *port source node*<br>c: IP *destination node* (jika -1 berarti *broadcast*)<br>d: *port destination node*<br>e: IP *header* ttl<br>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 [7].

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 [8].

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 [9].

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* [10].

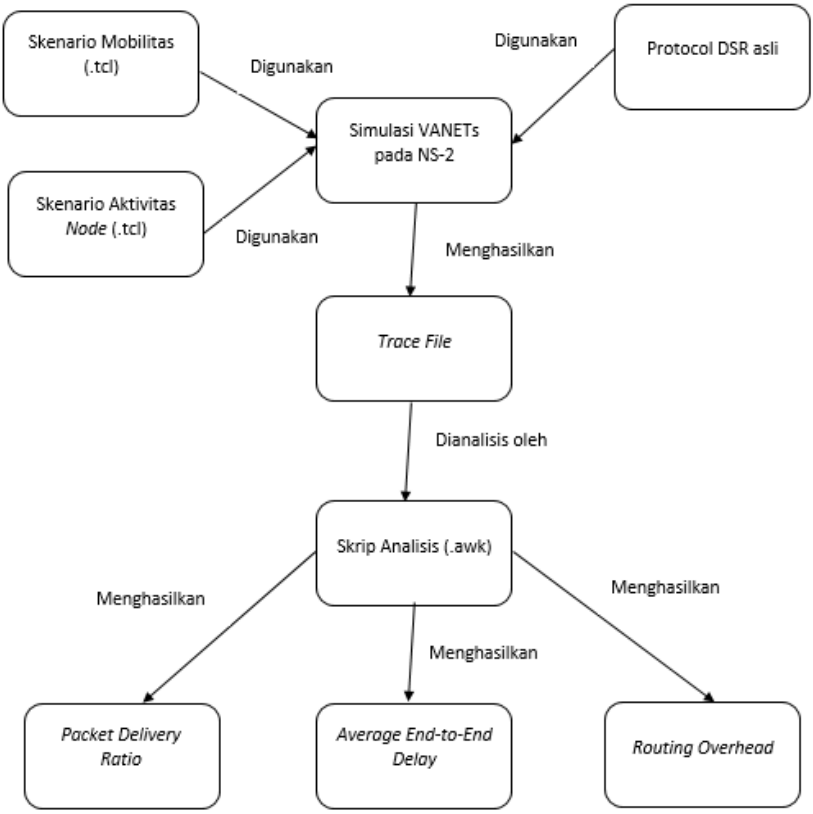
# BAB III
# PERANCANGAN

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.

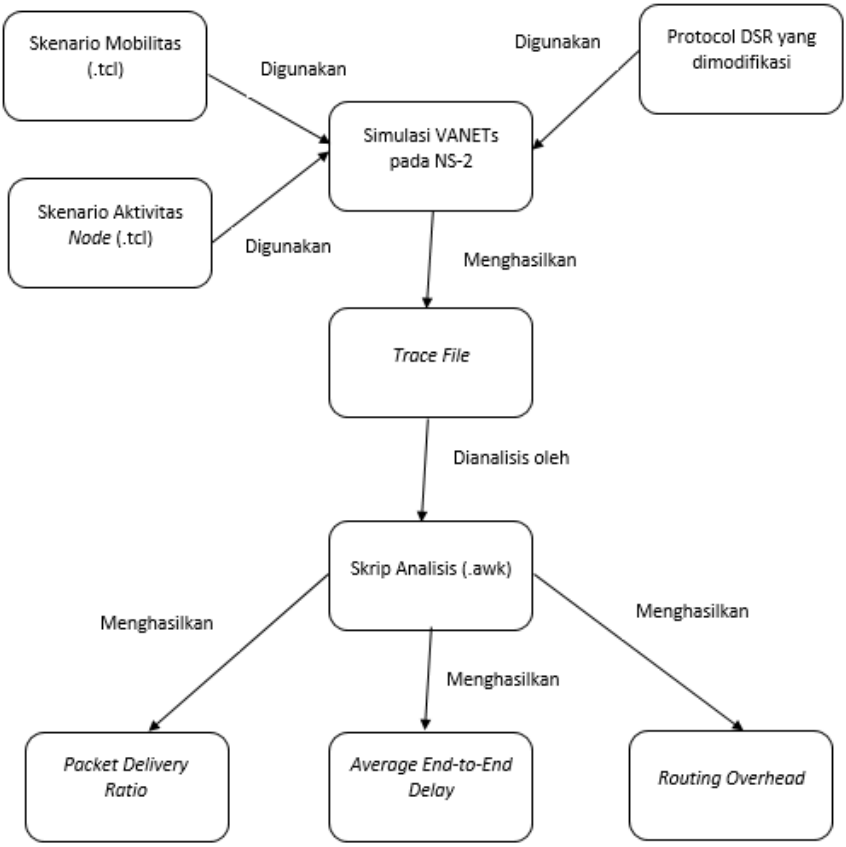
## 3.1 Deskripsi Umum

Pada Tugas Akhir ini penulis akan mengimplementasikan *routing protocol* DSR yang dimodifikasi dengan menambahkan proses seleksi pemilihan *forwarding node* yang berhak melakukan *rebroadcast* paket *route request* (RREQ) yang akan diteruskan. Modifikasi yang dilakukan yaitu dengan cara menyeleksi *node-node* mana saja yang mempunyai jumlah tetangga lebih dari *threshold*. Jika *node* tersebut mempunyai jumlah tetangga lebih dari *threshold* maka *node* tersebut berhak meneruskan paket RREQ, jika sebaliknya maka tidak meneruskan paket RREQ. Untuk simulasi yang akan dilakukan, penulis membuat dua sesi dalam satu simulasi. Pada sesi pertama, lingkungan akan dipelajari terlebih dahulu oleh seluruh *node*, lalu pada sesi kedua diimplementasikan proses seleksi pemilihan *forwarding node* yang akan meneruskan paket RREQ.

Modifikasi *routing protocol* DSR 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* DSR yang telah dimodifikasi dibandingkan dengan *routing protocol* DSR asli. Diagram rancangan simulasi dengan *routing protocol* DSR asli dan DSR yang telah dimodifikasi dapat dilihat pada Gambar 3.1 dan Gambar 3.2.




**Gambar 3.1** Diagram Rancangan Simulasi dengan *Routing Protocol* DSR Asli




**Gambar 3.2** Diagram Rancangan Simulasi dengan *Routing Protocol* DSR yang telah dimodifikasi

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

**Tabel 3.1** Daftar Istilah

| No. | Istilah | Penjelasan |
|---|---|---|
| 1 | DSR | *Dynamic Source Routing* |
| 2 | PDR | *Packet Delivery Ratio* |

| No. | Istilah | Penjelasan |
|---|---|---|
| 3 | E2E | *Average End-to-End Delay* |
| 4 | RO | *Routing Overhead* |
| 5 | RREQ | *Route Request* |
| 6 | RREP | *Route Reply* |
| 7 | *Threshold* | Batas nilai yang dijadikan sebagai acuan |

## 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.

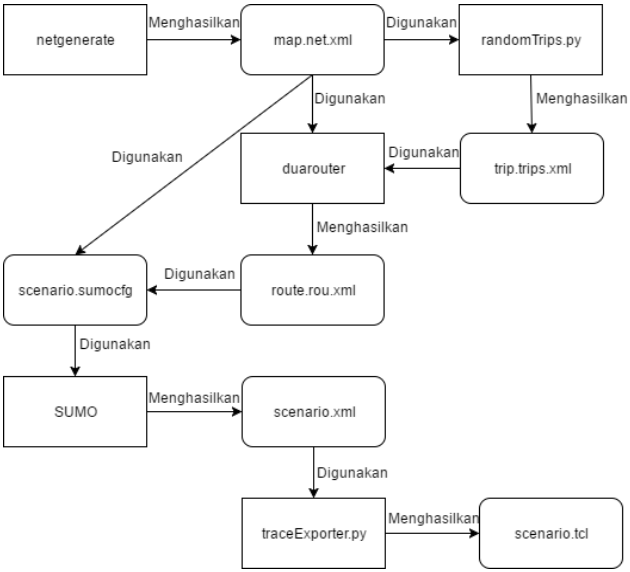
### 3.2.1 Perancangan Skenario Grid

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. Alur pembuatan skenario *grid* dapat dilihat pada Gambar 3.3.

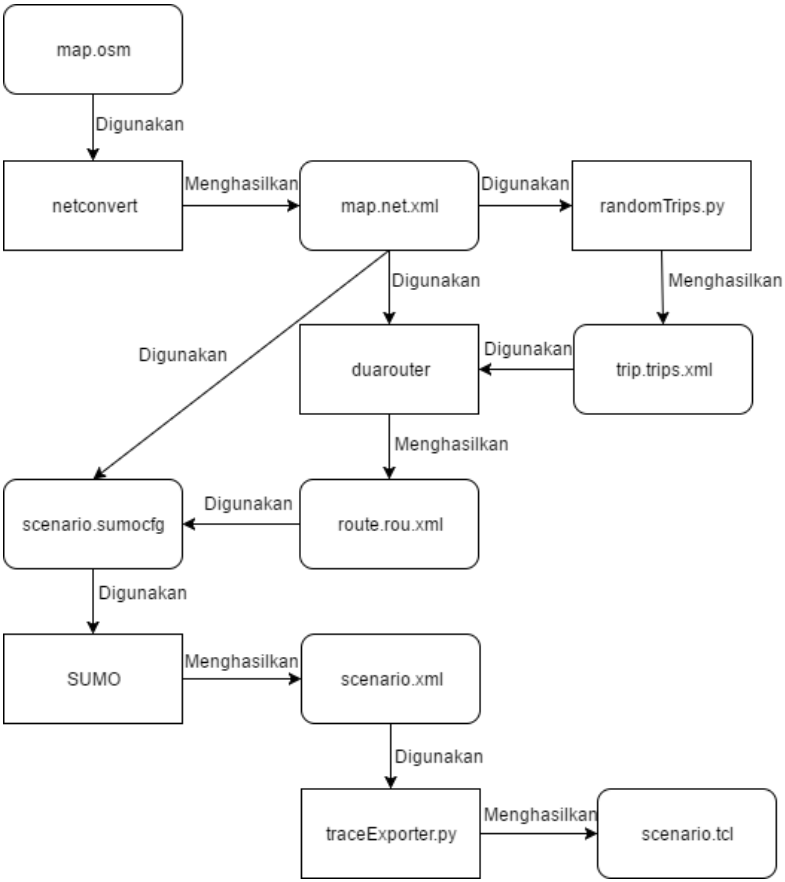



**Gambar 3.3** Alur Pembuatan Skenario Mobilitas *Grid*

### 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. Alur perancangan skenario mobilitas *real* dapat dilihat pada Gambar 3.4.




**Gambar 3.4** Alur Pembuatan Skenario Mobilitas *Real*

## 3.3 Analisis dan Perancangan Modifikasi *Routing Protocol* DSR

*Routing Protocol* DSR memiliki *cache* yang menyimpan jalur menuju *node* tujuan. DSR mengirim paket RREQ untuk mengetahui jalur mana yang menuju *node* tujuan lalu disimpan dalam *cache*. DSR

tidak menggunakan *hello message* sehingga DSR memerlukan dua sesi untuk melakukan satu kali simulasi. Dimana sesi pertama untuk mempelajari lingkungan dari simulasi tersebut dan sesi kedua menerapkan modifikasi yang telah dilakukan.

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

Ketika simulasi memasuki sesi pertama, maka DSR yang telah dimodifikasi akan menghitung jumlah *node* tetangga dari yang masuk *range* transmisi *node* tersebut. Karena simulasi dilakukan pada lingkungan VANETs, maka jumlah *node* tetangga dapat berubah dapat mempunyai jumlah tetangga yang banyak tergantung *range* transmisi sampai atau tidak. *Node* tetangga tersebut akan disimpan pada sebuah *array* agar *node* tetangga yang sudah masuk ke dalam list tidak terhitung dua kali. *Pseudocode* untuk penghitungan jumlah *node* tetangga dapat dilihat pada Gambar 3.5.

```
for ( i=0 to n)
      if(neighbor_list[id_node][i] == NULL)
            neighbor_list[id_node][i] =
neighbor_id
      else return
```

**Gambar 3.5** *Pseudocode* Penghitungan Jumlah *Node*

### 3.3.2 Perancangan Pemilihan *Forwarding Node*

Ketika simulasi memasuki sesi ke dua, maka DSR yang telah dimodifikasi akan menentukan *node* mana yang berhak *rebroadcast* paket RREQ. Penentuan *node* yang berhak melakukan *rebroadcast* adalah ketika jumlah tetangga yang dimiliki oleh *node* tersebut melebihi *threshold* atau batas nilai yang sudah ditentukan, maka *node* tersebut berhak melakukan *rebroadcast*, sedangkan sisanya tidak berhak. Langkah tersebut dilakukan agar mengurangi jumlah RREQ

yang dikirim sehingga lebih menghabiskan sedikit *cost.* Untuk pemilihan *forwarding node* dapat dilihat pada perintah *pseudocode* berikut:

```
if (neighbor >= threshold)
then
      receive route request
else
      return
```

## 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* | DSR |
| 3. | Waktu Simulasi | 200, 300, 500 |
| 4. | Area Simulasi | 1300 m x 1300 m dan 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* |

Kode yang ditambahkan diantaranya adalah deklarasi beberapa variable pada file dsragent.cc seperti *threshold* dan

penghitung jumlah tetangga. Ketika simulasi NS-2 dijalankan maka pada sesi ke dua saat simulasi, *routing protocol* DSR akan menyeleksi *node* mana saja yang bisa melakukan *rebroadcast*.

## 3.5 Perancangan Metrik Analisis

Berikut ini merupakan parameter-parameter yang akan dianalisis pada Tugas Akhir ini untuk dapat membandingkan performa dari *routing protocol* DSR yang telah dilakukan modifikasi dan *routing protocol* DSR 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.

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

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

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

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

# BAB IV
# IMPLEMENTASI

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

## 4.1 Implementasi Skenario Mobilitas

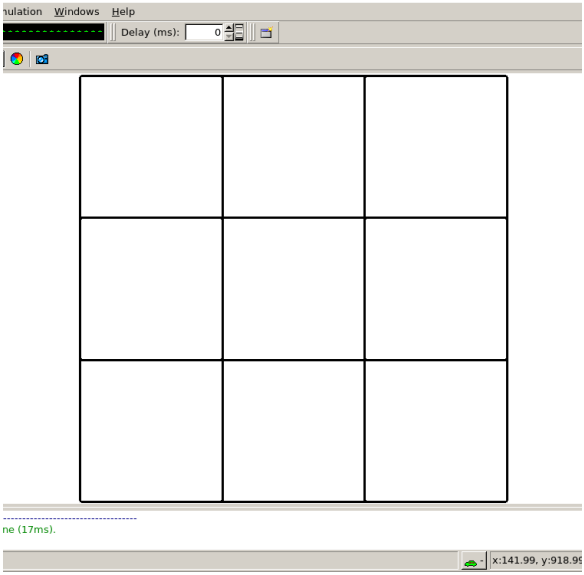
Implementasi skenario mobilitas VANETs dibagi menjadi dua, yaitu skenario *grid* yang menggunakan peta jalan berpetak dan skenario *real* yang menggunakan peta hasil pengambilan suatu area di kota Surabaya.

### 4.1.1 Skenario *Grid*

Dalam mengimplementasikan skenario *grid*, 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
```

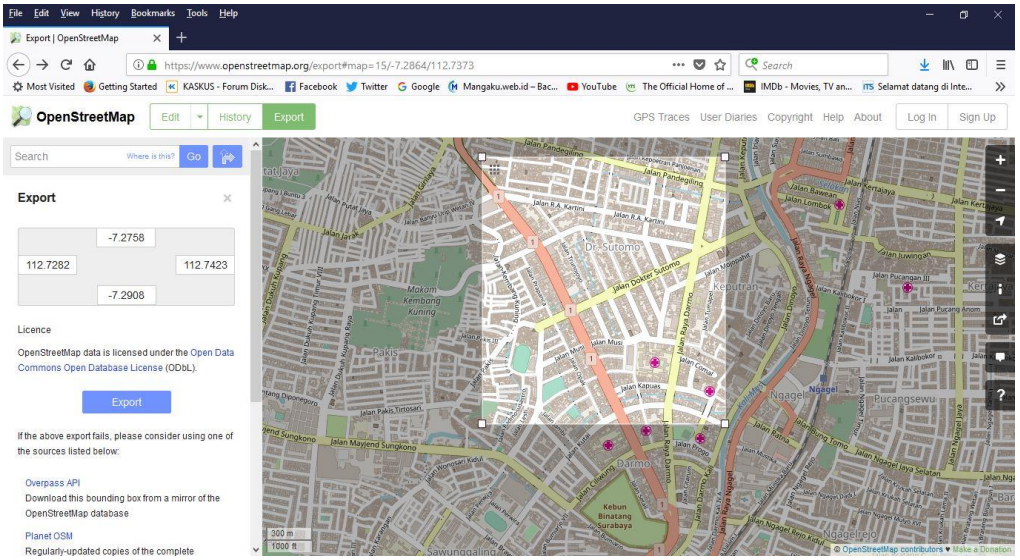
*File* skenario berekstensi .xml selanjutnya dikonversi ke dalam bentuk *file* berekstensi .tcl agar dapat disimulasikan menggunakan NS-2. *Tools* yang digunakn untuk melakukan konversi ini adalah traceExporter. Perintah untuk menggunakan traceExporter dapat dilihat pada Gambar 4.4.

```
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.

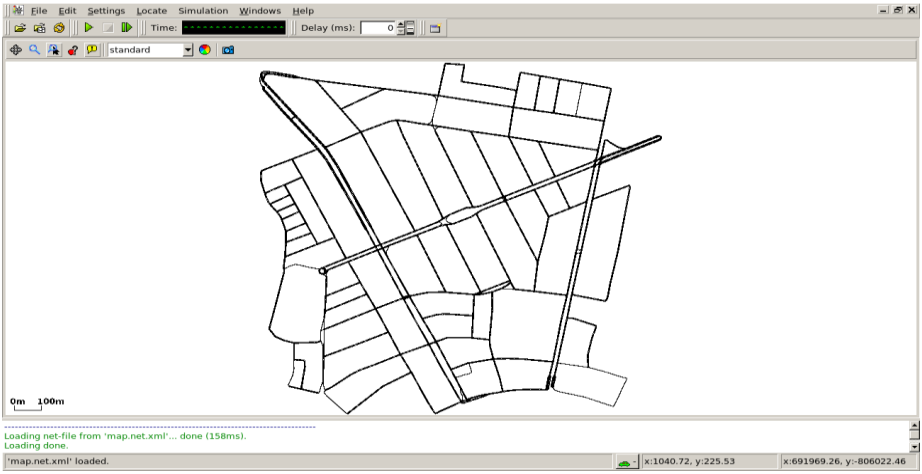



**Gambar 4.5** Ekspor Peta dari OpenStreetMap

*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* DSR untuk Memilih *Forwarding Node*

*Routing protocol* DSR 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.*

Pada Tugas Akhir ini dilakukan modifikasi pada *routing protocol* DSR agar dapat mengurangi jumlah *node* yang melakukan *rebroadcast* dengan memilih *forwarding node* yang lebih stabil yaitu berdasarkan jumlah tetangga yang lebih dari *threshold*. Sehingga pada Tugas Akhir ini bisa dilihat performa pada *routing protocol* DSR dapat meningkat.

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

- Implementasi Menghitung Jumlah *Node* Tetangga
- Implementasi Pemilihan *Forwarding Node*

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

### 4.2.1 Implementasi Menghitung Jumlah *Node* Tetangga

Langkah awal yang dilakukan untuk menghitung jumlah tetangga seperti yang telah dirancang pada subbab 3.3.1 adalah dengan menghitung *node-node* tetangga yang masuk *range* transmisi dari *node* yang sedang dieksekusi. Secara *default* DSR tidak dapat mengetahui jumlah *node* tetangga sehingga harus membuat kode sendiri untuk bisa mengetahui jumlah *node* tetangga. Dapat pula memanfaatkan fungsi `DSRAgent::recv()` pada kode sumber dsragent.cc. Ketika memasuki fungsi tersebut, kita bisa memanggil *node* mana yang sedang dieksekusi dengan cara menampilkannya menggunakan fungsi `net_id.dump()`. Karena tipe data untuk

setiap *node* yang ditampilkan oleh fungsi tersebut berbentuk string maka kita bisa mengubahnya menjadi *integer* menggunakan fungsi `atoi` yang terdapat pada bahasa C/C++ lalu dinamakan `id_int` yang berarti *id* dari sebuah *node* yang sudah menjadi *integer* dan `neighbor_id` yang berarti *node* tetangga yang sudah menjadi *integer* pula lalu menjadikan `id_int` menjadi *index* pada sebuah *array* yang sudah dideklarasikan terlebih dahulu yang bernama `list_neighbor[id_int][i]`. Deklarasikan juga sebuah *array* yang berguna untuk menjadi sebuah penghitung jumlah tetangga yang bernama `neighbor_counter[id_int]`. Untuk potongan kode tersebut bisa dilihat pada Gambar 4.7 dan Gambar 4.8.

```
int neighbor_list[1000][1000];
int neighbor_counter[1000];
int id_int;
int neighbor_id;
id_int = atoi(net_id.dump());
```

**Gambar 4.7** Potongan Kode untuk Deklarasi Variabel

Setelah itu bisa memanfaatkan fungsi `strtok()` yang digunakan untuk meng*split* beberapa karakter yang ada di fungsi `p.route.dump()` karena fungsi `p.route.dump()` menyimpan *path node* mana saja yang sudah dilalui ke dalam bentuk sebuah *array* sebelum memasuki fungsi `DSRAgent::recv()`. Dimana artinya fungsi `p.route.dump()` juga menyimpan *node* terakhir sebelum masuk ke fungsi `DSRAgent::recv()` dan *node* terakhir tersebut menjadi tetangga untuk *node* yang sedang dieksekusi lalu *id* dari *node* tetangga tersebut diubah menjadi integer dengan menggunakan fungsi `atoi()` karena akan dimasukkan ke dalam sebuah *array* bertipe data *integer*. Jadi array `list_neighbor[id_int][i]` menyimpan *node* terakhir pada *index* ke i dan `counter_neighbor[id_int]` di*increment* ketika menyimpan sebuah tetangga baru dan begitu seterusnya sampai semua *node*

tetangga sudah berhasil disimpan. Potongan kode sumber tersebut bisa dilihat pada Gambar 4.8.

```
while(token != NULL){
    tmptoken = token;
    token = strtok(NULL, " [](),");
    int flag = 0;
    if (token == NULL){
        neighbor_id = atoi(tmptoken);
        for (int n = 0; n <=
neighbor_counter[id_int]; n++) {
             if(neighbor_list[id_int][n] == -1){
                 neighbor_list[id_int][n] =
neighbor_id;
                 neighbor_counter[id_int]++;
                 flag = 1;
                 break;
             }
             else {
                  if (neighbor_list[id_int][n]
== neighbor_id){
                      flag = 1;
                      break;
                  }
             }
        }
        if (flag == 1 ) {
            break;
        }
    }
}
```

**Gambar 4.8** Potongan Kode untuk Menghitung Tetangga

Kode implementasi ini diletakkan pada fungsi DSRAgent::recv() dan dapat dilihat pada lampiran A.1 Kode Fungsi DSRAgent::recv().

### 4.2.2 Implementasi Pemilihan *Forwarding Node*

Pada tahap selanjutnya setelah dapat mengetahui jumlah tetangga yang disimpan dalam sebuah *counter* berbentuk *array* `neighbor_counter[id_int]` adalah pemilih untuk *forwarding node*. Sebelumnya perlu dijelaskan bahwa pada simulasi Tugas Akhir ini terdapat dua sesi dimana sesi pertama adalah sesi untuk setiap *node* mempelajari lingkungannya dan untuk mengetahui jumlah tetangga untuk masing-masing *node* dan untuk sesi ke dua adalah menerapkan *filter* ketika jumlah tetangga lebih dari *threshold* maka *node* yang sedang dieksekusi berhak mendapatkan paket RREQ yang artinya *node* tersebut bisa *rebroadcast* paket RREQ karena telah mendapatkan paket RREQ sebelumnya.

Selanjutnya memasuki kondisi `if (srh->route_request())` yang berarti fungsi tersebut akan mengeksekusi fungsi `handleroute request(p)` dimana jika paket yang diterima sebuah *node* berbentuk RREQ maka *node* tersebut berhak menerima RREQ. Untuk pembeda antara sesi pertama dengan sesi kedua dalam Tugas Akhir ini adalah dari waktu yang ditentukan. Pada lingkungan *node* yang jarang dipisahkan dengan jarak waktu ketika sebelum detik ke 100 maka masuk ke sesi pertama, setelah detik ke 101 maka masuk ke sesi kedua. Pada lingkungan *node* sedang dipisahkan dengan jarak waktu ketika sebelum detik ke 150 maka masuk ke sesi pertama, setelah detik ke 151 maka masuk ke sesi kedua. Pada lingkungan *node* yang padat dipisahkan dengan jarak waktu ketika sebelum detik ke 300 maka masuk ke sesi pertama, sedangkan setelah detik ke 301 maka masuk ke sesi kedua.

Pada sesi pertama masih belum di*filter* pemilihan *node* yang boleh menerima paket RREQ, jadi semua *node* dapat menerima paket RREQ. Pada sesi kedua *node-node* yang dapat menerima paket RREQ diseleksi berdasarkan jumlah *node* tetangga yang melebihi *threshold*. Potongan kode sumber untuk proses seleksi *node* yang dapat menerima paket RREQ menggunakan *threshold* dapat dilihat pada Gambar 4.9.

```
if (now <= 100.000000){
    handleRouteRequest(p);
}
else if (now >= 101.000000){
     if (neighbor_counter[id_int] >=
threshold) {
          handleRouteRequest(p);
     }
     else return;
}
```

**Gambar 4.9** Potongan Kode Untuk Sesi Pertama dan Sesi Kedua

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 dan *Traffic* Pengiriman Dua Sesi

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. 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)     CMUPriQueue
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) DSR
set val(stop)    200
set val(cp)      "cbr1.txt"
set val(sc)      "scen1modif.txt"
```

**Gambar 4.10** Konfigurasi Lingkungan Simulasi

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

Selain konfigurasi lingkungan simulasi dan pergerakan *node*, dilakukan juga konfigurasi untuk *traffic* dua sesi pengiriman paket. 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 untuk sesi pertama *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"
$ns_ at 100.0000000000000000 "$cbr_(0) stop"
```

**Gambar 4.11** Konfigurasi Sesi Pertama Pada *File Traffic*

Pada sesi pertama pengiriman dimulai pada detik ke 2.55 lalu berhenti pada detik ke 100 yang berarti sesi pertama sudah selesai di detik ke 100. Potongan konfigurasi untuk sesi kedua *file traffic* dapat dilihat pada Gambar 4.12.

```
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)
$ns_ at 101.0000000000000000 "$cbr_(0) start"
$ns_ at 200.0000000000000000 "$cbr_(0) stop"
```

**Gambar 4.12** Konfigurasi Sesi Kedua Pada *File Traffic*

Pada sesi kedua, pengiriman paket dimulai pada detik ke 101 lalu berhenti pada detik ke 200 yang berarti proses sesi kedua berjalan selama 100 detik. Implementasi konfigurasi *file traffic* untuk simulasi sesi pertama dan sesi kedua dapat dilihat pada lampiran A.3 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.4 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 4.13.

```
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 4.13** *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.5 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 4.14.

```
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 4.14** *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 sceanrio1.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.6 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 4.15.

```
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 4.15** *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.7 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 4.16.

```
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 4.16** *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) i7-7700HQ CPU @ 2.80GHz 2.81 GHz |
| **Sistem Operasi** | Ubuntu 14.04.5 LTS (Trusty Tahr) |
| **Linux Kernel** | Linux kernel 4.4 |
| **Memori** | 16.0 GB |
| **Penyimpanan** | 20 GB |

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.4 Kode Skrip AWK *Packet Delivery Ratio*, A.5 Kode Skrip AWK Rata-Rata *End-to-End Delay*, A.6 Kode Skrip AWK *Routing Overhead*, dan A.7 Kode Skrip AWK *Forwarded Route Request*.

## 5.2 Hasil Uji Coba

Untuk pra uji coba penentuan *threshold*, hasil uji coba dari skenario *grid* dan skenario *real* untuk Tugas Akhir ini dapat dilihat sebagai berikut:

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

Sebelum memasuki hasil uji coba untuk skenario *grid* dan skenario *real*, dilakukan terlebih dahulu pra uji coba untuk penentuan *threshold*. Penentuan *threshold* dilakukan dengan cara melakukan simulasi terlebih dahulu dengan menggunakan *threshold* dari 0 sampai batas *node* tertentu, dikarenakan untuk tiap lingkungan, ketika mencapai angka *threshold* tertentu dan seterusnya, untuk *packet delivery ratio*, *end-to-end delay*, dan *routing overhead* akan menghasilkan hasil yang sama jadi angka *threshold* tertentu sudah mencapai batasnya. Lalu dilakukan uji coba sebanyak 10 kali dan dilihat hasil yang unggul lalu dijadikan angka *threshold* untuk uji coba skenario mobilitas kedepannya. Untuk hasil pra uji coba penentuan *threshold* untuk lingkungan 60 *node* (jarang) yaitu didapatkan angka *threshold* 14. Untuk lingkungan 150 *node* (sedang) yaitu didapatkan angka *threshold* 17. Untuk lingkungan 300 *node* (padat) yaitu didapatkan angka *threshold* 19. Untuk salah satu hasil pra uji coba penentuan *threshold* dari lingkungan jarang, lingkungan sedang, dan lingkungan padat dapat dilihat pada Tabel 5.2, Tabel 5.3, dan Tabel 5.4.

**Tabel 5.2** Hasil Pra Uji Coba Lingkungan Jarang (60 *Node*)

| *Threshold* | *Packet Delivery Ratio* | *End-to-End Delay (ms)* | *Routing Packets* | *Route Request F* |
|---|---|---|---|---|
| 0 | 0,7512 | 1497,86 | 475 | 579 |
| 12 | 0,8485 | 1291,77 | 207 | 397 |
| 13 | 0,8159 | 40,1329 | 361 | 604 |

| *Threshold* | *Packet Delivery Ratio* | *End-to-End Delay (ms)* | *Routing Packets* | *Route Request F* |
|---|---|---|---|---|
| 14 | 0,8693 | 1146,69 | 200 | 429 |
| 15 | 0,8657 | 46,8432 | 207 | 413 |
| 16 | 0,7931 | 76,0052 | 397 | 599 |

**Tabel 5.3** Hasil Pra Uji Coba Lingkungan Sedang (150 *Node*)

| *Threshold* | *Packet Delivery Ratio* | *End-to-End Delay (ms)* | *Routing Packets* | *Route Request F* |
|---|---|---|---|---|
| 0 | 0,82 | 45,6457 | 289 | 872 |
| 15 | 0,7077 | 31,3225 | 587 | 830 |
| 16 | 0,7486 | 198,154 | 540 | 926 |
| 17 | 0,8834 | 18,7396 | 237 | 586 |
| 18 | 0,7485 | 43,8131 | 500 | 746 |
| 19 | 0,7286 | 61,1731 | 631 | 728 |

**Tabel 5.4** Hasil Pra Uji Coba Lingkungan Padat (300 *Node*)

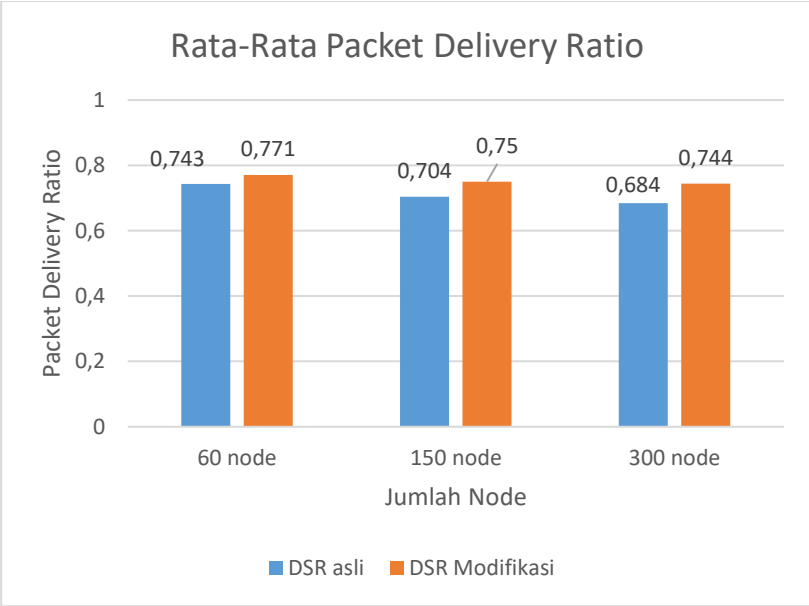
| *Threshold* | *Packet Delivery Ratio* | *End-to-End Delay (ms)* | *Routing Packets* | *Route Request F* |
|---|---|---|---|---|
| 0 | 0,7931 | 2202,88 | 965 | 1055 |
| 16 | 0,8248 | 81,7309 | 461 | 715 |
| 17 | 0,7972 | 25,4207 | 928 | 719 |
| 18 | 0,7907 | 56,4378 | 875 | 897 |
| 19 | 0,8323 | 130,978 | 905 | 695 |
| 20 | 0,8169 | 24,377 | 998 | 766 |

### 5.2.2 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* DSR asli dan *routing protocol* DSR yang telah dimodifikasi 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 menggunakan *threshold* yang berbeda-beda karena satu *threshold* tidak bisa disamakan untuk semua lingkungan disebabkan jumlah tetangga yang dimiliki oleh setiap *node* pada setiap lingkungan berbeda-beda, tergantung pada jumlah *node* pada lingkungan simulasi tersebut.

Hasil pengambilan data *packet delivery ratio* pada skenario *grid* 60, 150, dan 300 *node* dengan menggunakan DSR asli dan DSR yang telah dimodifikasi dapat dilihat pada Gambar 5.1.



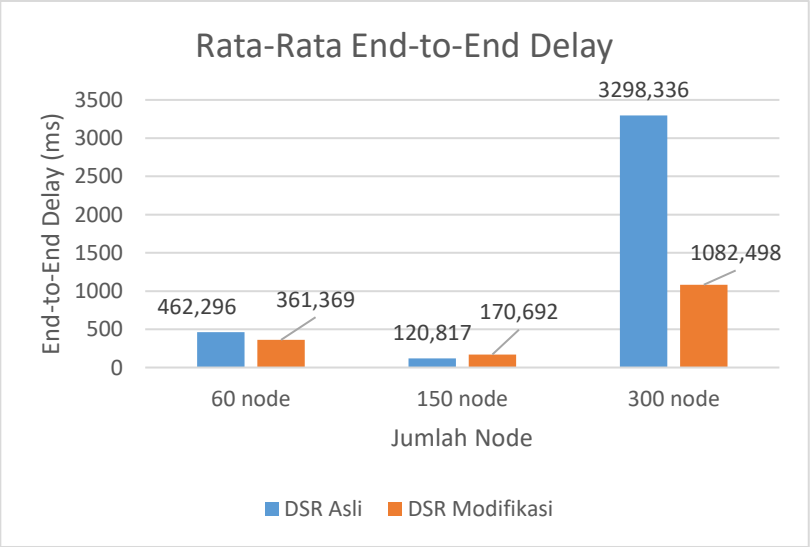

**Gambar 5.1** Grafik Rata-Rata *Packet Delivery Ratio* pada Skenario *Grid*

Berdasarkan grafik pada Gambar 5.1 dapat dilihat bahwa *routing protocol* DSR yang telah dimodifikasi dan juga *routing protocol* DSR asli mengalami penurunan yang signifikan pada *packet delivery ratio*. Pada lingkungan yang jarang dengan jumlah 60 *node*, menghasilkan perbedaan selisih PDR sebesar 0.028, atau naik menjadi sekitar 3.77% dimana *routing protocol* DSR yang telah dimodifikasi unggul dalam hal PDR tersebut. Pada lingkungan yang sedang dengan jumlah 150 *node*, menghasilkan perbedaan selisih PDR sebesar 0.046, atau naik menjadi sekitar 6.53% dimana *routing protocol* DSR yang telah dimodifikasi juga unggul dalam hal PDR tersebut dari DSR asli. Pada lingkungan yang padat dengan jumlah 300 *node*, menghasilkan perbedaan selisih PDR sebesar 0.06, atau

naik menjadi sekitar 8.77% dimana *routing protocol* DSR yang telah dimodifikasi juga unggul dalam hal PDR tersebut.

Dapat dilihat rata-rata kenaikan yang terjadi untuk PDR adalah 6,36%. Jika ketiga lingkungan tersebut dibandingkan dapat dilihat bahwa dengan jumlah *node* yang lebih sedikit atau pada lingkungan dengan *node* yang jarang, menghasilkan PDR yang lebih bagus daripada di lingkungan dengan jumlah *node* yang sedang maupun yang padat baik untuk *routing protocol* DSR asli maupun *routing protocol* DSR yang telah dimodifikasi. Dapat dilihat pula bahwa DSR yang telah dimodifikasi menghasilkan PDR yang lebih bagus daripada DSR asli dengan jumlah selisih PDR yang cukup signifikan.

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




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

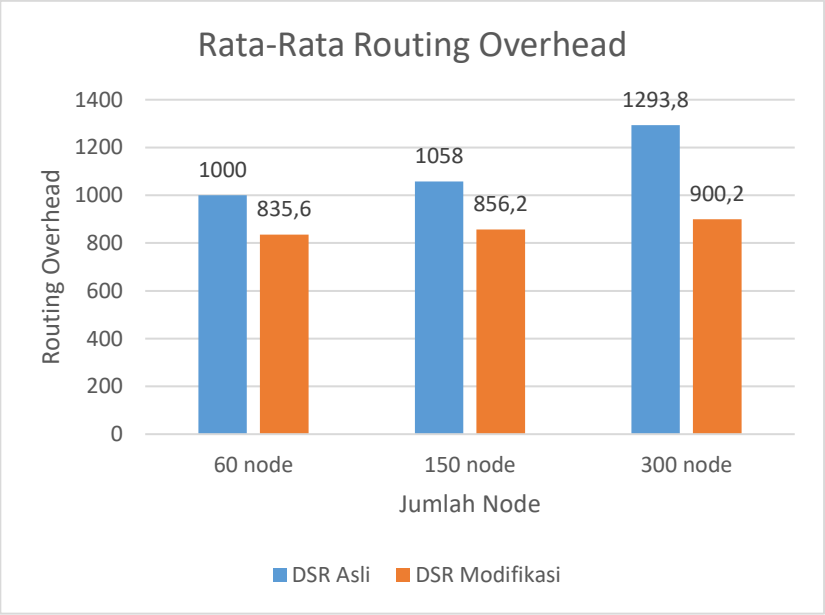
Berdasarkan grafik pada Gambar 5.2 dapat dilihat bahwa rata-rata *end-to-end delay* antara *routing protocol* DSR asli dan DSR yang telah dimodifikasi mengalami perubahan yang fluktuatif. Pada lingkungan yang jarang dengan jumlah 60 *node*, terjadi perbedaan selisih *end-to-end delay* sebesar 100.927 ms antara *routing protocol* DSR asli dengan *routing protocol* DSR yang telah dimodifikasi atau mengalami penurunan sebesar 27.93%, dimana *routing protocol* DSR yang telah dimodifikasi unggul dalam hal *end-to-end delay* tersebut. Sedangkan pada lingkungan sedang dengan jumlah 150 *node*, terjadi perbedaan selisih *end-to-end delay* sebesar 49.875 ms antara *routing protocol* DSR asli dengan *routing protocol* DSR yang telah dimodifikasi atau mengalami kenaikan sebesar 41.28%, dimana *routing protocol* DSR yang asli 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 2215.838 ms antara *routing protocol* DSR asli dengan *routing protocol* DSR yang telah dimodifikasi atau mengalami penurunan sebesar 204.70%, dimana *routing protocol* DSR yang telah dimodifikasi 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* DSR yang telah dimodifikasi tidak selalu lebih unggul dalam hal *end-to-end delay*. Juga, *routing protocol* DSR asli tidak selalu lebih unggul. Tetapi memang terlihat bahwa pada lingkungan jarang dan juga padat, *routing protocol* DSR yang telah dimodifikasi lebih unggul dalam hal *end-to-end delay* daripada *routing protocol* DSR asli. 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. Sedangkan jika lingkungan yang jarang atau jumlah *node* sedikit, dan diimplementasikan reduksi *forwarding node* maka akan mengurangi rute yang akan terbentuk dan paket yang terkirim juga semakin sedikit.

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



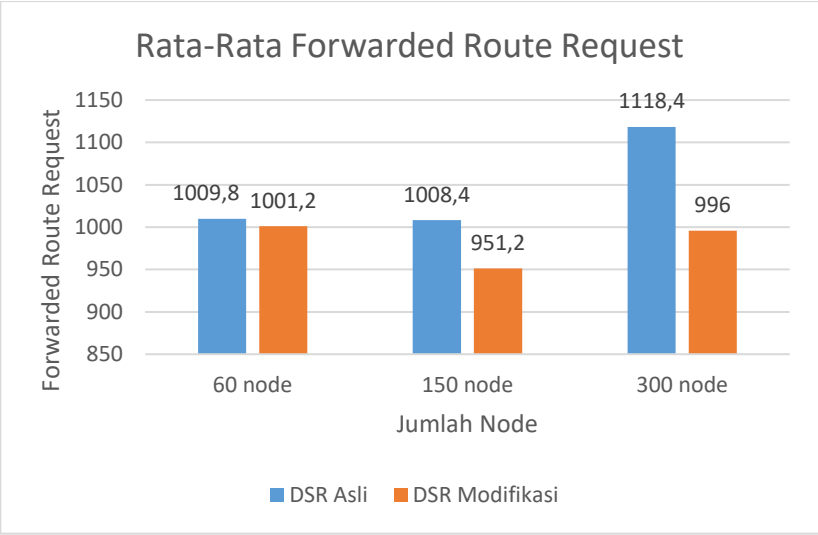

**Gambar 5.3** Grafik Rata-Rata *Routing Overhead* pada Skenario *Grid*

Berdasarkan grafik pada Gambar 5.3 dapat dilihat bahwa *routing protocol* DSR yang telah dimodifikasi dan juga *routing protocol* DSR asli mengalami kenaikan yang signifikan pada *routing overhead*. Pada lingkungan yang jarang dengan jumlah 60 *node*,

menghasilkan perbedaan selisih *routing overhead* sebesar 164.4 atau mengalami penurunan sebesar 19.67%, dimana *routing protocol* DSR yang telah dimodifikasi unggul dalam hal *routing overhead* tersebut karena menghasilkan *routing overhead* yang lebih rendah dari *routing* DSR asli. Pada lingkungan yang sedang dengan jumlah 150 *node*, menghasilkan perbedaan selisih *routing overhead* sebesar 201.8 atau mengalami penurunan sebeasr 23.57%, dimana *routing protocol* DSR yang telah dimodifikasi juga unggul dalam hal *routing overhead* tersebut dari DSR asli. Pada lingkungan yang padat dengan jumlah 300 *node*, menghasilkan perbedaan selisih *routing overhead* sebesar 393.6 atau mengalami penurunan sebesar 43.72%, dimana *routing protocol* DSR yang telah dimodifikasi juga unggul dalam hal *routing overhead* tersebut.

Dapat dilihat rata-rata penurunan yang terjadi untuk *routing overhead* adalah sebesar 28.99%. Jika ketiga lingkungan tersebut dibandingkan dapat dilihat bahwa dengan jumlah *node* yang lebih sedikit atau pada lingkungan dengan *node* yang jarang, menghasilkan *routing overhead* yang lebih bagus atau lebih sedikit daripada di lingkungan dengan jumlah *node* yang sedang maupun yang padat baik untuk *routing protocol* DSR asli maupun *routing protocol* DSR yang telah dimodifikasi. Dapat dilihat pula bahwa DSR yang telah dimodifikasi menghasilkan *routing overhead* yang lebih bagus atau dalam hal ini lebih rendah daripada DSR asli dengan jumlah selisih *routing overhead* yang cukup signifikan.

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* DSR yang telah dimodifikasi dan juga *routing protocol* asli mengalami perubahan *forwarded route request* (RREQ F) yang fluktuatif. Pada lingkungan yang jarang dengan jumlah 60 *node*, menghasilkan perbedaan selisih *forwarded route request* sebesar 8.6 atau mengalami penurunan sebesar 0.87%, dimana *routing protocol* DSR yang telah dimodifikasi unggul dalam hal *forwarded route request* tersebut karena menghasilkan *forwarded route request* yang lebih rendah dari *routing protocol* DSR asli. Pada lingkungan yang sedang dengan jumlah 150 *node*, menghasilkan perbedaan selisih *forwarded route request* sebesar 57.2 atau mengalami penurunan sebesar 6.01%, dimana *routing protocol* DSR yang telah dimodifikasi juga unggul dalam hal *forwarded route request* tersebut dari *forwarded route request* DSR asli. Pada lingkungan yang padat dengan jumlah 300 *node*, menghasilkan

perbedaan selisih *forwarded route request* sebesar 122.4 atau mengalami penurunan sebesar 12.29%, dimana *routing protocol* DSR yang telah dimodifikasi juga unggul dalam hal *forwarded route request* tersebut.

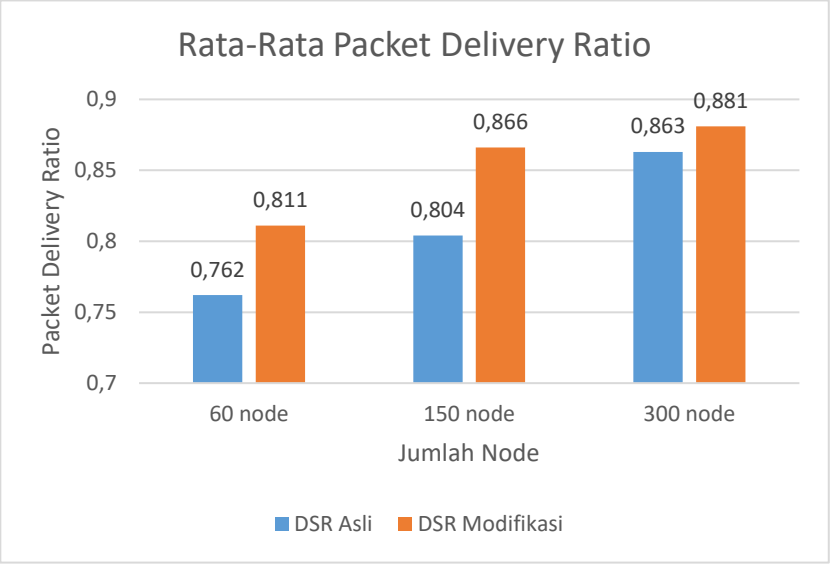
Dapat dilihat rata-rata penurunan yang terjadi untuk *forwarded route request* adalah sebesar 6.39%. Jika ketiga lingkungan tersebut dibandingkan dapat dilihat bahwa dengan jumlah *node* yang sedang, menghasilkan *forwarded route request* yang lebih bagus atau lebih sedikit daripada di lingkungan dengan jumlah *node* yang jarang maupun yang padat baik untuk *routing protocol* DSR asli maupun *routing protocol* DSR yang telah dimodifikasi. Dapat dilihat pula bahwa DSR yang telah dimodifikasi menghasilkan *forwarded route request* yang lebih bagus atau dalam hal ini lebih rendah daripada DSR asli dengan jumlah selisih *forwarded route request* yang cukup signifikan.

### 5.2.3 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* DSR asli dan *routing protocol* DSR yang telah dimodifikasi 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* untuk uji coba setiap lingkungan menggunakan *threshold* yang berbeda-beda karena satu *threshold* tidak bisa disamakan untuk semua lingkungan disebabkan jumlah tetangga yang dimiliki oleh setiap *node* pada setiap lingkungan berbeda-beda, tergantung pada jumlah *node* pada lingkungan simulasi tersebut. Hasil pengambilan data *packet delivery ratio* (PDR) pada

skenario *real* 60, 150, dan 300 *node* dengan menggunakan DSR asli masing dapat dilihat pada Gambar 5.5.



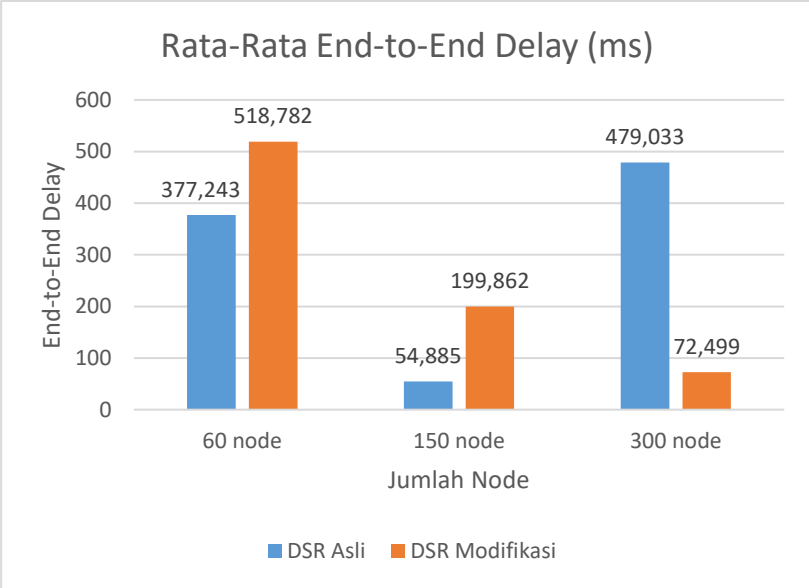

**Gambar 5.5** Grafik Rata-Rata *Packet Delivery Ratio* pada Skenario *Real*

Berdasarkan grafik pada Gambar 5.5 dapat dilihat bahwa *routing protocol* DSR yang telah dimodifikasi dan juga *routing protocol* DSR asli mengalami kenaikan yang signifikan pada *packet delivery ratio*. Pada lingkungan yang jarang dengan jumlah 60 *node*, menghasilkan perbedaan selisih PDR sebesar 0.049 atau mengalami kenaikan sebesar 6.43%, dimana *routing protocol* DSR yang telah dimodifikasi unggul dalam hal PDR tersebut. Pada lingkungan yang sedang dengan jumlah 150 *node*, menghasilkan perbedaan selisih PDR sebesar 0.062 atau mengalami kenaikan sebesar 7.71%, dimana *routing protocol* DSR yang telah dimodifikasi juga unggul dalam hal PDR tersebut dari DSR asli. Pada lingkungan yang padat dengan jumlah 300 *node*, menghasilkan perbedaan selisih PDR sebesar 0.018,

atau mengalami kenaikan sebesar 2.08%, dimana *routing protocol* DSR yang telah dimodifikasi juga unggul dalam hal PDR tersebut.

Dapat dilihat rata-rata kenaikan yang terjadi untuk PDR adalah sebesar 5.41%. 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 baik untuk *routing protocol* DSR asli maupun *routing protocol* DSR yang telah dimodifikasi. Dapat dilihat pula bahwa DSR yang telah dimodifikasi menghasilkan PDR yang lebih bagus daripada DSR asli dengan jumlah selisih PDR yang cukup signifikan.

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




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

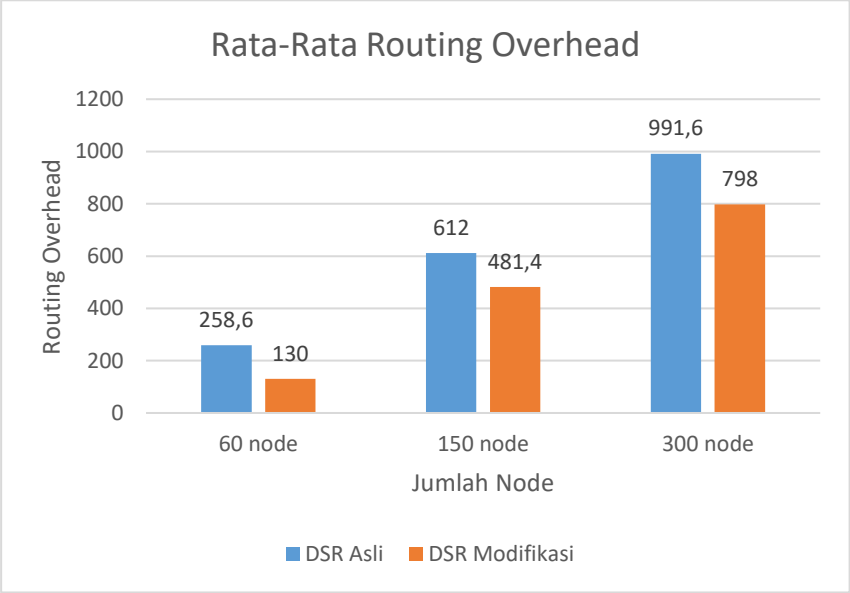
Berdasarkan grafik pada Gambar 5.6 dapat dilihat bahwa rata-rata *end-to-end delay* antara *routing protocol* DSR asli mengalami perubahan yang fluktuatif tetapi pada *routing protocol* DSR yang telah dimodifikasi mengalami penurunan yang signifikan. Pada lingkungan yang jarang dengan jumlah 60 *node*, terjadi perbedaan selisih *end-to-end delay* sebesar 141.539 ms antara *routing protocol* DSR asli dengan *routing protocol* DSR yang telah dimodifikasi atau mengalami kenaikan sebesar 37.52%, dimana *routing protocol* DSR yang asli 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 144.977 ms antara *routing protocol* DSR asli dengan *routing protocol* DSR yang telah dimodifikasi atau mengalami kenaikan sebesar 264.15%, dimana *routing protocol* DSR yang asli 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 406.534 ms antara *routing protocol* DSR asli dengan *routing protocol* DSR yang telah dimodifikasi atau mengalami penurunan sebesar 560.74%, dimana *routing protocol* DSR yang telah dimodifikasi jauh lebih unggul dalam hal *end-to-end delay* tersebut.

Jika ketiga lingkungan tersebut dibandingkan, memang pada lingkungan yang jarang, sedang, maupun padat tidak selalu lebih unggul dalam hal *end-to-end delay*. Juga, *routing protocol* DSR yang telah dimodifikasi tidak selalu lebih unggul daripada *routing protocol* DSR yang asli. Tetapi memang terlihat bahwa pada lingkungan jarang dan juga sedang, *routing protocol* DSR yang asli lebih unggul dalam hal *end-to-end delay* daripada *routing protocol* DSR yang telah dimodifikasi.

Untuk grafik *end-to-end delay* yang terjadi pada skenario *real* sama dengan grafik *end-to-end delay* yang terjadi pada skenario *grid*. 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. Sedangkan jika lingkungan yang jarang atau jumlah *node* sedikit, dan diimplementasikan reduksi *forwarding node* maka akan mengurangi rute yang akan terbentuk dan paket yang terkirim juga semakin sedikit.

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



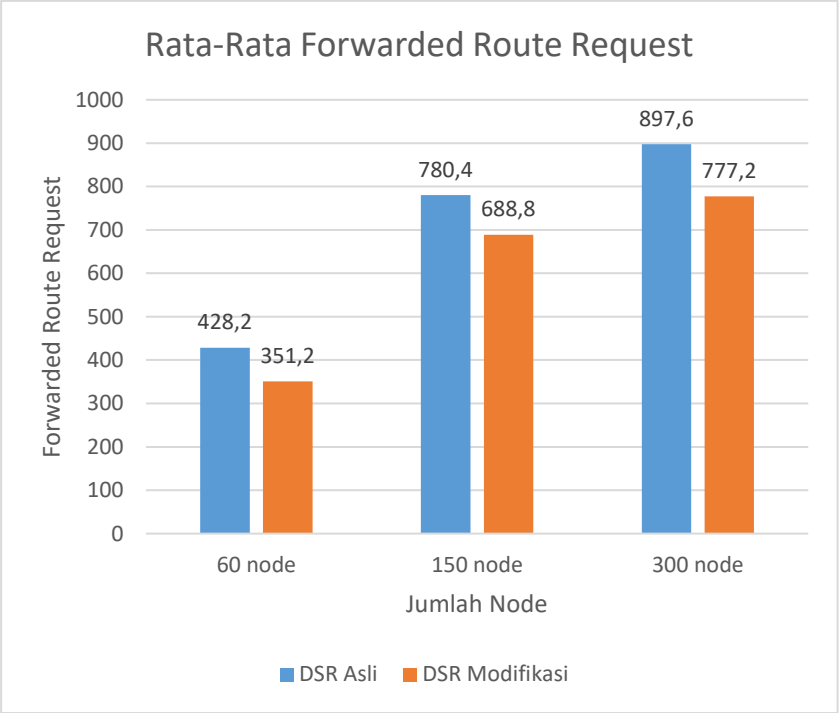

**Gambar 5.7** Grafik *Routing Overhead* pada Skenario *Real*

Berdasarkan grafik pada Gambar 5.7 dapat dilihat bahwa *routing protocol* DSR yang telah dimodifikasi dan juga *routing protocol* DSR asli mengalami kenaikan yang signifikan pada *routing overhead*. Pada lingkungan yang jarang dengan jumlah 60 *node*,

menghasilkan perbedaan selisih *routing overhead* sebesar 128.6 atau mengalami penurunan sebesar 98.92%, dimana *routing protocol* DSR yang telah dimodifikasi unggul dalam hal *routing overhead* tersebut karena menghasilkan *routing overhead* yang lebih rendah dari *routing* DSR asli. Pada lingkungan yang sedang dengan jumlah 150 *node*, menghasilkan perbedaan selisih *routing overhead* sebesar 130.6 atau mengalami penurunan sebesar 28.99%, dimana *routing protocol* DSR yang telah dimodifikasi juga unggul dalam hal *routing overhead* tersebut dari DSR asli. Pada lingkungan yang padat dengan jumlah 300 *node*, menghasilkan perbedaan selisih *routing overhead* sebesar 193.6 atau mengalami penurunan sebesar 24.26%, dimana *routing protocol* DSR yang telah dimodifikasi juga unggul dalam hal *routing overhead* tersebut.

Dapat dilihat rata-rata penurunan yang terjadi untuk *routing overhead* adalah sebesar 50.72%. Jika ketiga lingkungan tersebut dibandingkan dapat dilihat bahwa dengan jumlah *node* yang lebih sedikit atau pada lingkungan dengan *node* yang jarang, menghasilkan *routing overhead* yang lebih bagus atau lebih sedikit daripada di lingkungan dengan jumlah *node* yang sedang maupun yang padat baik untuk *routing protocol* DSR asli maupun *routing protocol* DSR yang telah dimodifikasi. Dapat dilihat pula bahwa *routing protocol* DSR yang telah dimodifikasi menghasilkan *routing overhead* yang lebih bagus atau dalam hal ini lebih rendah daripada DSR asli dengan jumlah selisih *routing overhead* yang cukup signifikan.

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* DSR yang telah dimodifikasi 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 77 atau mengalami penurunan sebesar 21.92%, dimana *routing protocol* DSR yang telah dimodifikasi unggul dalam hal *forwarded route request* tersebut karena menghasilkan *forwarded route request* yang lebih rendah dari *routing protocol* DSR asli. Pada

lingkungan yang sedang dengan jumlah 150 *node*, menghasilkan perbedaan selisih *forwarded route request* sebesar 91.6 atau mengalami penurunan sebesar 13.3%, dimana *routing protocol* DSR yang telah dimodifikasi juga unggul dalam hal *forwarded route request* tersebut dari *forwarded route request* DSR asli. Pada lingkungan yang padat dengan jumlah 300 *node*, menghasilkan perbedaan selisih *forwarded route request* sebesar 120.4 atau mengalami penurunan sebesar 15.49%, dimana *routing protocol* DSR yang telah dimodifikasi juga unggul dalam hal *forwarded route request* tersebut.

Dapat dilihat rata-rata penurunan yang terjadi untuk *forwarded route request* adalah sebesar 16.9%. Jika ketiga lingkungan tersebut dibandingkan dapat dilihat bahwa dengan jumlah *node* yang jarang, menghasilkan *forwarded route request* yang lebih bagus atau lebih sedikit daripada di lingkungan dengan jumlah *node* yang sedang maupun yang padat baik untuk *routing protocol* DSR asli maupun *routing protocol* DSR yang telah dimodifikasi. Dapat dilihat pula bahwa DSR yang telah dimodifikasi menghasilkan *forwarded route request* yang lebih bagus atau dalam hal ini lebih rendah daripada DSR asli dengan jumlah selisih *forwarded route request* yang cukup signifikan.

# 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. Dengan menambahkan perhitungan jumlah tetangga pada *routing protocol* DSR dan membatasi *node* mana saja yang bisa me*rebroadcast* paket RREQ dengan menggunakan *threshold* dari jumlah tetangga, maka dapat mengurangi jumlah *forwarding node* yang melakukan *rebroadcast*.
2. Dampak pembatasan *forwarding node* terhadap performa *routing protocol* DSR secara keseluruhan untuk skenario *real* menghasilkan peningkatan rata-rata *packet delivery ratio* sebesar 5.41%, rata-rata penurunan *routing overhead* sebesar 50.72%, dan juga rata-rata penurunan *forwarded route request* sebesar 16.9%.

## 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 lebih dari 10 kali percobaan untuk tiap skenario.
2. Kedepannya dapat melakukan perhitungan jumlah *node* tetangga yang lebih dinamis pada *routing protocol* DSR dan untuk lingkungan yang dinamis, misalnya pada *node* yang diluar jangkauan transmisi, dapat menghapus *node* tersebut dari

*list* simpanan *node* tetangga sehingga perhitungan jumlah *node* tetangga lebih akurat.

3. Ide untuk implementasi pengurangan jumlah *forwarding node* sudah bagus, kedepannya bisa mengimplementasikan dengan menambahkan aspek lain yang dijadikan untuk pembatasan *forwarding node* seperti energi, kecepatan, arah, dan lain-lain.

# DAFTAR PUSTAKA


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

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

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

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

[5] 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.

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

[7] "OpenStreetMap," OpenStreetMap, [Online]. Available: https://www.openstreetmap.org/about. [Diakses 6 December 2017].

[8] "JOSM," JOSM, [Online]. Available: http://wiki.openstreetmap.org/wiki/JOSM. [Diakses 6 December 2017].

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

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

*(Halaman ini sengaja dikosongkan)*

# LAMPIRAN

## A.1 Kode Fungsi DSRAgent::recv()

```
void
DSRAgent::recv(Packet* packet, Handler*)
/* handle packets with a MAC destination
address of this host, or
the MAC broadcast addr */
{
  hdr_sr *srh =  hdr_sr::access(packet);
  hdr_ip *iph =  hdr_ip::access(packet);
  hdr_cmn *cmh =
hdr_cmn::access(packet);
  // special process for GAF
  if (cmh->ptype() == PT_GAF) {
    if (iph->daddr() ==
(int)IP_BROADCAST) {
      if(cmh->direction() == hdr_cmn::UP)
        cmh->direction() = hdr_cmn::DOWN;

Scheduler::instance().schedule(ll,packet,
0);
      return;
    } else {
      target_->recv(packet, (Handler*)0);
      return;
    }
  }
  assert(cmh->size() >= 0);
  SRPacket p(packet, srh);
  //p.dest = ID(iph->dst(),::IP);
  //p.src = ID(iph->src(),::IP);
```

```
p.dest =
ID((Address::instance().get_nodeaddr(iph-
>daddr())),::IP);
  p.src =
ID((Address::instance().get_nodeaddr(iph-
>saddr())),::IP);
  assert(logtarget != 0);
  if (srh->valid() != 1) {
    unsigned int dst = cmh->next_hop();
    if (dst == IP_BROADCAST) {
    // extensions for mobileIP --Padma,
04/99.
    // Brdcast pkt - treat differently
      if (p.src == net_id)
        // I have originated this pkt
        sendOutBCastPkt(packet);
      else
        //hand it over to port-dmux
        port_dmux_->recv(packet,
(Handler*)0);
    }
    else {
      // this must be an outgoing packet,
it doesn't have a SR header on it
      srh->init();          // give
packet an SR header now
      cmh->size() += IP_HDR_LEN; // add
on IP header size
      if (verbose)
        trace("S %.9f _%s_ originating %s
-> %s", Scheduler::instance().clock(),
net_id.dump(), p.src.dump(),
p.dest.dump());
        handlePktWithoutSR(p, false);
      goto done;
    }
  }
```

```
else if (srh->valid() == 1) {
    if (p.dest == net_id || p.dest ==
IP_broadcast) {
      // this packet is intended for us
      handlePacketReceipt(p);
      goto done;
    }
    // should we check to see if it's an
error packet we're handling
    // and if so call
processBrokenRouteError to snoop
    if (dsragent_snoop_forwarded_errors
&& srh->route_error()) {
      processBrokenRouteError(p);
    }
    if (srh->route_request()) {
      // propagate a route_request that's
not for us
      id_int = atoi(net_id.dump());
      char *token;
      token = strtok(p.route.dump(), "
[]()");
      char *tmptoken;
      id_source = atoi(p.src.dump());
      while(token != NULL){
        tmptoken = token;
        token = strtok(NULL, " []()");
        int flag = 0;
        if (token == NULL){
          neighbor_id = atoi(tmptoken);
          for (int n = 0; n <=
neighbor_counter[id_int]; n++) {
            if(neighbor_list[id_int][n]
== -1){
                neighbor_list[id_int][n]
          = neighbor_id;
```

```
neighbor_counter[id_int]++;
              flag = 1;
              break;
            }
            else {
              if
(neighbor_list[id_int][n] ==
neighbor_id){
                flag = 1;
                break;
              }
            }
          }
          if (flag == 1 ) {
            break;
          }
        }
      }
      if (now <= 100.000000){
        handleRouteRequest(p);
      }
      else if (now >= 101.000000){
        if (neighbor_counter[id_int] >=
threshold) {
          handleRouteRequest(p);
        }
        else return;
      }
    }
    else {
      // we're not the intended final
recpt, but we're a hop
      handleForwarding(p);
    }
  }
```

```
else {
    // some invalid pkt has reached here
    fprintf(stderr,"dsragent: Error-
received Invalid pkt!\n");
    Packet::free(p.pkt);
    p.pkt =0; // drop silently
  }
  done:
  assert(p.pkt == 0);
  p.pkt = 0;
  return;
}
```

### A.2 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)   DSR;
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.3 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"

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)
$ns_ at 101.0000000000000000 "$cbr_(0) start"
$ns_ at 200.0000000000000000 "$cbr_(0) stop"
```

### A.4 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.5 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.6 Kode Skrip AWK *Routing Overhead*

```
BEGIN {
      rt_pkts = 0;
}
{
      if ($2 >= 101) {
            if (($1 == "s" || $1 == "f")
&& ($4 == "RTR") && ($7 == "DSR")) {

                  rt_pkts++;
            }
      }
}
END {
      printf "Routing Packets \t= %d \n",
rt_pkts;
}
```

### A.7 Kode Skrip AWK *Forwarded Route Request*

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

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

*(Halaman ini sengaja dikosongkan)*

## BIODATA PENULIS

Glleen Allan M lahir di Blitar pada 24 Mei 1996. Penulis menempuh pendidikan formal TK Aisyah (2001-2002), SDN 01 Jajar (2002-2003), SDN Kayu Putih 09 Pagi Jakarta (2003-2008), SMPN 99 Jakarta (2008-2011), SMAN 21 Jakarta (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 *Revolutionary Entertainment and Expo with Various Arts* (REEVA). Penulis pernah kerja praktik di PT. Telkom Indonesia Witel Denpasar periode Juli – Agustus 2017. Selama berkuliah, penulis juga menjadi administrator di Laboratorium Komputasi Berbasis Jaringan. Penulis dapat dihubungi melalui nomor *handphone* 085716411153 atau di email aglleen1@gmail.com.