TUGAS AKHIR - KI141502

# IMPLEMENTASI ADAPTIF *UPDATE PERIOD* PADA PROAKTIF *ROUTING PROTOCOL* DSDV BERDASARKAN FAKTOR KECEPATAN *VEHICLE* PADA VANETS


**KHARISMA MONIKA DIAN PERTIWI**
**NRP 05111440000092**

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

Dosen Pembimbing II
Henning Titi Ciptaningtyas, S.Kom., M.Kom.

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

TUGAS AKHIR – KI141502

# IMPLEMENTASI ADAPTIF *UPDATE PERIOD* PADA PROAKTIF *ROUTING PROTOCOL* DSDV BERDASARKAN FAKTOR KECEPATAN *VEHICLE* PADA VANETS

**KHARISMA MONIKA DIAN PERTIWI**
**NRP 05111440000092**

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

**Dosen Pembimbing II**
**Henning Titi Ciptaningtyas, S.Kom., M.Kom.**

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

*(Halaman ini sengaja dikosongkan)*


**UNDERGRADUATE THESES – KI141502**

# *ADAPTIVE UPDATE PERIOD IMPLEMENTATION ON PROACTIVE ROUTING PROTOCOL DSDV BASED ON VEHICLE SPEED FACTOR IN VANETS*

**KHARISMA MONIKA DIAN PERTIWI**
**NRP 05111440000092**

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

**Second Advisor**
**Henning Titi Ciptaningtyas, S.Kom., M.Kom.**

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

*(Halaman ini sengaja dikosongkan)*

## LEMBAR PENGESAHAN

# IMPLEMENTASI ADAPTIF *UPDATE PERIOD* PADA PROAKTIF *ROUTING PROTOCOL* DSDV BERDASARKAN FAKTOR KECEPATAN *VEHICLE* PADA VANETS

**TUGAS AKHIR**

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

Oleh:
**KHARISMA MONIKA DIAN PERTIWI**
**NRP: 05111440000092**

Disetujui oleh Pembimbing Tugas Akhir:

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

2. Henning Titi Ciptaningtyas, S.Kom., M.Kom. .....
   (NIP. 198407082010122004) (Pembimbing 2)

**SURABAYA**
**JUNI, 2018**

*[Halaman ini sengaja dikosongkan]*

**IMPLEMENTASI ADAPTIF *UPDATE PERIOD* PADA PROAKTIF *ROUTING PROTOCOL* DSDV BERDASARKAN FAKTOR KECEPATAN *VEHICLE* PADA VANETS**


| | |
|---|---|
| **Nama Mahasiswa** | **: KHARISMA MONIKA DIAN PERTIWI** |
| **NRP** | **: 05111440000092** |
| **Jurusan** | **: Informatika FTIK-ITS** |
| **Dosen Pembimbing 1** | **: Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.** |
| **Dosen Pembimbing 2** | **: Henning Titi Ciptaningtyas, S.Kom., M.Kom.** |


## Abstrak


VANETs (*Vehicular Ad hoc NETworks*) merupakan kumpulan kendaraan yang berkomunikasi nirkabel yang membentuk jaringan dinamis tanpa infrastruktur yang ditetapkan sebelumnya atau tanpa administrasi terpusat. Terdapat beberapa metode perutean untuk berkomunikasi antar kendaraan dalam jaringan. Mekanisme penentuan rute komunikasi dari *node* pengirim ke *node* penerima disebut dengan *routing*. Untuk berkomunikasi dalam jaringan VANETs dibutuhkan *protocol routing*, karena pengiriman pesan dari *node* pengirim ke penerima melewati beberapa *node* penghubung. *Protocol routing* akan mencarikan rute terbaik dari rute yang akan dilalui. Metode perutean tersebut digolongkan menjadi 2 macam proaktif dan reaktif. Proaktif dimana setiap kendaraan meng*update* rute ke setiap kendaraan yang terdapat dalam jaringan. Sedangkan metode reaktif dimana kendaraan akan mencari rute jika ingin berkomunikasi. Salah satu contoh *routing protocol* proaktif adalah DSDV.

Protokol DSDV memiliki 2 metode untuk memperbarui *routing table* yaitu full dump dan incemental. Full dump dilakukan

pada interval tertentu. Sedangkan incremental dilakukan pada kondisi tertentu, misalnya ketika *node* tetangga tidak merespon dalam waktu tertentu. Waktu interval untuk melakukan full dump disebut dengan *update period*. Pada protokol DSDV full dump dilakukan setiap 15 detik. Namun dalam VANETs, *node-node* bergerak sangat cepat sehingga dibutuhkan segera pembaruan *routing table*.

Pada Tugas Akhir ini, dilakukan modifikasi terhadap protokol DSDV dengan metode adaptif *update period*. Dengan metode ini, full dump pada protokol DSDV dilakukan berdasarkan kecepatan *node*. Semakin cepat *node*, semakin sering dilakukan pembaruan *routing table*. Dengan metode ini, mampu memperbarui performa protokol DSDV, hal ini dibuktikan dengan peningkatan nilai *packet delivery ratio* sebesar 6,10%.
**Kata kunci: VANETs, DSDV, Adaptif *Update Period***

***ADAPTIVE UPDATE PERIOD IMPLEMENTATION ON PROACTIVE ROUTING PROTOCOL DSDV BASED ON VEHICLE SPEED FACTOR IN VANETS***


| | |
|---|---|
| **Student's Name** | **: KHARISMA MONIKA DIAN PERTIWI** |
| **Student's ID** | **: 05111440000092** |
| **Department** | **: Informatics FTIK-ITS** |
| **First Advisor** | **: Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.** |
| **Second Advisor** | **: Henning Titi Ciptaningtyas, S.Kom., M.Kom.** |



***Abstract***

*VANETs (Vehicular Ad hoc NETworks) is a collection of wireless communicating vehicles that make up a dynamic network with no pre-defined infrastructure or centralized administration. There are some routing methods for communication between vehicle in the network. Routing is a mechanism to determine communication route from sender to receivers. Routing protocol needed in vanet, because packet sent in vanet network pass through some intermediate node. Routing protocol will look for best route to send the packet. Routing protocol categorized as proactive and reactive. Proactive routing protocol works by updating route in every nodes to another node in vanet network. Meanwhile reactive protocol only update route where node begin communication. Well known proactive protocol is DSDV.*

*DSDV protocol have 2 mechanisms to update their routing table, full dump and incremental. Full dump run only on some interval. Meanwhile incremental run only on certain condition such as neighbour's node unable to give response on certain time. Interval time for full dump called update period. In DSDV protocol, full dump run every 15 seconds. However, in vanet*

*network, nodes move so fast and make routing table less accurate. So, routing table must be updated often.*

*In this project, modification of DSDV protocol will be made using adaptive method to change update period of full dump mechanism. Full dump in DSDV will be run based on node speed. The faster the node, the more often routing table will be updated. Using this approach will increase DSDV packet delivery ratio by 6,10%.*

***Keywords: VANETs, DSDV, Adaptif Update Period***

## KATA PENGANTAR

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

Alhamdulillahirabbil'alamin, segala puji bagi Allah SWT, yang telah melimpahkan rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul "**IMPLEMENTASI ADAPTIF *UPDATE PERIOD* PADA PROAKTIF *ROUTING PROTOCOL* DSDV BERDASARKAN FAKTOR KECEPATAN *VEHICLE* PADA VANETS**". Tugas Akhir ini merupakan salah satu syarat dalam menempuh ujian sidang guna memperoleh gelar Sarjana Komputer. Dengan pengerjaan Tugas Akhir ini, penulis bisa belajar banyak untuk memperdalam dan meningkatkan apa yang telah didapatkan penulis selama menempuh perkuliahaan di Teknik Informatika ITS.

Selesainya Tugas Akhir ini tidak terlepas dari bantuan dan dukungan beberapa pihak, sehingga pada kesempatan ini penulis mengucapkan terima kasih kepada:

1. Allah SWT dan Nabi Muhammad SAW.
2. Keluarga penulis yang selalu memberikan dukungan 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 Ibu Henning Titi Ciptaningtyas, S.Kom., M.Kom. selaku dosen pembimbing penulis yang telah membimbing dan memberikan motivasi, nasehat dan bimbingan dalam menyelesaikan tugas akhir ini.
4. Bapak Darlis Herumurti, S.Kom., M.Kom. selaku kepala jurusan Informatika ITS.
5. Seluruh dosen dan karyawan Teknik Informatika ITS yang telah memberikan ilmu dan pengalaman kepada penulis selama menjalani masa studi di ITS.

6. Ibu Eva Mursidah dan Ibu Sri Budiati yang selalu mempermudah penulis dalam peminjaman buku di RBTC.
7. Teman-teman seperjuangan RMK AJK, yang telah menemani dan menyemangati penulis.
8. Teman-teman administrator NCC/KBJ, yang telah menemani dan menyemangati penulis selama penulis menjadi administrator, menjadi rumah kedua penulis selama penulis berkuliah.
9. Teman-teman angkatan 2014, yang sudah mendukung penulis selama perkuliahan.
10. Sahabat penulis yang tidak dapat disebutkan satu per satu yang selalu membantu, menghibur, menjadi tempat bertukar ilmu dan berjuang bersama-sama penulis.

Penulis menyadari bahwa Tugas Akhir ini masih memiliki banyak kekurangan sehingga dengan kerendahan hati penulis mengharapkan kritik dan saran dari pembaca untuk perbaikan ke depan.

Surabaya, Juni 2018

# DAFTAR ISI

*[Halaman ini sengaja dikosongkan]*

# DAFTAR GAMBAR

# DAFTAR TABEL

*[Halaman ini sengaja dikosongkan]*

# BAB I
# PENDAHULUAN

## 1.1 Latar Belakang

*Mobile Ad hoc NETworks* (MANETs) adalah kumpulan *mobile node* nirkabel yang dinamis membentuk jaringan tanpa infrastruktur yang ditetapkan sebelumnya. Dalam beberapa tahun terakhir, *Mobile Ad hoc NETworks* (MANETs) sangat diminati di seluruh dunia karena keuntungannya memiliki mobilitas dan fleksibilitas yang tinggi. Hal ini juga merupakan tantangan terbesar dalam komunikasi nirkabel. Untuk dapat berkomunikasi antar *node*, MANETs membutuhkan *routing*/perutean. Berdasarkan perlakuannya terhadap rute, *routing protocol* dalam MANETs dibedakan menjadi dua, yaitu *routing protocol* bersifat proaktif dan reaktif. Pada *routing protocol* proaktif, setiap *node* memperbarui *routing table* secara berkala, contohnya *Destination Sequence Distance Vector* (DSDV) dan *Optimized Link State Routing Protocols* (OSLR). Sedangkan pada *routing protocol* reaktif, *node* akan melakukan pencarian rute jika diperlukan, contohnya *Ad hoc On-demand Distance Vector* (AODV) dan *Dynamic Source Routing* (DSR). [1]

VANETs (*Vehicular Ad hoc NETworks*) adalah jenis MANETs yang bertujuan untuk komunikasi nirkabel antar kendaraan yang biasanya memiliki kecepatan relatif tinggi. Selain kecepatan tinggi, tidak seperti MANETs yang biasanya bergerak secara acak di area terbuka, *node* pada VANETs kebanyakan hanya bisa bergerak mengikuti pola tertentu dalam beberapa arah karena topologi jalan. Dengan demikian, perlu dilakukan perbaikan terhadap *protocol routing* MANETs [1].

Dalam beberapa tahun terakhir terdapat penelitian terkait metode untuk mengatasi meningkatkan kinerja *routing protocol* pada jaringan VANET, salah satunya adalah penerapan adaptif *update period* pada *routing protocol* proaktif DSDV [2]. *Update period* merupakan interval dalam memperbarui data rute. Pada *routing protocol* proaktif DSDV nilai *update period* adalah 15

detik. Dalam hal ini semua *node* akan memperbarui data rute setiap 15 detik.

Pada penelitian kali ini akan dilakukan cara untuk meningkatkan kinerja *routing protocol* DSDV dengan mengimplementasikan metode adaptif *update period*. Metode adaptif *update period* yang digunakan adalah berdasarkan faktor kecepatan pergerakan *node*. Diharapkan dengan metode ini setiap *node* memiliki *update period* berdasarkan kecepatan gerak *node* tersebut, semakin cepat *node* tersebut bergerak maka *update period* semakin sering. Sehingga dikemudian hari dapat memperbaiki algoritma *routing protocol* DSDV yang telah ada.

## 1.2 Rumusan Masalah

Tugas akhir ini mengangkat beberapa rumusan masalah sebagai berikut:

1. Bagaimana menerapkan metode adaptif *update period routing protocol* DSDV pada VANETs?
2. Bagaimana perbandingan performa *routing protocol* DSDV sebelum dan sesudah metode adaptif *update period* diterapkan pada skenario *grid* dan *real*?

## 1.3 Batasan Permasalahan

Permasalahan yang dibahas pada tugas akhir ini memiliki batasan sebagai berikut:

1. Simulator yang digunakan adalah simulator NS2.
2. *Routing protocol* yang digunakan adalah *routing protocol* DSDV.
3. Jaringan yang digunakan adalah VANETs.
4. Pembuatan skenario uji coba menggunakan *Simulation of Urban Mobility* (SUMO).

## 1.4 Tujuan

Tujuan dari tugas akhir ini adalah sebagai berikut:

1. Dapat menerapkan metode adaptif *update period* pada lingkungan VANETs dengan *routing protocol* DSDV.
2. Menganalisa perbandingan performa *routing protocol* DSDV sebelum dan sesudah metode adaptif *update period* diterapkan pada skenario *grid* dan *real*.

## 1.5 Manfaat

Manfaat dari hasil pembuatan tugas akhir ini adalah sebagai berikut:

1 Memberikan manfaat di bidang VANETs dengan mendapatkan performa *routing protocol* yang relatif stabil dalam lingkungan dinamis.

2 Memberikan informasi tentang pengaruh adaptif *update period* berdasarkan kecepatan *node* terhadap kinerja *routing protocol* DSDV dalam lingkungan VANETs.

3 Menerapkan ilmu yang dipelajari selama kuliah di Teknik Informatika ITS agar dapat berguna bagi masyarakat.

## 1.6 Metodologi

Tahapan-tahapan yang dilakukan dalam pengerjaan tugas akhir ini adalah sebagi berikut:

### 1.6.1 Penyusunan Proposal Tugas Akhir

Tahap awal tugas akhir ini adalah menyusun proposal tugas akhir. Pada proposal, berisi tentang deskripsi pendahuluan dari tugas akhir yang akan dibuat. Proposal juga berisi tentang garis besar tugas akhir yang akan dikerjakan sehingga memberikan gambaran untuk dapat mengerjakan tugas akhir sesuai dengan *timeline* yang dibuat. Gagasan untuk menerapkan metode adaptif

*update period* pada *routing protocol* DSDV untuk komunikasi V2V di lingkungan VANETs.

### 1.6.2 Studi Literatur

Pada tahap ini dilakukan untuk mencari informasi dan studi literatur apa saja yang dapat dijadikan sebagai referensi untuk membantu pengerjaan tugas akhir ini. Tahap ini merupakan tahap untuk memahami semua metode yang akan dikerjakan, sehingga memberi gambaran selama pengerjaan tugas akhir. Informasi didapatkan dari buku dan literatur yang berhubungan dengan metode yang digunakan. Informasi yang dicari mengenai *protocol* DSDV, VANETs, NS2 dan SUMO. Tugas akhir ini juga mengacu pada literatur jurnal dengan judul "*Performance Optimisation for DSDV in VANETs*" [2].

### 1.6.3 Implementasi Sistem

Implementasi merupakan perancangan sistem berdasarkan studi literatur dan pencarian informasi. Dalam tahap ini dilakukan implementasi metode yang telah diajukan pada proposal Tugas Akhir. Algoritma yang digunakan mengacu pada literatur jurnal. Implementasi dilakukan dengan menggunakan NS-2 sebagai *Network Simulator,* bahasa C/C++ sebagai bahasa pemrograman. Skenario uji coba dibuat dengan menggunakan SUMO dan JOSM.

### 1.6.4 Pengujian dan Evaluasi

Pada tahap ini, algortima yang telah diimplementasikan dan skenario yang telah dibuat akan dilakukan pengujian. Pengujian dilakukan pada NS2 *Network Simulator*. Kemudian dari pengujian tersebut akan didapatkan data yang digunakan untuk evaluasi kinerja protocol. Evaluasi dilakukan dengan mendapatkan nilai *packet delivery ratio* (PDR), rata-rata *end to end delay* (E2E), dan *routing overhead* (RO). Dari data hasil uji tersebut, dilakukan

analisa untuk membandingkan performa *routing protocol* yang telah dimodifikasi dengan *routing protocol original*.

### 1.6.5 Penyusunan Buku

Pada tahap ini dilakukan penyusunan laporan yang meliputi dasar teori, metode yang digunakan, implementasi serta hasil yang telah dikerjakan. Pada tahap ini disusun buku sebagai dokumentasi dari pelaksanaan tugas akhir yang mencangkup seluruh konsep, teori, implementasi, serta hasil yang telah dikerjakan. Laporan yang dihasilkan ini akan menjadi buku Tugas Akhir. Secara garis besar sistematika penulisan buku Tugas Akhir antara lain sebagai berikut:

1 Pendahuluan
   a. Latar Belakang
   b. Rumusan Permasalahan
   c. Tujuan
   d. Manfaat
   e. Metodologi
   f. Sistematika Penulisan
2 Tinjauan Pustaka
3 Perancangan Sistem
4 Implementasi
5 Pengujian dan Evaluasi
6 Kesimpulan dan Saran

## 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 dari metode dan algoritma yang digunakan dalam penyusunan tugas akhir ini. Secara garis besar, bab ini berisi tentang VANETs, DSDV, NS2, SUMO, OpenStreetMap dan AWK.

3. Bab III. Perancangan Perangkat Lunak
   Bab ini berisi pembahasan mengenai perancangan dari metode adaptif *update period* pada *routing protocol* DSDV yang digunakan dalam komunikasi V2V di lingkungan VANETs menggunakan NS-2 sebagai simulator. Dalam bab ini juga dibahas mengenai perancangan skenario *grid* dan skenario *real* menggunakan SUMO dan JOSM.

4. Bab IV. Implementasi
   Bab ini menjelaskan implementasi metode adaptif *update period* pada *routing protocol* DSDV yang digunakan dalam komunikasi V2V di lingkungan VANETs menggunakan NS-2 sebagai simulator. Dalam bab ini juga dibahas mengenai implementasi membuat skenario *grid* dan skenario *real* menggunakan SUMO dan JOSM.

5. Bab V. Pengujian dan Evaluasi
   Bab ini berisikan hasil uji coba dari implementasi metode adaptif *update period* pada *routing protocol* DSDV yang digunakan dalam komunikasi V2V di lingkungan VANETs menggunakan NS-2 sebagai simulator. Pengujian dilakukan dengan skenario yang di*generate* oleh SUMO untuk mendapatkan data uji PDR, E2E, dan RO yang nantinya akan dianalisa menggunakan skrip awk dan dilakukan perbandingan performa *routing protocol*.

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 program secara keseluruhan.

*[Halaman ini sengaja dikosongkan]*

# BAB II
# TINJAUAN PUSTAKA

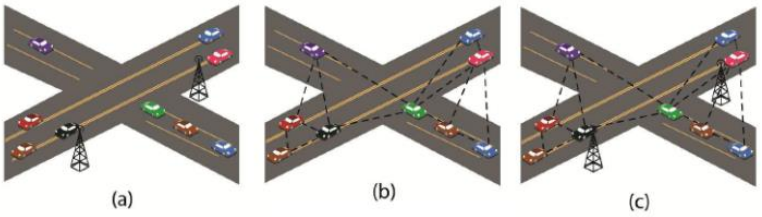
Bab ini berisi pembahasan mengenai teori-teori dasar dan *tools* yang digunakan dalam tugas akhir. Penjelesan ini bertujuan untuk memberikan gambaran atau definisi secara umum mengenai VANETs, *routing protocol* dan *tools* yang digunakan dalam pembuatan tugas akhir ini serta berguna sebagai penunjang dalam pengembangan riset yang berkaitan.

## 1.1 VANET (*Vehicular Ad hoc Network*)

VANETs adalah jenis jaringan yang dibuat dari konsep membangun jaringan kendaraan untuk kebutuhan atau situasi tertentu. VANET kini telah ditetapkan sebagai jaringan yang dapat diandalkan yang digunakan kendaraan untuk tujuan komunikasi di jalan raya atau lingkungan perkotaan. Tujuan utama dari VANET adalah untuk membantu sekelompok kendaraan untuk mengatur dan memelihara jaringan komunikasi di antara mereka tanpa menggunakan *base station* yang terpusat atau pengendali apapun. Salah satu aplikasi utama VANET adalah dalam situasi darurat medis kritis di mana tidak ada infrastruktur sementara sangat penting untuk menyampaikan informasi untuk menyelamatkan nyawa manusia [3]. Sedangkan manfaat lain dari VANETs antara lain untuk menghindari tabrakan antara 2 kendaraan dengan memperhitungkan jarak dan kecepatan dengan menggunakan sistem pengereman mendadak, mendeteksi keadaan yang berbahaya, misalnya kondisi jalan yang rusak, salju yang tebal, banjir, jalan yang diblokir dan kondisi yang licin, digunakan sebagai sinyal pengiriman untuk meminta bantuan secara otomatis pada saat terjadi kecelakaan. mendeteksi pengemudi yang tidak menaati peraturan lalu lintas, seperti menelepon dijalan, melewati batas kecepatan, mengemudi berlawanan arah, atau tidak menggunakan sabuk pengaman.

*Vehicular Ad hoc Network* (VANET) merupakan pengembangan dari *Mobile Ad hoc Network* (MANET) yang mempertimbangkan semua kendaraan dalam jaringan sebagai *node* tersebut untuk berkomunikasi dengan kendaraan lainnya pada radius tertentu [4]. Pada VANET maupun MANET, *node* yang bergerak bergantung pada *ad hoc routing protocol* untuk menentukan bagaimana mengirim pesan kepada tujuan. *Node* dalam jaringan dianggap sebagai *router* yang bebas bergerak dan bebas menentukan baik menjadi *client* maupun menjadi *router*. Perbedaan antara VANET dan MANET terletak pada kecepatan *node*. Pada VANET, *node* bergerak dengan kecepatan yang relatif tinggi dalam ruang gerak dan pola gerak yang terbatas. Hal tersebut merupakan sebuah tantangan dan memiliki peranan penting dalam menentukan desain jaringan seperti apakah yang akan dibuat.

Pada dasarnya, tidak ada topologi atau arsitektur yang tetap dalam VANETs. Pada umumnya VANETs terdiri dari kumpulan kendaraan yang saling berkomunikasi dan beberapa RSU *(road side unit*). Dalam arsitektur VANETs, kendaraan yang bergerak disebut dengan OBU (*unit on board*), dimana setiap kendaraan terdiri dari pemancar dan penerima sinyal jaringan nirkabel. Terdapat 3 kemungkinan komunikasi dalam VANETs yaitu komunikasi antara kendaraan dan RSU, komunikasi antara kendaraan dan kendaraan, komunikasi antara RSU dan RSU [5]. Pada Gambar 2.1 menunjukkan 3 kombinasi komunikasi dalam VANETs.




**Gambar 2.1 Kombinasi Komunikasi Dalam VANETs [5]**

Pada tugas akhir ini, VANETs digunakan sebagai jaringan nirkabel dalam menganalisis kinerja *routing protocol* DSDV.

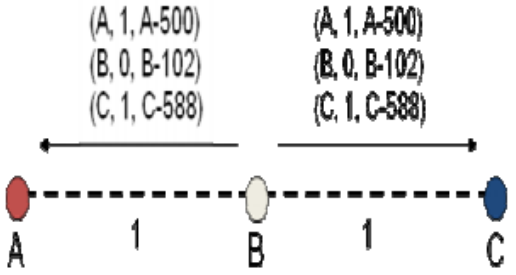
## 1.2 DSDV (*Destination Sequence Distance Vector*)

DSDV merupakan protokol proaktif yang diterapkan dalam jaringan MANETs. Protokol ini mengimplementasikan konsep *distance-vector*. DSDV menggunakan algoritma *routing* berdasarkan jarak terpendek untuk mendapatkan rute tunggal tanpa loop ke tujuan yang disimpan dalam tabel rute. DSDV meningkatkan beban pada jaringan karena pembaruan yang sering membuatnya tidak sesuai dengan jaringan berskala besar [6].

Protokol DSDV mampu mengatasi masalah utama yaitu *loop free path*. DSDV merupakan protokol proaktif atau *table driven* yang didasarkan pada algoritma *Bellman Ford*. DSDV diperkenalkan oleh C. Perkins dan P. Bhagwat pada tahun 1994 [7].

Dalam *routing protocol* DSDV, setiap *node* memiliki *routing table*. *Routing table* tersebut mencakup informasi rute ke semua *node* dalam jaringan tersebut. Setiap *routing table* berisi alamat IP tujuan, alamat IP hop berikutnya, jumlah hop untuk mencapai tujuan, *sequence number* dan waktu penyelesaian (*setting time*). Setiap *node* akan menginformasikan *routing table* yang dimiliki ke *node* tetangganya. Setiap *node* tetangga yang telah menerima informasi *routing table* akan memperbarui *routing table* yang dimiliki [8]. Dengan cara seperti itu setiap *node* dapat mengetahui rute ke semua *node*. Setiap *node* melakukan pembaruan rute secara berkala.

Terdapat 2 jenis pembaruan rute full dump dan incemental. Full dump digunakan untuk menyiarkan *routing table* secara lengkap. Paket full dump ini digunakan dalam kasus jaringan yang berubah dengan cepat. Paket incremental digunakan untuk mengirimkan entri *routing table* yang mengalami perubahan sejak pembaruan terakhir. Paket incremental ini digunakan dalam kasus jaringan yang relatif stabil [7].




| Dest. | Next | Metric | Seq |
|---|---|---|---|
| A | A | 0 | A-550 |
| B | B | 1 | B-102 |
| C | B | 2 | C-588 |

| Dest. | Next | Metric | Seq |
|---|---|---|---|
| A | A | 1 | A-550 |
| B | B | 0 | B-102 |
| C | C | 1 | C-588 |

| Dest. | Next | Metric | Seq. |
|---|---|---|---|
| A | B | 2 | A-550 |
| B | B | 1 | B-102 |
| C | C | 0 | C-588 |

**Gambar 2.2 *Node* B melakukan *broadcast routing table* [7]**

Pada Gambar 2.2 menunjukkan ilustrasi ketika *node* B melakukan *broadcast routing table* ke *node* tetangganya. Setiap *node* mencatat *destination* (tujuan) yang mungkin tercapai, *next node* yang mengarah ke *destination*, *cost* (*metric*), dan *sequence number*. Setiap *node* saling bertukar informasi secara rutin dengan melakukan *broadcast* ke *node* tetangga (*neighbor node*). Pembaruan *routing table* juga bisa terjadi apabila ada *event* tertentu, seperti rute putus atau pergerakkan *node* yang menyebabkan perubahan topologi jaringan dan perubahan informasi pada tabel. *Node* B melakukan *increment* terhadap *sequence number* menjadi 102 dan melakukan *broadcast* informasi *routing table* baru ke *node-node* tetangganya (A dan B). *Broadcast* ini akan terus dilakukan selama masih ada *node* pada jaringan yang terhubung sehingga *routing table* senantiasa baru.

Pada Tugas Akhir ini, *routing protocol* DSDV digunakan sebagai algoritma *routing protocol* untuk mengimplementasikan skenario pada lingkungan VANETs.

## 1.3 NS-2 (Network Simulator-2)

NS-2 merupakan sebuah *discrete event simulator* yang didesain untuk membantu penelitian pada bidang jaringan komputer. Pengembangan NS dimulai pada tahun 1989 sebagai sebuah varian dari REAL *network simulator*, pada tahun 1995 pengembangan NS didukung oleh DARPA melalui VINT project di LBL, Xerox PARC, UCB dan USC/ISI. NS kemudian memasuki versi dua pada tanggal 31 Juli 1995. Saat ini pengembangan NS didukung oleh DARPA melalui SAMAN dan NSF melalui CONSER beserta peneliti lainnya termasuk ACIRI [9].

NS bersifat *open source* di bawah GPL (*Gnu Public License*), sehingga NS dapat didownload dan digunakan secara gratis melalui web site http://www.isi.edu/nsnam/dist. Sifat *open source* juga mengakibatkan pengembangan NS menjadi lebih dinamis. *Network Simulator* (NS) mensimulasikan jaringan berbasis TCP/IP dengan berbagai medianya. Dapat mensimulasikan protokol jaringan (TCP/UDP/RTP), *Trafic behavior* (FTP, Telnet, CBR, dan lain-lain), *Queque management* (RED, FIFO, CBQ) algoritma *routing unicast* (*Distance Vector, Link State*) dan *multicast*, (PIM SM, PIM DM, DVMRP, *Shared Tree* dan *Bi Directional Shared Tree*), aplikasi multimedia yang berupa *layered video*, *quality of service* video-audio dan *transcoding*. NS juga mengimplementasikan beberapa MAC (IEEE 802.3, 802.11), di berbagai media, misalnya jaringan *wired* (LAN, WAN, point to point), *wireless* (seperti *mobile* IP, *Wireless* LAN), bahkan simulasi hubungan antar *node* jaringan yang menggunakan media satelit [10].

Pada Tugas Akhir ini, NS-2 digunakan untuk melakukan simulasi lingkungan VANET. *Trace file* yang dihasilkan oleh NS-2 juga digunakan sebagai informasi untuk mengukur performa *routing protocol*.

### 1.3.1 Instalasi

NS-2 membutuhkan beberapa *package* yang harus sudah terinstall sebelum memulai instalasi NS-2. Untuk menginstall *dependency* yang dibutuhkan dapat dilakukan dengan *command* yang ditunjukkan pada Gambar 2.3.

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

**Gambar 2.3 Perintah untuk menginstall *dependency* NS-2**

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

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

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

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

### 1.3.2 *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. Setiap jenis paket memiliki pola yang berbeda-beda. Secara umum format penulisan *trace file* ditunjukkan pada Gambar 2.5,Gambar 2.6 dan Gambar 2.7. Detail penjelasan *trace file* ditunjukkan pada Tabel 2.1.

```
s 0.556838879 _28_ AGT  --- 11 cbr 512 [0 0 0
0] ------- [28:0 29:0 32 0] [0] 0 0
```

**Gambar 2.5 Contoh Baris Pengiriman Data CBR**

```
r 72.042486122 _29_ AGT  --- 633 cbr 532 [13a
1d 4 800] ------- [28:0 29:0 27 29] [71] 6 0
```

**Gambar 2.6 Contoh Baris Penerimaan Data CBR**

```
s 0.522406133 _17_ RTR  --- 10 message 32 [0
ffffffff 16 800] ------- [22:255 -1:255 32 0]
```

**Gambar 2.7 Contoh Baris Pengiriman *Routing Packet***

**Tabel 2.1** Tabel Detail Penjelasan *Trace File*

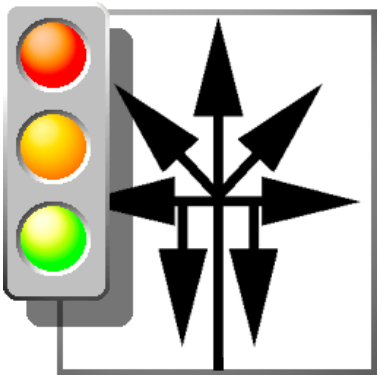
| Kolom ke- | Penjelasan | Isi |
|---|---|---|
| 1 | *Event* | s : *sent*<br>r : *received*<br>f : *forwarded*<br>D: *dropped* |
| 2 | *Time* | Waktu terjadinya *event* |
| 3 | ID *Node* | _x_ : dari 0 hingga banyak *node* pada topologi |
| 4 | *Layer* | AGT : *application*<br>RTR : *routing*<br>LL :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

| Kolom ke- | Penjelasan | Isi |
|---|---|---|
| 7 | Tipe Paket | message : paket *routing* DSDV<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* |
| 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*) |

## 1.4 SUMO (*Simulation of Urban Mobility*)

*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 [11]. Logo SUMO ditunjukkan pada Gambar 2.8.



**Gambar 2.8 Logo SUMO [11]**

Platform simulasi SUMO menawarkan beberapa fitur sebagai berikut:

- Simulasi mikroskopis - kendaraan, pejalan kaki dan transportasi umum dimodelkan secara eksplisit
- Interaksi *online* - kontrol simulasi dengan TraCI
- Simulasi lalu lintas multimodal, misalnya kendaraan, transportasi umum, dan pejalan kaki
- Jadwal waktu lampu lalu lintas dapat diimpor atau dihasilkan secara otomatis oleh SUMO
- Tidak ada batasan buatan dalam ukuran jaringan dan jumlah kendaraan simulasi
- Format impor yang didukung: OpenStreetMap, VISUM, VISSIM, NavTeq
- SUMO diimplementasikan dalam C ++ dan hanya menggunakan pustaka por*table*

SUMO memungkinkan pemodelan sistem lalu lintas intermodal termasuk kendaraan jalan, angkutan umum dan pejalan kaki. SUMO memiliki banyak alat pendukung atau *tools* yang menangani tugas-tugas seperti pencarian rute, visualisasi, impor jaringan dan perhitungan emisi. SUMO dapat ditingkatkan dengan model khusus dan menyediakan berbagai API untuk mengendalikan simulasi dari jarak jauh [12].

Pada Tugas Akhir ini, SUMO digunakan untuk membuat pemodelan lalu lintas. *Tools* pada SUMO yang digunakan dalam pembuatan Tugas Akhir ini adalah:

- *netgenerate* : *tools* yang digunakan untuk membuat peta jalan seperti *grid*. Selain itu *netgenerate* juga menentukan kecepatan maksimum kendaraan dalam jalan tersebut dan juga digunakan untuk membuat *traficlight* pada peta. Hasil dari *tools netgenerate* adalah sebuah *file* dengan ekstensi .net.xml. Dalam pembuatan Tugas Akhir ini, *netgenerate* digunakan untuk membuat peta skenario dalam bentuk peta *grid*.
- *netconvert* merupakan program CLI yang berfungsi untuk melakukan konversi dari peta dengan ekstensi lain seperti OpenStreetMap menjadi format *native* SUMO. Pada Tugas Akhir ini *netconvert* digunakan untuk mengkonversi peta dari OpenStreetMap.
- *RandomTrips.py* merupakan *tool* dalam SUMO untuk membuat rute acak yang akan dilalui oleh kendaraan dalam simulasi.
- *duarouter* merupakan *tool* untuk membuat detail perjalanan setiap kendaraan berdasarkan *output* dari *RandomTrips.py*.
- *sumo* adalah program yang melakukan simulasi lalu lintas berdasarkan data - data yang didapatkan dari map.net.xml dan route.rou.xml. Menghasilkan *file* skenario.xml.
- *sumo-gui* untuk melihat simulasi yang dilakukan oleh SUMO secara grafis.

- *traceExporter.py* yang bertujuan untuk mengkonversi *output* dari sumo menjadi format yang dapat digunakan pada simulator NS-2. Pada Tugas Akhir ini *traceExporter.py* digunakan untuk mengonversi skenario.xml menjadi skenario.tcl yang dapat digunakan pada NS-2.

## 1.5 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 [13]**.** Logo OpenStreetMap ditunjukkan pada Gambar 2.9.

**Gambar 2.9 Logo OpenStreetMap [13]**

Pada Tugas Akhir ini, OpenStreetMap digunakan untuk mendapatkan peta sebenarnya guna melakukan simulasi dengan skenario *real*.

## 1.6 JOSM

JOSM adalah editor tambahan untuk merubah data openstreetmap untuk Java 8. JOSM mendukung kemampuan dalam memuat jalur GPX, citra latar belakang dan data OSM dari data lokal maupun data online dan memungkinkan pengguna untuk merubah data OSM seperti *node*, jalan, dan relasi antar jalan serta metadata yang dimiliki oleh OSM [14].

JOSM dapat diunduh pada situs resmi JOSM. Pada Tugas Akhir ini JOSM diinstal dalam Ubuntu. Sebelum melakukan instalasi, perlu ditambahkan beberapa *package*. Selanjutnya menambahkan repo JOSM ke dalam Ubuntu. Setelah itu dilakukan pengunduhan JOSM dan registrasi *public key*. Setelah berhasil JOSM dapat diinstal.

Pada Tugas Akhir ini JOSM digunakan untuk menyunting peta osm yang telah diekspor dengan cara membersihkan bangunan-bangunan yang terdapat dalam peta. Dalam menyunting peta OSM, JOSM tidak memerlukan koneksi internet. Logo JOSM ditunjukkan pada Gambar 2.10.


**Gambar 2.10 Logo JOSM [14]**

## 1.7 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* [15].

Pada Tugas Akhir ini, AWK digunakan untuk memproses data yang diberikan oleh NS-2 agar mendapatkan analisis mengenai *packet delivery ratio*, *end to end delay*, dan *routing overhead*.

***[Halaman ini sengaja dikosongkan]***

# BAB III
# PERANCANGAN

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

## 3.1 Deskripi Umum

Pada Tugas Akhir ini akan diimplementasikan modifikasi protokol DSDV dengan menambahkan fungsi pembaruan *periodic update* masing-masing kendaraan berdasarkan kecepatan kendaraan tersebut. Modifikasi ini akan digunakan jika kecepatan kendaraan melebihi batas/*threshold*. Jika kecepatan kendaraan tidak melebihi *threshold* maka modifikasi tidak dilakukan.

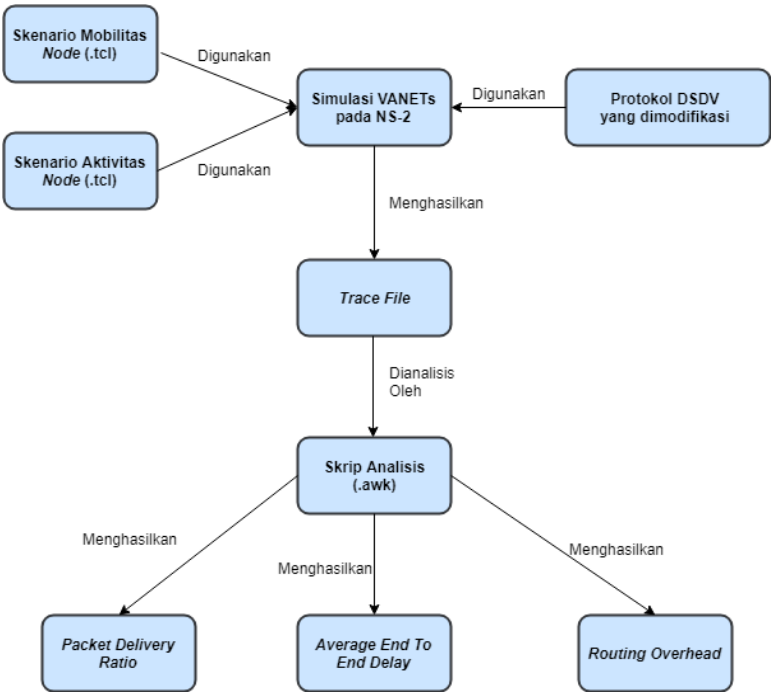
Modifikasi DSDV diawali dengan melakukan pengecekan data kecepatan masing-masing kendaraan. Data tersebut digunakan untuk menentukan periode *update* rute setiap kendaraan. Semakin cepat kendaraan tersebut bergerak, maka semakin semakin sering melakukan *update* rute, sehingga periode *update* semakin singkat. Diagram perancangan simulasi DSDV yang telah dimodifikasi dapat dilihat pada Gambar 3.1.

Dalam Tugas Akhir ini peta skenario yang digunakan adalah peta skenario *grid* dan peta skenario *real* lingkungan lalu lintas Kota Surabaya. Peta skenario *grid* dibuat dengan menggunakan *tools* dari SUMO, peta tersebut digunakan untuk melakukan simulasi lalu lintas pada SUMO. Sedangkan peta skenario *real* didapatkan dari peta OpenStreetMap.

Hasil pembuatan skenario dari SUMO tersebut kemudian digunakan untuk memodelkan lalu lintas pada NS-2 menggunakan protokol DSDV dan DSDV yang telah dimofikasi sebagai protokol pengiriman data komunikasi dalam VANETs. Simulasi yang telah dijalankan pada NS-2 tersebut menghasilkan *trace file* yang kemudian dianalisis menggunakan skrip AWK untuk mendapatkan

nilai dari *packet delivery ratio, end-to-end delay* dan *routing overhead* dari simulasi tersebut.

Hasil analisis tersebut dapat menjadi acuan dalam mengukur performa protokol DSDV dan protokol DSDV yang telah dimodfikasi dalam pengiriman paket data dengan menganilisis nilai *packet delivery ratio, end-to-end delay* dan *routing* overhead.




**Gambar 3.1 Diagram Perancangan Simulasi DSDV Modifikasi**

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

**Tabel 3.1 Daftar Istilah**

| No. | Istilah | Penjelasan |
|---|---|---|
| 1. | DSDV | *Destination Sequence Distance Vector* |
| 2. | NS-2 | *Network Simulator*-2 |
| 3. | SUMO | *Simulation Urban Mobility* |
| 4. | PDR | *Packet delivery ratio* |
| 5. | E2E | *Average End to end delay* |
| 6. | RO | *Routing Overhead* |
| 7. | *Threshold* | Batas kecepatan kendaraan |

## 3.2 Perancangan Skenario Mobilitas

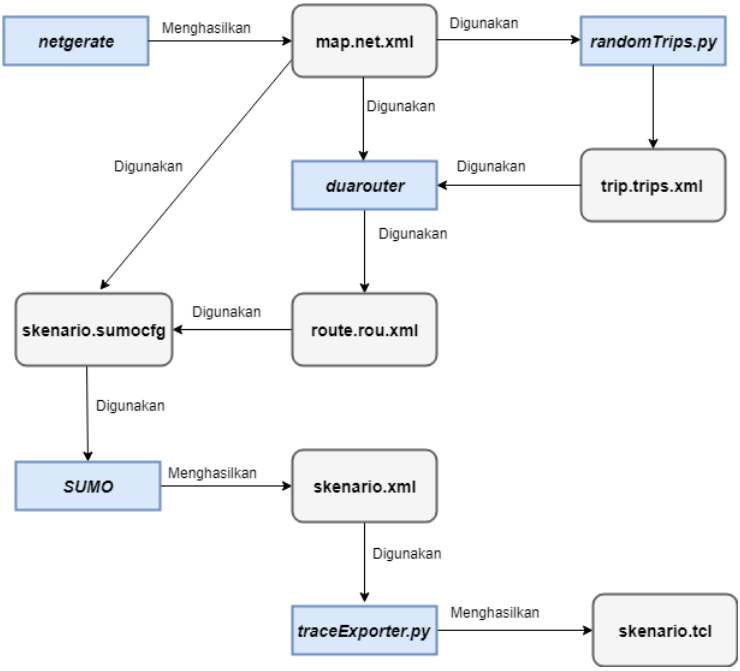
Perancangan skenario mobilitas dimulai dengan membuat area simulasi, pergerakan *node* dan implementasi pergerakan *node*. Dalam Tugas Akhir ini, terdapat dua macam peta skenario simulasi yang akan digunakan. Diantaranya adalah peta *grid* dan peta real. Peta *grid* merupakan peta jalan yang saling berpotongan, sehingga membentuk petak-petak. Peta *grid* digunakan sebagai simulasi awal VANET karena lebih seimbang dan stabil. Peta *grid* didapatkan dengan menentukan panjang dan jumlah petak area menggunakan SUMO. Sedangkan yang dimaksud peta *real* adalah peta asli yang digunakan sebagai area simulasi. Peta *real* didapatkan dengan mengambil area yang dimaksudkan sebagai area simulasi dari OpenStreetMap. Pada Tugas Akhir ini, peta *real* yang diambil adalah salah satu area di Surabaya.

### 3.2.1 Perancangan Skenario Mobilitas *Grid*

Perancangan skenario mobilitas *grid* diawali menentukan luas area simulasi yang dibutuhkan. Dari luas area tersebut ditentukan panjang dan lebar area. Selanjutnya dari panjang dan lebar area, ditentukan banyaknya persimpangan atau perpotongan jalan yang dibutuhkan, sehingga dapat diketahui jumlah petak. Setelah jumlah petak diketahui, dapat menentukan panjang dan lebar setiap petak. Misalnya area luas area simulasi 1000m2. Dari

luas tersebut didapatkan Panjang dan lebar area adalah 1000m x 1000m. Titik persimpangan yang dibutuhkan adalah 11 titik, sehingga dapat menghasilkan 10 petak. Dengan begitu untuk mendapatkan area 1000m x 1000m dengan 10 petak, maka Panjang dan lebar setiap petak adalah 100m x 100m.

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 jumlah kendaraan dan kecepatan maskimal 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*.




**Gambar 3.2 Alur Pembuatan Skenario *Grid***

Skenario mobilitas *grid* dihasilkan dengan menggabungkan *file* peta *grid* dan *file* pergerakan *node* yang telah digenerate, 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 trace*Exporter. Hasilnya berupa *file* yang berisi mobilitas dari setiap *node* (mobility.tcl) dan informasi lifetime dari setiap *node* (activity.tcl). Alur pembuatan skenario *grid* dapat dilihat pada Gambar 3.2.

Pada Tugas Akhir ini dirancang peta *grid* dengan luas area simulasi $1000m^2$ dengan 10 petak 100m x 100m. Simulasi dilakukan berdasarkan jumlah kendaraan yaitu 30, 40, 50 dan kecepatan maksimal kendaraan 25,30,35,40,45,50 m/s.

### 3.2.2 Perancangan Skenario Mobilitas *Real*

Perancangan skenario mobilitas *real* diawali dengan memilih area yang akan dijadikan simulasi. Pada Tugas Akhir ini, digunakan peta dari OpenStreetMap untuk mengambil area yang dijadikan model simulasi. Area yang diambil adalah sebagian area Kota Surabaya. Setelah memilih area, unduh dengan menggunakan fitur *export* dari OpenStreetMap. Peta hasil *export* dari OpenStreetMap ini memiliki ekstensi .osm.

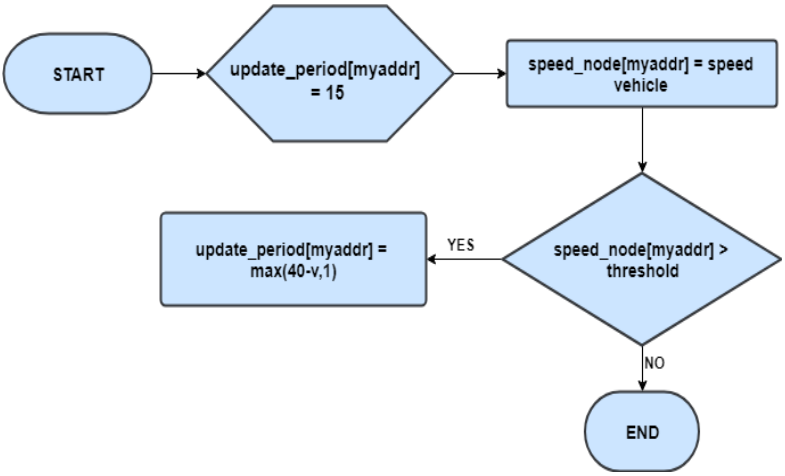
Setelah mendapatkan peta area yang dijadikan simulasi, peta tersebut dikonversi ke dalam bentuk *file* dengan ekstensi .net.xml menggunakan *tools* SUMO yaitu *netconvert*. Tahap berikutnya memiliki tahapan yang sama seperti tahapan ketika merancang skenario mobilitas *grid*, yaitu membuat pergerakan *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 digenerate. Hasil dari penggabungan tersebut merupakan *file* skenario yang berekstensi .xml. *File* yang dihasilkan tersebut dikonversi ke dalam bentuk *file* .tcl agar dapat diterapkan pada NS-2.

Alur pembuatan skenario *real* hampir sama dengan alur pembuatan skenario grid yang ditunjukkan pada Gambar 3.2. Namun peta map.net.xml dihasilkan dari peta osm yang telah dikonversi. Proses selanjutnya sama dengan pembuatan skenario grid.

## 3.3 Analisis dan Perancangan Implementasi Adaptif *Update Period* pada *Routing Protocol* DSDV

Protokol DSDV memiliki 2 metode untuk memperbarui *routing table* yaitu full dump dan incemental. Full dump dilakukan pada interval tertentu. Sedangkan incremental dilakukan pada kondisi tertentu, misalnya ketika *node* tetangga tidak merespon dalam waktu tertentu. Waktu interval untuk melakukan full dump disebut dengan *update period*. Pada protokol DSDV full dump dilakukan setiap 15 detik.

Protokol DSDV yang diajukan pada Tugas Akhir ini merupakan modifikasi dari protokol DSDV dengan mengubah mekanisme full dump pada protokol tersebut. Waktu full dump dilakukan berdasarkan pada kecepatan *node*. Sehingga setiap *node* memiliki *update period* masing-masing berdasarkan kecepatannya. Modifikasi DSDV ini diawali dengan cara mengecek kecepatan setiap *node*. Selanjutnya dilakukan perbandingan dengan *threshold* kecepatan. Jika kecepatan *node* melebihi *threshold*, maka dilakukan pembaruan nilai *update period* untuk *node* tersebut. Tetapi jika kecepatan *node* kurang dari atau sama dengan *threshold*, maka *update period* tetap pada nilai awal yaitu 15 detik. *Flow chart* perancangan modifikasi *routing protocol* DSDV dapat dilihat pada Gambar 3.3.



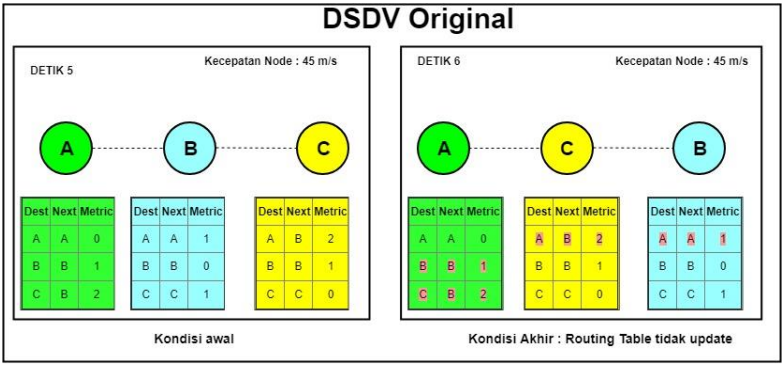

**Gambar 3.3 Perancangan Modifikasi *Routing Protocol* DSDV**

Sebelum melakukan modifikasi, penulis melakukan uji coba terhadap protokol DSDV. Uji coba dilakukan untuk mendapatkan batas kecepatan/*threshold*. Nilai *threshold* yang didapatkan akan digunakan dalam implementasi modifikasi. Berdasarkan pada literature jurnal yang diacu, threshold yang ditetapkan adalah 25 m/s. Oleh karena itu dilakukan pembuktian bahwa semakin cepat kendaraan bergerak maka performa protocol DSDV semakin menurun. Uji coba dilakukan dengan meng*generate* skenario dengan 30 node dan variasi kecepatan 15,20,25,30,35,40 m/s. Setiap variasi kecepatan di*generate* 10 skenario. Selanjutnya dilakukan evaluasi performa pada setiap skenario. Dari evaluasi tersebut, setiap 10 skenario pada setiap variasi kecepatan dilakukan perhitungan rata-rata. Hasil tersebut dibandingkan dengan variasi kecepatan lain.

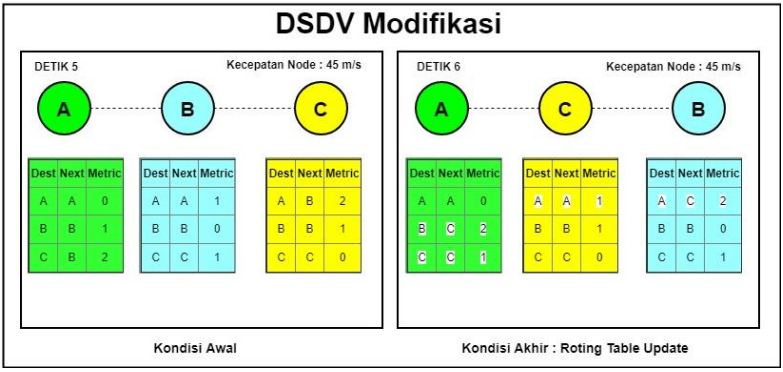
Hasil pengambilan data uji coba penentuan *threshold* dapat dilihat pada Tabel 3.2. Berdasarkan hasil evaluasi pada Tabel 3.2 terlihat bahwa semakin tinggi kecepatan, semakin menurun nilai PDR. Sedangkan pada *routing overhead* (RO) relatif stabil. Karena hal tersebut, ditentukan nilai *threshold* adalah 25.

**Tabel 3.2 Hasil Evaluasi Penentuan *Threshold***

| Kecepatan (m/s) | PDR (%) | E2D (ms) | RO |
|---|---|---|---|
| 15 | 61,07 | 40,13 | 1500 |
| 20 | 57,24 | 31,17 | 1499 |
| 25 | 56,71 | 108,94 | 1512 |
| 30 | 55,70 | 65,74 | 1510 |
| 35 | 53,57 | 32,40 | 1511 |
| 40 | 52,55 | 48,59 | 1516 |




**Gambar 3.4 Pembaruan Rute DSDV Original**




**Gambar 3.5 Pembaruan Rute DSDV Modifikasi**

Gambaran pembaruan rute pada DSDV original dan DSDV Modifikasi dapat dilihat pada Gambar 3.4 dan Gambar 3.5. Berdasarkan pada kedua gambar tersebut, pada detik ke-5 posisi *node* secara berturut-turut adalah A-B-C. Selanjutnya pada detik ke-6 posisi *node* berubah menjadi A-C-B. Pada detik ke-6 dengan DSDV *Original*, informasi perubahan rute yang tersimpan dalam *routing table* tidak diperbarui. Hal tersebut dapat terjadi karena DSDV *original* memperbarui rute setiap 15 detik. Sedangkan pada DSDV Modifikasi informasi pembaruan rute akan diperbarui berdasarkan kecepatan *node*. Karena kecepatan kendaraan adalah 45 m/s, maka pembaruan informasi rute dilakukan setiap 1 detik.

### 3.3.1 Perancangan Mengetahui Kecepatan Setiap *Node* dan *Default* Periode Update

Setiap *node* memiliki kecepatan pergerakan masing-masing. Pengecekan kecepatan *node* dan pembaruan nilai *update period* dilakukan pada setiap aktivitas *node*. Hal tersebut dilakukan agar data kecepatan merupakan data terbaru. Aktivitas *node* tersebut antara lain: penerimaan paket, pengiriman paket dan *update* rute. Pseudocode mengetahui kecepatan *node* dapat dilihat pada Gambar 3.6.

```
. . .
speed_node = speed_vehicle->myaddr
update_period(myaddr, speed_node, perup_)
. . .
```

**Gambar 3.6 Pseudocode Mengetahui Kecepatan *Node***

### 3.3.2 Perancangan Perhitungan *Update Period* Setiap *Node*

Dalam modifikasi ini, setiap *node* memiliki nilai *update* period. Data nilai *update period* disimpan dalam sebuah *array*. Nilai awal *update period* setiap *node* adalah 15, karena nilai *default update period protocol* DSDV adalah 15. Perhitungan nilai pembaruan *update period* dilakukan jika kecepatan *node* melebihi *threshold* yang telah ditentukan. Tetapi jika tidak melebihi *threshold* maka nilai *update period* menjadi nilai awal/*default*. Pseudocode perhitungan *update period node* dapat dilihat pada Gambar 3.7.

```
v = speed_node
If v > threshold :
    Perup_node[myaddr] = max (-v + 40, 1)
Else :
    Perup_node[myaddr] = 15
```

**Gambar 3.7 Pseudocode Perhitungan *Update* Period**

## 3.4 Perancangan Simulasi pada NS-2

Simulasi VANET 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 3.3 Konfigurasi Lingkungan Simulasi VANETs**

| No | Parameter | Spesifikasi |
|---|---|---|
| 1 | *Network simulator* | NS-2, 2.35 |

| No | Parameter | Spesifikasi |
|---|---|---|
| 2 | *Routing protocol* | DSDV |
| 3 | Waktu simulasi | 200 detik |
| 4 | Area simulasi | 1000m x 1000 m |
| 5 | Radius Transmisi | 400 m |
| 6 | Banyak kendaraan | 30, 40, 50 |
| 7 | Kecepatan Maksimal | 25,30,35,40,45,50 m/s |
| 8 | Tipe Koneksi | UDP |
| 9 | Agen pengirim | *Constant Bit Rate* (CBR) |
| 10 | Ukuran Paket | 512 bytes |
| 11 | Kecepatan generasi paket | 1 paket per detik |
| 12 | *Protocol* MAC | IEEE 802.11 |
| 13 | *Source/Destination* | Statis |
| 14 | Tipe Kanal | *Wireless channel* |
| 15 | Tipe Antena | *OmniAntena* |
| 16 | Tipe Interface Queue | Droptail/PriQueue |
| 17 | Tipe *Trace* | Old Wireless Format *Trace* |

Beberapa baris kode dan sebuah fungsi ditambahkan pada *file* dsdv.h dan dsdv.cc. Pada *file* dsdv.h ditambahkan deklarasi *array* penyimpanan data *update period* dan kecepatan setiap *node*. Pada *file* dsdv.cc ditambahkan sebuah fungsi untuk melakukan perbandingan kecepatan dengan *threshold* dan perhitungan *update period*. Perhitungan serta pembaruan *update period* dilakukan jika kecepatan *node* tersebut lebih dari *threshold*. Fungsi ini akan dipanggil pada fungsi yang lain pada *file* dsdv.cc. Pada saat simulasi NS-2 dijalankan, maka *routing protocol* DSDV akan

melakukan perubahan nilai *update period* berdasarkan kecepatannya.

## 3.5 Perancangan Metrik Analisis

Metrik yang akan dianalisis pada Tugas Akhir ini adalah *Packet delivery ratio* (PDR), *End-to-End Delay* (E2D), dan *Routing overhead* (RO). Parameter-parameter tersebut dianalisis untuk dapat membandingkan performa dari *routing protocol* DSDV yang telah dilakukan modifikasi dan *routing protocol* DSDV yang asli. Penjelasan untuk masing-masing metrik analisis adalah sebagai berikut:

### 3.5.1 *Packet Delivery Ratio* (PDR)

*Packet delivery ratio* merupakan perbandingan antara jumlah paket yang diterima dengan jumlah paket yang dikirimkan. *Packet delivery ratio* dihitung dengan persamaan 3.1, dimana nilai PDR dinyatakan dalam bentuk persentase.

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

Keterangan:

| | | |
|---|---|---|
| PDR | : | *Packet delivery ratio* |
| *received* | : | Jumlah paket data yang diterima |
| *sent* | : | Jumlah paket data yang dikirm |

*Packet delivery ratio* dapat menunjukkan keberhasilan paket yang dikirimkan. Semakin tinggi *packet delivery ratio*, 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 dan *node* tujuan menerima paket. Dari *delay* tiap paket tersebut semua dijumlahkan dan dibagi dengan jumlah paket yang berhasil diterima, maka akan didapatkan rata-rata *end to end delay*, yang dapat dihitung dengan persamaan 3.2.

$$\text{E2D} = \frac{\sum_{i \leq sent}^{i=0} (t_{received\ [i]} - t_{sent[i]})}{sent} \qquad (3.2)$$

Keterangan:

| | | |
|---|---|---|
| E2D | : | Rata-rata *End to end delay* (ms) |
| i | : | Urutan/ id paket ke-i |
| $t_{received}$ | : | Waktu paket diterima (ms) |
| $_{tsent}$ | : | Waktu paket dikirim (ms) |
| sent | : | Jumlah paket data yang dikirm |

### 3.5.3 *Routing overhead* (RO)

*Routing overhead* adalah jumlah paket kontrol *routing* yang ditransimisikan per data paket yang terkirim ke tujuan selama simulasi terjadi. RO dihitung berdasarkan jumlah paket *routing* yang ditransmisikan. Baris yang mengandung *routing overhead* pada *trace file* ditandai dengan paket yang bertipe *send* (s) / *forward* (f) dan terdapat *header* paket dari protokol DSDV. Perhitungan *routing overhead* dapat dilihat dengan persamaan 3.3.

Pada *routing protocol* DSDV, *routing overhead* dapat meningkatkan nilai PDR. Hal ini dapat terjadi karena DSDV merupakan *routing protocol* proaktif, dimana semakin sering melakukan perutean, maka *routing table* semakin *up-to-date*,

sehingga hal tersebut dapat meningkatkan PDR, karena rute yang digunakan merupakan rute yang paling baru.

$$RO = sentRTR + forwardedRTR \quad (3.3)$$

Keterangan:

| | | |
|---|---|---|
| RO | : | *Routing Overhead* |
| *sentRTR* | : | Jumlah paket *routing* yang dikirim |
| *forwardedRTR* | : | Jumlah paket *routing* yang diteruskan |

# BAB IV
# IMPLEMENTASI

Bab ini membahas mengenai implementasi dari perancangan sistem yang telah dijabarkan pada bab sebelumnya.

## 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*

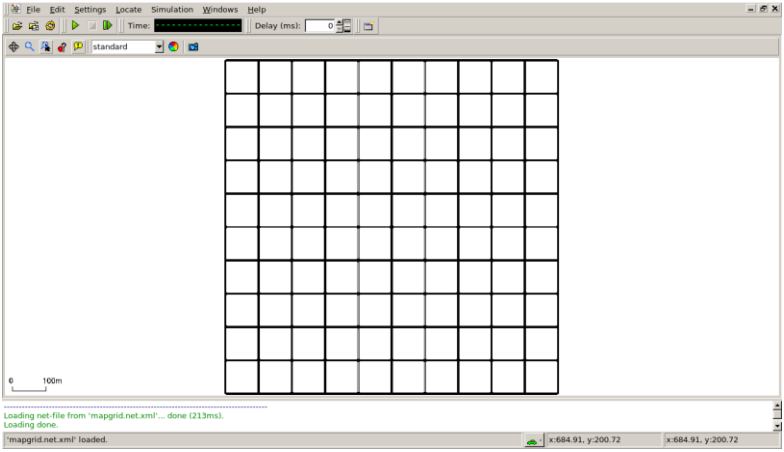
Pada Tugas Akhir ini, penulis membuat beberapa skenario *grid*. Beberapa skenario *grid* tersebut dibuat berdasarkan variasi jumlah *node* dan kecepatan maksimal kendaraan. Variasi jumlah *node* dan kecepatan maksimal kendaraan telah dijelaskan pada subbab 3.4.

Dalam mengimplementasikan skenario *grid*, SUMO menyediakan *tools* untuk membuat peta *grid* yaitu *netgenerate*. Pada Tugas Akhir ini, penulis membuat peta *grid* dengan luas 1000 m x 1000 m yang terdiri dari titik persimpangan antara jalan vertikal dan jalan horizontal sebanyak 11 titik x 11 titik. Dengan jumlah titik persimpangan sebanyak 11 titik tersebut, maka terbentuk 10 buah petak. Setiap persimpangan memiliki *traffic* light system. Sehingga untuk mencapai luas area sebesar 1000 m x 1000 m dibutuhkan luas per petak sebesar 100 m x 100 m. Perintah netgenerate untuk membuat peta tersebut dengan kecepatan *default* kendaraan sebesar 25 m/s dapat dilihat pada Gambar 4.1.

```
netgenerate --grid --grid.number=11 --
grid.length=100 --default.speed=25 --
tls.guess=1 --output-file=map.net.xml
```

**Gambar 4.1 Perintah *netgenerate***

Perintah *netgenerate* pada Gambar 4.1 menghasilkan sebuah *file* bernama map.net.xml. *File* tersebut dapat dibuka menggunakan *sumo-gui*. Gambar peta yang dibuat menggunakan *netgenerate* dapat dilihat pada Gambar 4.2.




**Gambar 4.2 Peta hasil netgenerate**

Setelah peta terbentuk, maka dilakukan pembuatan *node* dan pergerakan *node* dengan menentukan titik awal dan titik akhir setiap *node* secara random menggunakan *tools RandomTrips*. Perintah penggunaan *tool RandomTrips* untuk membuat titik awal dan titik akhir sebanyak 28 *node* dapat dilihat pada Gambar 4.3. Hanya dibuat 28 node karena 2 node merupakan node statis.

```
python /usr/share/sumo/tools/RandomTrips.py -n
mapgrid.net.xml -e 28 -l --trip-
attributes="departLane=\"best\"
departSpeed=\"max\" departPos=\"random_free\""
-o trips.trips.xml
```

**Gambar 4.3 Perintah Penggunaan *Tools RandomTrips***

Selanjutnya dibuatkan rute yang digunakan kendaraan untuk mencapai tujuan dari *file* hasil sebelumnya menggunakan *tools*

*duarouter*. *Tools duarouter* ini membaca *file* peta hasil generate dan *file* titik awal dan titik akhir *node*. *Tools* ini menghasilkan pergerakan setiap *node* dari titik awal menuju titik akhir yang telah didefinisikan. Secara *default* algoritma yang digunakan untuk membuat rute ini adalah algoritma dijkstra. Perintah penggunaan *tools duarouter* dapat dilihat pada Gambar 4.4.

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

**Gambar 4.4 Perintah Penggunaan *Tools Duarouter***

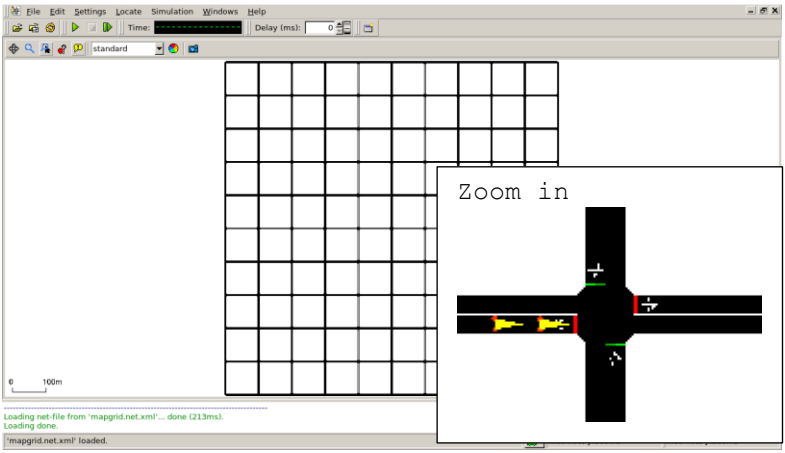
Untuk menjadikan peta dan pergerakan *node* yang telah di*generate* menjadi sebuah skenario dalam bentuk *file* berekstensi .xml, dibutuhkan sebuah *file* skrip dengan ekstensi .sumocfg guna menggabungkan *file* peta dan rute pergerakan *node*. Isi dari *file* skrip tersebut dapat dilihat pada Gambar 4.5.

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

**Gambar 4.5 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*. Cuplikan pergerakan kendaraan dapat dilihat pada Gambar 4.6.




**Gambar 4.6 Cuplikan Simulasi Pergerakan Kendaraan**

Kemudian dari *file* .sumocfg dibuat *file* skenario dalam bentuk *file* .xml menggunakan *tools* SUMO. Perintah untuk membuat *file* skenario tersebut menggunakan *tools* SUMO dapat dilihat pada Gambar 4.7.

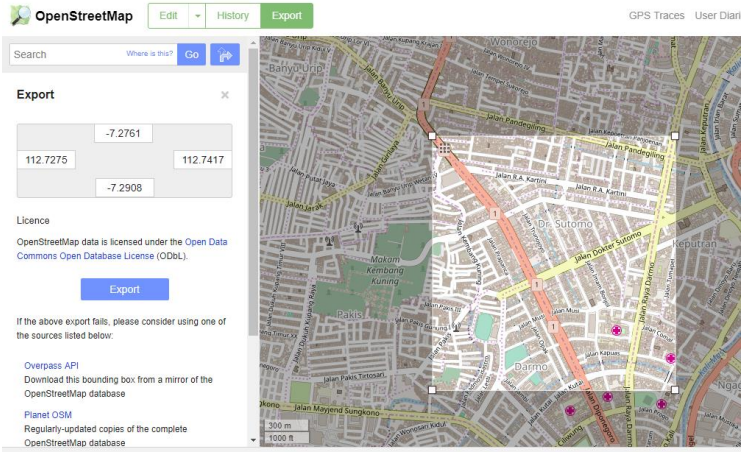
```
python sumo-0.27.1/tools/traceExporter.py --
fcd-input=scenario.xml --ns2-mobility-
output=scenario.tcl
```

**Gambar 4.7 Perintah Membuat *File* Skenario.tcl**

### 4.1.2 Skenario Real

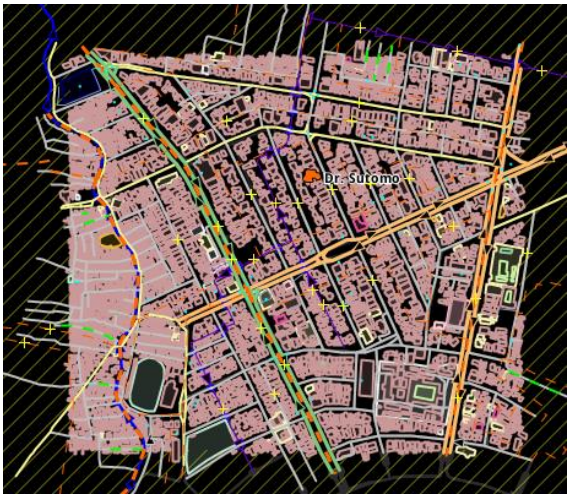
Dalam mengimplementasikan skenario *real*, langkah pertama adalah dengan menentukan area yang akan dijadikan area simulasi. Pada Tugas Akhir ini penulis mengambil area sekitar Jalan Dr.Soetomo Surabaya. Setelah menentukan area simulasi,

ekspor data peta tersebut dari OpenStreetMap seperti yang ditunjukkan pada Gambar 4.8.

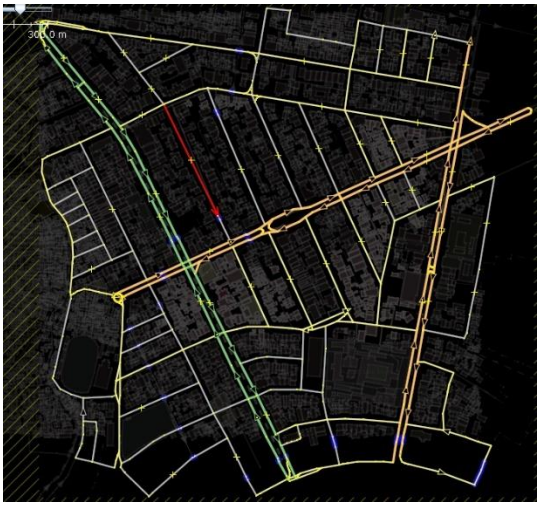



**Gambar 4.8 OpenStreetMap Area Sekitar Jl. Dr.Soetomo**

*File* hasil ekspor dari OpenStreetMap tersebut adalah *file* peta dengan ekstensi .osm. *File* .osm tersebut perlu diubah menggunakan JOSM. Pengubahan dilakukan untuk menghilangkan bangunan-bangunan yang terdapat dalam peta. Selain itu juga untuk menghilangkan jalan yang terputus. Gambar *file* .osm sebelum diubah dan setelah diubah dapat dilihat pada Gambar 4.9 dan Gambar 4.10.




**Gambar 4.9 Peta OSM sebelum diedit**



**Gambar 4.10 Peta OSM setelah diedit**

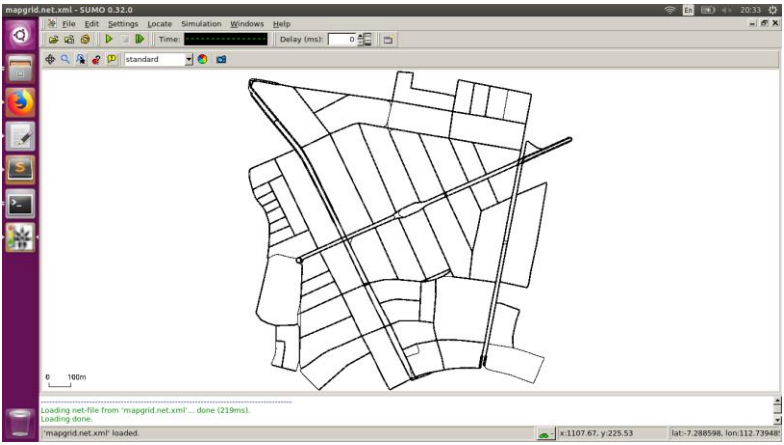
Selanjutnya *file* peta .osm tersebut dikonversi menjadi peta dalam bentuk *file* berekstensi .xml menggunakan *tools* netconvert.

Perintah untuk menggunakan netconvert dapat dilihat pada Gambar 4.11.

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

**Gambar 4.11 Perintah Penggunaan *Tools* Netconvert**

Hasil konversi peta dari *file* berekstensi .osm menjadi *file* berekstensi .xml dapat dilihat menggunakan *tools sumo-gui* seperti yang ditunjukkan pada Gambar 4.12.




**Gambar 4.12 Peta XML Hasil Konversi**

Langkah selanjutnya sama dengan ketika membuat skenario mobilitas *grid*, yaitu membuat *node*, asal, dan tujuan *node* 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 .xml tersebut menjadi *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 Adaptif *Update Period* Pada *Routing Protocol* DSDV

*Routing protocol* DSDV merupakan salah satu *routing protocol* yang umum digunakan dalam komunikasi *ad hoc*. Protokol DSDV merupakan salah satu protokol proaktif. Dalam hal ini protokol DSDV akan memperbarui *routing table* dalam interval tertertu. Namun protokol ini belum optimal dalam pengiriman paket ketika pergerakan *node* cepat. Hal ini disebabkan karena interval memperbarui *routing table* tidak mengikuti kecepatan *node*.

Pada Tugas Akhir ini dilakukan modifikasi pada *routing protocol* DSDV dengan mengubah mekanisme interval memperbarui *routing table* berdasarkan kecepatan *node*. Hal ini dilakukan agar protokol DSDV lebih optimal dalam pengiriman paket. Sehingga pada Tugas Akhir ini, mampu meningkatkan performa protokol DSDV.

Implementasi modifikasi *routing protocol* DSDV dibagi menjadi 2 bagian:

- Implementasi Mengetahui Kecepatan Setiap *Node*
- Implementasi Menghitung Nilai *Update Period* Setiap *Node*

Kode implementasi dari *routing protocol* DSDV pada NS-2 versi 2.35 berada pada direktori ns-2.35/dsdv. Di dalam diretori tersebut terdapat beberapa file, diantaranya dsdv.cc, dsdv.h dan sebagainya. Pada Tugas Akhir ini, penulis memodifikasi *file* dsdv.h dan dsdv.cc. Pada *file* dsdv.h, penulis menambahkan deklarasi fungsi untuk pembaruan nilai *update period* dan *array* untuk menyimpan data *update period* setiap *node*. Pada *file* dsdv.cc, penulis menambahkan fungsi penghitungan nilai *update period* dan melakukan pengecekan kecepatan *node* dalam beberapa fungsi yang telah ada. Setiap melakukan perubahan terhadap baris kode,

harus melakukan *remake* terhadap NS2. Hal ini dilakukan agar perubahan yang telah dibuat dapat digunakan untuk simulasi. Pada bagian ini penulis akan menjelaskan langkah-langkah dalam mengimplementasikan adaptif *update period* dalam *routing protocol* DSDV.

### 4.2.1 Implementasi Mengetahui Kecepatan Setiap *Node* dan *Default* Periode Update

Sesuai yang telah dijelaskan dalam subbab 3.3.1 , langkah awal yang dilakukan adalah mengetahui kecepatan setiap *node* dan *default* periode *update*. Nilai periode *update* dalam *protocol* DSDV telah dideklarasikan dalam *file* dsdv.h. Pada *file* dsdv.h, periode *update* dideklarasikan sebagai variabel `perup_` dengan nilai 15. Secara *default protocol* DSDV tidak mengetahui kecepatan setiap *node*. Namun nilai kecepatan *node* ini merupakan atribut dari *class node*. Sehingga untuk mengetahui kecepatan setiap *node* dapat memanfaatkan fungsi *`node_`*`->speed()`. Nilai kecepatan *node* tersebut dijadikan sebagai parameter untuk melakukan penghitungan nilai *update* period. Fungsi penghitungan nilai *update period* dipanggil dalam beberapa fungsi, diantaranya `DSDVTriggerHandler::handle()`, `DSDV_Agent::cancelTriggersBefore()`, `DSDV_Agent::needTriggeredUpdate()`, `DSDV_Agent::helper_callback()`, `DSDV_Agent::updateRoute()`, `DSDV_Agent::process`*`Update`* `()`, `DSDV_Agent::forwardPacket()`, `DSDV_Agent::recv()`. Potongan kode untuk mengetahui kecepatan *node* yang dilakukan dalam fungsi `DSDV_Agent::recv` dapat dilihat pada Gambar 4.13. Kode implementasi pemanggilan fungsi update*_period* dapat dilihat pada lampiran A1 sampai A8.

```
Void DSDV_Agent::recv (Packet * p, Handler *){
. . .
update_period(myaddr_,node_->speed(), perup_);
. . .
}
```

**Gambar 4.13 Potongan Kode Mengetahui Kecepatan Dalam Fungsi DSDV_Agent::recv**

### 4.2.2 Implementasi Perhitungan *Update Period* Setiap *Node*

Setelah mendapatkan nilai kecepatan setiap kendaraan dan nilai periode *update* maka dilakukan proses penghitungan. Nilai kecepatan yang didapatkan akan dibandingkan dengan *threshold*. Jika melebihi *threshold* maka dilakukan pembaruan nilai *update period*. Jika tidak, maka nilai *update period node* tersebut adalah nilai *default*.

Langkah awal yang dilakukan adalah deklarasi fungsi penghitungan nilai *update* period. Deklarasi dilakukan dalam *file* dsdv.h. Fungsi update_period() dideklarasikan sebagai fungsi global agar dapat digunakan oleh *class* yang lain. Potongan kode sumber deklarasi fungsi dapat dilihat pada Gambar 4.14 .

```
void update_period(int alamat, double speed,
double perup_ori);
```

**Gambar 4.14 Potongan Kode Deklarasi Fungsi *Update Period***

Selain deklarasi fungsi, pada *file* dsdv.h juga dilakukan deklarasi *array* yang digunakan untuk menyimpan nilai *update period* setiap *node*. Potongan kode sumber deklarasi *array* dapat dilihat pada Gambar 4.15.

```
double *perup_node = (double*) malloc (200 *
sizeof(double));
```

**Gambar 4.15 Potongan Kode Sumber Deklarasi *Array***

Selanjutnya adalah penulisan fungsi update_*period* dalam *file* dsdv.cc. Dalam fungsi ini dilakukan inisialisasi variabel dan penghitungan nilai sementara *update period*, `int temp_perup = round(40 - speed).` Selanjutnya dilakukan perbandingan nilai kecepatan dan *threshold*. Jika kecepatan melebihi *threshold* maka nilai *update period node* tersebut diubah dengan nilai `temp_perup`, namun jika nilai `temp_perup = 0` maka nilai *update period node* tersebut adalah 1. Dalam hal ini ketika kecepatan *node* mencapai 40 m/s atau setara dengan 144 kph, maka *node* tersebut akan memperbarui *routing table* setiap 1 detik. Tetapi jika kecepatan tidak melebihi *threshold*, maka nilai *update period node* tersebut adalah nilai *default update* period.

Dengan perhitungan tersebut, jika kecepatan *node* melebihi *threshold* maka nilai *update* periode setiap *node* dapat mengikuti kecepatan *node* tersebut. Semakin cepat *node* tersebut bergerak, maka semakin sering *node* tersebut memperbarui *routing table*. Potongan kode sumber fungsi update_*period* dapat dilihat pada Gambar 4.16.

```
void update_period(int alamat, double speed,
double perup_ori){
  int temp_perup = round(40 - speed);
  if(speed>25){
    if (temp_perup >= 1)
    {
      perup_node[alamat]= temp_perup;
    }
    else if (temp_perup < 1)
    {
      perup_node[alamat]= 1;
    }
  }
  else {
    perup_node[alamat]= perup_ori;
  }
}
```

**Gambar 4.16 Potongan Kode Sumber Fungsi *Update* Period**

## 4.3 Implementasi Simulasi pada NS-2

Implementasi simulasi VANETs diawali dengan pendeskripsian lingkungan simulasi pada sebuah *file* tcl. *File* ini berisikan konfigurasi setiap *node* dan langkah-langkah yang dilakukan selama simulasi. Potongan kode konfigurasi lingkungan simulasi dapat dilihat pada Gambar 4.17.

```
set val(chan)      Channel/WirelessChannel
set val(prop)      Propagation/TwoRayGround
set val(netif)     Phy/WirelessPhy
set val(mac)       Mac/802_11
set val(ifq)       Queue/DropTail/PriQueue
set val(ll)        LL
set val(ant)       Antenna/OmniAntenna
set val(ifqlen)    50
set val(nn)        30
set val(rp)        AODV
set val(energymodel)    EnergyModel
set val(initialenergy)  100
set val(lm)        "off"
set val(x)         1000
set val(y)         1000
set val(stop)      200
set val(cp)        "traffic"
set val(sc)        "scenario.tcl"
```

**Gambar 4.17 Potongan Kode Konfigurasi Lingkungan Simulasi**

Pada konfigurasi simulasi NS-2 dilakukan pemanggilan terhadap *file traffic* dan *file* scenario.tcl. *File traffic* merupakan *file* yang berisikan konfigurasi *node* pengirim, *node* penerima dan paket yang dikirim. *File* skenario merupakan *file* pergerakan *node* yang telah degenerate menggunakan SUMO. Kode implementasi pada NS-2 dapat dilihat pada lampiran A.9 Kode Skenario NS-2.

*File traffic* dapat di*generate* menggunakan *traffic generator* yang telah tersedia pada NS-2. Skrip *traffic generator* ini terdapat

pada direktori ~ns-allinone-2.35/ns-2.35/indep-utils/cmu-scen-gen dan disimpan dalam bentuk *file* cbrgen.tcl. *File* ini dapat digunakan untuk membuat *traffic* connection CBR ataupun TCP pada jaringan pergerakan antar *node*. Dalam menggunakan *traffic generator* harus mendefiniskan beberapa parameter seperti tipe koneksi yg digunakan, jumlah *node* dalam simulasi jaringan, jumlah seed, maksimum koneksi yang dikehendaki dan paket *rate*. Hasil dari *traffic generator* ini disimpan dalam *file* traffic. Perintah yang digunakan dalam *traffic generator* dapat dilihat pada Gambar 4.18.

```
ns cbrgen.tcl [-type cbr | tcp] [-nn nodes] [-
seed seed] [-mc connections] [-rate rate]
```

**Gambar 4.18 Perintah Menggunakan *Traffic Generator***

Dalam Tugas Akhir ini, penulis membuat koneksi CBR diantara 30 *node*, memiliki maksimal 1 koneksi dengan nilai *seed* 1.0 dan jumlah paket yang dikirimkan per detik sebanyak 1. Konfigurasi tersebut disimpan dalam *file traffic*. Perintah yang digunakan untuk membuat konfigurasi tersebut dapat dilihat pada Gambar 4.19.

```
ns cbrgen.tcl CBR –nn 30 –seed 1.0 –mc 1  rate
1 > traffic
```

**Gambar 4.19 Perintah Membuat *File Traffic***

Potongan konfigurasi *traffic* yang dihasilkan dapat dilihat pada Gambar 4.20. *Traffic* yang dihasilkan menggunakan koneksi CBR dan agent UDP. Dalam hal ini, koneksi UDP terjadi antara *node* 28 dan 29. Interval pengiriman paket data dilakukan setiap satu detik dengan besar paket data 512 byte dan maksimal pengiriman paket 1000.

```
# nodes: 30, max conn: 1, send rate: 1, seed:
1.0
#
# 1 connecting to 2 at time 2.5568388786897245
```

```
#
set udp_(0) [new Agent/UDP]
$ns_ attach-agent $node_(1) $udp_(28)
set null_(0) [new Agent/Null]
$ns_ attach-agent $node_(2) $null_(29)
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_ 1000
$cbr_(0) attach-agent $udp_(0)
$ns_ connect $udp_(0) $null_(0)
$ns_ at 2.5568388786897245 "$cbr_(0) start"
```

**Gambar 4.20 Kode Sumber Traffic**

## 4.4 Implementasi Metrik Analisis

Simulasi yang telah dijalankan oleh NS-2 menghasilkan keluaran sebuah *trace file* yang berisikan data mengenai apa saja yang terjadi selama simulasi dalam bentuk *plain text*. Dari data *trace file*, 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* (PDR)

Pada Tugas Akhir ini, penghitungan nilai PDR menggunakan skrip AWK. Persamaan untuk menghitung nilai PDR dapat dilihat pada persamaan 3.1. Skrip AWK menghitung nilai PDR berdasarkan pada *trace file* yang dihasilkan setelah menjalankan simulasi. Penjelasan mengenai *trace file* dijabarkan pada subbab 1.3.2. Kode skrip AWK untuk menghitung nilai PDR dapat dilihat pada lampiran A.10 Kode Skrip AWK *Packet delivery ratio*.

Proses perhitungan PDR dilakukan dengan menghitung jumlah paket data terkirim yang dilakukan oleh *node* pengirim dan

jumlah paket data yang diterima oleh *node* penerima pada satu *trace file*. Penambahan jumlah paket terkirim dilakukan apabila pada baris *trace* yang bersangkutan mengandung semua kondisi dimana huruf "s" yang menandakan kolom pertama mengandung *send packet*, kolom ke-3 menunjukkan bahwa *node* yang melakukan pengiriman adalah *node* pengirim, kolom ke-4 dan kolom ke-7 mengandung huruf "AGT" dan "cbr" yang menandakan pengiriman paket yang dilakukan adalah pengiriman paket data. Pencatatan jumlah paket yang diterima dilakukan apabila pada baris *trace* yang bersangkutan mengandung semua kondisi dimana huruf "r" yang menandakan kolom pertama mengandung *received packet*, kolom ke-3 menunjukkan bahwa *node* yang menerima paket data adalah *node* penerima, kolom ke-4 dan kolom ke-7 mengandung huruf "AGT" dan "cbr" yang menandakan penerimaan paket yang diterima adalah paket data. Perhitungan dilakukan sampai baris terakhir *trace* file, dan hasilnya adalah hasil hitung nilai PDR simulasi skenario. Setelah itu nilai PDR dapat dihitung dengan persamaan yang telah dijelaskan. Pseudocode untuk menghitung PDR dapat dilihat pada Gambar 4.21.

```
ALGORITMA PDR( trace file)
//Input: trace file simulasi skenario
//Ouput: PDR
BEGIN (
sent ← 0
recv ← 0
recv_id ← 0
pdr ← 0
)
#count packet send
(if ($1 == "s" and $3 == "_28_" and $4 ==
"AGT" and $7 == "cbr" )
   sent +1;
)
#count packet receive
```

```
if ($1 == "r" and $3 == "_29_" and $4 == "AGT"
and $7 == "AGT")
    recv +1;
)
END (
Pdr ← ( recv / sent ) * 100
print pdr)
```

**Gambar 4.21 Pseudocode Menghitung PDR**

Perintah mengeksekusi skrip awk untuk menganalisis *trace file* adalah `awk –f pdr.awk 30`*node*`.tr`. Hasil keluaran dari skrip yang telah dieksekusi tersebut adalah `pdr: xx.xx %`.

### 4.4.2 Implementasi Rata-rata *End to end delay* (E2E)

Pada Tugas Akhir ini, penghitungan nilai rata-rata E2D menggunakan skrip AWK. Persamaan untuk menghitung nilai E2D dapat dilihat pada persamaan 3.2. Skrip AWK menghitung nilai E2D berdasarkan pada *trace file* yang dihasilkan setelah menjalankan simulasi. Kode skrip AWK untuk menghitung nilai rata-rata E2D dapat dilihat pada lampiran A.11 Kode Skrip AWK *End to end delay*.

Perhitungan nilai rata-rata E2D dilakukan dengan menghitung selisih waktu paket data terkirim yang dilakukan oleh *node* pengirim dan waktu paket data diterima oleh *node* penerima di dalam satu *trace* file. Pencatatan waktu paket terkirim pada kolom ke-2 dilakukan apabila pada baris *trace* mengandung beberapa kondisi yaitu kolom ke-1 mengandung huruf “s” yang menandakan send packet, kolom ke-3 menunjukkan ID *node* pengirim, kolom ke-4 dan kolom ke-7 mengandung kata “AGT” dan “cbr” yang menunjukkan pengiriman paket data. Perhitungan waktu dan pencatatan ID serta jumlah paket yang diterima dilakukan apabila baris *trace* mengandung beberapa kondisi yaitu kolom ke-1 mengandung huruf “r” yang menandakan *received packet*, kolom ke-3 menunjukkan ID *node* penerima, kolom ke-4 dan kolom ke-7 mengandung kata “AGT” dan “cbr” yang

menunjukkan pengiriman paket data. Perhitungan dilakukan sampai baris terakhir *trace file*. Selanjutnya dilakukan penghitungan nilai rata-rata E2D. Pseudocode E2D ditunjukkan pada Gambar 4.22.

```
ALGORITMA E2D (trace file)
//Input: trace file
//Ouput:  E2D
BEGIN (
for i in pkt_id
   pkt_id[i] ← 0
for i in pkt_sent
   pkt_sent[i] ← 0
for i in pkt_recv
   pkt_recv[i]← 0
delay = avg_delay ← 0
recv ← 0
recv_id ← 0
)
(if ($1 == "s" and $3 == "_28_" and $4 ==
"AGT" and $7 == "cbr)
   pkt_sent[$6] ← $2
if ($1 == "r" and $3 == "_29_" and $4 == "AGT"
and $7 == "cbr" and recv_id != $6 )
   recv +1
   recv_id ← $6
   pkt_recv[$6]← $2
)
END (
for i in pkt_recv
   delay += pkt_recv[i] - pkt_sent[i]
avg_delay ← delay/recv
print avg_delay)
```

**Gambar 4.22 Pseudocode Menganalisis *End to end delay***

Perintah mengeksekusi skrip awk untuk menganalisis *tracefile* adalah `awk awk –f e2d.awk 30node.tr`. Hasil keluaran dari skrip yang telah dieksekusi tersebut adalah `E2D: xx.xx (float number)`.

### 4.4.3 Implementasi *Routing overhead* (RO)

Pada Tugas Akhir ini, penghitungan nilai *routing overhead* menggunakan skrip AWK. Persamaan untuk menghitung nilai RO dapat dilihat pada persamaan 3.3. Skrip AWK menghitung nilai RO berdasarkan pada *trace file* yang dihasilkan setelah menjalankan simulasi. Kode skrip AWK untuk menghitung nilai RO dapat dilihat pada lampiran A.12 Kode Skrip AWK *Routing* Overhead.

Implementasi penghitungan metrik *routing overhead* DSDV dihitung apabila beberapa kondisi terpenuhi yaitu kolom ke-1 diawali dengan huruf "s" yang berarti send packet atau huruf "f" yang berarti forward packet, kolom ke-4 mengandung kata "RTR" yang berarti layer *routing*, kolom ke-7 mengandung kata "message" yang menunjukkan paket *routing* DSDV. Pseudocode menghitung RO dapat dilihat pada Gambar 4.23.

```
ALGORITMA RO-DSDV (trace file)
//Input: trace file
//Ouput: jumlah routing overhead protocol DSDV
BEGIN (
rt_pkts ← 0
)
(if (($1=="s" || $1=="f") && ($4 == "RTR") &&
($7 = "message"))
   rt_pkts +1)
)
END (
print rt_pkts)
```

**Gambar 4.23 Pseudocode Analisa *Routing* Overhead**

Perintah mengeksekusi skrip awk untuk menganalisis *tracefile* adalah `awk awk –f RO.awk 30node.tr`. Hasil keluaran dari skrip yang telah dieksekusi tersebut adalah `RO: xx.xx (float number)`.

# BAB V
# HASIL UJI COBA DAN EVALUASI

Bab ini membahas mengenai uji coba dan evaluasi dari skenario yang telah disimulasikan di NS-2.

## 5.1 Lingkungan Uji Coba

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

**Tabel 5.1** Spesifikasi Perangkat yang Digunakan

| Komponen | Spesifikasi |
|---|---|
| **CPU** | Intel(R) Core(TM) i3-3217U CPU @ 1.80 GHz x 2 |
| **Sistem Operasi** | Linux Ubuntu 16.04 64-bit |
| **Linux Kernel** | 3.19.0-32-generic |
| **Memori** | 1.8 GiB |
| **Penyimpanan** | 26.5 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*, dan *routing overhead* menggunakan kode pada lampiran masing-masing pada A.10 Kode Skrip AWK *Packet delivery ratio*, A.11 Kode Skrip AWK *End to end delay*, A.12 Kode Skrip AWK *Routing* Overhead.

## 5.2 Hasil Uji Coba

Hasil uji coba skenario *grid* dan skenario *real* dapat dilihat sebagai berikut:

### 5.2.1 Hasil Uji Coba Skenario *Grid*

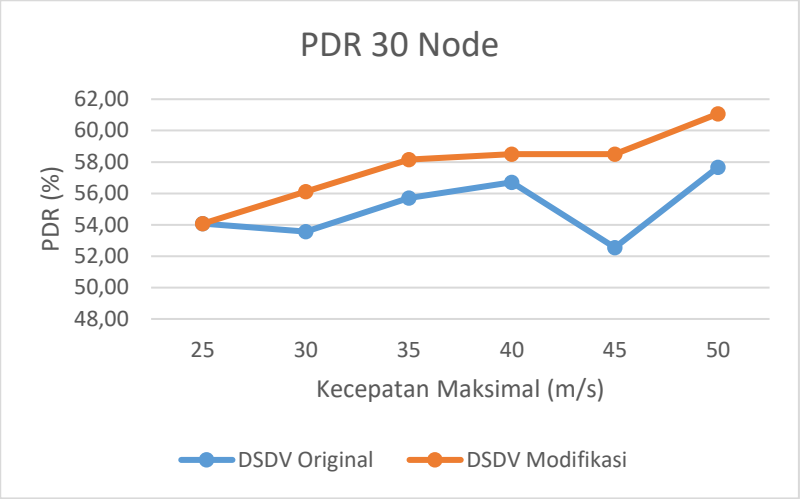
Pengujian pada skenario *grid* digunakan untuk melihat perbandingan *packet delivery ratio*, rata-rata *end to end delay*, dan *routing overhead* antara *routing protocol* DSDV *original* dan *routing protocol* DSDV yang telah dimodifikasi. Pengujian dilakukan sesuai dengan perancangan parameter lingkungan simulasi yang dapat dilihat pada Tabel 3.3.

Pengambilan data uji PDR, rata-rata *end to end delay*, dan *routing overhead* dengan luas area 1000 m x 1000 m dan *node* sebanyak 30, 40, dan 50 dengan variasi kecepatan maksimal 25, 30, 35, 40, 45 dan 50 m/s. Dalam hal ini, variasi jumlah *node* digunakan untuk melihat pengaruh jumlah *node* terhadap performa. Sedangkan variasi kecepatan digunakan untuk melihat pengaruh kecepatan terhadap performa. Setiap variasi skenario tersebut di*generate* 10 kali. Hasil skenario yang didapatkan, disimulasikan menggunakan *routing protocol* DSV *original* dan DSDV yang telah dimodifikasi. Setelah selesai melakukan simulasi, setiap skenario dievaluasi menggunakan metrik analisis. Selanjutnya dilakukan perhitungan rata-rata, sehingga menghasilkan nilai metrik analisis setiap variasi skenario. Nilai tersebut yang akan dibandingkan antara DSDV *original* dan DSDV yang telah dimodifikasi.

Hasil pengambilan data nilai *packet delivery ratio* (PDR) pada skenario *grid* 30 *node* dapat dilihat pada Gambar 5.1 dan Tabel 5.2. Pada kecepatan maksimal *node* 25m/s nilai PDR tidak mengalami perubahan. Hal ini terjadi karena batas kecepatan melakukan adaptif *update period* adalah lebih dari 25 m/s. Nilai PDR pada setiap kecepatan cenderung naik dan stabil. Hal ini dapat dilihat pada kecepatan maksimal 45 m/s, dimana nilai PDR pada

DSDV *original* mengalami penurunan, namun pada DSDV modifikasi stabil.

Pada kecepatan maksimal 30 m/s nilai PDR mengalami peningkatan sebesar 2,55%. Pada kecepatan maksimal 35 m/s nilai PDR mengalami peningkatan sebesar 2,46%. Pada kecepatan 40 m/s nilai PDR mengalami peningkatan sebesar 1,79%. Pada kecepatan 45 m/s nilai PDR mengalami peningkatan sebesar 5.95%. Pada kecepatan 50 m/s nilai PDR mengalami peningkatan sebesar 3,40%. Rata-rata peningkatan nilai PDR dalam 30 *node* adalah 3,23%.




**Gambar 5.1 Grafik PDR terhadap kecepatan maksimal *node* dalam skenario 30 *node***

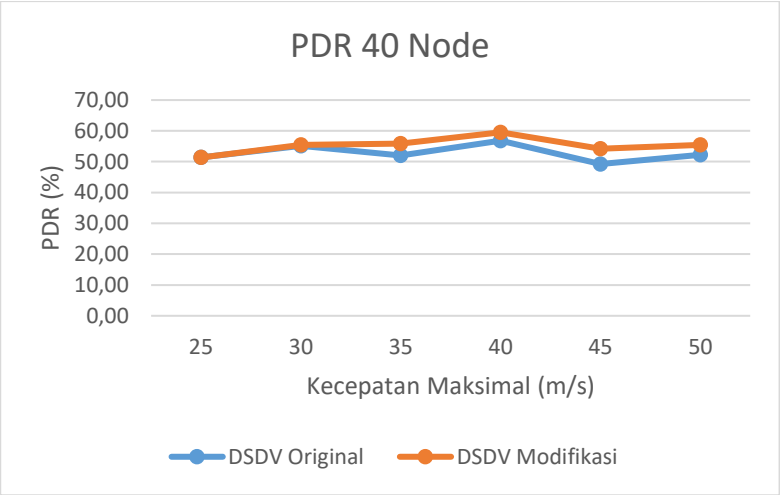
**Tabel 5.2 Data PDR terhadap kecepatan maksimal *node* dalam skenario 30 *node***

| Kecepatan (m/s) | DSDV *Original* | DSDV Modifikasi |
|---|---|---|
| 25 | 54,1 | 54,1 |
| 30 | 53,57 | 56,12 |

| Kecepatan (m/s) | DSDV *Original* | DSDV Modifikasi |
|---|---|---|
| 35 | 55,70 | 58,15 |
| 40 | 56,71 | 58,50 |
| 45 | 52,55 | 58,49 |
| 50 | 57,67 | 61,07 |

Hasil pengambilan data nilai *packet delivery ratio* (PDR) pada skenario *grid* 40 *node* dapat dilihat pada Gambar 5.2 dan Tabel 5.3. Pada kecepatan maksimal *node* 25 m/s nilai PDR tidak mengalami perubahan. Hal ini terjadi karena batas kecepatan melakukan adaptif *update period* adalah lebih dari 25 m/s. Nilai PDR pada setiap kecepatan mengalami peningkatan dan penurunan. Peningkatan dan penurunan ini terjadi karena nilai PDR dari grafik tersebut merupakan hasil rata-rata nilai PDR dari 10 skenario, dimana setiap skenario memiliki gerakan acak. Meskipun mengalami peningkatan dan penurunan, nilai PDR pada DSDV yang telah dimodifikasi masih lebih baik dibandingkan dengan DSDV *original*.

Pada kecepatan maksimal 30 m/s nilai PDR mengalami peningkatan sebesar 0,6%. Pada kecepatan maksimal 35 m/s nilai PDR mengalami peningkatan sebesar 2,4%. Pada kecepatan 40 m/s nilai PDR mengalami peningkatan sebesar 0,4%. Pada kecepatan 45 m/s nilai PDR mengalami peningkatan sebesar 4,9%. Pada kecepatan 50 m/s nilai PDR mengalami peningkatan sebesar 0,5%. Rata-rata peningkatan nilai PDR dalam 40 *node* adalah 3%.




**Gambar 5.2 Grafik PDR terhadap kecepatan maksimal *node* dalam skenario 40 *node***

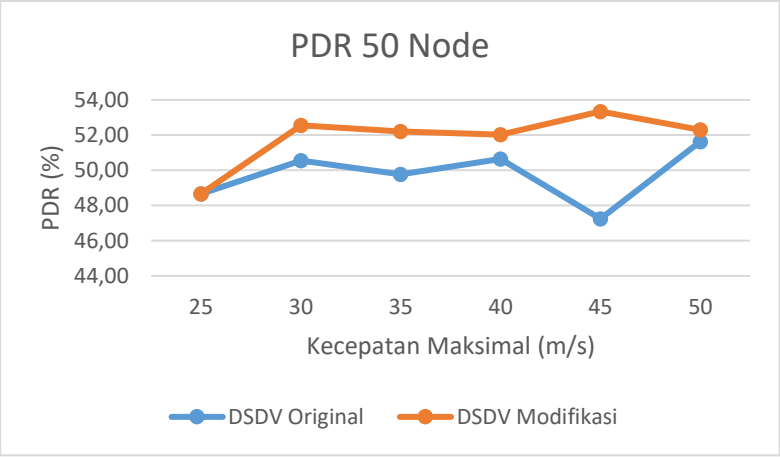
**Tabel 5.3 Data PDR terhadap kecepatan maksimal *node* dalam skenario 40 *node***

| Kecepatan (m/s) | DSDV *Original* | DSDV Modifikasi |
|---|---|---|
| 25 | 51,36 | 51,36 |
| 30 | 55,18 | 55,44 |
| 35 | 52,01 | 55,86 |
| 40 | 56,82 | 59,54 |
| 45 | 49,24 | 54,16 |
| 50 | 52,22 | 55,46 |

Hasil pengambilan data nilai *packet delivery ratio* (PDR) pada skenario *grid* 50 *node* dapat dilihat pada Gambar 5.3 dan Tabel 5.4. Pada kecepatan maksimal *node* 25m/s nilai PDR tidak mengalami perubahan. Hal ini terjadi karena batas kecepatan melakukan adaptif *update period* adalah lebih dari 25 m/s. Nilai

PDR pada setiap kecepatan mengalami peningkatan dan penurunan. Peningkatan dan penurunan ini terjadi karena nilai PDR dari grafik tersebut merupakan hasil rata-rata nilai PDR dari 10 skenario, dimana setiap skenario memiliki gerakan acak. Meskipun mengalami peningkatan dan penurunan, nilai PDR pada DSDV yang telah dimodifikasi masih lebih baik dibandingkan dengan DSDV *original*.

Pada kecepatan maksimal 30 m/s nilai PDR mengalami peningkatan sebesar 2%. Pada kecepatan maksimal 35 m/s nilai PDR mengalami peningkatan sebesar 2,44%. Pada kecepatan 40 m/s nilai PDR mengalami peningkatan sebesar 1,39%. Pada kecepatan 45 m/s nilai PDR mengalami peningkatan sebesar 6,10%. Pada kecepatan 50 m/s nilai PDR mengalami peningkatan sebesar 0,66 %. Rata-rata peningkatan nilai PDR dalam 50 *node* adalah 2,52 %.




**Gambar 5.3 Grafik PDR terhadap kecepatan maksimal *node* dalam skenario 50 *node***

**Tabel 5.4 Data PDR terhadap kecepatan maksimal *node* dalam skenario 50 *node***

| Kecepatan (m/s) | DSDV *Original* | DSDV Modifikasi |
|---|---|---|
| 25 | 48,67 | 48,67 |
| 30 | 50,54 | 52,55 |
| 35 | 49,77 | 52,20 |
| 40 | 50,65 | 52,03 |
| 45 | 47,25 | 53,34 |
| 50 | 51,63 | 52,29 |

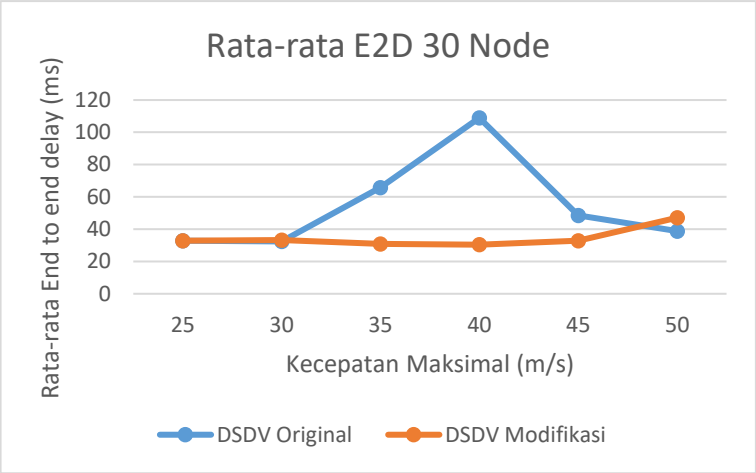
Berdasarkan Gambar 5.1, Gambar 5.2 dan Gambar 5.3 dapat dilihat bahwa *protocol* DSDV yang telah dimodifikasi mampu meningkatan nilai PDR. Meskipun dalam setiap kecepatan nilai PDR tidak selalu meningkat, namun nilai PDR masih lebih baik dibandingkan dengan DSDV *original*.

Hasil pengambilan data nilai rata-rata *end to end delay* (E2D) pada skenario *grid* 30 *node* dapat dilihat pada Gambar 5.4 dan

Tabel 5.5. Pada kecepatan maksimal *node* 25m/s nilai E2D tidak mengalami perubahan. Hal ini terjadi karena batas kecepatan melakukan adaptif *update period* adalah lebih dari 25 m/s. Nilai rata-rata E2D pada setiap kecepatan cenderung stabil, namun pada kecepatan 30 m/s dan 50 m/s nilai *end to end delay* pada DSDV yang telah dimodifikasi sedikit lebih besar dibandingkan pada DSDV *original*.

Pada kecepatan 25 m/s, nilai rata-rata E2D DSDV modifikasi sama dengan DSDV *original*. Pada kecepatan 30 m/s nilai rata-rata E2D DSDV modifikasi lebih besar 0,95 dibandingkan dengan DSDV *original*. Pada kecepatan maksimal 35 m/s DSDV modifikasi mampu memperkecil nilai rata-rata E2D sebesar 34,81 ms. Pada kecepatan maksimal 40 m/s DSDV modifikasi mampu memperkecil nilai rata-rata E2D sebesar 78,54

ms. Pada kecepatan maksimal 45 m/s DSDV modifikasi mampu memperkecil nilai rata-rata E2D sebesar 15,79 ms. Sedangkan pada kecepatan maksimal 50 m/s nilai E2D lebih besar 8,20 ms.



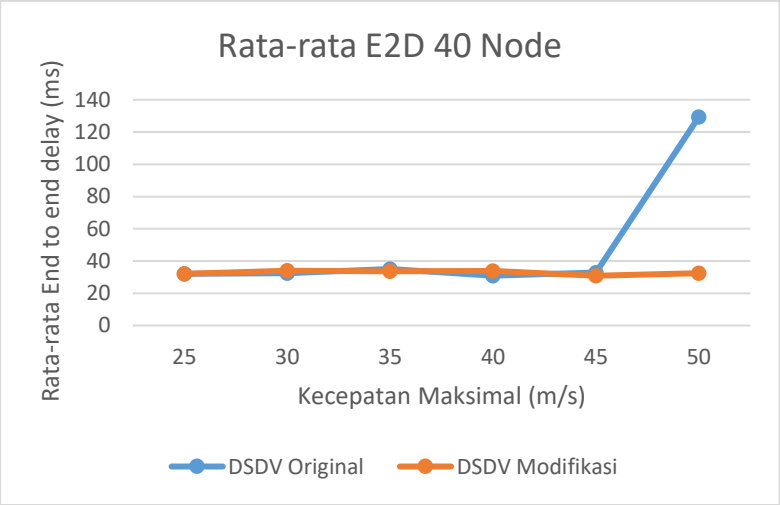

**Gambar 5.4 Grafik Rata-rata E2D terhadap kecepatan maksimal *node* dalam skenario 30 node**

**Tabel 5.5 Data rata-rata E2D terhadap kecepatan maksimal *node* dalam skenario 30 *node***

| Kecepatan (m/s) | DSDV *Original* | DSDV Modifikasi |
|---|---|---|
| 25 | 32,80 | 32,80 |
| 30 | 32,40 | 33,35 |
| 35 | 65,74 | 30,92 |
| 40 | 108,94 | 30,40 |
| 45 | 48,59 | 32,80 |
| 50 | 38,89 | 47,09 |

Hasil pengambilan data nilai rata-rata *end to end delay* (E2D) pada skenario *grid* 40 *node* dapat dilihat pada Gambar 5.5 dan Tabel 5.6. Pada kecepatan maksimal *node* 25m/s nilai E2D tidak mengalami perubahan. Hal ini terjadi karena batas kecepatan melakukan adaptif *update period* adalah lebih dari 25 m/s. Nilai rata-rata E2D DSDV modifikasi pada setiap kecepatan cenderung stabil.

Pada kecepatan maksimal 30 m/s nilai rata-rata E2D DSDV modifikasi lebih besar 1,58 ms dibandingkan dengan DSDV *original*. Pada kecepatan 35 m/s DSDV modifikasi mampu memperkecil nilai rata-rata E2D sebesar 1.40 ms. Pada kecepatan 40 m/s nilai rata-rata E2D pada DSDV modifikasi lebih besar 2,91 ms dibandingkan dengan DSDV *original*. Pada kecepatan 45 m/s DSDV modifikasi mampu memperkecil nilai rata-rata E2D sebesar 2,04. Pada kecepatan maksimal 50 m/s DSDV modifikasi mampu memperkecil nilai rata-rata E2D sebesar 96,88 ms.



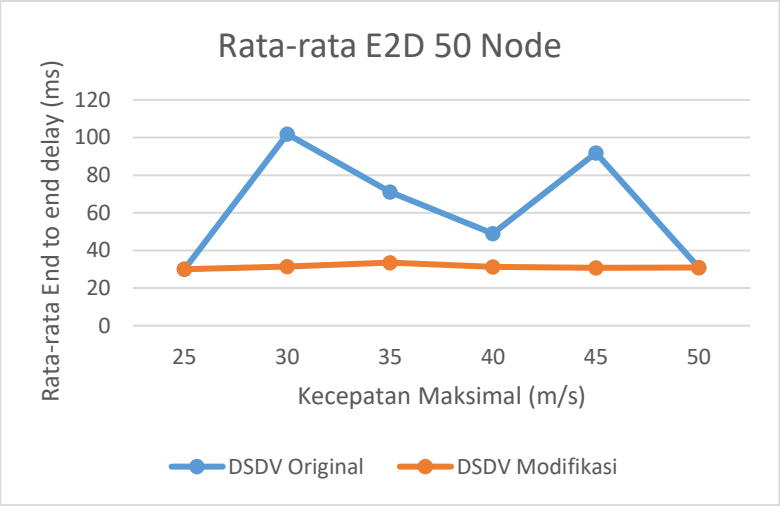

**Gambar 5.5 Grafik rata-rata E2D terhadap kecepatan maksimal *node* dalam skenario 40 node**

**Tabel 5.6 Data rata-rata E2D terhadap kecepatan maksimal *node* dalam skenario 40 *node***

| Kecepatan (m/s) | DSDV *Original* | DSDV Modifikasi |
|---|---|---|
| 25 | 32,12 | 32,12 |
| 30 | 32,39 | 33,98 |
| 35 | 35,00 | 33,60 |
| 40 | 30,93 | 33,84 |
| 45 | 32,92 | 30,88 |
| 50 | 129,31 | 32,43 |

Hasil pengambilan data nilai rata-rata *end to end delay* (E2D) pada skenario *grid* 50 *node* dapat dilihat pada Gambar 5.6 dan Tabel 5.7. Pada kecepatan maksimal *node* 25 m/s nilai E2D tidak mengalami perubahan. Hal ini terjadi karena batas kecepatan melakukan adaptif *update period* adalah lebih dari 25 m/s. Nilai rata-rata E2D pada setiap kecepatan cenderung stabil.

Pada kecepatan maksimal 30 m/s DSDV modifikasi mampu memperkecil nilai rata-rata E2D sebesar 70,46 ms. Pada kecepatan maksimal 35 m/s DSDV modifikasi mampu memperkecil nilai rata-rata E2D sebesar 37,54 ms. Pada kecepatan maksimal 40 m/s DSDV modifikasi mampu memperkecil nilai rata-rata E2D sebesar 17,59 ms. Pada kecepatan maksimal 45 m/s DSDV modifikasi mampu memperkecil nilai rata-rata E2D sebesar 60,93 ms. Sedangkan pada kecepatan maksimal 50 m/s nilai rata-rata E2D DSDV modifikasi lebih besar 0,04 ms dibandingkan dengan DSDV *original*.




**Gambar 5.6 Grafik rata-rata E2D terhadap kecepatan maksimal *node* dalam skenario 50 node**

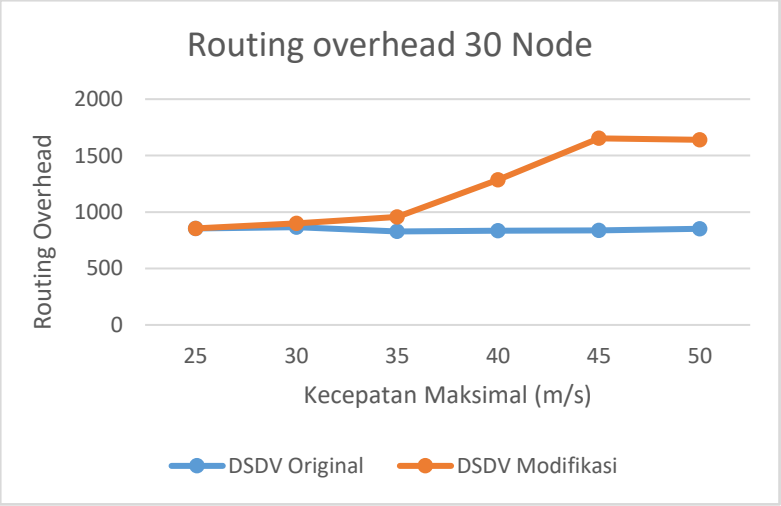
**Tabel 5.7 Data rata-rata E2D terhadap kecepatan maksimal *node* dalam skenario 50 *node***

| Kecepatan (m/s) | DSDV *Original* | DSDV Modifikasi |
|---|---|---|
| 25 | 30,01 | 30,01 |
| 30 | 101,83 | 31,37 |
| 35 | 71,07 | 33,53 |
| 40 | 48,93 | 31,33 |
| 45 | 91,76 | 30,83 |
| 50 | 30,97 | 31,01 |

Berdasarkan Gambar 5.4, Gambar 5.5 dan Gambar 5.6 dapat dilihat bahwa *protocol* DSDV yang telah dimodifikasi lebih baik dibandingkan *protocol* DSDV *original*. Hal ini karena semakin

rendah nilai *end to end delay*, maka semakin cepat paket data sampai pada tujuan. Dalam setiap variasi kecepatan, nilai *end to end delay* cenderung stabil.

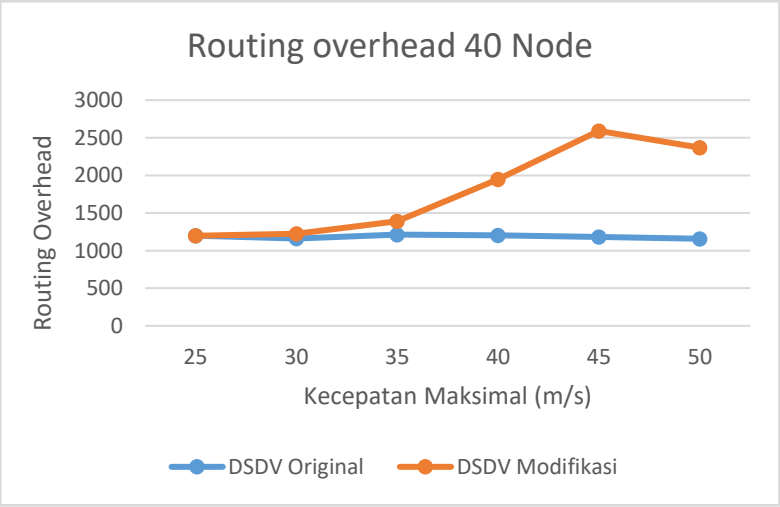
Hasil pengambilan data nilai *routing overhead* (RO) pada skenario *grid* 30 *node* dapat dilihat pada Gambar 5.7 dan Tabel 5.8, skenario *grid* 40 *node* dapat dilihat pada Gambar 5.8 dan Tabel 5.9 dan pada skenario *grid* 50 *node* dapat dilihat pada Gambar 5.9 dan Tabel 5.10. Pada kecepatan maksimal *node* 25m/s nilai RO tidak mengalami perubahan. Hal ini terjadi karena batas kecepatan melakukan adaptif *update period* adalah lebih dari 25 m/s. Nilai RO DSDV modifikasi pada setiap variasi kecepatan cenderung naik dibandingkan DSDV *original*. Hal ini terjadi karena semakin cepat kecepatan gerak *node*, maka semakin sering *node* tersebut melakukan *update routing table*.




**Gambar 5.7 Grafik *Routing overhead* terhadap kecepatan maksimal *node* dalam *skenario* 30 *node***

**Tabel 5.8 Data *Routing overhead* terhadap kecepatan maksimal *node* dalam *skenario* 30 *node***

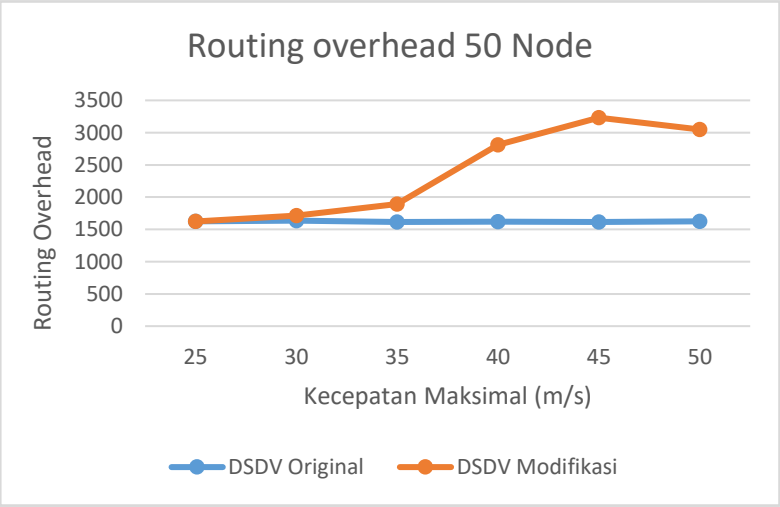
| Kecepatan (m/s) | DSDV *Original* | DSDV Modifikasi |
|---|---|---|
| 25 | 855 | 855 |
| 30 | 866 | 901 |
| 35 | 828 | 956 |
| 40 | 835 | 1285 |
| 45 | 839 | 1653 |
| 50 | 851 | 1639 |




**Gambar 5.8 Grafik *Routing overhead* terhadap kecepatan maksimal *node* dalam *skenario* 40 node**

**Tabel 5.9 Data *Routing overhead* terhadap kecepatan maksimal *node* dalam *skenario* 40 *node***

| Kecepatan (m/s) | DSDV *Original* | DSDV Modifikasi |
|---|---|---|
| 25 | 1197 | 1197 |
| 30 | 1161 | 1224 |
| 35 | 1213 | 1391 |
| 40 | 1201 | 1947 |
| 45 | 1184 | 2591 |
| 50 | 1158 | 2367 |




**Gambar 5.9 Grafik *Routing overhead* terhadap kecepatan maksimal *node* dalam skenario50 node**

**Tabel 5.10 Data *Routing overhead* terhadap kecepatan maksimal *node* dalam *skenario* 50 *node***

| Kecepatan (m/s) | DSDV *Original* | DSDV Modifikasi |
|---|---|---|
| 25 | 1623 | 1623 |
| 30 | 1635 | 1716 |
| 35 | 1615 | 1891 |
| 40 | 1620 | 2811 |
| 45 | 1617 | 3232 |
| 50 | 1625 | 3051 |

### 5.2.2 Hasil Uji Coba Skenario *Real*

Pengujian pada skenario *real* digunakan untuk melihat perbandingan *packet delivery ratio*, rata-rata *end to end delay*, dan *routing overhead* antara *routing protocol* DSDV *original* dan *routing protocol* DSDV yang telah dimodifikasi. Pengujian dilakukan pada peta sekitar Jalan Dr.Soetomo Surabaya yang telah dikonversi menjadi peta XML.

Pengambilan data uji PDR, rata-rata *end to end delay*, dan *routing overhead* dilakukan pada peta *real* dengan variasi jumlah *node* 30, 40 dan 50. Dalam hal ini, variasi jumlah *node* digunakan untuk melihat pengaruh jumlah *node* terhadap performa. Setiap variasi skenariotersebut di*generate* 10 kali. Hasil scenario yang didapatkan, disimulasikan menggunakan *routing protocol* DSDV *original* dan DSDV yang telah dimodifikasi. Setelah selesai melakukan simulasi, setiap skenario dievaluasi menggunakan metrik analisis. Selanjutnya dilakukan perhitungan rata-rata, sehingga menghasilkan nilai metrik analisis setiap variasi scenario. Nilai tersebut yang akan dibandingkan antara DSDV *original* dan DSDV yang telah dimodifikasi.

Hasil pengambilan data nilai *packet delivery ratio* (PDR) pada skenario *real* dapat dilihat pada Gambar 5.10 dan Tabel 5.11.

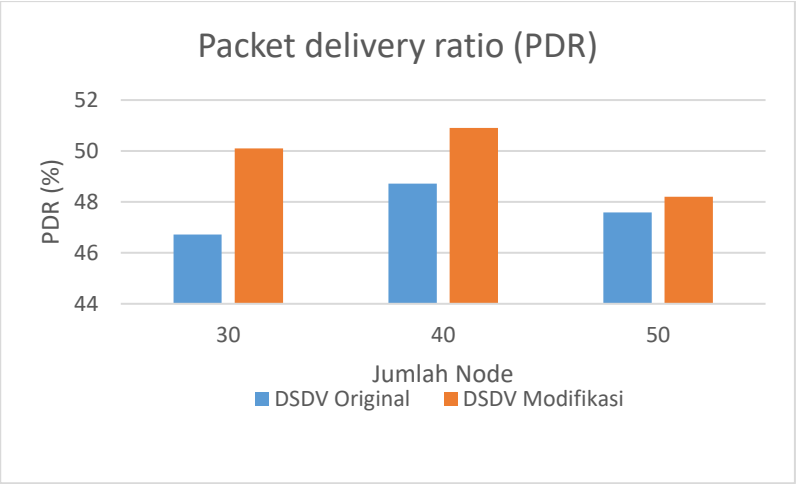
Nilai PDR pada DSDV yang telah dimodifikasi selalu lebih baik dibandingkan dengan DSDV *original*. Pada skenario dengan jumlah *node* 30, peningkatan nilai PDR sebesar 3,38%. Pada skenario dengan jumlah *node* 40, peningkatan nilai PDR sebesar 2,18%. Pada skenario dengan jumlah *node* 50, peningkatan nilai PDR sebesar 0,61%. Rata-rata peningkatan nilai PDR dengan jumlah *node* 30, 40,50 adalah 2.06%.

Hasil pengambilan data nilai rata-rata *end to end delay* (E2D) pada skenario *real* dapat dilihat pada Gambar 5.11 dan Tabel 5.12. DSDV yang telah dimodifikasi mampu menurunkan nilai rata-rata *end to end delay*. Namun nilai E2D pada DSDV yang telah dimodifikasi tidak selalu lebih baik dibandingkan dengan DSDV *original*. Hal ini dapat dilihat pada jumlah *node* 40 dan 50, rata-rata *end to end delay* pada DSDV modifikasi sedikit lebih besar dibandingkan dengan DSDV *original*. Peningkatan dan penurunan tersebut disebabkan karena nilai rata-rata E2D dari grafik tersebut merupakan hasil rata-rata nilai E2D dari 10 skenario, dimana setiap skenario memiliki gerakan acak.

Hasil pengambilan data nilai *routing overhead* (RO) pada skenario *real* dapat dilihat pada Gambar 5.12 dan Tabel 5.13. Pada DSDV modifikasi, nilai *routing overhead* selalu lebih tinggi dibandingkan dengan DSDV *original*. Semakin banyak jumlah *node*, maka nilai *routing overhead* juga semakin tinggi. Hal ini terjadi karena *routing overhead* merupakan akumulasi dari semua *node*.

**Tabel 5.11 Data PDR pada Skenario Real**

| Jumlah *node* | DSDV *Original* | DSDV Modifikasi |
|---|---|---|
| 30 | 46,72 | 50,10 |
| 40 | 48,72 | 50,90 |
| 50 | 47,59 | 48,20 |



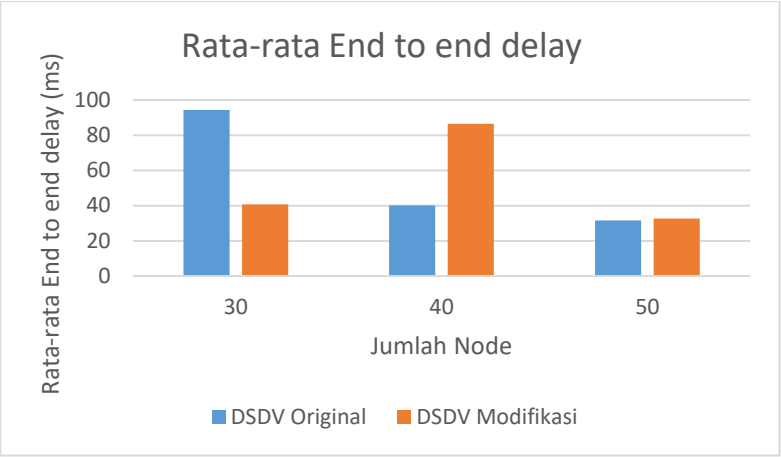

**Gambar 5.10 Grafik PDR pada Skenario Real**

**Tabel 5.12 Data rata-rata E2D pada Skenario Real**

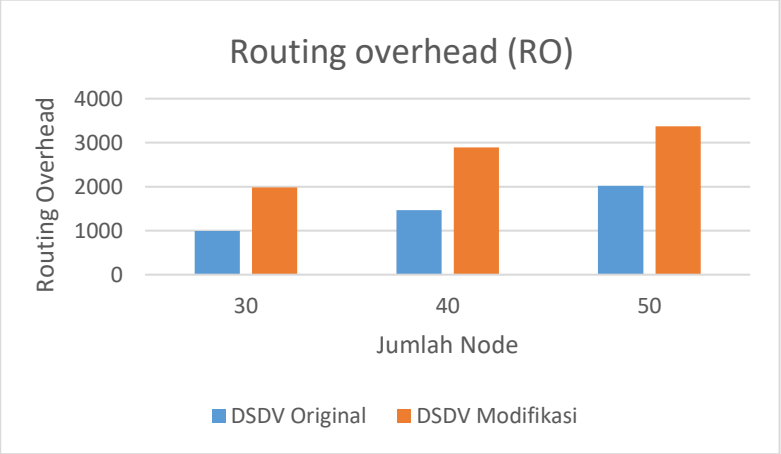
| Jumlah *node* | DSDV *Original* | DSDV Modifikasi |
|---|---|---|
| 30 | 94,36 | 40,72 |
| 40 | 40,15 | 86,56 |
| 50 | 31,65 | 32,65 |

**Tabel 5.13 Data Routing Overhead pada Skenario Real**

| Jumlah *node* | DSDV *Original* | DSDV Modifikasi |
|---|---|---|
| 30 | 996 | 1980 |
| 40 | 1469 | 2895 |
| 50 | 2021 | 3375 |




**Gambar 5.11 Grafik Rata-rata E2D pada Skenario Real**




**Gambar 5.12 Grafik *Routing overhead* pada Skenario Real**

# 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. Metode adaptif *update period routing protocol* DSDV pada VANETs dapat diterapkan berdasarkan kecepatan *node*. Jika kecepatan *node* melebihi *threshold*, maka dilakukan perubahan nilai *update* period.
2. Pada skenario *grid* dan real, secara umum setelah diterapkan metode adaptif *update period* terdapat peningkatan nilai PDR dan penurunan nilai rata-rata *end to end delay*. Pada skenario *grid*, peningkatan PDR terbesar adalah 6,10% dan pada skenario *real*, peningkatan PDR terbesar adalah 3,38%. Sedangkan untuk nilai rata-rata end to end delay, pada skenario grid penurunan terbesar adalah 96,88 ms dan pada skenario real penurunan terbesar adalah 53,64 ms. Namun dalam hal ini nilai *routing overhead* selalu lebih besar, karena semakin cepat kendaraan maka *routing* semakin sering.

## 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 *smooth*, dapat dilakukan uji coba lebih banyak, misal 20 kali untuk tiap skenario.

2. Melakukan penelitian yang berkaitan dengan metode perhitungan nilai *update* period.
3. Menerapkan metode adaptif *update period* pada lingkungan simulasi yang lain.

# DAFTAR PUSTAKA


[1] H. kaur, "*Routing* Protocols AODV and DSDV for MANET," *International Journal of Advanced Research in Computer Science,* vol. 8, pp. 630-635, 2017.

[2] X. Yang, "Performance Optimisation for DSDV in VANETs," in *UKSIM-AMSS International Conference on Modelling and Simulation,* 2015.

[3] S. u. Rehman*, "Vehicular Ad-Hoc Networks (VANETs) - An Overview and Challenges," *Journal of Wireless Networking and Communications,* vol. 3, pp. 29-38, 2013.

[4] R. H. Y.-S. C. A. I. a. A. H. S. Zeadally, "Vehicular *ad hoc* networks (VANETS): status, results, and challenges," *Spring Sci. Media LLC,* 2010.

[5] M. Watfa, "Advances in Vehicular Ad-Hoc Networks: Devel-opments and Challenges," *Intelligent Transport Systems. IGI Global,* 2010.

[6] Z. S. Houssaini, "Comparative Study of *Routing* Protocols Performance for Vehicular Ad-hoc Networks," *International Journal of Applied Engineering Research,* vol. 12, no. Research India Publications, pp. 3867-3878, 2017.

[7] T. Tamanna, "STUDY AND ANALYSIS OF DSDV AND OLSR," *International Journal of Computer Science and Mobile Computing,* vol. 5, p. 283 – 290, 2016.

[8] K. M. E. J. S. H. Afrah Daas, "Comparison between AODV and DSDV *Routing* protocols in Mobile Ad-hoc Network," *IEEE,* 2015.

[9] "Information Sciences Institute," USC University of Southern California , [Online]. Available:

https://www.isi.edu/nsnam/ns/. [Accessed 25 April 2018].

[10] J. Sasongko, "Network Simulator dan Network Animator menggunakan Cygnus Windows dalam Windows XP," *Jurnal Teknologi Informasi DINAMIK,* vol. XIV, pp. 60 - 69, 2009.

[11] "Simulation of Urban MObility," [Online]. Available: http://sumo.dlr.de/index.html. [Accessed 25 April 2018].

[12] "Institute of Transportation Systems," [Online]. Available: http://www.dlr.de/ts/en/desktop*default*.aspx/tabid-9883/16931_read-41000/. [Accessed 25 April 2018].

[13] "OpenStreetMap Indonesia," [Online]. Available: https://openstreetmap.id/about/tentang-openstreetmap/. [Accessed 25 April 2018].

[14] "JOSM," [Online]. Available: https://josm.openstreetmap.de/. [Accessed 10 5 2018].

[15] "GNU Operating System," [Online]. Available: https://www.gnu.org/software/gawk/manual/gawk.html. [Accessed 25 April 2018].

# LAMPIRAN

## A.1 Kode Fungsi DSDVTriggerHandler::handle

```
void
DSDVTriggerHandler::handle(Event *e)
 // send a triggered update (or a full update
if one's needed)
{
      //DEBUG
      //printf("(%d)-->triggered update with
e=%x\n", a->myaddr_,e);

  Scheduler & s = Scheduler::instance ();
  Time now = s.clock ();
  rtable_ent *prte;
  int update_type;       // we want periodic
(=1) or triggered (=0) update?
  Time next_possible = a->lasttup_ +
DSDV_MIN_TUP_PERIOD;

  update_period(a->myaddr_, a->node_->speed(),
a->perup_);

  for (a->table_->InitLoop(); (prte = a-
>table_->NextLoop());)
        if (prte->trigger_event == e) break;

  assert(prte && prte->trigger_event == e);

  if (now < next_possible)
    {
          //DEBUG
          //printf("(%d)..Re-scheduling
triggered update\n",a->myaddr_);
      s.schedule(a->trigger_handler, e,
next_possible - now);
      a->cancelTriggersBefore(next_possible);
      return;
```

```
    }

  update_type = 0;
  Packet * p = a->makeUpdate(/*in-
out*/update_type);

  if (p != NULL)
    {
      if (update_type == 1)
      { // we got a periodic update, though we
only asked for triggered
        // cancel and reschedule periodic
update
        s.cancel(a->periodic_callback_);
        //DEBUG
        //printf("we got a periodic update,
though asked for trigg\n");
        s.schedule (a->helper_, a-
>periodic_callback_,
                    a->perup_ * (0.75 + jitter
(0.25, a->be_random_)));
        if (a->verbose_) a->tracepkt (p, now,
a->myaddr_, "PU");
      }
      else
      {
        if (a->verbose_) a->tracepkt (p, now,
a->myaddr_, "TU");
      }
      assert (!HDR_CMN (p)->xmit_failure_);
      // DEBUG 0x2
      s.schedule (a->target_, p,
jitter(DSDV_BROADCAST_JITTER, a->be_random_));

      a->lasttup_ = now; // even if we got a
full update, it still counts
      // for our last triggered update time
    }
```

```
  // free this event
  for (a->table_->InitLoop (); (prte = a-
>table_->NextLoop ());)
    if (prte->trigger_event && prte-
>trigger_event == e)
      {
      prte->trigger_event = 0;
      delete e;
      }
}
```

## A.2 Kode Fungsi DSDV_Agent::cancelTriggersBefore

```
void
DSDV_Agent::cancelTriggersBefore(Time t)
  // Cancel any triggered events scheduled to
take place *before* time
  // t (exclusive)
{
  rtable_ent *prte;
  Scheduler & s = Scheduler::instance ();

  update_period(myaddr_, node_->speed(),
perup_);

  for (table_->InitLoop (); (prte = table_-
>NextLoop ());)
    if (prte->trigger_event && prte-
>trigger_event->time_ < t)
      {
            //DEBUG
            //printf("(%d) cancel event
%x\n",myaddr_,prte->trigger_event);
      s.cancel(prte->trigger_event);
      delete prte->trigger_event;
      prte->trigger_event = 0;
      }
}
```

### A.3 Kode Fungsi DSDV_Agent::needTriggeredUpdate

```
void
DSDV_Agent::needTriggeredUpdate(rtable_ent
*prte, Time t)
  // if no triggered update already pending,
make one so
{
  Scheduler & s = Scheduler::instance();
  Time now = Scheduler::instance().clock();
  update_period(myaddr_, node_->speed(),
perup_);

  assert(t >= now);

  if (prte->trigger_event)
    s.cancel(prte->trigger_event);
  else
    prte->trigger_event = new Event;
  //DEBUG
  //printf("(%d)..scheduling trigger-update
with event %x\n",myaddr_,prte->trigger_event);
  s.schedule(trigger_handler, prte-
>trigger_event, t - now);
}
```

### A.4 Kode Fungsi DSDV_Agent::helper_callback

```
void
DSDV_Agent::helper_callback (Event * e)
{
  Scheduler & s = Scheduler::instance ();
  double now = s.clock ();
  rtable_ent *prte;
  rtable_ent *pr2;
  int update_type;       // we want periodic
(=1) or triggered (=0) update?
  //DEBUG
```

```
  //printf("Triggered handler on 0x%08x\n",
e);
  update_period(myaddr_, node_->speed(),
perup_);

  // Check for periodic callback
  if (periodic_callback_ && e ==
periodic_callback_)
    {
      update_type = 1;
      Packet *p = makeUpdate(/*in-
out*/update_type);
      if (verbose_)
      {
        trace ("VPC %.5f _%d_", now, myaddr_);
        tracepkt (p, now, myaddr_, "PU");
      }


      if (p) {
            assert (!HDR_CMN (p)-
>xmit_failure_);  // DEBUG 0x2
            // send out update packet jitter
to avoid sync
            //DEBUG
            //printf("(%d)..sendout update pkt
(periodic=%d)\n",myaddr_,update_type);
            s.schedule (target_, p,
jitter(DSDV_BROADCAST_JITTER, be_random_));
      }

      // put the periodic update sending
callback back onto the
      // the scheduler queue for next time....
      s.schedule (helper_, periodic_callback_,
perup_node[myaddr_]  * (0.75 + jitter (0.25,
be_random_)));
```

```
      //s.schedule (helper_,
periodic_callback_, perup_node[myaddr_] +
(jitter (0.25, be_random_)));

      // this will take the place of any
planned triggered updates
      lasttup_ = now;
      return;
    }

  // Check for timeout
  // If it was a timeout, fix the routing
table.
  for (table_->InitLoop (); (prte = table_-
>NextLoop ());)
    if (prte->timeout_event && (prte-
>timeout_event == e))
      break;

  // If it was a timeout, prte will be non-
NULL
  // Note that in the if we don't touch the
changed_at time, so that when
  // wst is computed, it doesn't consider the
infinte metric the best
  // one at that sequence number.
  if (prte)
        {
      if (verbose_)
      {
        trace ("VTO %.5f _%d_ %d->%d", now,
myaddr_, myaddr_, prte->dst);
        /*    trace ("VTO %.5f _%d_ trg_sch %x
on sched %x time %f", now, myaddr_,
             trigupd_scheduled,
             trigupd_scheduled ?
s.lookup(trigupd_scheduled->uid_) : 0,
             trigupd_scheduled ?
trigupd_scheduled->time_ : 0); */
```

```
      }

      for (table_->InitLoop (); (pr2 = table_-
>NextLoop ()); )
      {
        if (pr2->hop == prte->dst && pr2-
>metric != BIG)
          {
            if (verbose_)
            trace ("VTO %.5f _%d_ marking %d",
now, myaddr_, pr2->dst);
            pr2->metric = BIG;
            pr2->advertise_ok_at = now;
            pr2->advert_metric = true;
            pr2->advert_seqnum = true;
            pr2->seqnum++;
            // And we have routing info to
propogate.
            //DEBUG
            //printf("(%d)..we have routing
info to propagate..trigger update for dst
%d\n",myaddr_,pr2->dst);
            needTriggeredUpdate(pr2, now);
          }
      }

      // OK the timeout expired, so we'll free
it. No dangling pointers.
      prte->timeout_event = 0;
    }
  else
    { // unknown event on  queue
      fprintf(stderr,"DFU: unknown queue
event\n");
     abort();
    }
  if (e)
    delete e;
}
```

## A.5 Kode Fungsi DSDV_Agent::updateRoute

```
void
DSDV_Agent::updateRoute(rtable_ent *old_rte,
rtable_ent *new_rte)
{
  int negvalue = -1;
  assert(new_rte);

  Time now = Scheduler::instance().clock();
  update_period(myaddr_, node_->speed(),
perup_);

  char buf[1024];
  //  snprintf (buf, 1024, "%c %.5f _%d_
(%d,%d->%d,%d->%d,%d->%d,%lf)",
  snprintf (buf, 1024, "%c %.5f _%d_ (%d,%d-
>%d,%d->%d,%d->%d,%f)",
           (new_rte->metric != BIG
            && (!old_rte || old_rte->metric !=
BIG)) ? 'D' : 'U',
           now, myaddr_, new_rte->dst,
           old_rte ? old_rte->metric :
negvalue, new_rte->metric,
           old_rte ? old_rte->seqnum :
negvalue,  new_rte->seqnum,
           old_rte ? old_rte->hop : -1,
new_rte->hop,
           new_rte->advertise_ok_at);

  table_->AddEntry (*new_rte);
  //printf("(%d),Route table
updated..\n",myaddr_);
  if (trace_wst_)
    trace ("VWST %.12lf frm %d to %d wst
%.12lf nxthp %d [of %d]",
         now, myaddr_, new_rte->dst, new_rte-
>wst, new_rte->hop,
```

```
          new_rte->metric);
  if (verbose_)
    trace ("VS%s", buf);
}
```

### A.6 Kode Fungsi DSDV_Agent::processUpdate

```
void
DSDV_Agent::processUpdate (Packet * p)
{
  hdr_ip *iph = HDR_IP(p);
  Scheduler & s = Scheduler::instance ();
  double now = s.clock ();

  // it's a dsdv packet
  int i;
  unsigned char *d = p->accessdata ();
  unsigned char *w = d + 1;
  rtable_ent rte;       // new rte learned
from update being processed
  rtable_ent *prte;           // ptr to entry
*in* routing tbl

  update_period(myaddr_, node_->speed(),
perup_);

  //DEBUG
  //int src, dst;
  //src =
Address::instance().get_nodeaddr(iph->src());
  //dst =
Address::instance().get_nodeaddr(iph->dst());
  //printf("Received DSDV packet from %d(%d)
to %d(%d) [%d)]\n", src, iph->sport(), dst,
iph->dport(), myaddr_);

  for (i = *d; i > 0; i--)
```

```
    {
      bool trigger_update = false;  // do we
need to do a triggered update?
      nsaddr_t dst;
      prte = NULL;

      dst = *(w++);
      dst = dst << 8 | *(w++);
      dst = dst << 8 | *(w++);
      dst = dst << 8 | *(w++);

      if ((prte = table_->GetEntry (dst)))
      {
        bcopy(prte, &rte, sizeof(rte));
      }
      else
      {
        bzero(&rte, sizeof(rte));
      }

      rte.dst = dst;
      //rte.hop = iph->src();
      rte.hop =
Address::instance().get_nodeaddr(iph-
>saddr());
      rte.metric = *(w++);
      rte.seqnum = *(w++);
      rte.seqnum = rte.seqnum << 8 | *(w++);
      rte.seqnum = rte.seqnum << 8 | *(w++);
      rte.seqnum = rte.seqnum << 8 | *(w++);
      rte.changed_at = now;
      if (rte.metric != BIG) rte.metric += 1;

      if (rte.dst == myaddr_)
      {
        if (rte.metric == BIG &&
periodic_callback_)
          {
```

```
            // You have the last word on
yourself...
            // Tell the world right now that
I'm still here....
            // This is a CMU Monarch
optimiziation to fix the
            // the problem of other nodes
telling you and your neighbors
            // that you don't exist described
in the paper.
            s.cancel (periodic_callback_);
            s.schedule (helper_,
periodic_callback_, 0);
          }
        continue;       // don't corrupt your
own routing table.

      }

      /**********  fill in meta data for new
route ***********/
      // If it's in the table, make it the
same timeout and queue.
      if (prte)
      { // we already have a route to this dst
        if (prte->seqnum == rte.seqnum)
          { // we've got an update with out a
new squenece number
            // this update must have come
along a different path
            // than the previous one, and is
just the kind of thing
            // the weighted settling time is
supposed to track.

            // this code is now a no-op left
here for clarity -dam XXX
            rte.wst = prte->wst;
```

```
            rte.new_seqnum_at = prte-
>new_seqnum_at;
          }
        else
          { // we've got a new seq number, end
the measurement period
            // for wst over the course of the
old sequence number
            // and update wst with the
difference between the last
            // time we changed the route
(which would be when the
            // best route metric arrives) and
the first time we heard
            // the sequence number that
started the measurement period

            // do we care if we've missed a
sequence number, such
            // that we have a wst measurement
period that streches over
            // more than a single sequence
number??? XXX -dam 4/20/98
            rte.wst = alpha_ * prte->wst +
            (1.0 - alpha_) * (prte->changed_at
- prte->new_seqnum_at);
            rte.new_seqnum_at = now;
          }
      }
      else
      { // inititallize the wst for the new
route
        rte.wst = wst0_;
        rte.new_seqnum_at = now;
      }

      // Now that we know the wst_, we know
when to update...
```

```
      if (rte.metric != BIG && (!prte || prte-
>metric != BIG))
      rte.advertise_ok_at = now + (rte.wst *
2);
      else
      rte.advertise_ok_at = now;

      /*********** decide whether to update
our routing table *********/
      if (!prte)
      { // we've heard from a brand new
destination
        if (rte.metric < BIG)
          {
            rte.advert_metric = true;
            trigger_update = true;
          }
        updateRoute(prte,&rte);
      }
      else if ( prte->seqnum == rte.seqnum )
      { // stnd dist vector case
        if (rte.metric < prte->metric)
          { // a shorter route!
            if (rte.metric == prte-
>last_advertised_metric)
            { // we've just gone back to a
metric we've already advertised
              rte.advert_metric = false;
              trigger_update = false;
            }
            else
            { // we're changing away from the
last metric we announced
              rte.advert_metric = true;
              trigger_update = true;
            }
            updateRoute(prte,&rte);
          }
        else
```

```
            { // ignore the longer route
            }
        }
        else if ( prte->seqnum < rte.seqnum )
        { // we've heard a fresher sequence
number
          // we *must* believe its rt metric
          rte.advert_seqnum = true;   // we've
got a new seqnum to advert
          if (rte.metric == prte-
>last_advertised_metric)
            { // we've just gone back to our old
metric
              rte.advert_metric = false;
            }
          else
            { // we're using a metric different
from our last advert
              rte.advert_metric = true;
            }

          updateRoute(prte,&rte);

#ifdef TRIGGER_UPDATE_ON_FRESH_SEQNUM
          trigger_update = true;
#else
          trigger_update = false;
#endif
        }
        else if ( prte->seqnum > rte.seqnum )
        { // our neighbor has older sequnum info
than we do
          if (rte.metric == BIG && prte->metric
!= BIG)
            { // we must go forth and educate
this ignorant fellow
              // about our more glorious and
happy metric
              prte->advertise_ok_at = now;
```

```
            prte->advert_metric = true;
            // directly schedule a triggered
update now for
            // prte, since the other logic
only works with rte.*
            needTriggeredUpdate(prte,now);
          }
        else
          { // we don't care about their stale
info
          }
      }
      else
      {
        fprintf(stderr,
              "%s DFU: unhandled adding a
route entry?\n", __FILE__);
        abort();
      }

      if (trigger_update)
      {
        prte = table_->GetEntry (rte.dst);
        assert(prte != NULL && prte-
>advertise_ok_at == rte.advertise_ok_at);
        needTriggeredUpdate(prte, prte-
>advertise_ok_at);
      }

      // see if we can send off any packets
we've got queued
      if (rte.q && rte.metric != BIG)
      {
        Packet *queued_p;
        while ((queued_p = rte.q->deque()))
        // XXX possible loop here
        // while ((queued_p = rte.q->deque()))
        // Only retry once to avoid looping
```

```
        // for (int jj = 0; jj < rte.q-
>length(); jj++){
        //  queued_p = rte.q->deque();
          recv(queued_p, 0); // give the
packets to ourselves to forward
        // }
        delete rte.q;
        rte.q = 0;
        table_->AddEntry(rte); // record the
now zero'd queue
      }
    } // end of all destination mentioned in
routing update packet

  // Reschedule the timeout for this neighbor
  prte = table_-
>GetEntry(Address::instance().get_nodeaddr(iph
->saddr()));
  if (prte)
    {
      if (prte->timeout_event)
      s.cancel (prte->timeout_event);
      else
      {
        prte->timeout_event = new Event ();
      }

      s.schedule (helper_, prte-
>timeout_event, min_update_periods_ * perup_);
    }
  else
    { // If the first thing we hear from a
node is a triggered update
      // that doesn't list the node sending
the update as the first
      // thing in the packet (which is
disrecommended by the paper)
      // we won't have a route to that node
already.  In order to timeout
```

```
      // the routes we just learned, we need a
harmless route to keep the
      // timeout metadata straight.

      // Hi there, nice to meet you. I'll make
a fake advertisement
      bzero(&rte, sizeof(rte));
      rte.dst =
Address::instance().get_nodeaddr(iph-
>saddr());
      rte.hop =
Address::instance().get_nodeaddr(iph-
>saddr());
      rte.metric = 1;
      rte.seqnum = 0;
      rte.advertise_ok_at = now + 604800; //
check back next week... :)

      rte.changed_at = now;
      rte.new_seqnum_at = now;
      rte.wst = wst0_;
      rte.timeout_event = new Event ();
      rte.q = 0;

      updateRoute(NULL, &rte);
      s.schedule(helper_, rte.timeout_event,
min_update_periods_ * perup_);
    }

  /*
   * Freeing a routing layer packet --> don't
need to
   * call drop here.
   */
  Packet::free (p);

}
```

## A.7 Kode Fungsi DSDV_Agent::forwardPacket

```
void
DSDV_Agent::forwardPacket (Packet * p)
{
  hdr_ip *iph = HDR_IP(p);
  Scheduler & s = Scheduler::instance ();
  double now = s.clock ();
  hdr_cmn *hdrc = HDR_CMN (p);
  int dst;
  rtable_ent *prte;

  update_period(myaddr_, node_->speed(),
perup_);

  // We should route it.
  //printf("(%d)-->forwardig pkt\n",myaddr_);
  // set direction of pkt to -1 , i.e downward
  hdrc->direction() = hdr_cmn::DOWN;

  // if the destination is outside
mobilenode's domain
  // forward it to base_stn node
  // Note: pkt is not buffered if route to
base_stn is unknown

  dst = Address::instance().get_nodeaddr(iph-
>daddr());
  if (diff_subnet(iph->daddr())) {
        prte = table_->GetEntry (dst);
        if (prte && prte->metric != BIG)
              goto send;

        //int dst = (node_->base_stn())-
>address();
        dst = node_->base_stn();
        prte = table_->GetEntry (dst);
        if (prte && prte->metric != BIG)
              goto send;
```

```
        else {
              //drop pkt with warning
              fprintf(stderr, "warning: Route
to base_stn not known: dropping pkt\n");
              Packet::free(p);
              return;
        }
  }

  prte = table_->GetEntry (dst);

  //  trace("VDEBUG-RX %d %d->%d %d %d 0x%08x
0x%08x %d %d",
  //  myaddr_, iph->src(), iph->dst(), hdrc-
>next_hop_, hdrc->addr_type_,
  //  hdrc->xmit_failure_, hdrc-
>xmit_failure_data_,
  //  hdrc->num_forwards_, hdrc-
>opt_num_forwards);

  if (prte && prte->metric != BIG)
    {
       //printf("(%d)-have route for
dst\n",myaddr_);
       goto send;
    }
  else if (prte)
    { /* must queue the packet */
         //printf("(%d)-no route, queue
pkt\n",myaddr_);
      if (!prte->q)
      {
        prte->q = new PacketQueue ();
      }

      prte->q->enque(p);

      if (verbose_)
```

```
      trace ("VBP %.5f _%d_ %d:%d -> %d:%d",
now, myaddr_, iph->saddr(),
             iph->sport(), iph->daddr(), iph-
>dport());

      while (prte->q->length () >
MAX_QUEUE_LENGTH)
            drop (prte->q->deque (),
DROP_RTR_QFULL);
      return;
    }
  else
    { // Brand new destination
      rtable_ent rte;
      double now = s.clock();

      bzero(&rte, sizeof(rte));
      rte.dst = dst;
      rte.hop = dst;
      rte.metric = BIG;
      rte.seqnum = 0;

      rte.advertise_ok_at = now + 604800; //
check back next week... :)
      rte.changed_at = now;
      rte.new_seqnum_at = now;      // was now
+ wst0_, why??? XXX -dam
      rte.wst = wst0_;
      rte.timeout_event = 0;

      rte.q = new PacketQueue();
      rte.q->enque(p);

      assert (rte.q->length() == 1 && 1 <=
MAX_QUEUE_LENGTH);
      table_->AddEntry(rte);

      if (verbose_)
```

```
            trace ("VBP %.5f _%d_ %d:%d ->
%d:%d", now, myaddr_,
                  iph->saddr(), iph->sport(),
iph->daddr(), iph->dport());
      return;
    }


 send:
  hdrc->addr_type_ = NS_AF_INET;
  hdrc->xmit_failure_ = mac_callback;
  hdrc->xmit_failure_data_ = this;
  if (prte->metric > 1)
        hdrc->next_hop_ = prte->hop;
  else
        hdrc->next_hop_ = dst;
  if (verbose_)
        trace ("Routing pkts outside domain: \
VFP %.5f _%d_ %d:%d -> %d:%d", now, myaddr_,
iph->saddr(),
             iph->sport(), iph->daddr(), iph-
>dport());

  assert (!HDR_CMN (p)->xmit_failure_ ||
        HDR_CMN (p)->xmit_failure_ ==
mac_callback);
  target_->recv(p, (Handler *)0);
  return;

}
```

### A.8 Kode Fungsi DSDV_Agent::recv

```
void
DSDV_Agent::forwardPacket (Packet * p)
{
  hdr_ip *iph = HDR_IP(p);
  Scheduler & s = Scheduler::instance ();
```

```
  double now = s.clock ();
  hdr_cmn *hdrc = HDR_CMN (p);
  int dst;
  rtable_ent *prte;

  update_period(myaddr_, node_->speed(),
perup_);

  // We should route it.
  //printf("(%d)-->forwardig pkt\n",myaddr_);
  // set direction of pkt to -1 , i.e downward
  hdrc->direction() = hdr_cmn::DOWN;

  // if the destination is outside
mobilenode's domain
  // forward it to base_stn node
  // Note: pkt is not buffered if route to
base_stn is unknown

  dst = Address::instance().get_nodeaddr(iph-
>daddr());
  if (diff_subnet(iph->daddr())) {
         prte = table_->GetEntry (dst);
        if (prte && prte->metric != BIG)
              goto send;

        //int dst = (node_->base_stn())-
>address();
        dst = node_->base_stn();
        prte = table_->GetEntry (dst);
        if (prte && prte->metric != BIG)
              goto send;

        else {
              //drop pkt with warning
              fprintf(stderr, "warning: Route
to base_stn not known: dropping pkt\n");
              Packet::free(p);
              return;
```

```
        }
  }

  prte = table_->GetEntry (dst);

  //  trace("VDEBUG-RX %d %d->%d %d %d 0x%08x
0x%08x %d %d",
  //  myaddr_, iph->src(), iph->dst(), hdrc-
>next_hop_, hdrc->addr_type_,
  //  hdrc->xmit_failure_, hdrc-
>xmit_failure_data_,
  //  hdrc->num_forwards_, hdrc-
>opt_num_forwards);

  if (prte && prte->metric != BIG)
    {
       //printf("(%d)-have route for
dst\n",myaddr_);
       goto send;
    }
  else if (prte)
    { /* must queue the packet */
         //printf("(%d)-no route, queue
pkt\n",myaddr_);
      if (!prte->q)
      {
        prte->q = new PacketQueue ();
      }

      prte->q->enque(p);

      if (verbose_)
      trace ("VBP %.5f _%d_ %d:%d -> %d:%d",
now, myaddr_, iph->saddr(),
             iph->sport(), iph->daddr(), iph-
>dport());

      while (prte->q->length () >
MAX_QUEUE_LENGTH)
```

```
            drop (prte->q->deque (),
DROP_RTR_QFULL);
      return;
    }
  else
    { // Brand new destination
      rtable_ent rte;
      double now = s.clock();

      bzero(&rte, sizeof(rte));
      rte.dst = dst;
      rte.hop = dst;
      rte.metric = BIG;
      rte.seqnum = 0;

      rte.advertise_ok_at = now + 604800; //
check back next week... :)
      rte.changed_at = now;
      rte.new_seqnum_at = now;      // was now
+ wst0_, why??? XXX -dam
      rte.wst = wst0_;
      rte.timeout_event = 0;

      rte.q = new PacketQueue();
      rte.q->enque(p);

      assert (rte.q->length() == 1 && 1 <=
MAX_QUEUE_LENGTH);
      table_->AddEntry(rte);

      if (verbose_)
            trace ("VBP %.5f _%d_ %d:%d ->
%d:%d", now, myaddr_,
                  iph->saddr(), iph->sport(),
iph->daddr(), iph->dport());
      return;
    }
```

```
 send:
  hdrc->addr_type_ = NS_AF_INET;
  hdrc->xmit_failure_ = mac_callback;
  hdrc->xmit_failure_data_ = this;
  if (prte->metric > 1)
        hdrc->next_hop_ = prte->hop;
  else
        hdrc->next_hop_ = dst;
  if (verbose_)
        trace ("Routing pkts outside domain: \
VFP %.5f _%d_ %d:%d -> %d:%d", now, myaddr_,
iph->saddr(),
             iph->sport(), iph->daddr(), iph-
>dport());

  assert (!HDR_CMN (p)->xmit_failure_ ||
        HDR_CMN (p)->xmit_failure_ ==
mac_callback);
  target_->recv(p, (Handler *)0);
  return;

}
```

### A.9 Kode Skenario NS-2

```
#
=============================================
=======================
# Define options
#
=============================================
=======================

set val(chan)
      Channel/WirelessChannel       ;# channel
type
```

```
set val(prop)
      Propagation/TwoRayGround      ;# radio-
propagation model
set val(netif)                  Phy/WirelessPhy
                  ;# network interface type
set val(mac)                    Mac/802_11
            ;# MAC type
set val(ifq)
      Queue/DropTail/PriQueue       ;#
interface queue type
set val(ll)                     LL
            ;# link layer type
set val(ant)
      Antenna/OmniAntenna           ;# antenna
model
set opt(x)                      1000
                  ;# X dimension of the
topography
set opt(y)                      1000
                  ;# Y dimension of the
topography
set val(ifqlen)                 1000
            ;# max packet in ifq
set val(nn)                     30
            ;# how many nodes are simulated
set val(seed)                           1.0
                  ;
set val(adhocRouting)           DSDV
            ;# routing protocol
set val(stop)                   200
            ;# simulation time
set val(cp)               "trafic"
      ;#<-- traffic file
set val(sc)               "dsdv-update.tcl"
            ;#<-- mobility file
```

```
#
# ======================================================
# Main Program
#
# ======================================================
# Initialize Global Variables
# create simulator instance

set ns_            [new Simulator]

# setup topography object

set topo    [new Topography]

#if { $val(adhocRouting) == "DSR" } {
#set val(ifq)            CMUPriQueue
#} else {
#set val(ifq)
Queue/DropTail/PriQueue
#}
# create trace object for ns and nam

set tracefd [open 30node.tr w]
set namtrace    [open 30node.nam w]

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

# set up topology object
set topo                        [new Topography]
$topo load_flatgrid $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 skenariofile..."
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) 100
}

# 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.10 Kode Skrip AWK *Packet delivery ratio*

```
# AWK Script for Packet Delivery Calculation
for OLD Trace Format

BEGIN {
sent=0;
received=0;
}

{

# count packet send
  if($1=="s" && $3=="_28_" && $4=="AGT" &&
$7=="cbr")
   {
    sent++;
   }

  else if($1=="r" && $3=="_29_" && $4=="AGT"
&& $7=="cbr")
   {
     received++;
   }

}
END {
printf "%.2f|",(received/sent)*100;

}
```

### A.11 Kode Skrip AWK *End to end delay*

```
BEGIN {
      seqno = -1;
      #droppedPackets = 0;
      #receivedPackets = 0;
      count = 0;
}
```

```
{
      if($4 == "AGT" && $1 == "s" && seqno <
$6) {
            seqno = $6;
      }
      #else if(($4 == "AGT") && ($1 == "r")) {
      #receivedPackets++;
      #} else if ($1 == "D" && $7 == "cbr" &&
$8 > 512){
      #droppedPackets++;
      #}
      #end-to-end delay

      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;
        #print "\n";
        #print "GeneratedPackets = " seqno+1;
        # print "ReceivedPackets = "
receivedPackets;
        #print "Packet delivery ratio = "
receivedPackets/(seqno+1)*100 "%";
        #print "Total Dropped Packets = "
droppedPackets;
        #print "Average End - to - End Delay = "
n_to_n_delay * 1000 " ms";

        printf "%.2f|", n_to_n_delay * 1000;
}
```

## A.12 Kode Skrip AWK *Routing* Overhead

```
BEGIN {
rt_pkts = 0;
}

{
if (($1 == "s" || $1 == "f") && ($4 ==
"RTR")&& ($7== "message") )
rt_pkts++;
}

END {
#printf ("Total number of routing
packets\t%d\n",rt_pkts);
printf ("%d|",rt_pkts);
}
```

## BIODATA PENULIS

Kharisma Monika Dian Pertiwi lahir di Kota Madiun pada 06 Agustus 1995. Penulis menempuh pendidikan formal dimulai dari TK Dharma Wanita (2001-2002), SDN 02 Manisrejo Kota Madiun (2002-2008), SMPN 4 Madiun (2008-2011), SMAN 2 Madiun (2011-2014) dan S1 Teknik Informatika ITS (2014-2018). Bidang studi yang diambil oleh penulis pada saat berkuliah di Teknik Informatika ITS adalah Arsitektur Jaringan dan Komputer (AJK). Penulis aktif dalam organisasi seperti Himpunan Mahasiswa Teknik Computer-Informatika (2015-2016) dan Keluarga Muslim Informatika (2015-2016). Penulis juga aktif dalam kegiatan kepanitiaan seperti SCHEMATICS 2015 dan SCHEMATICS 20165 divisi NLC. Penulis pernah kerja praktik di Trust Solution Surabaya periode Juli – Agustus 2017. Selama berkuliah, penulis juga menjadi administrator Laboratorium Komputasi Berbasis Jaringan. Penulis dapat dihubungi melalui email: kharismamonika@gmail.com.