TUGAS AKHIR - KI141502

IMPLEMENTASI PENAMBAHAN FAKTOR PERGERAKAN DALAM PROSES ROUTE DISCOVERY PADA AODV DI LINGKUNGAN VANET

M VIJAY FATHUR RAHMAN
NRP 5112100043

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

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

Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2017

TUGAS AKHIR – KI141502

IMPLEMENTASI PENAMBAHAN FAKTOR PERGERAKAN DALAM PROSES ROUTE DISCOVERY PADA AODV DI LINGKUNGAN VANET

M VIJAY FATHUR RAHMAN
NRP 5112100043

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

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

Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2017

(Halaman ini sengaja dikosongkan)

UNDERGRADUATE THESES – KI141502

IMPLEMENTATION OF ADDING MOVEMENT FACTORS ON AODV DISCOVERY ROUTE PROCESS IN VANET

M VIJAY FATHUR RAHMAN
NRP 5112100043

Advisor I
Dr. Radityo Anggoro, S.Kom., M.Sc.

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

Department of Informatics
Faculty of Information Technology
Sepuluh Nopember Institute of Technology
Surabaya 2017

(Halaman ini sengaja dikosongkan)

LEMBAR PENGESAHAN

IMPLEMENTASI PENAMBAHAN FAKTOR PERGERAKAN DALAM PROSES ROUTE DISCOVERY PADA AODV DI LINGKUNGAN VANET

TUGAS AKHIR

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

Oleh:
M VIJAY FATHUR RAHMAN
NRP: 5112100043

Disetujui oleh Pembimbing Tugas Akhir:

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

2. Prof. Ir. Supeno Djanali, M.Sc., Ph.D. .....
   (NIP. *194806191973011001*) (Pembimbing 2)

SURABAYA
JUNI, 2017

(Halaman ini sengaja dikosongkan)

IMPLEMENTASI PENAMBAHAN FAKTOR PERGERAKAN DALAM PROSES ROUTE DISCOVERY PADA AODV DI LINGKUNGAN VANET

Nama Mahasiswa	**: M VIJAY FATHUR RAHMAN**
NRP	**: 5112100043**
Jurusan	**: Teknik Informatika FTIF-ITS**
Dosen Pembimbing 1	**: Dr. Radityo Anggoro, S.Kom., M.Sc.**
Dosen Pembimbing 2	**: Prof. Ir. Supeno Djanali, M.Sc., Ph.D**

Abstrak

Terdapat banyak sekali routing protocol pada lingkungan VANET. AODV merupakan salah satu routing protocol yang populer dan terdapat berbagai macam variasi. Namun banyaknya variasi AODV belum tentu menjamin akan menghasilkan kestabilan yang tinggi dalam komunikasi antar kendaraan.

Oleh karena itu dilakukan penelitian mengenai faktor-faktor yang mempengaruhi kinerja AODV. Pada tugas akhir ini, penulis mencoba meneliti pengaruh faktor pergerakan kendaraan, seperti kecepatan dan percepatan pada proses route discovery AODV. Setiap node akan memiliki bobot TWR (Total Weight of Route) agar dipilih sebagai relay node. Selain itu juga memperhatikan predisksi faktor kecepatan pada beberapa detik yang akan datang dan memperhitungkan faktor kualitas komunikasi antar kendaraan. Dari hasil perhitungan prediksi tersebut juga akan menjadi pertimbangan dalam memilih relay node yang akan melanjutkan paket route request dan disimpan sebagai bobot Future-TWR.

Dari uji coba yang telah dilakukan, penambahan faktor kecepatan tidak memberikan dampak yang signifikan pada efektifitas rasio pesan tersampaikan. Meskipun pada beberapa skenario rasio pesan diterima lebih rendah dibanding AODV biasa.

Kata kunci: VANET, AODV, NS-2, TWR, Future-TWR

(Halaman ini sengaja dikosongkan)

IMPLEMENTATION OF ADDING MOVEMENT FACTORS ON AODV DISCOVERY ROUTE PROCESS IN VANET

Student's Name	**: M VIJAY FATHUR RAHMAN**
Student's ID	**: 5112100043**
Department	**: Teknik Informatika FTIF-ITS**
First Advisor	**: Dr. Radityo Anggoro, S.Kom., M.Sc.**
Second Advisor	**: Prof. Ir. Supeno Djanali, M.Sc., Ph.D**

Abstract

There are so many routing protocol in the VANET environment. And AODV is one of the popular routing protocol and has so many variations. However, the many variations of AODV may not guarantee a high stability in vehicle-to-vehicle communication.

Therefore, factors that affect the performance of AODV are being researched.In this thesis, the author tries to examine the effect of vehicle movement factors, such as speed and acceleration in AODV discovery route process. Each node will have TWR(Total Weight of Route) to be selected as a candidate of the relay node. It also takes the prediction of movement factors in the next few second into account. From the calculation of the prediction, will also be a consideration in choosing a relay node that will relay the route request packet and stored as a Future-TWR

The simulation results show that, adding movement factors doesn't make a big difference to affect the packet delivery ratio results, altough in some case(s) it's lower than using normal AODV.

Keywords: VANET, AODV, NS-2, TWR, Future-TWR

(Halaman ini sengaja dikosongkan)

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 Penambahan Faktor Pergerakan dalam Proses Route Discovery pada AODV di Lingkungan Vanet**"

yang merupakan salah satu syarat dalam menempuh ujian sidang guna memperoleh gelar Sarjana Komputer. Selesainya Tugas Akhir ini tidak terlepas dari bantuan dan dukungan beberapa pihak, sehingga pada kesempatan ini penulis mengucapkan terima kasih kepada :

1. Allah Subhanahu wa ta'ala atas semua berkah, ilmu serta petunjuk yang diberikan sehingga penulis dapat menyelesaikan Tugas Akhir ini.
2. Bapak Jumain dan Ibu Tasmiatun selaku orang tua penulis yang selalu memberikan dukungan motivasi, do'a dan kasih sayang yang tak terhingga kepada penulis sehingga penulis dapat menyelesaikan Tugas Akhir ini.
3. BapakDr. Radityo Anggoro, S.Kom., M.Sc., sertaBapak Prof. Ir. Supeno Djanali, M.Sc., Ph.D., selaku dosen pembimbing Tugas Akhir yang telah membimbing dan memberikan motivasi, nasehat dan bimbingan dalam menyelesaikan Tugas Akhir ini.
4. Ibu Bilqis Amaliah S.Kom, M.Kom., selaku dosen wali penulis yang telah memberikan arahan, masukan dan motivasi kepada penulis selama menjalani masa studi di ITS.
5. Bapak Darlis Herumurti, S.Kom, M.Kom., selaku kepala jurusan Teknik Informatika ITS dan segenap dosen dan karyawan Teknik Informatika ITS yang telah memberikan ilmu dan pengalaman kepada penulis selama menjalani masa studi di ITS.

6. Teman seperjuangan dalam topik tugas akhir (Reyhan Arief, Putu Pradnyana, Nabila Tsurayya) yang selalu memberikan masukan dalam menyelesaikan Tugas Akhir, teman seperjuangan semasa kuliah penulis (Iskandar Zulkarnain, Ardhinata Juari, Faishal Azka) yang selalu memberikan dukungan moril, serta teman-teman satu angkatan yang sudah memberi dukungan, informasi dan hal lainnya yang membantu penulis menyelesaikan tugas akhir ini.
7. Fatimatus Zuhro serta sahabat penulis lainnya yang tidak bisa disebutkan satu-persatu, yang selalu membantu, menghibur, menghadapi persoalan sulit bersama, menjadi tempat bertukar ilmu dan berjuang bersama-sama penulis.

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

DAFTAR ISI

(Halaman sengaja dikosongkan)

DAFTAR GAMBAR

(Halaman sengaja dikosongkan)

DAFTAR TABEL

(Halaman ini sengaja dikosongkan)

BAB I
PENDAHULUAN

Bab ini membahas garis besar penyusunan Tugas Akhir yangmeliputi latar belakang, tujuan pembuatan, rumusan dan batasan permasalahan, metodologi penyusunan Tugas Akhir dan sistematika penulisan.

1.1 Latar Belakang

Seiring dengan berkembangnya teknologi komunikasi dengan memanfaatkan teknologi nirkabel secara ad-hoc, telah muncul teknologi MANET (*Mobile Ad-hoc Network*) yang memudahkan komunikasi antar perangkat tanpa adanya jaringan infrastruktur yang tetap. Sedangkan jaringan VANET (*Vehicular Ad-hoc Network*) merupakan pengembangan dari MANET yang diterapkan pada pergerakan kendaraan. Pada VANET terdapat banyak sekali algortima *routing protocol*, namun performanya tidak stabil dikarenakan topologi VANET yang dinamis. Sehingga modifikasi *routing protocol* dapat diharapkan meningkatkan kinerja *routing protocol* pada VANET.

Ada beberapa jenis *routing protocol* yang telah tersedia pada NS-2. Salah satunya adalah AODV (*Ad-hoc On-demand Distance Vector*). Pada dasarnya, AODV dirancang untuk penggunaan dalam lingkungan MANET, sehingga pada lingkungan VANET perlu dilakukan modifikasi agar dapat digunakan dengan baik. Selain itu, pergerakan kendaraan dan topologi yang terus berubah membuat komunikasi data dalam VANET sering terjadi kegagalan. Salah satu proses yang dapat dimodifikasi adalah proses pengiriman RREP (*Route-Reply*). RREP merupakan pesan balasan yang dikirimkan oleh node tujuan ketika telah berhasil menerima RREQ (Route-Request) untuk pertama kalinya, dan menghiraukan RREQ yang datang setelahnya.

Pada tugas akhir ini akan dilakukan penelitian terhadap studi kinerja dari RREP dengan memodifikasi cara kerjanya. Perbedaan antara penelitian yang akan dilakukan pada Tugas Akhir ini dengan algoritma AODV yang sudah ada adalah, penambahan faktor pergerakan pada proses *route request* dalam menentukan node mana saja yang akan meneruskan paket ke node tujuan. Selain itu juga diperhitungkan faktor prediksi tren node sebelum atau setelah mengirim paket, apakah layak dipertahankan atau tidak.

Dalam Tugas Akhir ini, penulis menggunakan Network-Simulator-2 (NS2) dalam mengimplementasikan *routing protocol* AODV yang baru. Penulis akan melakukan studi kinerja dengan simulasi pada berbagai macam skenario yang memiliki jumlah dan kecepatan node yang bervariasi. Kemudian hasilnya akan dibandingkan dengan kinerja AODV murni.

Parameter yang akan dibandingkan dalam penelitian ini adalah perubahan faktor pergerakan yang mempengaruhi *delay*, jumlah, dan perserntase paket yang akan diterima.

1.2 Rumusan Masalah

Tugas Akhir ini mengangkat beberapa rumusan masalah sebagai berikut:

1. Bagaimana perbedaan kinerja antara penambahan faktor pergerakan pada protokol AODV dibandingkan dengan AODV biasa?
2. Bagaimana cara mengimplementasikan protokol AODV dengan memperhatikan faktor pergerakan?

1.3 Batasan Permasalahan

Permasalahan yang dibahas pada Tugas Akhir ini memiliki batasan sebagai berikut:

1. Proses pengujian menggunakan NS-2.
2. Modifikasi *routing protocol* berdasarkan protokol AODV.
3. Menggunakan SUMO sebagai *mobility generator*.

1.4 Tujuan

Tujuan dari Tugas Akhir ini adalah sebagai berikut:

1. Dapat mengetahui pengaruh faktor pergerakan terhadap kinerja protokol AODV pada lingkungan VANET.
2. Mengimplementasikan protokol AODV dengan modifikasi penambahan faktor pergerakan.

1.5 Manfaat

Manfaat dari Tugas Akhir ini adalah:

1. Dengan dibuatnya tugas akhir ini diharapkan dapat memberikan manfaat pada penelitian AODV selanjutnya, terutama pada penelitian pengaruh faktor pergerakan kendaraan.
2. Mengetahui perbandingan performa AODV biasa dengan AODV yang mempertimbangkan faktor pergerakan.

1.6 Metodologi

Pembuatan Tugas Akhir ini dilakukan dengan menggunakan metodologi sebagai berikut:

1.6.1 Penyusunan Proposal

Tahap awal Tugas Akhir ini adalah menyusun proposal Tugas Akhir. Pada proposal, dijelaskan mengenai garis besar alur modifikasi *routing protocol.*

1.6.2 Studi Literatur

Pada tahap ini dilakukan untuk mencari informasi dan studi literatur apa saja yang dapat dijadikan referensi untuk membantu pengerjaan Tugas Akhir ini.Informasi yang digunakan berupa dokumentasi *tools, protocol* dan penelitian sebelumnya terkait Tugas Akhir ini.

1.6.3 Implementasi Protokol

Pada tahap implementasi *routing protocol*, akan dilakukan modifikasi pada *routing protocol* AODV yang sudah ada pada NS-2 menjadi AODV yang memperhitungkan faktor pergerakan. Implementasi *routing protocol* akan dilakukan pada platform Linux.

1.6.4 Pengujian dan Evaluasi

Pengujian dilakukan dengan cara menjalankan simulasi dengan parameter tertentu pada NS-2 dengan *routing protocol* AODV dan AODV yang dimodifikasi. Evaluasi dilakukan dengan membandingkan hasil simulasi AODV dan AODV yang memperhitungkan faktor pergerakan. Berikut adalah parameter perbandingan yang digunakan:

1. *Routing overhead*
2. *Average delay*
3. PDR (*Packet Delivery Ratio*)
4. Jumlah RREQ

1.6.5 Penyusunan Buku

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

1.7 Sistematika Penulisan Laporan

Sistematika penulisan laporan Tugas Akhiradalah 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 penjelasan secara detail mengenai dasar-dasar penunjang untuk mendukung pembuatan Tugas Akhir ini.

3. Bab III. Perancangan

   Bab ini berisi perancangan metode yang nantinya akan diimplementasikan dan dilakukan uji coba.

4. Bab IV. Implementasi

   Bab ini menjelaskan implementasi dari implementasi yang telah dibuat padabab sebelumnya. Penjelasan berupa implemetasi mobilitas *vehicular*, konfigurasi sistem serta *script* analisis yang digunakan untuk menguji performa *routing protocol.*

5. Bab V. Hasil Uji Coba dan Evaluasi

   Bab ini menjelaskan tahap hasil pengujian skenario yang telah dibuat yang dijalankan dengan *routing protocol*. Menampilkan perbandingan setiap skenario dengan *routing protocol* yang lama dengan yang telah dimodifikasi.

6. Bab VI. Penutup

   Bab ini merupakan bab terakhir yang menyampaikan kesimpulan dari hasil uji coba yang dilakukan terhadap

rumusan masalah yang ada dan saran untuk pengembangan lebih lanjut.

7. Daftar Pustaka

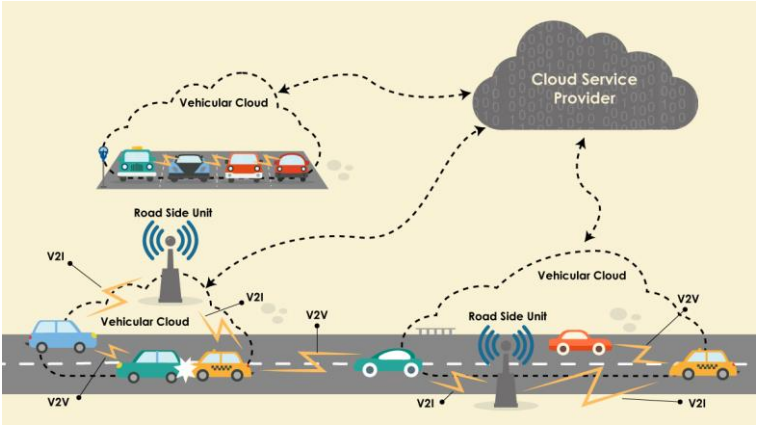
   Bab ini berisi daftar pustaka yang dijadikan literatur dalam TugasAkhir.

BAB II
TINJAUAN PUSTAKA

Bab ini berisi penjelasan teori yang berkaitan dengan implementasi perangkat lunak. Penjelasan tersebut bertujuan untuk memberikan gambaran sistem yang akan dibangun dan sebagai penunjang dalam pengembangan perangkat lunak.

2.1 Vehicular Ad-Hoc Network (VANET)

VANET diciptakan berdasarkan prinsip-prinsip *Mobile Ad-hoc Network* atau MANET, namun untuk implementasinya dilakukan pada jaringan komunikasi antar kendaraan. Jaringan VANET dibentuk dengan menghubungkan kendaraan dengan RSU (*Roadside Unit*) atau dengan kendaraan yang lain. Bentuk komunikasi antar kendaraan disebut sebagai komunikasi V2V (*vehicle to vehicle*) sedangkan komunikasi antara kendaraan dengan RSU disebut dengan V2I (*vehicle to infrastructure*). VANET bertujuan untuk meningkatkan keselamatan dalam berkendara dan mempermudah manajemen lalu lintas, serta memberikan akses internet kepada pengendara ataupun penumpang kendaraan. RSU dapat berkontribusi sebagai penyedia bantuan navigasi dan beberapa informasi yang bermanfaat lainnya seperti informasi lokasi tempat makan, stasiun pengisian bahan bakar, dan lain lain. Pada komunikasi V2V, pengendara dapat mendapatkan informasi yang lebih baik dalam mengambil tindakan awal ketika menghadapi situasi yang abnormal di jalan raya. Untuk dapat melakukan hal tersebut, OBU (*On-Board Unit*) atau alat yang terpasang pada setiap kendaraan untuk melakukan komunikasi *wireless*, secara teratur mengirimkan broadcast informasi posisi kendaraan, arah laju kendaraan, kecepatan, percepatan, waktu saat ini, kondisi lalu lintas, dan variabel lain yang dapat membantu agar informasi kendaraan dapat diketahui sepenuhnya oleh kendaraan lain [1].



Gambar 2.1: Ilustrasi VANET

2.2 Ad-hoc on Demand Distance Vector (AODV)

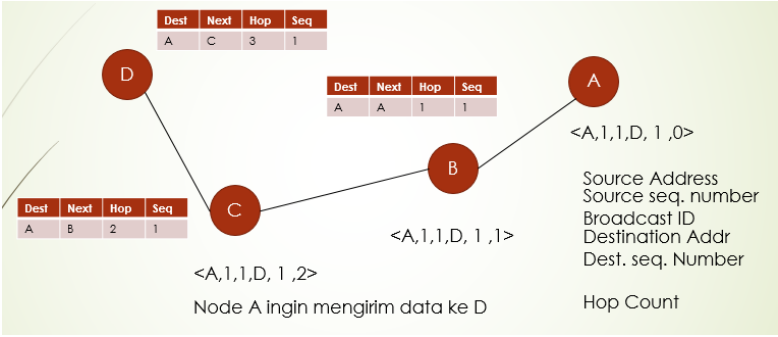
Protokol AODV dimaksudkan untuk digunakan oleh *mobilenode* pada suatu jaringan ad-hoc. protokol ini menawarkan adaptasi yang cepat pada kondisi topologi jaringan yang dinamis.AODV adalah *distance vector routing protocol* yang termasuk dalam klasifikasi *reactive routing protocol*, sehingga hanya akan melakukan request sebuah *route* saat dibutuhkan. AODV memiliki ciri utama dengan menjaga *lifetime* pada setiap *node* sesuai dengan penggunaan tabel *routing*, dan tabel *routing* akan kadaluarsa ketika jarang digunakan. Informasi tabel *route* harus selalu disimpan, meskipun untuk sebuah rute dengan masa hidup yang sebentar, seperti rute sementara yang digunakan untuk pengiriman RREP kembali ke *node* asal, adapun *field*yang disimpan pada setiap entri tabel *routing*:

- *Destination IP Address* : alamat IP node tujuan
- *Destination Sequence Number*: untuk memastikan tidak terjadi *loop*

- *Next Hop*: ’Loncatan’ (*hop*) berikutnya, bisa berupa tujuan atau *node* lainnya, field ini dirancang untuk meneruskan paket ke *node* tujuan.
- *Hop Count*: Jumlah *hop* dari alamat IP sumber sampai ke alamat IP tujuan.
- *Lifetime*: Waktu dalam milidetik yang digunakan untuk node menerima RREP.
- *Routing Flags*: Status sebuah rute, *up*(valid), *down*(tidak valid) atau sedang diperbaiki

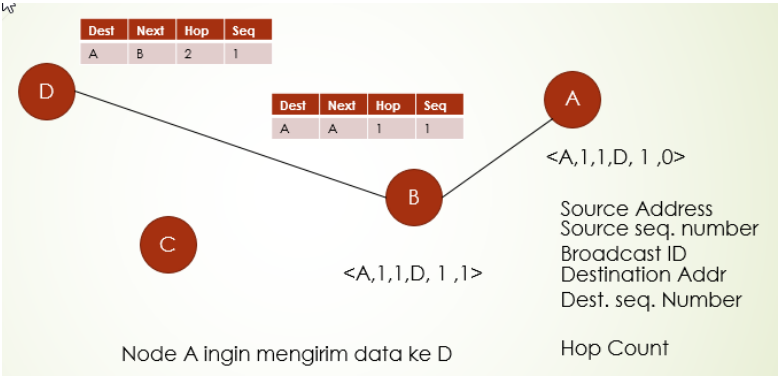
AODV memiliki beberapa proses, yaitu *route discovery* dan *route maintenance*. Pada *route discovery node* asal mengirimkan *Route Request* (RREQ) hingga menerima *Route Reply* (RREP) dari *node* tujuan. Sedangkan *route maintenance* mencakup data, *routeupdate*, dan *Route Error* (RERR).

Proses *route discovery* diinisialisasi oleh pengecekan area disekitar *node* asal, jika di dalam area tersebut telah ditemukan *node* tujuan, maka pesan RREQ akan langsung dikirimkan ke *node* tujuan. Jika tidak ditemukan, maka *node* asal akan melakukan *broadcast* pesan RREQ ke seluruh *node* disekitarnya, dan diteruskan oleh *node* lain hingga menemukan *node* tujuan. Ketika telah sampai di *node* tujuan, pesan RREP akan dikirimkan ke *node* asal yang bersangkutan.



Gambar 2.2 : Ilustrasi AODV

Pada gambar 2.2, diperlihatkann ilustrasi AODV yang terdapat pada ns2. Sedangkan pada gambar 2.3 merupakan ilustrasi AODV yang telah dimodifikasi, yaitu AODV yang memperhitungkan faktor pergerakan. **Harapannya** bentuk komunikasi akan lebih singkat dan efektif.



Gambar 2.3 : Ilustrasi AODV yang Telah Dimodifikasi

2.3 Ad-Hoc On-Demand Distance Vector Predicting Node Trend (AODV-PNT)

AODV-PNT [2] merupkan *routing protocol* AODV yang telah dilakukan modifikasi pada beberapa parameter *node*. Modifikasi yang dilakukan adalah penambahan mekanisme TWR atau *Total Weight Route*dan mekanisme prediksi *future*-TWR. Dengan demikian pemilihan *relay node* akan ditentukan oleh kedua parameter tersebut. TWR dihitung sebelum *node* pengirim/sumber mengirim paket *request*/RREQ dan sebelum *relay node* meneruskan paket RREQ.

Pada Tugas Akhir ini, perhitungan TWR dipertimbangkan beberapa faktor, yakni :

- Kecepatan dan Akselerasi
  Dengan perbedaan kecepatan yang besar antara *node* satu dengan *node* lainnya, akan membuat *node-node* tersebut

saling tersebar berjauhan. Akibatnya membuat pelebaran jarak yang berdampak pada kecilnya kemungkinan paket tersampaikan. Oleh karena itu, jarak dapat mempengaruhi bobot TWR, karena ketika kecepatan antar *node* sangat stabil dan tidak saling mendahului maka probabilitas diterimanya paket akan tinggi.

- Kualitas hubungan antar *node*
  Node yang akan mengirim paket dan *node* yang akan menerima paket idealnya akan berada pada dalam sebuah radius komunikasi dan dapat berkomunikasi antara pengirim dan penerima. Namun kenyataannya, dalam lingkungan VANET banyak sekali *neighbor node*, bangunan yang ada di sekitar jalan, serta halangan lain yang dapat mempengaruhi kualitas hubungan antara *node* pengirim dengan *node* tujuan. Maka diperlukan sebuah indeks untuk menghitung indeks stabilitas antar *node*[2]. Formula indeks stabilitas dapat dilihat pada persamaan berikut[2.1].

$$S_{ij} = 1 - \frac{\min\left(\sqrt{(i_x - j_x)^2 + (i_y - j_y)^2}\,; r\right)}{r} \quad (2.1)$$

dengan,

i_x , i_y : Koordinat dari kendaran i
j_x , j_y : Koordinat dari kendaraan j
r : Jarak transmisi maksimum (radius)

Nilai indeks stabilitas yang paling stabil adalah 1, jadi semakin mendekati 1, nilai indeks akan semakin baik. Kondisi ketika indeks stabilitas=1 adalah ketika kendaraan *i* dan *j* mempunyai vektor pergerakan yang sama. Sedangkan indeks stabilitas yang buruk adalah indeks mendekati nilai 0, yaitu ketika kendaraan *i* dan *j* memiliki jarak yang lebih jauh dari pada jarak transmisi maksimum(*r)*. Hal tersebut mengindikasikan kendaraan *i* dan *j* berada di luar radius komunikasi satu sama lain.

Nilai kualitas hubungan Q [2] dapat dihasilkan dengan rumus pada persamaan [2.2].

$$Q = \frac{1}{S_{ij}} \qquad (2.2)$$

Berdasarkan faktor-faktor di atas, rumus TWR[2] yang dimodifikasi, diekspresikan dengan persamaan[2.3].

$$TWR = f_s \times |S_n - S_d| + f_a \times |A_n - A_d| + f_q \times Q \qquad (2.3)$$

dimana,

S_n , A_n : Faktor pergerakan (kecepatan & akselerasi) dari *next-hop node*

S_d , A_d : Faktor pergerakan (kecepatan & akselerasi) dari *node* tujuan

f_s : Faktor pengali kecepatan

f_a : Faktor pengali akselerasi

f_q : Faktor pengali kualitas hubungan

Q : Kualitas hubungan antar *node* sumber dengan *next-hop*

Nilai TWR ditentukan oleh perbedaan faktor pergerakan, yaitu kecepatan dan akselerasi, serta kualitas hubungan antara *node* sumber dengan *next-hop node*. Dari perhitungan diatas, dapat disimpulkan bahwa *next-hop node* yang baik adalah yang memiliki nilai TWR rendah, karena memiliki kecepatan, akselerasi, dan kualitas hubungan yang cukup bagus dengan *node* sumber[2].

Pada topologi VANET yang memiliki topologi yang dinamis dan cepat berubah dengan cepat, maka setelah dilakukan perhitungan TWR, perlu dilakukan perhitungan prediksi TWR pada 3 detik selanjutnya. Maka yang perlu diprediksi adalah :

- Prediksi kecepatan dan akselerasi
  Diasumsikan akselerasi pada *future-TWR* adalah konstan, karena interval dari waktu TWR dengan *future-*

TWR sangat singkat. Sedangkan kecepatan *future*-TWR dapat dihitung dengan rumus penambahan kecepatan (akselerasi).

- Prediksi kualitas hubungan antar kendaraan
  Indeks stabilitas dihitung dengan memanfaatkan koordinat kendaraan, sedangkan posisi koordinat kendaraan dapat diperoleh dengan memanfaatkan informasi kecepatan dan akselerasi yang telah dihitung pada TWR.

Hasil prediksi nilai di atas akan digunakan untuk perhitungan *future*-TWR. Setelah mendapatkan hasil TWR dan *future*-TWR, akan dilakukan proses pemilihan *next-hop node* atau dapat disebut juga sebagai *relay node*, dengan aturan pemilihan sebagai berikut. Tabel [2.1].

Tabel 2.1*: Aturan Pemilihan *Relay Node *[2]*

TWR	*State*	*Future* TWR	Dipilih ?
Optimal	Tidak stabil	Lebih baik	Ya
Optimal	Sabil	Lebih buruk	Ya
Sub-optimal	Tidak stabil	Lebih baik	Ya
Sub-optimal	Stabil	Lebih buruk	Ya
Kondisi lain			Tidak

Penjelasan aturan pemilihan *relay node* :

- TWR
  Jika *node* pengirim sudah menghitung TWR dari seluruh *neighbor node* yang ada disekitarnya,

nilai-nilai TWR tersebut akan dievaluasi. TWR dikatakan "**optimal**" apabila TWR tersebut adalah nilai yang rendah, selain itu nilai TWR akan dianggap sebagai "**sub-optimal**".

- *State*
  Merupakan perbandingan antara *treshold* W dengan nilai absolut dari delta TWR (selisih TWR dengan *future* TWR). Nilai *treshold* sebelumnya akan ditentukan sebagai pembanding apakah nilai delta TWR (ΔTWR) lebih besar atau lebih kecil. Jika lebih besar dari *treshold*, maka calon *relay node* tersebut dianggap tidak stabil.

- *Future* TWR
  Dikatakan "**lebih baik**" ketika nilainya rebih rendah dibandingkan dengan nilai TWR sebelumnya (semakin kecil TWR semakin baik). Sedangkan *future* TWR akan dinilai buruk ketika lebih besar dibandingkan TWR sebelumnya.

Dalam proses pemilihan *relay node*, *node* pengirim mendapatkan informasi pergerakan dari *neighbor nodes*nya. Kemudian dilakukan proses evaluasi apakah nilai TWR *node* di sekitarnya optimal, apakah *node* tersebut stabil dan yang terakhir apakah *future* TWRnya lebih baik dari TWR sekarang. Setelah proses evaluasi tersbut, akan dilakukan pemilihan *relay node* dan selanjutnya *node* pengirim akan melakukan *multicast* ke seluruh *relay node* yang telah dievaluasi dan dipilih.

Pada proses pengiriman paket AODV, *node-node* akan melakukan *broadcast* HELLO *message* untuk *route maintenance* secara periodik. Dalam Tugas Akhir ini perlu dilakukan modifikasi HELLO *message* agar *node* dapat melakukan perhitungan TWR. Struktur HELLO *message* dapat dilihat pada Tabel 2.2.

Tabel 2.2: Struktur Paket HELLO

Dest_IP_Address	*Dest_Sequence_Num*	*Hop_count*
Speed	*Acceleration*	*Lifetime*

Setelah *node* pengirim menerima HELLO *message*, baru akan dilakukan perhitungan TWR dan menghasilkan *relay node set*. Kemudian *node* pengirim melakukan *multicast* RREQ ke semua *relay node set.* Jika *relay node* mengetahui rute menuju *node* destinasi, maka *relay node* tersebut mengirimkan RREP dan *reverse path*-nya. Jika *relay node* belum mengetahui rute ke destinasi, maka dia akan membuat *relay node set* baru lagi dan melakukan proses *multicast* RREQ ke *neighbor node* yang ada disekitarnya, dilakukan proses pencarian *node* sekitar (*node discovery)*, melakukan *route maintenance* dengan mengirim HELLO *message*, melakukan perhitungan TWR dan future TWR, memilih *relay node set* yang berpotensi untuk meneruskan pesan. Proses tersebut akan terus berulang hingga RREQ sampai ke *node* tujuan atau berhenti jika *node* tujuan tidak ditemukan.

2.4 SUMO (*Simulation of Urban Mobility*)

SUMO merupakan sebuah simulator yang digunakan untuk membuat simulasi pergerakan kendaraan pada suatu skenario tertentu dengan topologi yang dapat ditentukan. SUMO merupakan program *open source*, *free,* berukuran cukup kecil, dan simulasi trafik inter-modal. SUMO memungkinkan untuk mensimulasikan pergerakan kendaraan, transportasi publik, interaksi rambu lalu lintas, bahkan sampai interaksi pejalan kaki[3]. SUMO dikembangkan mulai sekitar awal tahun 2000-an yang bertujuan untuk dapat memudahkan penelitian yang melibatkan pergerakan kendaraan di jalan raya. Publikasi refernsi

terbaru tentang SUMO ditulis oleh Krajzewicz et al. [4] pada tahun 2012.

Pada SUMO, terdapat berbagai macam *tools* yang membantu pembuatan simulasi lalu lintas dengan tahapan yang berbeda. Beberapa *tools* SUMO yang digunakan pada Tugas Akhir ini adalah :

- netgenerate
  Merupakan sebuah *tool* yang berfungsi sebagai pembuatan *map*/peta berbentuk *grid, spider,* bahkan *random network.* Netgenerate juga dapat membuat peta berdasarkan peta nyata yang ada di dunia. Pada proses netgenate, *user* dapat memasukkan parameter seperti kecepatan maksimum *nodes*/kendaraan, dan membuat *traffic light* pada peta. Hasil dari netgenerate merupakan *file* peta yang berekstensikan .net.xml. Pada Tugas Akhir ini, netgenerate digunakan untuk pembuatan peta grid dan peta skenario riil.

- netconvert
  Merupakan *tool* yang digunakan untuk mengkonversi peta riil yang nantinya didapatkan dari OpenStreetMap yang berekstensi .osm agar dikonversi menjadi peta yang sesuai dengan format sumo yaitu .net.xml. Pada Tugas Akhir ini netconvert digunakan untuk mengkonversi sebuah peta riil dari OpenStreetMap kedalam SUMO.

- randomTrips.py
  Merupakan tools yang digunakan untuk pembuatan titik awal *node* beserta tujuannya secara *random.* Pada Tugas Akhir ini hasil dari randomTrips.py berupa *file* berekstensi .trips.xml. *Tool* ini digunakan baik untuk peta grid maupun peta riil.
  - duarouter

Berfungsi sebagai pembuat rute pergerakan yang sudah ada asal dan tujuannya. Merupakan *tool* untuk menggabungkan hasil dari netgenerate/netconvert dengan randomTrips.py. Hasil dari duarouter adalah *file* dengan ekstensi .duarouter.xml.

- sumo-gui
  SumoGUI berfungsi untuk menampilkan pergerakan *node* yang sudah dibuat sebelumnya. Cara kerjanya adalah dengan memperlihatkan pergerakan *node* secara GUI dengan membaca pengaturan *file* yang berekstensi .sumocfg. Pada *file* .sumocfgdidalamnya harus mendefinisikan nama *file* peta yang berekstensi .net.xml dengan rutenya yang berekstensi .duarouter.xml.

- sumo
  Buat sebuah file konfigurasi sumo yang berekstensikan .sumocfg yang mendefinisikan lokasi *file* peta dengan ekstensi .net.xml dan *file*rute yang berekstensi .duarouter.xml. Selanjutnya *file* .sumocfg akan diolah ke *file* skenario yang berekstensi .xml.

- traceExporter.py
  Pada Tugas Akhir ini, traceExporter.py digunakan untuk mengolah file skenario.xml dari keluaran proses sumo agar dapat digunakan oleh NS-2 nantinya. Dari *tool* ini akan didapatkan 3 output berekstensi .tcl, yaitu mobility.tcl, activity.tcl, dan config.tcl. Setelah menggunakan *tool* traceExporter.py selanjutnya dilakukan proses pembuatan skenario dan dijalankan dengan menggunakan NS-2.

2.5 OpenStreetMap

OpenStreetMap (OSM)[5] merupakan sebuah proyek kolaboratif untuk mengumpulkan data spasial dan digunakan secara bebas (*open data)*. Data-data tersebut digunakan untuk membangun peta dunia dan peta khusus yang dimanfaatkan untuk berbagai kebutuhan, salah satunya adalah navigasi. OpenStreetMap memungkinkan untuk siapapun yang mengaksesnya dapat menggunakan data geografis yang telah dibangun dan dikumpulkan secara kolaboratif.

Tujuan dari proyek ini adalah, membangun sebuah *database* geografis yang dapat dimanfaatkan berdasarkan lisensi *open database*. Pengembangan OSM dirintis sejak tahun 2004 yang terinspirasi oleh kesuksesan wikipedia dalam pembuatan sumber informasi yang jumlah kontributornya sangat banyak serta tersebar di seluruh dunia. Hingga saat ini, pada komunitas OSM, ribuan kontributor terus bekerja keras secara sukarela untuk menambah dan memperbaiki akurasi, detil, dan aktualitas peta, agar paling tidak dapat memiliki kualitas sebaik peta-peta digital komersial. Seperti layaknya Wikipedia, OpenStreetMap banyak dibangun oleh ribuan relawan dari seluruh dunia yang terus memasukkan data ke dalamnya.

Pada Tugas Akhir ini, penulis menggunakan OSM untuk mengambil peta riil pada area Surabaya, yang nantinya akan dikonversi menjadi peta yang sesuai dengan format SUMO. OpenStreetMap dapat diakses pada laman https://www.openstreetmap.org/.

2.6 JOSM (Java OpenStreetMap Editor)

JOSM (Java OpenStreetMap Editor) adalah alat untuk menyunting data yang telah didapat dari OSM. Aplikasi JOSM dapat diunduh pada website JOSM (https://josm.openstreetmap.de/). Penulis menggunakan JOSM

untuk menyunting dan merapikan potongan peta yang telah didapat dari OSM.

2.7 AWK

AWK merupakan *Domain Spesific Language* yang didesain untuk *text processing*. AWK bersifat *data-driven* yang berisikan sekumpulan perintah yang digunakan untuk memproses banyak *tasks* hanya dengan beberapa baris kode. Tujuan penggunaan AWK pada Tugas Akhir ini aadalah memproses *output* dari simulasi NS-2 menjadi hasil yang dapat dianalisa.

2.8 Network Simulator – 2 (NS - 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.Saat ini terdapat dua buah major version dari NS yang masih dikembangkan yaitu NS-2 dan NS-3. Pada Tugas Akhir ini digunakan versi NS-2.35 yang dirilis 4 November 2011. Dalam membuat sebuah simulasi, NS-2 menggunakan dua buah bahasa pemrograman, yaitu C++ dan Otcl. Bahasa C++ digunakan untuk mengimplementasi bagian jaringan yang akan disimulasikan, sedangkan Otcl digunakan untuk menulis skenario simulasi jaringan. NS-2 mendukung sistem operasi GNU/Linux, FreeBSD, OS X dan Solaris. NS-2 juga dapat berjalan pada sistem operasi windows dengan menggunakan Cygwin.

2.8.1 Instalasi Network Simulator – 2

Sebelum meng-*install* NS-2 perlu dilakukan instalasi dependensi yang diperlukan. Untuk meng-*install package* yang diperlukan masukkan perintah berikut.

```
sudo apt-get install build-essential autoconf
automake libxmu-dev gcc-4.4
```

Gambar 2.4: **Perintah untuk Meng*installPackage* Dependensi**

Setelah instalasi dependensi dilakukan, selanjutnya adalah mengunduh *source code* NS-2. Setelah selesai mengunduh, lakukan ekstraksi *file*. Proses unduh dan ekstraksi dapat dilakukan

```
wget
http://jaist.dl.sourceforge.net/project/nsnam/a
llinone/nsallinone-2.35/ns-allinone-2.35.tar.gz

tar -xvf ns-allinone-2.35.tar.gz
```

seperti gambar berikut.

Gambar 2.5: Perintah untuk Mengunduh dan Mengekstrak ns2

Selanjutnya masuk ke folder “linkstate” yang terletak pada *ns-allinone-2.35/ns-2.35/linkstate*. Buka dan edit *file* “ls.h” dan pergi ke baris 137. Ubah “error” menjadi “this->error” seperti yang ditunjukkan pada gambar 2.4. Kemudian ubah konfigurasi makefile pada ns-allinone-2.35/otcl-1.14/Makefile.in. Ubah konfigurasi CC menggunakan gcc-4.4 seperti pada gambar 2.5.

```
136
137         void eraseAll() { this->erase(baseMap::begin(), baseMap::end()); }
138         T* findPtr(Key key) {
139                 iterator it = baseMap::find(key);
140                 return (it == baseMap::end()) ? (T *)NULL : &((*it).second);
141         }
142 };
143
```

Gambar 2.6: Pengubahan pada *File* ls.h

```
 6
 7 CC=              gcc-4.4
 8 CFLAGS=          @CFLAGS@
 9 RANLIB=          @RANLIB@
10 INSTALL=         @INSTALL@
11
```

Gambar 2.7: Perubahan pada Makefile.in

Setelah semua tahap selesai, jalankan skrip instalasi NS-2 dengan memasukkan perintah *./install* pada terminal direktori NS-2 dan tunggu hingga proses instalasi selesai. Setelah instalasi selesai, lakukan *setting environment path* pada .bashrc, restart sistem dan NS dapat digunakan.

```
# LD_LIBRARY_PATH
OTCL_LIB=/home/vertikal/Documents/Workspace_TA/ns-allinone-2.35/otcl-1.14
NS2_LIB=/home/vertikal/Documents/Workspace_TA/ns-allinone-2.35/lib
X11_LIB=/usr/X11R6/lib
USR_LOCAL_LIB=/usr/local/lib
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$OTCL_LIB:$NS2_LIB:$X11_LIB:$USR_LOCAL_LIB
# TCL_LIBRARY
TCL_LIB=/home/vertikal/Documents/Workspace_TA/ns-allinone-2.35/tcl8.5.10/library
USR_LIB=/usr/lib
export TCL_LIBRARY=$TCL_LIB:$USR_LIB
# PATH
XGRAPH=/home/vertikal/Documents/Workspace_TA/ns-allinone-2.35/bin:/home/vertikal/Documents/
Workspace_TA/ns-allinone-2.35/tcl8.5.10/unix:/home/vertikal/Documents/Workspace_TA/ns-
allinone-2.35/tk8.5.10/unix
#the above two lines beginning from xgraph and ending with unix should come on the same line
NS=/home/vertikal/Documents/Workspace_TA/ns-allinone-2.35/ns-2.35/
NAM=/home/vertikal/Documents/Workspace_TA/ns-allinone-2.35/nam-1.15/
PATH=$PATH:$XGRAPH:$NS:$NAM
```

Gambar 2.8:Penambahan *EnvironmentPath* pada .bashrc

2.8.2 Penggunaan Skrip OTcl

OTcl [6] merupakan bahasa *scripting* yang digunakan pada NS-2 untuk pengaturan skenario lingkungan simulasi. Setiap *class* pada OTcl memiliki *binding* pada C++. Hal ini memungkinkan pembuatan skenario simulasi tanpa harus membuat fungsi C++ secara langsung. Gambar 2.7 menunjukkan potongan kode OTcl yang digunakan untuk mengatur lingkungan simulasi.

```
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 val(ifqlen) 50 ;# max packet in ifq
set val(rp) AODV ;# routing protocol
set val(nn) 50
set val(cbrsize) 512    ;# 512 Bytes
set val(cbrrate) 2KB
set val(cbrinterval) 1  ;# 1 packet per second
set val(stop) 360
set val(mobilityfile) "mobility.tcl"
set val(activityfile) "activity.tcl"

# Initialize ns
set ns [new Simulator]

# Set up topography object
set topo       [new Topography]

set channel_ [new $val(chan)]
$ns node-config \-adhocRouting $val(rp) \-llType $val(ll) \-macType $val(mac) \-ifqType $val(ifq) \-ifqLen $val(ifqlen) \-antType $val(ant)
\-propType $val(prop) \-phyType $val(netif) \-channel  $channel_ \-agentTrace ON \-routerTrace ON \-macTrace OFF \-movementTrace OFF \-
topoInstance $topo
```

Gambar 2.9: Contoh Kode Pengaturan Lingkungan Simulasi

2.8.3 NS-2 *Tracefile*

Tracefile merupakan hasil simulasi dari sebuah skenario yang telah dijalankan dengan NS-2. Isi dari sebuah file *tracefile* adalah catatan dari setiap paket yang dikirim dan diterima oleh setiap *node* dalam simulasi. Terdapat beberapa jenis paket yang digunakan, antara lain, HELLO, RREQ, RREP, RRER, dan paket data.

Contoh jenis paket yang dikirimkan ditunjukkan pada gambar 2.8. Gambar 2.8 hingga 2.12 menunjukkan contoh paket routing control AODV dari NS-2. Paket *routing control* selalu ditandai dengan tulisan "RTR" pada kolom keempat. Kolom ketujuh menunjukkan informasi nama *routing protocol*. Kolom kedelapan menunjukkan ukuran dari paket *routing control*. Kolom terakhir menunjukkan jenis paket *routing control*. Sedangkan pada gambar 2.12 menunjukkan paket data dari agen CBR (*Constant Bit Rate).* Cirinya, paket selalu ditandai dengan tag “AGT” pada kolom ke empat. Kolom ketujuh menunjukkan informasi *agent.* Kolom ke delapan menunjukkan ukuran paket data.

Informasi penting lainnya adalah, tag “s” untuk paket yang dikirim, sedangkan tag “r” untuk paket yang diterima, tag tersebut terdapat pada kolom pertama. Sedangkan kolom kedua

merupakan waktu dalam detik, ketika *event* terjadi. Dengan mengetahui informasi pada *tracefile*, analisa simulasi dapat dilakukan.

```
r 150.001408063 _23_ RTR  --- 0 AODV 48 [0 ffffffff 64
800] ------- [100:255 -1:255 30 0] [0x2 1 1 [101 0] [100
4]] (REQUEST)
```

Gambar 2.10: RREQ *Packet*

```
s 150.019645028 _101_ RTR  --- 0 AODV 44 [0 0 0 0] ------
- [101:255 100:255 30 63] [0x4 1 [101 4] 10.000000]
(REPLY)
```

Gambar 2.11: RREP *Packet*

```
r 153.057009186 _91_ RTR  --- 0 AODV 32 [0 ffffffff 2e
800] ------- [46:255 -1:255 1 0] [0x8 1 [101 0] 0.000000]
```

Gambar 2.12: RRER *Packet*

```
r 154.098765111 _101_ AGT  --- 4 cbr 532 [13a 65 f 800] -
------ [100:0 101:0 27 101] [4] 4 0
```

Gambar 2.13: HELLO *Packet*

```
r 154.050270930 _83_ RTR  --- 0 AODV 48 [0 ffffffff 2a
800] ------- [42:255 -1:255 5 0] [0x2 3 2 [101 5] [100
6]] (HELLO)
```

Gambar 2.14: Contoh Packet Data

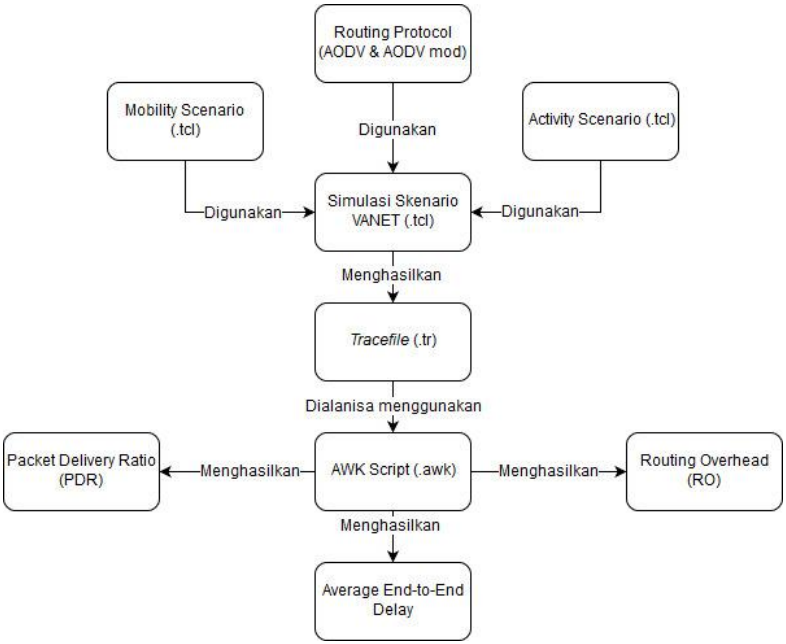
(Halaman sengaja dikosongkan)

BAB III
PERANCANGAN

Bab ini membahas mengenai implementasi sistem maupun metode yang akan digunakan. Bab ini secara khusus menjelaskan rancangan sistem yang dibuat pada Tugas Akhir. Bagian yang akan dijelaskan pada bab ini berawal dari deskripsi umum hingga perancangan skenario grid, riil, dan alur implementasinya.

3.1 Deskripsi Umum

Pada Tugas Akhir ini dilakukan implementasi serta analisis *routing protocol* AODV dan dilakukan modifikasi untuk mengetahui pengaruh faktor kecepatan. Diagram rancangan simulasi dapat dilihat pada gambar 3.1.



Gambar 3.1: Diagram Alur Rancangan Simulasi

Dalam Tugas Akhir ini terdapat dua jenis skenario yang digunakan sebagai bahan analisa, yaitu skenario yang berbentuk grid dan skenario yang berdasarkan peta riil yang berada di lingkungan kota Surabaya. Skenario grid digunakan menggunakan sumo, sedangkan skenario riil membutuhkan OpenStreetMap lalu dirapikan menggunakan JOSM dan selanjutnya dikonversi menggunakan SUMO. Setalah mendapatkan file peta yang diinginkan, dilakukan proses simulasi oleh NS-2. Setelah proses simulasi selesai, dilakukan analisa hasil *tracefile* menggunakan skrip awk untuk menghitung *packet delivery ratio, average end-to-end-delay, routing overhead.*

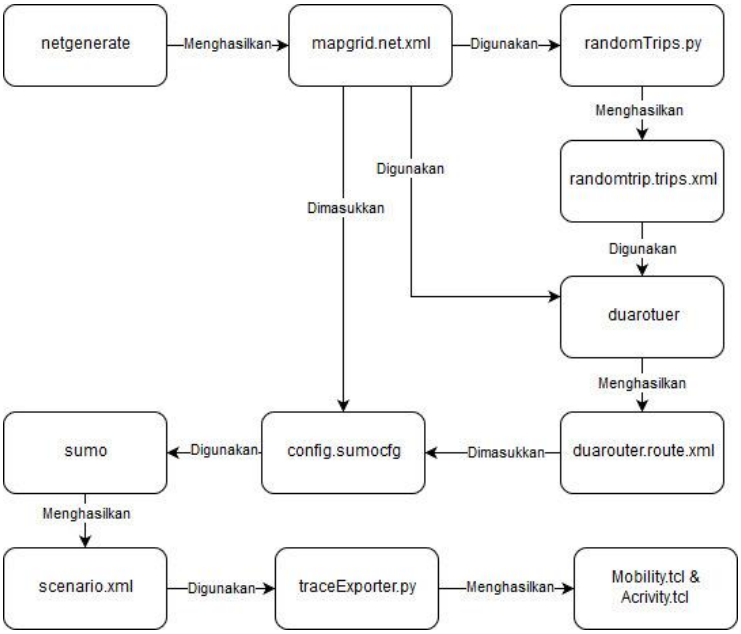
3.2 Perancangan Skenario Grid

Pembuatan skenario grid dapat langsung menggunakan *tool* yang dimiliki SUMO, yaitu netgenerate. Pada Tugas Akhir ini, kecepatan pada skenario grid dibedakan menjadi 3 kecepatan, yaitu 10 m/s, 15 m/s dan 20 m/s. Ukuran dari grid adalah 1000 m x 1000 m. Dan jumlah blok gridnya adalah 5, secara horizontal maupun vertikal. Selain kecepatan, jumlah *node* atau kendaraan juga divariasikan, yaitu 50 *nodes,* 75 *nodes* dan 100 *nodes.* Ilustrasi alur perancangan skenario grid dapat dilihat pada gambar 3.2.

3.3 Perancangan Skenario Riil

Pada perancangan skenario riil prosesnya mirip dengan perancangan skenario grid. Namun untuk mendapatkan peta riilnya memerlukan OpenStreetMap. Maka langkah pertama adalah, pilih daerah yang diinginkan pada website OpenStreetMap, ketika sudah memilih area yang diinginkan, lakukan *export* area dan simpan sebagai .osm *file.* Setelah itu buka *file* .osm dengan JOSM, edit peta yang telah didapat. Setelah dilakukan edit menggunakan JOSM, lakukan konversi ke map.net.xml menggunakan netconvert dari SUMO. Untuk langkah selanjutnya sama dengan proses skenario grid, hingga

akhirnya menghasilkan activity dan mobility tcl. Alur selengkapnya dapat dilihat pada gambar 3.3.



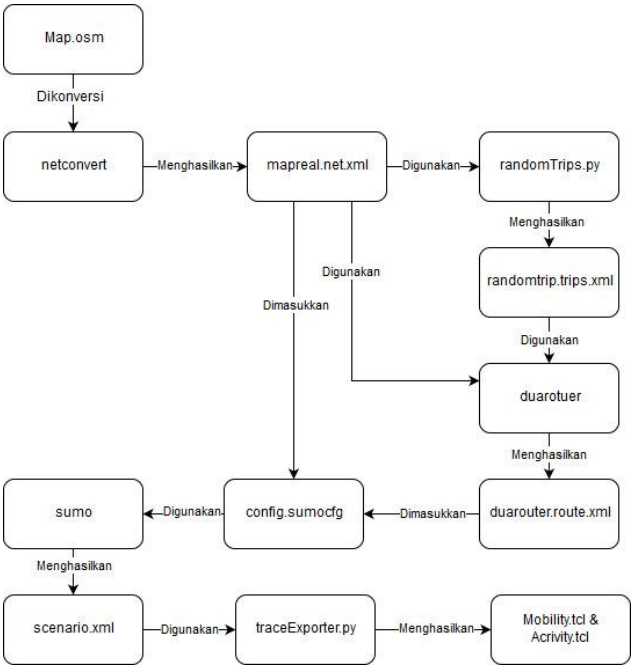
Gambar 3.2: Alur Perancangan Skenario Grid

3.4 Perancangan Implementasi AODV dengan Penambahan Faktor Pergerakan

Dalam pemilihan *relaynode*, protokol AODV pada Tugas Akhir ini, menggunakan perhitungan TWR dan *future* TWR dari masing-masing *neighbor* node. Perhitungan TWR dilakukan dengan memanfaatkan faktor pergerakan dari *neighbor node.* Sedangkan *future* TWR menggunakan rumus fisika gerak lurus untuk memprediksi informasi yang akan datang.

Diasumsikan *node* pengirim dapat mengetahui letak dari *neighbornode* dan *node* tujuan melalui layanan GPS, namun implementasi *routing protocol* AODV pada NS-2 tidak

menggunakan location service, sehinga tidak memungkinkan mendapat lokasi dari setiap *node*. Oleh karena itu dilakukan modifikasi untuk mendapatkan informasi posisi *neighbor node*. Modifikasi dilakukan dengan penambahan *field* koordinat x dan y pada struktur paket HELLO. Modifikasi struktur yang baru dapat dilihat pada tabel 3.1.



Gambar 3.3: Alur Perancangan Skenario Riil

Tabel 3.1: Modifikasi Sturktur HELLO

Dest_IP_Address	*Dest_Sequence_Num*	*Hop_count*

Speed	*Acceleration*	*Lifetime*	*x*	*y*

Pada Tugas Akhir ini, *node* sumber dan *node* tujuan diasumsikan berada dalam posisi diam *(stationary node)*sehingga

perlu dilakukan penyesuaian terhadap formula TWR. Perhitungan TWR terdiri atas faktor pergerakan (kecepatan dan akselerasi) dan faktor kualitas hubungan. Faktor pergerakan didapat dari selisih nilai yang dimiliki oleh *next-hop node* terhadap *node* tujuan. Untuk formula TWR tidak dilakukan modifikasi karena tidak menambahkan faktor lain selain faktor pergerakan dan faktor kualitas hubungan.

Namun pada formula indeks stabilitas perlu diubah karena jika hasil fungsi minimum antara jarak dan radius transmisi maksimum menghasilkan nilai dari radius transmisi maksimum, akan terjadi *division by zero* pada persamaan 2.2. Maka dilakukan perubahan formula indeks stabilitas sebagai berikut:

$$S_{ij} = 1 - \frac{\min\left(\sqrt{(i_x - j_x)^2 + (i_y - j_y)^2}\,; r\right)}{r+1} \quad \textbf{(3.2)}$$

Untuk menghitung *future* TWR membutuhkan prediksi nilai faktor pergerakan yang akan datang. Berikut adalah beberapa rumus fisika gerak lurus yang akan digunakan dalam menghitung nilai prediksi:

- Perhitungan akselerasi

$$a = \frac{\Delta v}{\Delta t} \quad \textbf{(3.3)}$$

dimana,
Δv : Selisih kecepatan
Δt : Selisih waktu

- Perhitungan kecepatan

$$v' = v + a \times t \quad \textbf{(3.4)}$$

dimana,
v : Kecepatan saat ini

a : Akselerasi
t : Waktu dalam detik

- Perhitungan posisi yang akan datang

$$x' = x + v_0 t + \frac{1}{2} a t^2 \textbf{(3.5)}$$

$$y' = y + v_0 t + \frac{1}{2} a t^2 \textbf{(3.6)}$$

dimana,
x,*y* : Koordinat sekarang
v_0: Kecepatan sekarang
a : Akselerasi
t : Waktu dalam detik

Nilai waktu yang digunakan pada rumus-rumus diatas adalah bernilai konstan, 3 detik. Sedangkan nilai kecepatan dan akselerasi didapatkan dari paket HELLO yang dikirim oleh *neighbornode* secara periodik.

3.5 Perancangan Metrik Analisis

Berikut ini merupakan beberapa metrik analisis yang digunakan untuk menganalisa hasil simulasi pada Tugas Akhir ini:

3.5.1 *Packet Delivery Ratio* / PDR

PDR merupakan perbandingan dari jumlah paket data yang dikirim, dengan paket data yang diterima. PDR dihitung dengan persamaan 3.7.

$$PDR = \frac{Data_{received}}{Data_{sent}} \textbf{(3.7)}$$

3.5.2 *Average End-to-End Delay*

Average end-to-end delay adalah rata-rata waktu yang dibutuhkan setiap paket dari *node* pengirim hingga ke *node* tujuan. Semua paket, termasuk *delay* yang diakibatkan oleh paket *routing*, juga akan diperhitungkan. Paket yang akan dimasukkan ke dalam perhitungan hanya paket yang berhasil sampai ke tujuan. Persamaan *average end-to-end delay* dapat dilihat pada persamaan 3.8.

$$Delay = \frac{\sum_{i=0}^{n} t_{received}[i] - t_{sent}[i]}{receivedPacket} \textbf{(3.8)}$$

3.5.3 *Routing Overhead*

Routing Overhead (RO) adalah jumlah paket *routing control* yang ditransmisikan selama simulasi. Paket kontrol yang dimaksud adalah RREQ, RREP dan RERR. Perhitungan jumlah RO dapat dilihat pada persamaan 3.9.

$$RO = RREQ_{sent} + RREP_{sent} + RERR_{sent} \textbf{(3.9)}$$

(Halaman ini sengaja dikosongkan)

BAB IV
IMPLEMENTASI

Bab ini berisi mengenai implementasi dari perancangan sistem yang dijelaskan pada bab sebelumnya.

4.1 Implementasi Skenario *Grid*

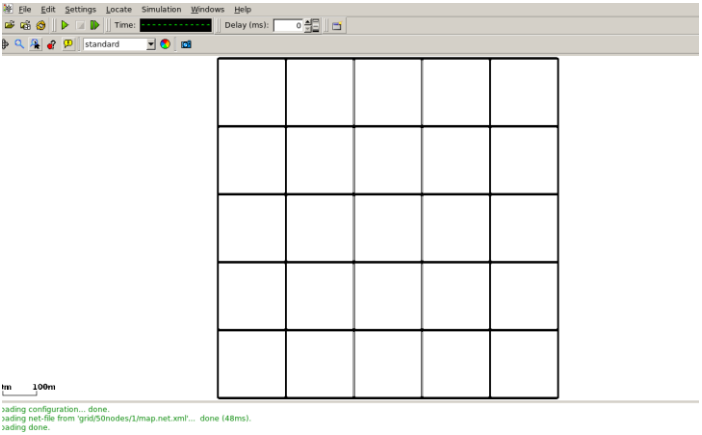
Pada pembuatan skenario *grid* akan digunakan sebuah *tool* dari SUMO yaitu netgenerate. Skenario *grid* yang akan dibuat pada Tugas Akhir ini berukuran 1000 m x 1000 m dengan jumlah titik persimpangan sebanyak 6 x 6 titik pada persimpangan vertikal dan horizontal. Variasi kecepatan yang diperbolehkan adalah 10 m/s, 15 m/s dan 20 m/s.

Untuk membuat peta grid dengan spesifikasi tersebut, digunakan perintah seperti gambar 4.1

```
Netgenerate --grid --grid.number=6 --grid.length=200 --
default.speed=15 --tls.guess=1 --output-file=map.net.xml
```

Gambar 4.1: Perintah untuk Membuat Peta *Grid*

Hasil peta *grid* dari *tool* netgenerate dapat dilihat pada gambar 4.2. Langkah selanjutnya, lakukan pembuatan titik asal dan tujuan untuk setiap kendaraan secara *random* melalui modul randomTrips.py yang sudah tersedia pada SUMO. Penggunaan randomTrips.py ditunjukkan pada gambar 4.3. Jumlah *node* dapat didefinisikan pada opsi **-e**. Agar setiap *node* dapat memiliki banyak alternatif *route*, gunakan **--intermediate** dengan *value* yang cukup besar (contohnya dengan menggunakan $RANDOM) agar setiap *node* dapat aktif hingga simulasi berakhir.



Gambar 4.2: Hasil Pembuatan Peta *Grid* dengan Netgenerate

```
/usr/share/sumo/tools/randomTrips.py -n map.net.xml -e
$NUM_NODES --seed=$RANDOM --fringe-factor 5.0 --
intermediate=$RANDOM --trip-attributes='departLane="best"
departPos="random_free"' -o randomtrip.trips.xml
```

Gambar 4.3: Perintah untuk Membuat Titik Asal dan Titik Tujuan Setiap Kendaraan Secara Acak dengan RandomTrips.py

Proses selanjutnya adalah, pembuatan rute dengan *tool* duarouter yang akan digunakan oleh setiap *node*. Duarouter akan menggabungkan file peta *grid* .net.xml dengan *file* .randomtrip.trips yang telah dibuat sebelumnya. Perintah dapat dilihat pada gambar 4.4.

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

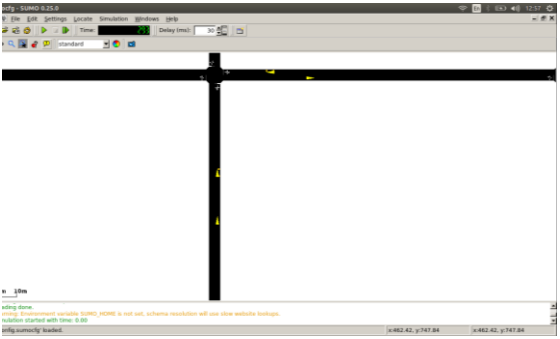
Gambar 4.4: Perintah untuk Membuat Rute Setiap Kendaraan

Langkah selanjutnya, buat file konfigurasi berekstensi .sumocfg yang berisi lokasi map.net.xml dan lokasi *file* rute yang dihasilkan duarouter. Gambar 4.5 menunjukkan contoh isi *file* config.sumocfg.

```
<?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="${net_filename}"/>
    <route-files value="${route_filename}"/>
  </input>
  <time>
    <begin value="0"/>
    <end value="${end_time}"/>
  </time>
</configuration>
```

Gambar 4.5: Isi dari *File* .sumocfg

Untuk melihat visualisasi dari file config.sumocfg dapat menggunakan perintah *"sumo-gui -c config.sumocfg"* , hasil visualisasi sumogui dapat dilihat pada gambar 4.6. Langkah selanjutnya lakukan simulasi lalu lintas dengan perintah pada gambar 4.7.



Gambar 4.6: Visualisasi dengan Sumo-gui

```
sumo -c config.sumocfg --fcd-output simulation-result.xml
```

Gambar 4.7: Simulasi Lalu Lintas dengan Sumo

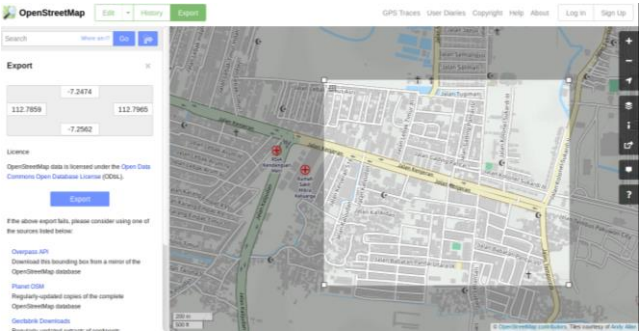
Hasil dari simulasi dengan sumo, selanjutnya akan dikonversi ke dalam format yang dapat dijalankan oleh NS-2 dengan *tool* traceExporter.pyyang terdapat pada gambar 4.8.

```
$SUMO_HOME/tools/traceExporter.py                    --fcd-
input=simulation_result.xml                  --ns2mobility-
output=mobility.tcl     --ns2config-output=config.tcl     --
ns2activity-output=activity.tcl
```

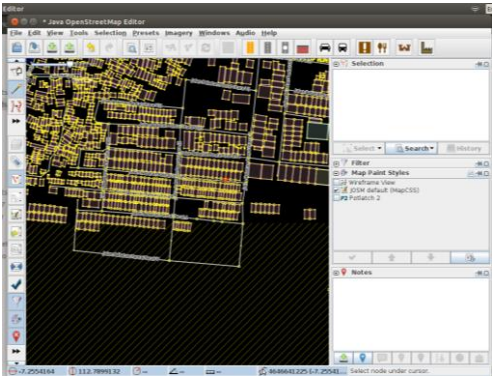
Gambar 4.8: Perintah Mengkonversi Keluaraan dari Sumo Menjadi *File* .tcl

4.2 Implementasi Skenario Riil

Skenario riil menggunakan bagian peta wilayah kota Surabaya yang diambil dari OpenStreetMap. Cara pengambilan peta dengan menseleksi area yang akan diambil, lalu gunakan perintah export yang telah tersedia seperti pada Gambar 4.9. Selanjutnya lakukan penyuntingan peta .osm dengan menggunakan JOSM. Tujuan dari penyuntingan adalah untuk menghapus jalan yang tidak digunakan, seperti bangunan di sekitar jalan, dan hal-hal lain yang tidak digunakan. Selain itu, penambahan jalan baru juga perlu ditambahkan untuk penyesuaian skenario dan agar tidak ada jalan yang buntu. Contoh dari proses penyuntingan dapat dilihat pada Gambar 4.10. Setelah selesai melakukan penyuntingan, peta tersbut akan dikonversi ke dalam file yang sesuai dengan ekstensi SUMO yaitu .net.xml dengan menggunkan *tool* netconvert.



Gambar 4.9: Contoh Proses Pengambilan Peta Pada OpenStreetMap



Gambar 4.10: Contoh Proses Penyuntingan Peta Pada JOSM

```
Netconvert --remove-edges.by-vclass pedestrian,rail --
tls.join --osm-files peta_darmo.osm --output-file
map.net.xml --remove-edges.isolated
```

Gambar 4.11: Konversi dengan Netconvert ke dalam Format .net.xml

Setelah proses netconvert selesai, proses selanjutnya sama dengan proses pembuatan skenario *grid*, hingga mendapatkan *file activity* serta *mobility.tcl* .

4.3 Implementasi Modifikasi Protokol AODV

Protokol AODV yang akan dimodifikasi memiliki beberapa perubahan dibandingkan dengan protokol AODV biasa yang sudah tersedia pada NS-2. Perubahan yang dilakukan adalah sebagai berikut:

- Aktivasi HELLO
- Modifikasi struktur paket HELLO
- Modifikasi dan penambahan atribut pada kelas AODV
- Penyusunan struktur *neighbor cache*
- Penanganan paket HELLO
- Penanganan paket RREQ

Secara *default*, kode implementasi protokol AODV terdapat pada direktori ns2/aodv. Daftar kode sumber yang akan dimodifikasi pada direktori tersebut adalah sebagai berikut:

- aodv.h untuk mengaktifkan pesan HELLO dan penambahan atribut pada kelas AODV
- aodv.cc untuk modifikasi penanganan HELLO dan RREQ
- aodv_rtable.h untuk mengubah *neighbor cache*
- aodv_packet.h untuk mengubah stuktur paket HELLO dan RREQ

Pada bagian ini akan dijelaskan langkah-langkah dalam mengimplementasikan modifikasi AODV dengan penambahan faktor pergerakan.

4.3.1 Pengaktifan Pesan HELLO

Pada AODV biasa, secara *default*, HELLO *message* dinon-aktifkan. Sehingga perlu menghilangkan baris tertentu. Untuk dapat menggunakan HELLO, *comment* baris *link layer detection* pada baris 58 dan *use LL metric* pada baris 66 pada *file* aodv.h seperti gambar 4.12.

```
58  // #define AODV_LINK_LAYER_DETECTION //-->comment this
59
60  /*
61    Only applies if AODV_USE_LL_METRIC is defined.
62    Causes AODV to apply omniscient knowledge to the feedback received
63    from 802.11.  This may be flawed, because it does not account for
64    congestion.
65  */
66  //#define AODV_USE_GOD_FEEDBACK //-->comment this
```

Gambar 4.12: Comment untuk Mengaktifkan Pesan HELLO

4.3.2 Modifikasi Struktur Paket HELLO

Secara *default* implemetasi *routing* protocol AODV pada NS-2, *struct* paket HELLO menggunakan *struct* yang sama dengan paket RREP, karena pada dasarnya paket HELLO merupakan sebuah RREP dengan TTL bernilai 1 [3]. Dalam modifikasi AODV dengan faktor kecepatan, perlu adanya penambahan variabel pada *struct* paket HELLO, yaitu kecepatan, akselerasi, dan koordinat x dan y. Untuk itu dilakukan penambahan atribut rp_speed, rp_accel, rp_x, dan rp_y pada *struct* hdr_aodv_reply di dalam *file* aodv_packet.h seperti pada Gambar 4.13.

4.3.3 Modifikasi Class AODV

Implementasi modifikasi AODV sangat bergantung pada informasiyang didapatkan dari objek MobileNode. Salah satu atribut yang digunakan untuk perhitungan akselerasi node adalah position_update_time. Informasi kecepatan pun juga diperlukan

untuk mendapatkan informasi akselerasi. Oleh karena itu penambahan atribut seperti lastUpdateTime, lastSpeed, dan lastAccel dalam skrip aodv.h dapat berfungsi sebagai alat untuk memonitoring dampak faktor pergerakan. Penambahan atribut dapat dilihat pada Gambar 4.14.

```
struct hdr_aodv_reply {
        u_int8_t        rp_type;        // Packet Type
        u_int8_t        reserved[2];
        u_int8_t        rp_hop_count;           // Hop Count
        nsaddr_t        rp_dst;                 // Destination IP Address
        u_int32_t       rp_dst_seqno;           // Destination Sequence Number
        nsaddr_t        rp_src;                 // Source IP Address
        double          rp_lifetime;            // Lifetime
        double          rp_timestamp;           // when corresponding REQ sent;
        //penambahan attribut HELLO
        double          rp_speed;
        double          rp_accel;
        double          rp_x;
        double          rp_y;
        double          rp_xBefore;
        double          rp_yBefore;
```

Gambar 4.13: Penambahan Atribut Paket HELLO

```
//penambahan atribut pada class AODV
nsaddr_t        index;
u_int32_t       seqno;
int             bid;
double          lastX;
double          lastY;
double          lastUpdateTime;
double          lastAccel;
double          lastSpeed;
```

Gambar 4.14: Penambahan Atribut *Class* AODV

4.3.4 Modifikasi *Neighbor Cache Entry*

AODV menggunakan *neighbor cache* sebagai tempat penyimpanan dari daftar *neighbor node*. Setiap kali menerima HELLO, informasi pada *neighbor cache* akan terus diperbarui. Sebelumnya list tersebut tidak menyimpan informasi kecepatan,

akselerasi, dan koordinat *neighbor node* sehingga perlu penambahan atribut nb_speed, nb_accel, nb_x dan nb_y dalam skrip aodv_rtable.h. Kode implementasi dapat dilihat pada gambar 4.15.

```
/*
   AODV Neighbor Cache Entry
*/
class AODV_Neighbor {
        friend class AODV;
        friend class aodv_rt_entry;
 public:
        AODV_Neighbor(u_int32_t a) { nb_addr = a; }

 protected:
        LIST_ENTRY(AODV_Neighbor) nb_link;
        nsaddr_t        nb_addr;
        double          nb_expire;      // ALLOWED_HELLO_LOSS * HELLO_INTERVAL

        //PENAMBAHAN ATRIBUT
        double          nb_speed;
        double          nb_accel;
        double          nb_x;
        double          nb_y;
        double          nb_xBefore;
        double          nb_yBefore;
};
```

Gambar 4.15: Penambahan atribut AODV *Neighbor Cache* pada *File* aodv_rtable.h

4.3.5 Modifikasi Proses Pengiriman HELLO

Dalam AODV yang telah dimodifikasi, paket HELLO digunakan untuk mengirim informasi kecepatan, akselerasi, dan koordinat dari *node* pengirim. Pengisian nilai dari atribut paket HELLO yang baru dapat dilakukan di fungsi sendHello pada aodv.cc.Pada proses pengiriman HELLO terdapat beberapa tahap, yaitu pembuatan paket,pengisian atribut paket, dan penjadwalan event pengiriman dalam NS-2.

Kecepatan dan koordinat *node* pengirim paket HELLO didapatkan dari kelas MobileNode. Langkah pertama, cari *node* pengirim, kemudian, ambil nilai kecepatan dengan fungsi speed(),

dan ambil nilai lain seperti nilai waktu, korrdinat x dan y dengan memanfaatkan fungsi getUpdateTime(), X(). Proses selanjutnya, hitung nilai akselerasi dengan pembagian selisih kecepatan dan selisih waktu. Langkah terakhir, informasi yang telah didapat tinggal dimasukkan ke masing-masing atribut paket HELLO. Hasil modifikasi dapat dilihat pada Gambar 4.16.

4.3.6 Modifikasi Proses Penerimaan HELLO

Pada saat paket HELLO datang, *node*penerima paket akan mengecek *listneighbor cache*. Jika alamat pengirim paket HELLOtersebut sudah ada dalam *list neighbor cache* lakukan *update* informasi kecepatan, akselerasi, koordinat dan batas waktu penyimpanan informasi *node* tersebut di dalam *cache*. Jika alamat pengirim paket HELLO tidak ditemukan dalam *list*, maka tambahkan alamat tersebut ke dalam *list* dan simpan informasi kecepatan, akselerasi, dan koordinatnya. Hasil modifikasi dapat dilihat pada Gambar 4.17

4.3.7 Implementasi Perhitungan TWR, *Future* TWR, dan Pengolahan *Relay Node Set*

*Node*yang akan melakukan perhitungan TWR hingga proses pengolah *relay node set* adalah node pengirim/penerus paket RREQ. Nilai *treshold* W dan faktor-faktor pengali dari kecepatan, akselerasi serta kualitas hubungan harus didefinisikan terlebih dahulu. Setelah nilai faktor pengali ditentukan, informasi kecepatan, akselerasi, dan koordinat dari *node* pengirim akan disimpan. Untuk dapat dilakukan perhitungan TWR,dibutuhkan informasi kecepatan, akselerasi, dan koordinat dari *neighbor node* dan *node* tujuan. Dengan memanfaatkan *neighbor cache*informasi diatas dapat diperoleh dengan cara mengarahkan pointer ke alamat *neighbor cache* dan lakukan perulangan hingga akhir *list* seperti pada Gambar 4.18.

```
void AODV::sendHello() {
Packet *p = Packet::alloc();
struct hdr_cmn *ch = HDR_CMN(p);
struct hdr_ip *ih = HDR_IP(p);
struct hdr_aodv_reply *rh = HDR_AODV_REPLY(p);

MobileNode *iNode;
iNode = (MobileNode *) (Node::get_node_by_address(index));
double iSpeed = ((MobileNode *) iNode)->speed();
double now = ((MobileNode *) iNode)->getUpdateTime();
double posX = iNode->X(); //modifikasi penambahan x coordinat
double posY = iNode->Y(); //modifikasi penambahan y coordinat

 rh->rp_type = AODVTYPE_HELLO;
 rh->rp_hop_count = 1;
 rh->rp_dst = index;
 rh->rp_dst_seqno = seqno;
 rh->rp_lifetime = (1 + ALLOWED_HELLO_LOSS) * HELLO_INTERVAL;

 // modifikasi masukkan speed dan accel info ke hello packet

 if (now - lastUpdateTime == 0) { //if not updated
   rh->rp_accel = lastAccel;
 }
 else {
   rh->rp_accel = iSpeed / (now - lastUpdateTime); // a = delta v/delta t
   lastAccel    = rh->rp_accel;
   lastSpeed    = iSpeed;
 }
 rh->rp_speed = iSpeed;
 lastUpdateTime = now;
 rh->rp_x     = posX;
 rh->rp_y     = posY;
 rh->rp_xBefore = lastX;
 rh->rp_yBefore = lastY;
```

Gambar 4.16: Modifikasi Fungsi sendHello pada aodv.cc

Pada Tugas Akhir ini*node* tujuan memiliki kecepatan dan akselerasi yang bernilai nol karena *node* tersebut diam (stationary node). Koordinat *node* tujuan akandituliskan langsung pada kode sumber.

Proses perhitungan TWR akan dilakukan setelah informasi *node* pengirim, *neighbor node*, dan *node* tujuan telah diperoleh. Proses perhitungan TWR dimulai dengan menghitung kualitas hubungan antara *node* pengirim dan *next-hop node* dengan formulaindeks stabilitas yang dinormalisasi, lalu kecepatan dan akselerasi dari *next-hop node* masing-masing dikurangkan dengan kecepatan dan

akselerasi dari node tujuan. Karena *node* tujuan telah diasumsikan selalu diam, maka kecepatan dan akselerasi dari *next-hop node* masing-masing dikurangi nol. Nilai TWR didapat dengan menjumlahkan hasil perkalian antara faktor pengali dengan kecepatan, akselerasi dan kualitas hubungan. Perhitungan TWR dapat dilihat pada Gambar 4.19.

```
void AODV::recvHello(Packet *p) {
struct hdr_aodv_reply *rp = HDR_AODV_REPLY(p);
AODV_Neighbor *nb;
 nb = nb_lookup(rp->rp_dst);
 //modifikasi mencari neighbor
 if(nb == 0) { //jika tidak ditemukan
   nb_insert(rp->rp_dst);
   // modif lookup neighbor
   nb = nb_lookup(rp->rp_dst);
   // modif dapatkan data speed dan accel dari hello message
   nb->nb_speed = rp->rp_speed;
   nb->nb_accel = rp->rp_accel;
   nb->nb_x     = rp->rp_x;
   nb->nb_y     = rp->rp_y;
   nb->nb_xBefore = rp->rp_xBefore;
   nb->nb_yBefore = rp->rp_yBefore;
 }
 else {
   // modif dapatkan data speed dan accel dari hello message
   nb->nb_speed = rp->rp_speed;
   nb->nb_accel = rp->rp_accel;
   nb->nb_x     = rp->rp_x;
   nb->nb_y     = rp->rp_y;
   nb->nb_xBefore = rp->rp_xBefore;
   nb->nb_yBefore = rp->rp_yBefore;

   nb->nb_expire = CURRENT_TIME +
                   (1.5 * ALLOWED_HELLO_LOSS * HELLO_INTERVAL);
 }
 Packet::free(p);
}
```

Gambar 4.17: Modifikasi Fungsi recvHello pada aodv.cc

Perhitungan *future* TWR dilakukan setelah melakukan perhitungan TWR. *Future* TWR memanfaatkan informasi yang sudah didapatkan sebelumnya dan prediksi nilai denganmenggunakan beberapa rumus fisika gerak lurus yang sudah dirancang pada bab sebelumnya. Kecepatan next-hop node yang akan datang, koordinat x dan y yang akan datang, masing-

masing dapat menggunakan persamaan 3.4, 3.5 dan 3.6. Implementasi dapat dilihat pada Gambar 4.20.

```
//pembuatan looping pada cache entry
 // Traverse neighbor list
 AODV_Neighbor *nb = nbhead.lh_first;

 for(; nb; nb = nb->nb_link.next) {
   //get speed, accel, coordinate
   // TWR  dan future TWR calculation
   //buat list dari calculated twr
  // . . . codes  ...
```

Gambar 4.18: *Looping* pada *Neighbor Cache Entry*

```
// radius antara node ini dengan node neighbor
// minimum radius -> min(V(i x - j x )^2 + (i y - j y )^2 ; r)
double radius = std::min(
  sqrt(pow((nb->nb_x - posX), 2) + pow((nb->nb_y - posY), 2)), (double) maxTxRange);

double quality = 1.0 / (1.0 - (radius / ((double) maxTxRange + 1.0)));

double modSpeed     = fSpeed * nb->nb_speed;
double modAccel     = fAccel * nb->nb_accel;
double modDistance  = fDistance * nb_distance;
double modQuality   = fQuality * quality;

// TWR = f s × |S n − S d | + f a × |A n − A d | + f q × Q
double TWR = modSpeed + modAccel  + modQuality;
```

Gambar 4.19: Perhitungan TWR

Untuk dapat memilih *relay node* yang baik, nilai TWR dan *future* TWR dapat digunakan seba gai penentunya. Pada tabel 2.1 telah disebutkan aturan pemilihan *relay node* berdasarkan TWR dan *future* TWR.

Untuk setiap *node* yang masuk ke dalam kriteria *relay*, alamatnya akan dimasukkan ke dalam *array* bertipe data nsaddr_t untuk disimpan sebagai daftar *node* yang terpilih sebagai *relay node set*. Ketiga tahapan ini merupakan bagian dari modifikasi fungsi sendRequest dan recvRequest. Implementasi dari sendRequest dan recvRequest masing-masing dapat dilihat pada lampiran A.2 dan lampiran A.3.

```
// Future speed v' = v + a × t
double nb_speedFuture = nb->nb_speed + (nb->nb_accel * timeModifier);
// Formula: x' = x + v0 t × 0.5 a t^2
// Future neighbor position
double nb_xFuture = nb->nb_x +
            (nb->nb_speed * timeModifier)
              + (0.5 * nb->nb_accel * timeModifier * timeModifier);
double nb_yFuture = nb->nb_y +
            (nb->nb_speed * timeModifier)
              + (0.5 * nb->nb_accel * timeModifier * timeModifier);
// Future this_node position
double iXFuture = posX +
            (iSpeed * timeModifier)
              + (0.5 * iAccel * timeModifier * timeModifier);
double iYFuture = posY +
            (iSpeed * timeModifier)
              + (0.5 * iAccel * timeModifier * timeModifier);
double futureQuality = 1.0 / (1.0 - (futureRadius / ((double) maxTxRange + 1.0)));

modSpeed     = fSpeed * nb_speedFuture;
modAccel     = fAccel * nb->nb_accel;
modQuality   = fQuality * futureQuality;
double futureTWR = modSpeed + modAccel  + modQuality;
```

Gambar 4.20: Perhitungan *Future* TWR

4.3.8 Modifikasi Proses Pengiriman RREQ

Proses pengiriman RREQ dilakukan setelah menjalani proses perhitungan TWR dan *future* TWR terlebih dahulu. Modifikasi dilakukan pada fungsi sendRequest() pada aodv.cc. *Pseudocode* modifikasi dapat dilihat pada gambar 4.21. Tahap akhir dari sendRequest adalah pembuatan paket RREQ dan*multicast* ke *relay node set*. Namun pada AODV, pengiriman paket dilakukan secara *broadcast*. Oleh karena itu perlu adanya modifikasi agar paket RREQ dikirim sebagai *multicast*. Perubahan dilakukan dengan menambahkan array rq_eligible_nodes untuk menyimpan *relay node set* dan nodes_list_len untuk menyimpan panjang *array* seperti pada Gambar 4.22. Kemudian ketika *neighbor node* menerima

broadcast paket RREQ, dilakukan pengecekan apakah *node* tersebut ada di dalam *array relay node set.*

Jika *node* tersebut ada di dalam *array relay node set*, maka lakukan proses selanjutnya. Namun jika *node* tersebut tidak ada di dalam *array relay node set*, maka *drop* paket RREQ tersebut. Penjelasan lebih lanjut mengenai modifikasi penerimaan paket RREQ akan dijelaskan pada subbab selanjutnya. Modifikasi dari sendRequest dapat dilihat pada lampiran A.2

```
Input: rreq
.
.
set treshold;
calculate TWR of neighbor nodes;
calculate future TWR of neighbor nodes;
classify TWR, stability, future TWR
according to treshold;
list ← relay node;
.
.
rreq_pckt ← new RREQ packet;
write RREQ fields;
          rreq_pckt_list_leng ← list.size();
          rreq_pckt_relay_node_set ← list;
broadcast rreq_pckt
```

Gambar 4.21:*Pseudocode*RREQyang TelahDimodifikasi

4.3.9 Modifikasi Proses Pengiriman RREP

Sama seperti proses penerimaan RREQ sebelumnya, proses penerimaan RREP juga perlu dimodifikasi agar penerimaan paket secara *broadcast* dapat memiliki perilaku yang mirip dengan penerimaan paket secara *multicast*. Pada kondisi normal, ketika paket RREQ diterima oleh suatu *node*, akan dilakukan

pengecekan kondisi. Seperti, apakah *node* tersebut merupakan *node*sumber, apakah*node*tersebut sebelumnya sudah pernah menerima paket RREQ tersebut, atau mengecek apakah *node* tersebut adalah *node* tujuan, dan lain-lain. Agarperilaku dari penerimaan paket RREQ yang di-*broadcast* mirip dengan *multicast*, maka perlu modifikasi penambahan kondisi baru dimana *node* yang tidak seharusnya menerima paket RREQ harus melakukan *packet drop*. Apabila *node* tersebut berhak menerima paket RREQ dan bukan merupakan *node* tujuan, maka lakukan proses perhitungan TWR hingga pembuatan *relay node set* yang baru dan teruskan paket tersebut. *Pseudocode* perubahan yang dilakukan di fungsi recvRequest padaAodv.cc dapat dilihat pada gambar 4.23. Keseluruhan modifikasi fungsi dapat dilihat pada lampiran A.3.

```
struct hdr_aodv_request {
        u_int8_t        rq_type;  // Packet Type
        u_int8_t        reserved[2];
        u_int8_t        rq_hop_count;   // Hop Count
        u_int32_t       rq_bcast_id;    // Broadcast ID

        nsaddr_t        rq_dst;         // Destination IP Address
        u_int32_t       rq_dst_seqno;   // Destination Sequence Number
        nsaddr_t        rq_src;         // Source IP Address
        u_int32_t       rq_src_seqno;   // Source Sequence Number

        double          rq_timestamp;   // when REQUEST sent;
          //modifikasi : penambahan
        nsaddr_t        *rq_eligible_nodes = NULL;
        u_int32_t       nodes_list_len;
```

Gambar 4.22: Modifikasi Paket RREQ pada aodv_packet.h

```
Input: rreq
.
.
index←this_node address;
if index is not in relay_node_set then
drop packet;
else
    if index is not destinationthen
        define treshold;
        calculate TWR of neighbor nodes;
        calculate future TWR neighbor nodes;
        classify TWR, stability, future TWR
        according to treshold;
        new_list ← new relay node set;
        rreq_relay_node_set ← new_list;
end
end
```

Gambar 4.23: *Pseudocode* Modifikasi Proses RREQ yang Diterima

4.4 Implementasi Metrik Analisis

Hasil dari simulasi skenario dalam NS-2 adalah sebuah *tracefile*. Analisa simulasi yang dilakukan memerlukan *tracefile*sebagai *output*yang diharapkan. Pada Tugas Akhir ini, terdapat tiga metrik yang akan menjadi parameter analisis, yaitu *packet delivery ratio*, *average end-to-end delay*, dan *routing overhead*.

4.4.1 Implementasi PDR

Packet delivery ratio didapatkan dengan cara menghitung semua proses pengiriman dan penerimaan paket datayang dikirim melalui agen pada *tracefile*. *File* skrip untuk mendapatkan informasi analisa PDR dapat dilihat pada lampiran A.4. Pada skrip tersebut dilakukan pencarian pada setiap baris yang mengandung string AGT karena event tersebut berhubungan dengan paket data. Paket yang dikirim dan diterima dibedakan dari kolom pertama pada baris yang telah dipilih. Setelah setiap baris dilakukan pencarian, simpan baris-baris yang dibutuhkan, selanjutnya dilakukan perhitungan PDR dengan persamaan 3.7. Untuk dapat menjalankan skrip pdr.awk dapat menggunakan perintah "awk –f pdr.awk tracefile.tr". Hasil keluaran PDR dapat berupa pesan seperti berikut, "Sent: 150 Recv: 125 Ratio: 0.8333".

4.4.2 Implementasi Average End-to-End Delay

Dalam pembacaan baris *tracefile* untuk perhitungan *average end-to-end delay* terdapat lima kolom yang harus diperhatikan, yaitu penanda eventpengiriman atau penerimaan, waktu terjadinya event, informasi layer komunikasi paket, ID paket, dan tipe paket. Untuk menghitung *delay* dari setiap paket kurangi waktu penerimaan dengan waktu pengiriman berdasar ID paket. Hasilpengurangan waktu dari masing-masing paket dijumlahkan dan dibagi dengan jumlah paket CBR yang ID-nya terlibat dalam perhitungan pengurangan waktu. Kode implementasi dari perhitungan average end-to-end delaydapat dilihat pada lampiran A.5. Perintah untuk menjalankan skrip delay.awk dapat menggunakan "awk –f delay.awk tracefile.tr". Contoh hasil keluarannya dapat berupa pesan sebagai berikut, "Average Delay: 0.4068".

4.4.3 Implementasi *Routing Overhead*

Routing overhead didapatkan dengan cara menyaring setiap baris yang mengandung *string* REQUEST, REPLY, dan ERROR. Kode implementasi dari perhitungan *routing overhead*dapat dilihat pada lampiran A.6.Contoh perintah untuk memanggil skrip AWK untuk menganalisis *trace file*dapat menggunakan perintah "awk –f ro.awk tracefile.tr". Hasil dari ro.awk dapat berupa pesan sebagai berikut, "Overhead : 1211".

4.5 Implementasi Simulasi pada NS-2

Untuk melakukan simulasi VANET pada NS-2, dibutuhkan sebuah *file* OTcl yang mendefinisikan lingkungan simulasi. *File*tersebut berisikan pengaturan untuk setiap *node* dan beberapa event yang perlu diatur agar berjalan pada waktu tertentu. Contoh potongan skrip pengaturan *node* dapat dilihat pada Gambar 4.24.

```
//Otcl file example
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 val(ifqlen) 50 ;# max packet in ifq
set val(rp) AODV ;# routing protocol
set val(nn) ${num_nodes}
set val(cbrsize) ${cbrsize}    ;# 512 Bytes
set val(cbrrate) ${cbrrate}
set val(cbrinterval) ${cbrinterval}  ;# 1 packet per second
set val(stop) ${end_time}
# Initialize ns
set ns_ [new Simulator]
# Set up topography object
set topo       [new Topography]
$topo load_flatgrid ${length_x} ${length_y}
$ns_ node-config \
      -adhocRouting $val(rp) -llType $val(ll) \
      -macType $val(mac) -ifqType $val(ifq) \
      -ifqLen $val(ifqlen) -antType $val(ant) \
      -propType $val(prop) -phyType $val(netif) \
      -channel  $channel_ -agentTrace ON \
      -routerTrace ON -macTrace OFF \
      -movementTrace OFF -topoInstance $topo
```

Gambar 4.24 Potongan Skrip Pengaturan N*ode*

Pengaturan yang dilakukan pada *file* tersebut antara lain pengaturan lokasi penyimpanan dan nama*tracefile*, lokasi *file* mobilitas *node*, konfigurasi *node* sumber dan *node* tujuan, dan konfigurasi event pengiriman paket data. Kode implementasi skenario yang digunakan pada Tugas Akhir ini dapat dilihat pada lampiran A.1.Penjelasan dari pengaturan skrip Otcl dapat dilihat pada Tabel 4.1.

Tabel 4.1: Penjelasan dari Setiap Parameter pada Skrip Pengaturan *Node*

Parameter	Value	Keterangan
llType	LL	Menggunakan *link layer* standar
macType	Mac/802_11	Menggunakan MAC 802.11 karena *wireless*
ifqType	Queue/DropTail/PriQueue	Menggunakan *priority queue* sebagai antrian paket dan paket yang dihapus saat antrian penuh adalah paket yang baru
ifqLen	50	Jumlah maksimum paket pada *queue*
antType	Antena/Omni Antena	Jenis antena yang digunakan adalah Omni antena
propType	Propagation/ TwoRayGround	Tipe propagasi sinyal wireless adalah *two ray ground*
phyType	Phy/Wireless Phy	Komunikasi menggunakan nirkabel
topoInstance	$topo	Topologi yang digunakan skenario
agentTrace	ON	Pencatatan aktifitas dari agen *routing protocol* bernilai aktif
routerTraace	ON	Pencatatan pada aktifitas *routing protocol* bernilai aktif
macTrace	OFF	Matikan *trace* MAC *layer* pada *tracefile*
movementTrace	OFF	Matikan pencatatan pergerakan *node*
channel	Channel/Wireless channel	*Channel* komunikasi yang digunakan

Hasil dari simulasi adalah *tracefile* dengan format .tr. Skenario simulasi dijalankan dengan perintah “ns scenario.tcl”. Isi *tracefile* merupakan catatan seluruh *event* dari setiap paket yang tersebar pada lingkungan simulasi.

(Halaman sengaja dikosongkan)

BAB V
UJI COBA DAN EVALUASI

Bab ini berisikan hasil uji coba dan evaluasi dari simulasi skenario NS-2 yang telah dilakukan.

4.6 Lingkungan Uji Coba

Spesifikasi perangkat keras yang digunakan pada Tugas Akhir ini adalah sebagai berikut:

Tabel 4.2: Spesifikasi Perangkat Keras

Komponen	**Spesifikasi**
CPU	Intel Core i3-2330M Processor (2.20 GHz, 3M Cache) Dual Core
Sistem Operasi	Linux Ubuntu 14.04 LTS 64-bit
RAM	6 GB
Penyimpanan	1 TB

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

- SUMO versi 0.25.0 untuk pembuatan skenario mobilitas VANET.
- JOSM versi 10301 untuk penyuntingan peta OpenStreetMap.
- ns2.35 untuk simulasi skenario VANET.

Sedangkan parameter lingkungan uji coba yang digunakan pada NS-2 dapat dilihat pada Tabel 5.2.

Tabel 4.3: Parameter simulasi NS-2

No.	Parameter	Value/Keterangan
1.	Network Simulator	NS-2.35
2.	Routing Protocol	AODV dan AODV-Penambahan Faktor Pergerakan
3.	Waktu Simulasi	360 detik
4.	Area Simulasi	Grid 1000m x 1000m sedangkan Riil 1800m x 1800m
5.	Jumlah Kendaraan	50,75,100
6.	Radius Transmisi	250 m
7.	Kecepatan Maksimum	10 m/s, 15 m/s,20 m/s
8.	Agen	Constant Bit Rate (CBR)
9.	Source & Destination position	Stationary (tetap)
10.	Ukuran Paket	512Bytes
11.	Packet Rate	2KBps
12.	Packet Interval	1 pckt/s
13.	Protokol MAC	IEEE 802.11p
14.	Model Propagasi	*Two-way ground*
15.	Parameter AODV-modifikasi	fs = 15(faktor pengali kecepatan), fa = 10(faktor pengali akselerasi), fq = 50(faktor pengali kualitas), W = 100(treshold)

4.7 Hasil Uji Coba

Hasil uji coba skenario *grid* dan skenario riil dapat dilihat sebagai berikut

4.7.1 Hasil Uji Coba Skenario *Grid*

Uji coba skenario *grid* dilakukan sebanyak masing-masing 5 kali dengan skenario mobilitas *random* pada peta *grid* dengan luas 1000m x 1000m dengan variasi jumlah *node* dan kecepatan sebanyak 9 variasi yang dibentuk dari 50 *nodes*,100 *nodes*,75 *nodes* dan 10 m/s, 15 m/s,20 m/s.

Hasil analisa dengan metrik PDR, *average delay* dan *routing overhead* dapat dilihat pada tabel 5.3,5.4 dan 5.5. Tampilan grafik hasil analisa dapat dilihat pada gambar 5.1,5.2,5.3.

Tabel 4.4: Hasil Rata-rata PDR pada Skenario *Grid*

Jumlah Node	Kecepatan Maks. (m/s)	AODV	Modifikasi-AODV	Margin
50	10	0,856	0.852	-0.004
	15	0.850	0.877	+0.027
	20	0.838	0.887	+0.049
75	10	0.855	0.824	-0.031
	15	0.89	0.899	+0.009
	20	0.894	0.858	-0.036
100	10	0.832	0.843	+0.011
	15	0.894	0.842	-0.052
	20	0.876	0.86	-0.016

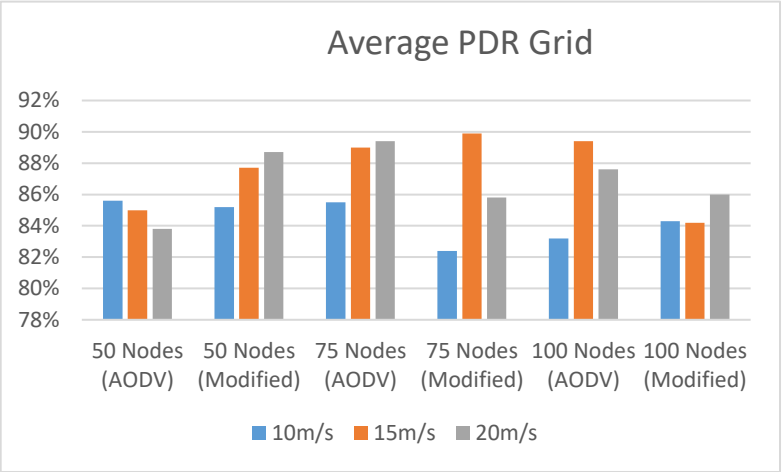
Pada skenario *grid* dapat dilihat bahwa margin rata-rata PDR tidak terlalu memberikan dampak yang signifikan, sedangkan pada *end to end delay* memberikan dampak yang cukup bervariasi, namun pada kasus skenario 100 *node* terjadi peningkatan *delay* sekitar 40-50 persen. Untuk *routing overhead*, juga banyak sekali variasi yang terjadi, namun secara umum, terdapat peningkatan jumlah nilai *routing overhead*.

Tabel 4.5: Hasil Rata-rata *End-to-End Delay* pada Skenario *Grid*

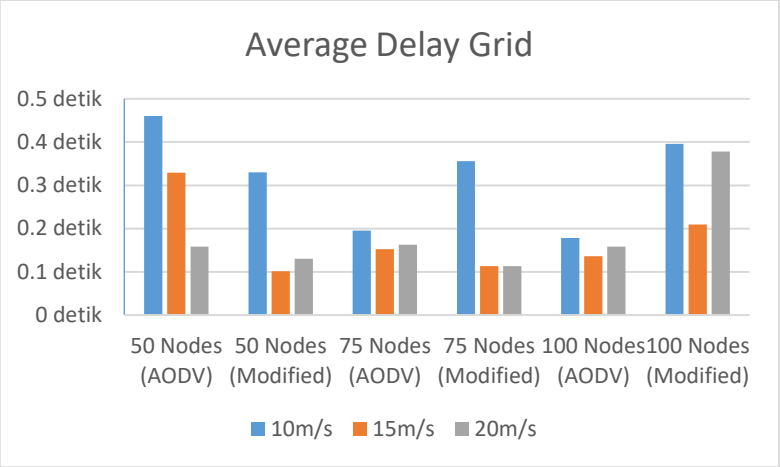
Jumlah Node	Kecepatan Maks. (m/s)	AODV	Modifikasi-AODV	Margin
50	10	0.460	0.33	-0.13
	15	0.329	0.101	-0.228
	20	0.158	0.130	-0.028
75	10	0.195	0.356	+0.361
	15	0.152	0.113	-0.039
	20	0.163	0.113	-0.05
100	10	0.178	0.396	+0.218
	15	0.136	0.209	+0.073
	20	0.158	0.378	+0.22

Tabel 4.6: Hasil Rata-rata *Routing Overhead* pada Skenario *Grid*

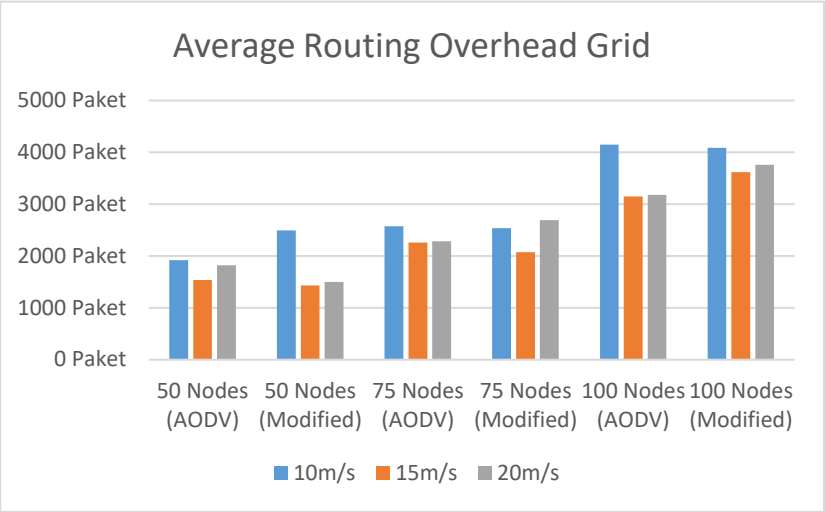
Jumlah Node	Kecepatan Maks. (m/s)	AODV	Modifikasi-AODV	Margin
50	10	1920	2497.2	+577.2
	15	1534.8	1430.2	-104.6
	20	1820.6	1496.8	-323.8
75	10	2574.8	2539.8	-35
	15	2258.4	2074.4	-184
	20	2282.2	2691.8	+409.6
100	10	4151.8	4085.8	-66
	15	3148.4	3617.6	+469.2
	20	3181.6	3760.8	+579.2



Gambar 4.25: Grafik Rata-rata Analisa PDR pada Skenario *Grid*



Gambar 4.26: Grafik Rata-rata Analisa *End-to-End Delay* pada Skenario *Grid*



Gambar 4.27: Grafik Rata-rata Analisa R*outing Overhead* pada Skenario *Grid*

4.7.2 Hasil Uji Coba Skenario Riil

Uji coba skenario riil dilakukan sebanyak 15 kali dengan skenario mobilitas *random* pada peta daerah surabaya dengan luas 1800 m x 1800 m dengan variasi jumlah node sebanyak 3 variasi yang dibentuk dari 50 *nodes*,100 *nodes*,75 *nodes*.

Hasil analisa dengan metrik PDR, *average delay* dan *routing overhead* dapat dilihat pada tabel 5.6,5.7 dan 5.8.

Tabel 4.7: Hasil Rata-rata PDR pada Skenario Riil

Jumlah Node	AODV	Modifikasi-AODV	Margin
50	0.32	0.286	-0.034
75	0.45	0.54	+0.09
100	0.56	0.68	+0.12

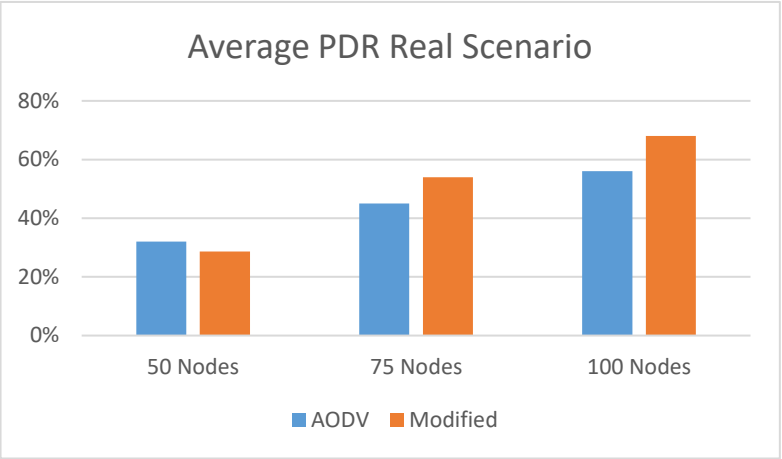
Tabel 4.8: Hasil Rata-rata *End-to-End Delay* pada Skenario Riil

Jumlah Node	AODV	Modifikasi-AODV	Margin
50	4.71	3.418	-1.292
75	2.877	2.832	-0.045
100	1.607	1.13	-0.477

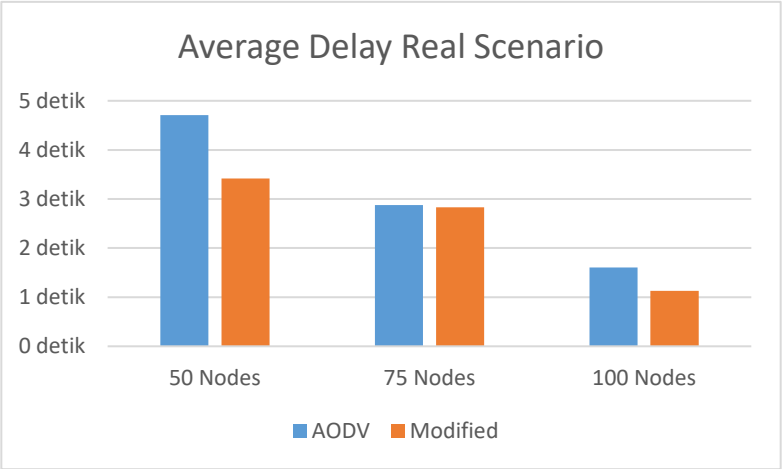
Tabel 4.9: Hasil Rata-rata *Routing Overhead* pada Skenario Riil

Jumlah Node	AODV	Modifikasi-AODV	Margin
50	2155.2	1776.8	-378.4
75	4504.6	4043.4	-461.2
100	5290.8	6777.2	1486.4

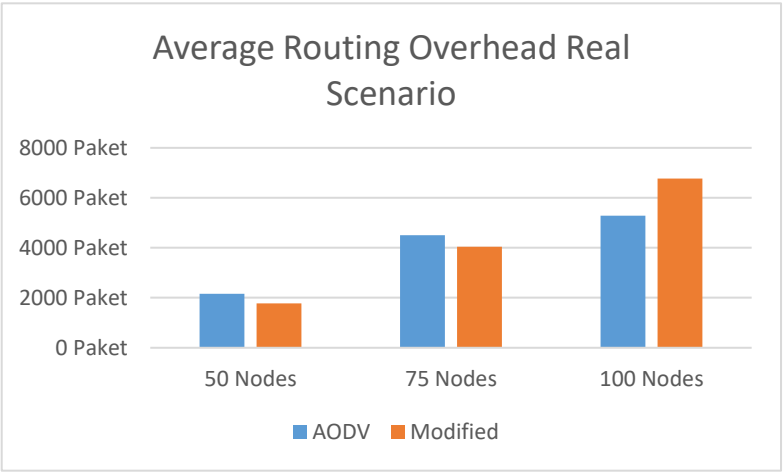
Pada skenario riil dapat dilihat bahwa margin rata-rata PDR cukup memberikan dampak yang baik, bisa dilihat pada jumlah *node* 75, menaikkan nilai PDR sebesar 9% dan pada *node* 100 naik sebesar 12%. Berbeda dengan skenario *grid*, pada skenario riil diperlihatkan bahwa terjadi penurunan *delay*sebanyak 27% pada 50 *node*, 1.5% pada 75 *node* dan 29% pada 100 *node.* Meski begitu, *delay* pada skenario riillebih tinggi dibandingkan dngan skenario *grid.* Sedangkan untuk jumlah RO marginnya bervariasi, namun secara umum pada skenario riil semakin besar node yang ada pada simulasi, semakin besar RO yang didapatkan. Grafik analisa pada skenario riil dapat dilihat pada gambar 5.4,5.5 dan 5.6.



Gambar 4.28: Grafik Rata-rata Analisa PDR pada Skenario Riil



Gambar 4.29: Grafik Rata-rata Analisa *End-to-End Delay* pada Skenario Riil



Gambar 4.30: Grafik Rata-rata Analisa R*outing Overhead* pada Skenario Riil

(Halaman ini sengaja dikosongkan)

BAB VI
Penutup

Bab ini berisikan kesimpulan yang dapat diambil dari hasil uji coba yang telah dilakukan. Selain kesimpulan, terdapat juga saran yang ditujukan untuk pengembangan mekanisme yang diajukan ini nantinya.

5.1 Kesimpulan

Kesimpulan yang diperoleh berdasarkan uji coba dan evaluasi yang dilakukan antara lain:

1) Pada rancangan skenario *grid* modifikasi dengan menggunakan faktor kecepatan tidak terlalu memberikan efek yang signifikan terutama pada kasus dengan jumlah *node* antara 50 hingga 100 dengan variasi kecepatan antara 10 hingga 20 m/s.
2) Pada rancangan skenario riil modifikasi dengan menggunkan faktor kecepatan cukup memberikan dampak peningkatan sebesar 9% pada skenario yang *node*nya berjumlah 75 dan peningkatan sebesar 12% pada skenario yang *node*nya berjumlah 100.
3) Hasil perancangan skenario riil lebih baik dibandingkan dengan hasil skenario *grid* meskipun menggunakan *routing protocol* yang sama.

5.2 Saran

Saran yang diberikan terkait pengembangan pada Tugas Akhir ini adalah:

1. Menambah jumlah *node* dan variasi kecepatan maksimum.
2. Diperlukan studi lebih lanjut mengenai pemilihan *treshold* W serta koefisien faktor pengali yang digunakan pada rumus TWR.
3. Diperlukan penanganan sistem lalu lintas yang lebih baik. Pada Tugas Akhir ini sistem lalu lintas ditangani oleh *simulator* bukan berdasarkan desain yang telah disiapkan.
4. Perbanyak jumlah skenario tiap variasi, pada Tugas Akhir ini jumlah skenario tiap variasi berjumlah 5 skenario.

DAFTAR PUSTAKA

[1] C. Perkins, E. Belding-Royer and S. Das, "Ad Hoc On-Demand Distance Vector (AODV) Routing," RFC 3561, July 2003. [Online]. Available: https://www.rfc-editor.org/rfc/rfc3561.txt. [Accessed 10 05 2017].

[2] X. Shen, Y. Wu, Z. Xu and X. Lind., "AODV-PNT: An Improved Version of AODV Routing Protocol with Predicting Node Trend in VANET," pp. 91-97, November 2014.

[3] C. Sommer and F. Dressler, Vehicular Networking, Cambridge University Press, 2015.

[4] D. Krajzewicz, J. Erdmann, M. Behrisch dan L. Bieker, "Recent Development and Applications of SUMO - Simulation of Urban Mobility," *International Journal On Advances in Systems and Measurements*, no. 5, pp. 128-138, December 2012.

[5] "OpenStreetMap," OpenStreetMap Foundation, [Online]. Available: https://www.openstreetmap.org. [Diakses 01 05 2017].

[6] D. Wetherall, "OTcl," 1997. [Online]. Available: http://otcl-tclcl.sourceforge.net/otcl/. [Diakses 01 05 2017].

(Halaman ini sengaja dikosongkan)

LAMPIRAN

A.1 Kode Skenario.tcl

```
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 val(ifqlen) 50 ;# max packet in ifq
set val(rp) AODV ;# routing protocol
set val(nn) 50
set val(cbrsize) 512    ;# 512 Bytes
set val(cbrrate) 2KB
set val(cbrinterval) 1  ;# 1 packet per second
set val(stop) 360
set val(mobilityfile) "mobility.tcl"
set val(activityfile) "activity.tcl"

# Initialize ns
set ns [new Simulator]

#Fixing the co-ordinate of simutaion area
set val(x) 1000
set val(y) 1000

set tracefd [open traceout-50-10.tr w]
$ns trace-all $tracefd

# Set up topography object
set topo       [new Topography]
$topo load_flatgrid $val(x) $val(y)
```

```
create-god [expr $val(nn) + 2]
set channel_ [new $val(chan)]
$ns node-config \
      -adhocRouting $val(rp) \
      -llType $val(ll) \
      -macType $val(mac) \
      -ifqType $val(ifq) \
      -ifqLen $val(ifqlen) \
      -antType $val(ant) \
      -propType $val(prop) \
      -phyType $val(netif) \
      -channel  $channel_ -agentTrace ON \
      -routerTrace ON  -macTrace OFF \
      -movementTrace OFF -topoInstance $topo

for {set i 0} {$i < $val(nn)} {incr i} {
set node_($i) [$ns node]
        $node_($i) random-motion 0
}

puts "Loading mobility file..."
set where [file dirname [info script]]
source [file join $where $val(mobilityfile)]
source [file join $where $val(activityfile)]

# Stationary nodes
set sender [$ns node]
$sender set X_ 198.05
$sender set Y_ 809.45
$sender set Z_ 0.0

set receiver [$ns node]
$receiver set X_ 801.29
$receiver set Y_ 189.51
$receiver set Z_ 0.0
```

```
#Setup a UDP connection
set udp [new Agent/UDP]
$ns attach-agent $sender $udp
set null [new Agent/Null]
$ns attach-agent $receiver $null
$ns connect $udp $null

#Setup a CBR over UDP connection
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set type_ CBR
$cbr set packet_size_ $val(cbrsize)
$cbr set rate_ $val(cbrrate)
$cbr set interval_ $val(cbrinterval)

#Schedule events for the CBR and FTP agents
$ns at 150.0 "$cbr start"
$ns at 300.0 "$cbr stop"

# Tell nodes when the simulation ends
for {set i 0} {$i < $val(nn) } {incr i} {
    $ns at [expr $val(stop) +1.0] "$node_($i)
reset";
}

# Tell static node when simulation ends
$ns at [expr $val(stop) +1.0] "$sender reset"
$ns at [expr $val(stop) +1.0] "$receiver reset"

# What to do when scenario finished
$ns at [expr $val(stop) +1.0] "finish"
$ns at [expr $val(stop) +2.0] "puts \"Exiting
NS2...\"; $ns halt"

proc finish {} {
global ns tracefd
  $ns flush-trace
close $tracefd
}
```

```
puts "Starting simulation..."
$ns run
```

A.2 Kode Implementasi sendRequest

```
void AODV::sendHello() {
Packet *p = Packet::alloc();
struct hdr_cmn *ch = HDR_CMN(p);
struct hdr_ip *ih = HDR_IP(p);
struct hdr_aodv_reply *rh = HDR_AODV_REPLY(p);

MobileNode *iNode;
iNode          =          (MobileNode          *)
(Node::get_node_by_address(index));
double iSpeed = ((MobileNode *) iNode)->speed();
double    now    =    ((MobileNode    *)    iNode)-
>getUpdateTime();
double posX = iNode->X(); //x coordinat
double posY = iNode->Y(); //y coordinat

rh->rp_type = AODVTYPE_HELLO;
 //rh->rp_flags = 0x00;
rh->rp_hop_count = 1;
rh->rp_dst = index;
rh->rp_dst_seqno = seqno;
rh->rp_lifetime  =  (1  +  ALLOWED_HELLO_LOSS)  *
HELLO_INTERVAL;

 // embed speed and accel info to hello packet

if  (now  -  lastUpdateTime  ==  0)  {  //if  not
updated
rh->rp_accel = lastAccel;
 }
else {
rh->rp_accel  =  iSpeed  /  (now  -  lastUpdateTime);
// a = delta v/delta t
lastAccel    = rh->rp_accel;
lastSpeed    = iSpeed;
```

```
rh->rp_speed = iSpeed;
lastUpdateTime = now;
rh->rp_x     = posX;
rh->rp_y     = posY;
rh->rp_xBefore = lastX;
rh->rp_yBefore = lastY;

 // ch->uid() = 0;
ch->ptype() = PT_AODV;
ch->size() = IP_HDR_LEN + rh->size();
ch->iface() = -2;
ch->error() = 0;
ch->addr_type() = NS_AF_NONE;
ch->prev_hop_ = index;          // AODV hack

ih->saddr() = index;
ih->daddr() = IP_BROADCAST;
ih->sport() = RT_PORT;
ih->dport() = RT_PORT;
ih->ttl_ = 1;

 Scheduler::instance().schedule(target_,       p,
0.0);

 // Update its latest position
lastX          = posX;
lastY          = posY;

}
```

A.3 Kode Implementasi recvRequest

```
void AODV::recvRequest(Packet *p) {
// struct hdr_cmn *ch = HDR_CMN(p);
struct hdr_ip *ih = HDR_IP(p);
struct        hdr_aodv_request        *rq        =
HDR_AODV_REQUEST(p);
aodv_rt_entry *rt;
bool isEligible = true;

  /*
* Drop jika:
   *      - adalah source
   *      - mendapatkan request ini sebelumnya
   *      - tdk dalam list relay node
   */

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

    Packet::free(p);
return;
  }


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

#ifdef DEBUG
fprintf(stderr,   "%s:   discarding   request\n",
__FUNCTION__);
#endif // DEBUG

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

```
}
 // jika list null drop packet
if(rq->rq_eligible_nodes == NULL) {

   Packet::free(p);
return;
 }
else if (rq->rq_dst != index) {
isEligible = false;
for (u_int32_t i = 0; i < rq->nodes_list_len;
++i) {
     // ada dalam list, terima
if (rq->rq_eligible_nodes[i] == index) {
isEligible = true;
break;
     }
   }
 }

// jika tidak didalam list.drop packet
if (!isEligible) {
  Packet::free(p);
return;
}
// Jika di dalam list, re-compute untuk membuat
list baru
else if (rq->rq_dst != index) {
  /*
*  re-compute TWR
   */

 // TWR list
 std::vector<TWRContainer> listTWR;
 std::vector<nsaddr_t> eligibleAddrList;
   // same seq no., update the rt entry. Else
```

```
nsaddr_t *eligibleNodes;

 // Get current position, speed, time
 MobileNode *iNode;
iNode          =          (MobileNode          *)
(Node::get_node_by_address(index));
double iSpeed = ((MobileNode *) iNode)->speed();
double    now    =    ((MobileNode    *)    iNode)-
>getUpdateTime();

double iAccel;
if (now - lastUpdateTime == 0) {
iAccel = lastAccel;
 }
else {
iAccel = iSpeed / (now - lastUpdateTime);
lastAccel = iAccel;
 }

double posX = iNode->X();
double posY = iNode->Y();

u_int32_t fSpeed = F_SPEED;
u_int32_t fAccel = F_ACCEL;
u_int32_t fQuality = F_QUALITY;
u_int32_t timeModifier = TIME_MOD;
u_int32_t maxTxRange = TX_RANGE; // ns2 default
range (based on Pt_)
u_int32_t thresholdW = THRES_W;


double xDst = DST_X;
double yDst = DST_Y;
```

```
 // Traverse neighbor list
 AODV_Neighbor *nb = nbhead.lh_first;

for(; nb; nb = nb->nb_link.next) {
   //get speed, accel, coordinate
   // TWR  and future TWR calculation
double radius = std::min(
sqrt(pow((nb->nb_x - posX), 2) + pow((nb->nb_y -
posY), 2)), (double) maxTxRange);

double quality = 1.0 / (1.0 - (radius /
((double) maxTxRange + 1.0)));

double modSpeed     = fSpeed * nb->nb_speed;
double modAccel     = fAccel * nb->nb_accel;
double modQuality   = fQuality * quality;

   // TWR = f s × |S n − S d | + f a × |A n − A
d | + f q × Q
double TWR = modSpeed + modAccel + modQuality;

   // Calculate future TWR dalam [timeModifier]
seconds

   // Future speed v' = v + a × t
double nb_speedFuture = nb->nb_speed + (nb-
>nb_accel * timeModifier);
```

```
    // Formula: x' = x + v0 t × 0.5 a t^2
   // Future neighbor position
double nb_xFuture = nb->nb_x +
               (nb->nb_speed * timeModifier)
                 +   (0.5   *   nb->nb_accel   *
timeModifier * timeModifier);
double nb_yFuture = nb->nb_y +
               (nb->nb_speed * timeModifier)
                 +   (0.5   *   nb->nb_accel   *
timeModifier * timeModifier);
   // Future this_node position
double iXFuture = posX +
               (iSpeed * timeModifier)
                 + (0.5 * iAccel * timeModifier
* timeModifier);
double iYFuture = posY +
               (iSpeed * timeModifier)
                 + (0.5 * iAccel * timeModifier
* timeModifier);
double   futureQuality   =   1.0   /   (1.0   -
(futureRadius / ((double) maxTxRange + 1.0)));

modSpeed     = fSpeed * nb_speedFuture;
modAccel     = fAccel * nb->nb_accel;
double  futureTWR  =  modSpeed  +  modAccel   +
modQuality;


double futureDistance;
futureDistance  =  sqrt(pow((nb_xFuture  -  xDst),
2) + pow((nb_yFuture - yDst), 2));

   //  Future  radius  antara  node  ini  dengan
neighbord node dengan future result
double futureRadius = std::min(
```

```
sqrt(pow((nb_xFuture - iXFuture), 2) +
pow((nb_yFuture - iYFuture), 2)), (double)
maxTxRange);

double futureQuality = 1.0 / (1.0 -
(futureRadius / ((double) maxTxRange + 1.0)));

   // Calculate future TWR

   // Store TWR futureTWR in a list
   TWRContainer container;
   container.TWR = TWR;
   container.futureTWR = futureTWR;
   container.address = nb->nb_addr;
   container.isOptimal = false;
   container.isStable = false;
   container.isFutureBetter = false;

   listTWR.push_back(container);
 }
 //  Relay  decision  making  based  on  TWR  and
futureTWR using some rules
 // ascending sort listTWR based on TWR value
 std::sort(listTWR.begin(),        listTWR.end(),
minTWR());

 // minimum TWR in the list
listTWR[0].isOptimal = true;

for (u_int32_t i = 0; i < listTWR.size(); i++) {

if (listTWR[i].futureTWR < listTWR[i].TWR) {
listTWR[i].isFutureBetter = true;
   }
```

```
double deltaTWR = fabs(listTWR[i].TWR -
listTWR[i].futureTWR);
if (deltaTWR < thresholdW) {
listTWR[i].isStable = true;
   }

   // Optimal, unstable, better -> Relay
if (listTWR[i].isOptimal && !listTWR[i].isStable
&& listTWR[i].isFutureBetter) {

eligibleAddrList.push_back(listTWR[i].address);
   }
   // Optimal, stable, not better -> Relay
else      if      (listTWR[i].isOptimal      &&
listTWR[i].isStable                          &&
!listTWR[i].isFutureBetter) {

eligibleAddrList.push_back(listTWR[i].address);
   }
   // Suboptimal, unstable, better -> Relay
else      if      (!listTWR[i].isOptimal      &&
!listTWR[i].isStable                         &&
listTWR[i].isFutureBetter) {

eligibleAddrList.push_back(listTWR[i].address);
   }
   // Suboptimal, stable, not better -> Relay
else      if      (!listTWR[i].isOptimal      &&
listTWR[i].isStable                          &&
!listTWR[i].isFutureBetter) {

eligibleAddrList.push_back(listTWR[i].address);
   }
 }

 // replace rq_eligible_nodes with the re-
computed list
rq->nodes_list_len = eligibleAddrList.size();
```

```
eligibleNodes      =      new      nsaddr_t[rq-
>nodes_list_len];
for (u_int32_t i = 0; i < rq->nodes_list_len;
i++) {
eligibleNodes[i] = eligibleAddrList[i];
 }
rq->rq_eligible_nodes = eligibleNodes;

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


aodv_rt_entry *rt0; // rt0 is the reverse route

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

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

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

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

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

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

 // First check if I am the destination ..

if(rq->rq_dst == index) {

#ifdef DEBUG
fprintf(stderr,  "%d  -  %s:  destination  sending
reply\n",
```

```
index, __FUNCTION__);
#endif // DEBUG

seqno = max(seqno, rq->rq_dst_seqno)+1;
if (seqno%2) seqno++;

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

delete[] rq->rq_eligible_nodes;
rq->rq_eligible_nodes = NULL;
   Packet::free(p);
 }

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


assert(rq->rq_dst == rt->rt_dst);
sendReply(rq->rq_src,
rt->rt_hops + 1,
rq->rq_dst,
rt->rt_seqno,
           (u_int32_t)      (rt->rt_expire      -
CURRENT_TIME),
           //                     rt->rt_expire  -
CURRENT_TIME,
rq->rq_timestamp);
   // Insert nexthops to RREQ source and RREQ
destination in the
   // precursor lists of destination and source
```

```
respectively
rt->pc_insert(rt0->rt_nexthop);  //  nexthop  to
RREQ source
   rt0->pc_insert(rt->rt_nexthop); // nexthop to
RREQ destination

#ifdef RREQ_GRAT_RREP

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

// TODO: send grat RREP to dst if G flag set in
RREQ using rq->rq_src_seqno, rq->rq_hop_counT

// DONE: Included gratuitous replies to be sent
as per IETF aodv draft specification. As of now,
G  flag  has  not  been  dynamically  used  and  is
always  set  or  reset  in  aodv-packet.h  ---  Anant
Utgikar, 09/16/02.

      Packet::free(p);
 }
 /*
  * Can't reply. So forward the  Route Request
  */
else {

ih->saddr() = index;
ih->daddr() = IP_BROADCAST;
rq->rq_hop_count += 1;
   // Maximum sequence number seen en route
```

```
if (rt) rq->rq_dst_seqno = max(rt->rt_seqno, rq-
>rq_dst_seqno);
forward((aodv_rt_entry*) 0, p, DELAY);
 }
}
```

A.4 Kode awk PDR

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

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

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

 END {
printf "Sent: %d Recv: %d Ratio: %.4f\n",
sendLine, recvLine, (recvLine/sendLine);
 }
```

A.5 Kode awk RO

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

 $0 ~/^s.* \(REQUEST\)/ {
sendLine ++ ;
 }

 $0 ~/^s.* \(REPLY\)/ {
recvLine ++ ;
 }

 $0 ~/^s.* \(ERROR\)/ {
errLine ++ ;
 }

 END {
printf "Overhead: %d\n", (sendLine + recvLine +
errLine);
 }
```

A.6 Kode awk *End to End Delay*

```
BEGIN {
  highest_packet_id = 0;
 }
{
  # Using old trace file format
action = $1;
time = $2;
layer = $4;
  packet_id = $6;
  # Check for the latest packet id
if ( packet_id > highest_packet_id ) {
    highest_packet_id = packet_id;
  }
  # If start_time of packet_id is empty, then
get read time
if ( start_time[packet_id] == 0 ) {
    start_time[packet_id] = time;
  }
if ( $7 == "cbr" ) {
    # If packet is not dropped
if ( action != "D" ) {
      # If packet is received
if ( action == "r" ) {
        # Get received time
        end_time[packet_id] = time;
        }
      }
    # If packet is dropped
else {
      # There is no "end time"
      end_time[packet_id] = -1;
        }
  }
 }
```

```
 END {
    sigma_duration = 0;
count = 0;
for    (    packet_id    =    0;    packet_id    <=
highest_packet_id; packet_id++ )
      {
type = packet_type[packet_id];
start = start_time[packet_id];
end = end_time[packet_id];
        packet_duration = end - start;RBTC

if ( start < end ) {
          sigma_duration += packet_duration;
count++;
          }
      }

if ( count == 0 ) {
printf("no_packet_counted\n");
    }
else {
printf("Average Delay: %.4f\n", sigma_duration /
count);
    }
 }
```

BIODATA PENULIS

Muhamad Vijay Fathur Rahman. Merupakan anak pertama dari pasangan Bapak Jumaindengan Ibu Tasmiatun. Lahir di Bojonegoro tanggal 17 Desember 1994. Penulis menempuh pendidikan formal dimulai dari TK Al-Muttaqien (1999-2000), SD Al-Muttaqien (2000-2006), SMP Negeri 9 Surabaya (2006-2009), SMA Negeri 1 Surabaya (2009-2012), dan S1 Teknik Informatika ITS (2012-2017). Bidang studi yang diambil oleh penulis pada saat berkuliah di Teknik Informatika ITS adalah Arsitektur dan Jaringan Komputer (AJK). Penulis aktif dalam mengikuti lomba dan menjadi finalis di berbagai lomba seperti Gemastik 7 dalam kategori keamanan jaringan (2014), dan Gemastik 8 dalam kategori keamanan jaringan (2015). Penulis memiliki hobi membaca, pengembangan game, dan menyukai hal-hal baru yang berhubungan dengan sesuatu yang bersifat ilmiah. Penulis dapat dihubungi melalui alamat email: vijay.fathur@gmail.com.