TUGAS AKHIR - KI141502

IMPLEMENTASI *VELOCITY AWARE* AOMDV MENGGUNAKAN NS-2 PADA LINGKUNGAN VANET

I PUTU PRADNYANA
NRP 5112100128

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

Dosen Pembimbing II
Ir. F.X. Arunanto, M.Sc.

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2016

TUGAS AKHIR - KI141502

IMPLEMENTASI *VELOCITY AWARE* AOMDV MENGGUNAKAN NS-2 PADA LINGKUNGAN VANET

I PUTU PRADNYANA
NRP 5112100128

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

Dosen Pembimbing II
Ir. F.X. Arunanto, M.Sc.

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2016

Halaman ini sengaja dikosongkan

UNDERGRADUATE THESIS - KI141502

IMPLEMENTATION OF VELOCITY AWARE AOMDV USING NS-2 IN VANET ENVIRONMENT

I PUTU PRADNYANA
NRP 5112100128

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

Supervisor II
Ir. F.X. Arunanto, M.Sc.

DEPARTMENT OF INFORMATICS
Faculty of Information Technology
Institut Teknologi Sepuluh Nopember
Surabaya, 2016

Halaman ini sengaja dikosongkan

IMPLEMENTASI *VELOCITY AWARE* AOMDV MENGGUNAKAN NS-2 PADA LINGKUNGAN VANET

TUGAS AKHIR

Diajukan Guna Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Arsitektur dan Jaringan Komputer
Program Studi S1 Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember

Oleh :
I Putu Pradnyana
NRP: 5112100128

Disetujui oleh Dosen Pembimbing Tugas Akhir :

Dr. Radityo Anggoro , S.Kom., M.Sc. .....
NIP: 19841016 200812 1 002 (Pembimbing 1)

Ir. F.X. Arunanto, M.Sc. .....
NIP: 19570101 198303 1 004 (Pembimbing 2)

SURABAYA
Mei 2016

Halaman ini sengaja dikosongkan

IMPLEMENTASI *VELOCITY AWARE* AOMDV MENGGUNAKAN NS-2 PADA LINGKUNGAN VANET

Nama	: I PUTU PRADNYANA
NRP	: 5112100128
Jurusan	: Teknik Informatika FTIf
Pembimbing I	: Dr. Radityo Anggoro , S.Kom.,M.Sc.
Pembimbing II	: Ir. F.X. Arunanto, M.Sc.

Abstrak

Topologi jaringan pada VANET berubah dengan sangat cepat sehingga *routing protocol* yang digunakan harus sangat adaptif terhadap perubahan. AOMDV adalah sebuah varian dari AODV yang memberikan lebih dari satu jalur dari sumber ke tujuan. Tetapi dalam kondisi topologi yang berubah dengan cepat pada VANET implementasi AOMDV harus lebih dioptimasi. Salah satu variabel yang dapat digunakan untuk mengoptimasi kinerja AOMDV adalah kecepetan kendaraan.

Dalam buku tugas akhir ini penulis menjelasan implementasi dari HM-AOMDV *(High Mobility Ad hoc On-Demand Multipath Distance Vector)* dalam NS-2. HM-AOMDV adalah implementasi AOMDV yang menggunakan kecepatan kendaraan sebagai variabel optimasi. HM-AOMDV menggunakan jumlah *hop* dan kecepatan relatif dari tiap kendaraan untuk memilih rute terbaik untuk mengirim data.

Dari uji coba yang dilakukan, HM-AOMDV memberikan rata - rata PDR *(Packet Delivery Ratio)* hingga lebih baik 8,5% hingga 10% dibandingkan AOMDV.

Kata-Kunci: VANET, AOMDV, NS-2, HM-AOMDV.

Halaman ini sengaja dikosongkan

IMPLEMENTATION OF VELOCITY AWARE AOMDV USING NS-2 IN VANET ENVIRONMENT

Name	: I PUTU PRADNYANA
NRP	: 5112100128
Major	: Informatics FTIf
Supervisor I	: Dr. Radityo Anggoro , S.Kom.,M.Sc.
Supervisor II	: Ir. F.X. Arunanto, M.Sc.

Abstract

Topology in VANET is changing very fast making the routing protocol used had to be very adaptive to changes. AOMDV is a variant of AODV that implement multiple path from source to destination. But in the quickly ever changing topology of VANET the AOMDV implementatin have to be optimized. One variable that can be used to optimize AOMDV is the vehicles speed.

In this thesis writer explain the implementation of HM-AOMDV (High Mobility Ad hoc On-Demand Multipath Distance Vector) in NS-2. HM-AOMDV is AOMDV implementation that use the vehicles speed as an optimization variable. HM-AOMDV uses hop count and relative speed of the vehicles to choose the best route to send data.

The result of experiment pointed that the average PDR (Packet Delivery Ratio) of HM-AOMDV are 8,5% to 10% better than AOMDV.

***Keywords**: VANET, AOMDV, NS-2, HM-AOMDV.*

Halaman ini sengaja dikosongkan

DAFTAR ISI

Halaman ini sengaja dikosongkan

DAFTAR TABEL

Halaman ini sengaja dikosongkan

DAFTAR GAMBAR

BAB 1

PENDAHULUAN

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

1.1 Latar Belakang

Seiring dengan kemajuan teknologi, aspek komunikasi antar komputer menjadi semakin penting. Salah satu cara komputer dapat berkomunikasi satu sama lain adalah dengan menggunakan jaringan *ad hoc*. Dalam jaringan *ad hoc* setiap komputer terkoneksi satu sama lain secara langsung tanpa bergantung kepada infrastruktur yang sudah ada seperti *router* atau *access point*.[4] Penentuan *node* mana yang akan meneruskan data dilakukan secara dinamis berdasarkan konektifitas jaringan. Karena karakteristik tersebut jaringan *ad hoc* cocok diaplikasikan dimana topologi jaringan mudah berubah sehingga *central node* tidak dapat diandalkan. Salah satu sistem yang bergantung pada jaringan *ad hoc* adalah *"Intelligent Transportation System"*[5] dalam bentuk VANET[6].

Terdapat beberapa buah *routing protocol* yang dapat digunakan dalam lingkungan VANET, salah satunya adalah AOMDV[7]. AOMDV adalah *routing protocol* yang merupakan ekstensi dari AODV[8]. AOMDV membuat beberapa jalur yang bersifat *loop-free* dan *link-disjoint* yang menghubungkan pengirim dan penerima.[7] Oleh karena itu jika jalur utama yang digunakan untuk mengirim data terputus, jaringan dapat dengan mudah mengganti jalur pengganti dari beberapa jalur *backup* yang telah terseida.

Tetapi protokol AOMDV masih hanya bergantung pada jumlah hop pada pemilihan jalur. Salah satu usaha untuk memberikan aspek *velocity aware* AOMDV adalah HM-AOMDV *(High Mobility Ad hoc On-Demand Distance Vector)*. Optimasi yang diajukan pada

HM-AOMDV adalah dengan menggunakan jumlah *hop* dan kecepatan relatif antar kendaraan untuk memilih *next hop* pada *routing table*.[1]

Dalam Tugas Akhir ini penulis akan mengimplenentasikan *routing protocol* HM-AOMDV pada *network simulator* NS-2[9]. Penulis akan mengujicoba dan membandingakan performa HM-AOMDV pada beberapa peta skenario yang didasarkan pada data yang terdapat pada OpenStreetMap[10]. Simulasi akan dilakukan dengan standar deviasi kecepatan kendaraan 10% dan 15% kecepatan standar dari jalur kendaraan. Dari simulasi tersebut akan didapatkan kesimpulan terhadap kemampuan adaptasi protokol HM-AOMDV dibandingkan denagn AOMDV pada skenario dimana kendaraan memiliki kecepatan yang berbeda beda. Hasil dari simulasi ini juga dapat menjadi dasar untuk mengoptimasi protokol AOMDV lebih jauh lagi.

1.2 Rumusan Masalah

Berikut ini adalah rumusan masalah yang diangkat pada Tugas Akhir ini:

1. Bagaimana cara mengimplementasikan protokol HM-AOMDV dan AOMDV pada peta daerah di Surabaya?
2. Bagaimana perbedaan performa antara AOMDV dan HM-AOMDV pada skenario yang telah disiapkan?
3. Bagaimana cara mengimplementasikan protokol HM-AOMDV pada lingkungan NS-2?

1.3 Batasan Masalah

Permasalahan yang dibahas pada Tugas Akhir ini memiliki beberapa batasan, yaitu:

1. Protokol yang diimplementasikan adalah HM-AOMDV.
2. Protokol yang diujicobakan adalah AOMDV dan HM-AOMDV.

3. Pembuatan skenario simuasi VANET dilakukan dengan menggunakan SUMO.
4. Data peta Surabaya diambil dari OpenStreetMap.

1.4 Tujuan

Berikut merupakan tujuan dari Tugas Akhir ini, yaitu:

1. Mengetahui pengaruh kecepatan antar kendaraan terhadap stabilitas koneksi pada lingkungan VANET.
2. Mendapatkan faktor lain yang dapat digunakan untuk memperkirakan stabilitas koneksi.

1.5 Manfaat

Manfaat dari pengerjaan Tugas Akhir ini adalah:

1. Menjadi acuan untuk optimasi lebih lanjut pada protokol AOMDV.
2. Memberikan gambaran mengenai faktor - faktor yang dapat diperhatikan dalam mendesain *routing protocol* pada lingkungan VANET.

1.6 Metodologi

Adapun langkah - langkah yang ditempuh dalam pengerjaan Tugas Akhir ini adalah sebagai berikut:

1. Penyusunan proposal Tugas Akhir
   Tahap awal untuk memulai pengerjaan Tugas Akhir adalah penyusunan proposal Tugas Akhir. Pada proposal tersebut dijelaskan secara garis besar tentang alur pembuatan sistem.
2. Studi literatur
   Pada tahap ini dilakukan studi literatur mengenai alat dan metode yang digunakan. Literatur yang dipelajari dan digunakan meliputi buku referensi, artikel, jurnal dan dokumentasi dari internet.

3. Implementasi protokol
   Tahap ini meliputi perancangan sistem berdasarkan studi literatur dan pembelajaran konsep teknologi dari perangkat lunak yang ada. Tahap ini merupakan tahap yang paling penting dimana bentuk awal aplikasi yang akan diimplementasikan didefinisikan. Pada tahapan ini dibuat prototype sistem, yang merupakan rancangan dasar dari sistem yang akan dibuat. Serta dilakukan desain suatu sistem dan desain proses-proses yang ada.
4. Uji coba dan evaluasi
   Pada tahapan ini dilakukan uji coba terhadap aplikasi yang telah dibuat. Pengujian dan evaluasi akan dilakukan dengan melihat kesesuaian dengan perencanaan. Tahap ini dimaksudkan juga untuk mengevaluasi jalannya sistem, mencari masalah yang mungkin timbul dan mengadakan perbaikan jika terdapat kesalahan.
5. Penyusunan buku Tugas Akhir
   Pada tahapan ini disusun buku sebagai dokumentasi dari pelaksanaan tugas akhir.

1.7 Sistematika Penulisan

Buku Tugas Akhir ini disusun degnan sistematika penulisan sebagai berikut:

BAB I. PENDAHULUAN

Bab yang berisi mengenai latar belakang, tujuan, dan manfaat dari pembuatan Tugas Akhir. Selain itu permasalahan, batasan masalah, metodologi yang digunakan, dan sistematika penulisan juga merupakan bagian dari bab ini.

BAB II. TINJAUAN PUSTAKA

Bab ini berisi penjelasan secara detail mengenai dasar-dasar penunjang untuk mendukung pembuatan tugas akhir ini.

BAB III. PERANCANGAN

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

BAB IV. IMPLEMENTASI

Bab ini membahas implementasi dari desain yang telah dibuat pada bab sebelumnya. Penjelasan berupa implementasi mobilitas vehicular, konfigurasi sistem dan skrip analisis yang digunakan untuk menguji performa protokol routing.

BAB V. UJI COBA DAN EVALUASI

Bab ini menjelaskan tahap pengujian sistem dan pengujian performa dalam skenario mobilitas vehicular yang dibuat.

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.

Halaman ini sengaja dikosongkan

BAB 2

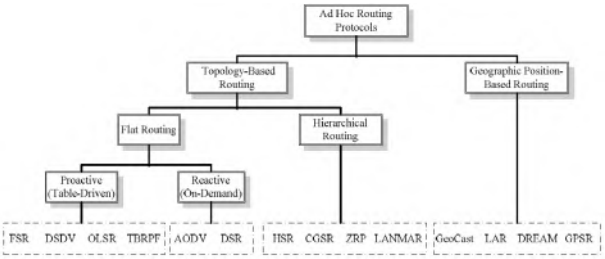
TINJAUAN PUSTAKA

Bab ini berisi penjelasan teori-teori yang berkaitan dengan pengimplementasian perangkat lunak. Penjelasan ini bertujuan untuk memberikan gambaran secara umum terhadap protokol routing, alat, serta definisi yang digunakan dalam pembuatan Tugas Akhir.

2.1 *Vehicular Ad hoc Network* (VANET)

VANET merupakan pengembangan teknologi dari *Mobile Ad Hoc Network* MANET yang mempertimbangkan semua kendaraan dalam jaringan sebagai *node* untuk berkomunikasi dengan kendaraan lainnya pada cakupan tertentu dengan menjadikan kendaraan berperan sebagai *router*.

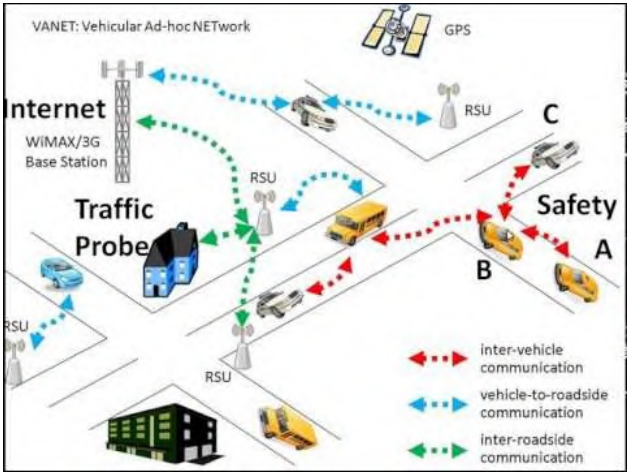
Pada VANET maupun MANET, *node* yang bergerak bergantung pada *ad hoc routing protocol* untuk menentukan bagaimana mengirim pesan kepada node tujuan. Hal ini menunjukkan bahwa protokol yang digunakan memiliki peranan penting dalam jaringan nirkabel dan tingkat kesulitannya bergantung kepada kecapat karier dan lingkungannya. Sebagaimana diilustrasikan pada gambar 2.1,*ad hoc routing protocol* dapat diklasifikasikan menjadi dua kategori yaitu *Topology-Based Routing* dan *Geographic Position-Based Routing*.



Gambar 2.1: *Ad Hoc Routing Protocol*[2]

Namun, VANET memiliki karakter yang berbeda dari MANET. Batasan pergerakan dan kecepatan yang tinggi menciptakan keunikan karakteristik VANET. Karakter inilah yang memegang pernana penting untuk menentukan desain jaringan seperti apakah yang akan dibuat.

Terdapat dua jenis *node* yang tergabung pada VANET, yaitu RSU *(Road-side Unit)* dan OBU *(On-board Unit)* dan karenanya, ada dua tipe komunikasi yang memungkinkan dengan VANET, yaitu antara kendaraan (yang mempunyai OBU) dengan kendaraan dan antara kendaraan dengan RSU seperti pada gambar 2.2. RSU merupakan *node* yang terpasang pada beberapa bagian jalan yang nantinya dapat terhubung dengan jaringan *backbone* untuk memberikan informasi - informasi penting kepada OBU. Contoh informasi yang dapat diberikan oleh RSU adalah batas kecepatan, status lampu lalu - lintas, keberadaan infrastruktur penting dan lain - lain. Selain itu RSU juga dapat memberikan jaringan internet kepada OBU yang terhubung.



Gambar 2.2: Ilustrasi VANET[3]

OBU merupakan *node* yang bergerak atau dalam kata lain kendaraan yang berada di jalan. Informasi yang dapat diberikan oleh OBU diantaranya adalah kecepatan, status rem, sudut kemudi, lampu sen dan kondisi lalu lintas. Dari komunikasi antar OBU, pengendara dapat mengambil tidakan awal jika terindikasi terdapat situasi abnormal pada lalu - lintas.

Agar komunikasi antar *node* pada VANET dapat berjalan dengan lancar, diperlukan sebuah *routing protocol* yang dapat beradaptasi dengan karakteristik VANET yang sudah dijelaskan tadi. Pada Tugas Akhir ini penulis akan mengimplementasikan *routing protocol* HM-AOMDV pada NS-2 dan menguji performa protokol tersebut pada lingkungan VANET.

2.2 *High Mobility Ad hoc On-Demand Multipath Distance Vector* (HM-AOMDV)

HM-AOMDV merupakan sebuah *routing protocol* yang dikembangkan oleh Lee et al. [1] sebagai pengembangan dari protokol AOMDV dengan menggunakan jumlah *hop* dan kecepatan relatif antar kendaraan untuk menentukan jalur yang digunakan untuk mengirim paket data. AOMDV sendiri merupakan pengembangan dari protokol AODV dengan menggunakan lebih dari satu jalur yang bersifat *loop free* dan *link disjoint* untuk menghubungkan pengirim dan penerima. Berikut ini penulis akan menjelaskan implementasi dari masing - masing protokol terkait.

2.2.1 AODV

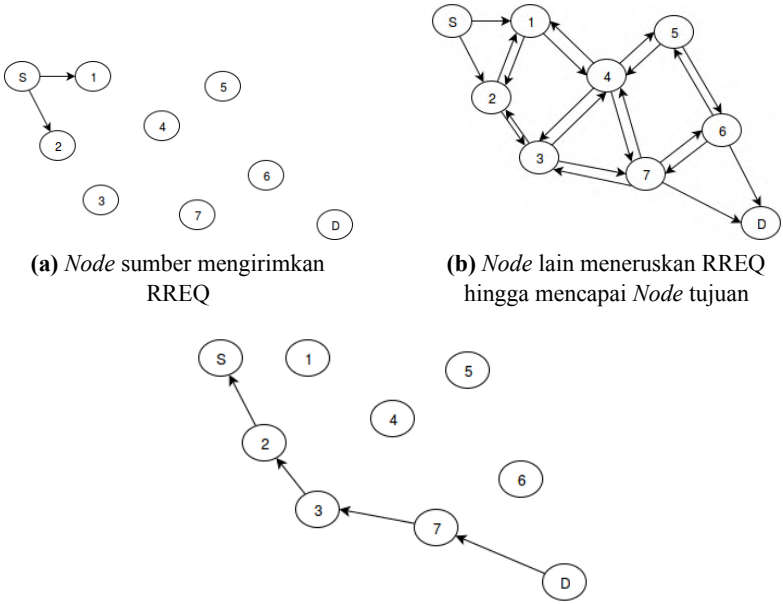
Ad ho On-Demand Distance Vector merupakan sebuah *routing protocol* yang bersifat reaktif, hal ini berarti protokol ini hanya mengurimkan *request* untuk sebuah rute hanya saat *node* ingin melakukan komunikasi. Spesifikasi AODV ditulis oleh Perkins et al. [8] pada RFC 3561. AODV memiliki dua buah fungsi, yaitu *route discovery* dan *route maintenance*. *Route discovery* dilakukan menggu-

nakan menggunakan paket *Route Request* (RREQ) dan *Route Reply* (RREP). Sedangkan fungsi *Route Maintenance* dibantu dengan dengan paket *Route Error* (RERR).

Pada AODV setiap *node* harus memiliki sebuah *routing table* yang memiliki field berikut:

- *Destination Address*: berisi alamat dari *node* tujuan.
- *Destination Sequence Number*: *sequence number* dari jalur komunikasi.
- *Next Hop*: *node* yang akan digunakan untuk meneruskan paket data.
- *Hop Count*: jumlah lompatan yang harus dilakukan paket untuk mencapai *node* tujuan.
- *Lifetime*: Waktu dalam milidetik yang diperlukan *node* untuk menerima RREP.
- *Routing Flags*: status jalur. *Up* (valid), *down* (tidak valid) atau sedang diperbaiki.

Gambar 2.3 menunjukkan langkah - langkah yang dilakukan pada proses *route discovery* dalam AODV. Saat *node* akan menginisiasi sebuah komunikasi maka *node* pengirim akan melakukan *route discovery* dengan melakukan *broadcast* RREQ seperti ditunjukkan pada gambar 2.3a. Saat *node* menerima paket RREQ maka *node* itu akan memeriksa *routing table* miliknya. Jika *destination sequence number* yang terdapat pada paket RREQ sama atau lebih kecil dari data yang terdapat pada *routing table* dan jalur menuju *node* tujuan belum ditemukan, maka paket tersebut tidak akan dilanjutkan tetapi kalau ternyata jalur menuju *node* tujuan tercatat pada *routing table* dan memiliki *routing flags "up"* maka *node* akan mengirimkan paket RREP ke *node* pengirim. Jika *destination sequence number* pada RREQ lebih besar dibandingkan yang terdapat pada *routing table* maka *entry* pada *routing table* akan diperbarui dan paket tersebut akan diteruskan sekaligus membuat *reverse path* menuju *node* pengirim. Proses *route discovery* dilanjutkan pada gambar 2.3b dimana *node* yang menerima paket RREQ juga akan mem-*broadcast*



(a) *Node* sumber mengirimkan RREQ

(b) *Node* lain meneruskan RREQ hingga mencapai *Node* tujuan

(c) *Node* tujuan mengirimkan RREP sekaligus memvalidasi rute yang akan digunakan

Gambar 2.3: Proses *Route Discovery* pada AODV

paket itu ke *node* sekitarnya. Tahap terakhir dari *route discovery* seperti ditunjukkan pada gambar 2.3c adalah saat *node* tujuan menerima paket RREQ dan mengirimkan paket RREP menuju *node* pengirim melalui *reverse path* yang telah dibangun pada proses pengiriman RREQ.

Fungsi *route maintenance* berguna untuk memastikan semua jalur yang tercatat pada *routing table* memiliki status yang tepat. Contohnya adalah saat proses pengiriman data berlangsung. Jika terdapat koneksi antar *node* yang terputus maka *node* yang mendeteksi kejadian tersebut akan mengirimkan paket RERR kepada semua

node yang bekomunikasi menggunakan jalur terkait. Kemudian semua *node* yang menerima paket RERR akan menandai jalur tersebut menjadi *down* dan meneruskan paket tersebut hingga sampai pada *node* pengirim. Jika *node* pengirim masih memerlukan komunikasi data dengan *node* penerima, maka proses *route discovery* antara *node* pengirim dan tujuan akan dilakukan kembali.

2.2.2 AOMDV

Ad hoc On-Demand Multipath Distance Vector merupakan ekstensi dari protokol AODV. Pada AOMDV *node* menyimpan lebih dari satu jalur menuju *node* tujuan. Hal ini dimungkinkan dengan merubah proses *route discovery* pada protokol AODV. Berikut ini adalah perubahan yang terdapat pada protokol AOMDV jika dibandingkan dengan protokol AODV:

- Pada AODV jika *destination sequence number* yang diterima dari paket RREQ bernilai sama dengan yang berada pada *routing table* maka paket tersebut akan di-*drop*. Sedangkan pada AOMDV *node* akan memeriksa jumlah *hop* yang tercatat pada paket RREQ tersebut, jika jumlah *hop* lebih sedikit dari yang tercatat pada *routing table* maka *node* tersebut akan menambahkan *reverse route* berdasarkan paket RREQ tersebut.
- Pada AOMDV *node* akan meneruskan paket RREP ke semua jalur yang tercatat pada *routing table* dengan *destination sequence number* yang sama.
- Sebuah node bisa mendapatkan lebih dari satu paket RREP dengan *destination sequence number* yang sama tetapi dari *node* yang berbeda. Saat hal tersebut terjadi maka *node* tersebut akan membandingkan jumlah *hop* yang tercatat pada paket tersebut dengan paket RREP pertama yang mencapai *node* tersebut. Jika jumlah *hop* yang terbaru lebih kecil maka jalur alternatif melalui *node* pengirim paket RREP akan ditambahkan kedalam *routing table*.

Dengan adanya lebih dari satu jalur menuju satu *node* maka proses pencarian jalur baru saat jalur yang sedang digunakan terputus-bisa dilakukan dengan lebih cepat. Proses *route discovery* di tengah - tengah komunikasi hanya dilakukan saat semua jalur yang tercatat pada *route table* menuju *node* tujuan berstatus *down*.

2.2.3 HM-AOMDV

High Mobility Ad hoc On-Demand Multipath Distance Vector merupakan ekstensi dari protokol AOMDV yang menggunakan jalur yang tersimpan pada *routing table* untuk mengoptimasi kinerja protokol tersebut. HM-AOMDV menggunakan sebuah variabel yang disebut *mobility factor* untuk memilih jalur utama yang akan digunakan untuk pengiriman data. *Mobility factor* adalah nilai yang didapat setelah menghitung kecepatan relatif sebuah *node* dengan *node* tetangganya. Untuk memungkinkan perhitungan *mobility factor* maka diperlukan perubahan pada struktur paket data yang digunakan untuk *route discovery* dan *route maintenance*. Paket yang diubah dari AOMDV adalah paket RREP dan HELLO, struktur paket tersebut pada HM-AOMDV dapat dilihat pada tabel 2.1.

Type	*Source IP Address*	*Speed*(km/s)

(a) Struktur HELLO message

Type	*Reversed*	*Last hop*	*Hop count*
RREQ ID		*Destination IP Address*	
Destination Sequence Number		*Originator IP Address*	
Originator Sequence Number		*Mobility Factor*	

(b) Struktur RREP

Tabel 2.1: Struktur paket HELLO dan RREP pada HM-AOMDV[1]

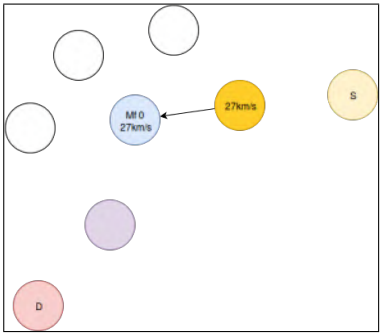
Rumus untuk menghitung *mobility factor* adalah:

$$Mf = \frac{(1+r)^{-w} + (Mf \times (N-1))}{N} \tag{2.1}$$

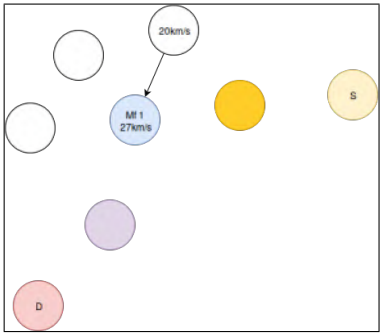
Gambar 2.4 akan menunjukkan ilustrasi dari penghitungan Mf. Selain itu, berikut adalah penjelasan mengenai variabel yang digunakan dalam menghitung *mobility factor*:

- Mf: *mobility factor* kendaraan yang diskalakan berdasarkan kecepatan dari kendaraan tetangga. Variabel ini memiliki nilai dari 0 hingga 1 dan memiliki nilai maksimum ketika kecepatan relatifnya dengan kendaraan lain adalah nol. Setiap kendaraan menyimpan Mf dari kendaraan tetangganya. Kendaraan dengan Mf tertinggi akan dipilih menjadi *next hop* pada komunikasi yang sedang berjalan. Nilai Mf akan diubah menjadi nol setiap *hello interval*.
- r: nilai relatif antara kecepatan kendaraan saat ini dengan kecepatan yang tercantum pada HELLO.
- N: Jumlah HELLO yang diterima dalam satu *hello interval*. Nilainya akan diubah menjadi nol setiap *hello interval*.
- w: konstanta untuk perhitungan Mf.

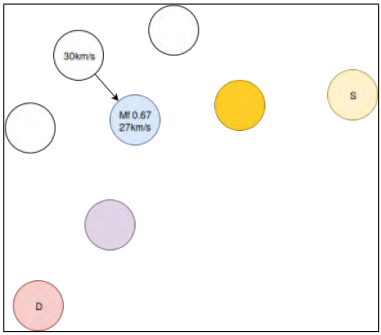
Gambar 2.5 menunjukkan *flowchart* saat protokol HM-AOMDV menerima paket RREP. Setelah menerima paket HELLO, *node* akan menghitung nilai Mf dan mengirimnya melalui paket RREP. *Node* yang menerima paket tersebut pertama - tama akan membaca *IP address* yang tercantum dan mencari jalur yang melalui *node* tersebut. Kemudian *node* tersebut membandingkan *destination sequence number* yang tercantum pada RREP dan *routing table*. Jika *destination sequence number* pada RREP lebih kecil dari yang terdapat pada *routing table* maka paket tersebut akan dihapus. Sebaliknya jika *destination sequence number* pada RREP lebih besar artinya paket tersebut berisikan informasi yang lebih baru dari *routing table*, sehingga *node* tersebut menghapus data mengenai jalur tersebut dan menggantinya berdasarkan data yang terdapat pada RREP. Tetapi



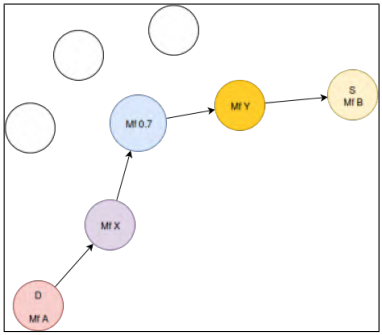
(a) *Node* dengan *default* $Mf = 0$ mendapatkan HELLO dari node tetangga



(b) *Node* menghitung nilai Mf lalu menerima HELLO dari tetangga lainnya



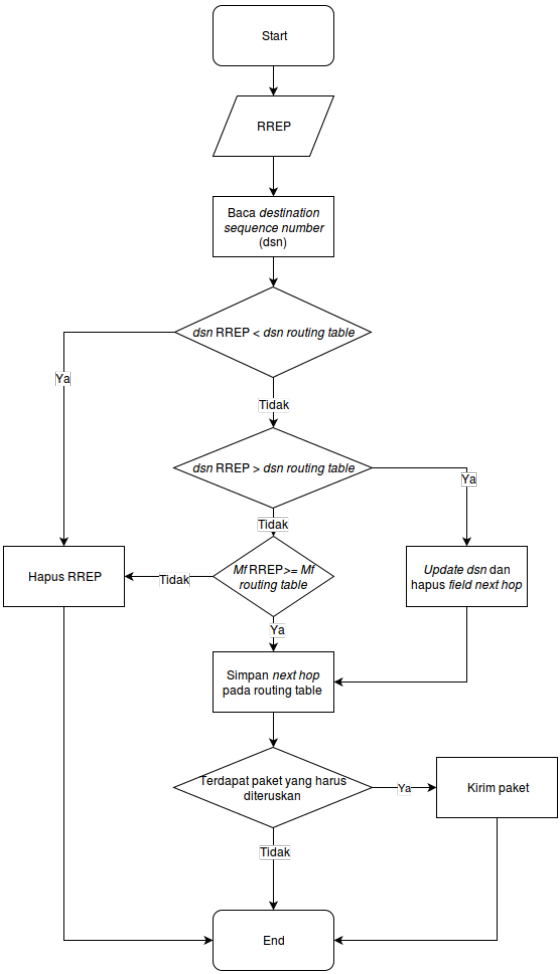
(c) Proses terus diulang setiap kali *Node* menerima paket HELLO



(d) Nilai Mf yang telah dimiliki oleh *Node* nantinya akan dikirimkan bersamaan dengan paket RREP

Gambar 2.4: Ilustrasi penghitungan nilai Mf

jika kedua *destination sequence number* bernlai sama *node* tersebut akan membandingkan nilai Mf pada paket RREP dan *routing table*. Jika nilai Mf pada RREP lebih besar atau sama dengan nilai Mf pada *routing table* maka *node* akan menambahkan jalur alternatif berdasarkan RREP, sebaliknya jika nilai Mf pada RREP lebih



Gambar 2.5: Flowchart penanganan paket RREP pada HM-AOMDV[1]

kecil maka paket tersebut akan dihapus. Dengan cara ini protokol HM-AOMDV dapat memilih jalur yang paling sehat untuk komuni-

kasi data.

2.3 *Simulation of Urban Mobility* (SUMO)

Simulation of Urban Mobility atau SUMO merupakan sebuah program *simulator* lalu lintas. SUMO dikembangakan sejak tahun 2000 dengan tujuan untuk mengakimodasi penelitian - penelitian yang melibatkan pergerakan kendaraan di jalan raya, terutama dalam daerah dengan penduduk yang padat (*urban*). Publikasi referensi terbaru tentang SUMO ditulis oleh Krajzewicz et al. [11] pada tahun 2012.

SUMO terdiri dari beberapa *tools* yang dapat membantu pembuatan simulasi lalu lintas pada tahap - tahap berbeda. Berikut penjelasan fungsi tools yang digunakan dalam pembuatan Tugas Akhir ini:

- netconvert
  netconvert merupakan program CLI yang berfungsi untuk melakukan konversi dari peta seperti OpenStreetMap atau VISUM menjadi format *native* SUMO. Pada Tugas Akhir ini penulis menggunakan netconvert untuk mengkonversi peta dari OpenStreetMap.
- randomTrips.py
  Tool dalam SUMO untuk membuat rute acak yang akan dilalui oleh kendaraan dalam simulasi.
- route2trips.py
  Membuat detail perjalanan setiap kendaraan berdasarkan output dari randomTrips.py.
- sumo
  Program yang melakukan simulasi lalu lintas berdasarkan data - data yang didapatkan dari netconvert dan route2trips.py. Hasil simulasi dapat di *export* ke sebuah file untuk nantinya dikonversi menjadi format lain.
- sumo-gui

GUI untuk melihat simulasi yang dilakukan oleh SUMO secara grafis.

- traceExporter.py
  Tool yang bertujuan untuk mengkonversi output dari sumo menjadi format yang dapat digunakan pada *simulator* lain. Pada Tugas Akhir ini penulis menggunakan traceExporter.py untuk mengkonversi data menjadi format .tcl yang dapat digunakan pada NS-2.

2.4 OpenStreetMap

OpenStreetMap (OSM) [10] merupakan sebuah proyek kolaboratif untuk membuat sebuah peta dunia yang dapat dengan bebas diubah oleh siapapun. Dua buah faktor pendukung dalam pembuatan dan perkembangan OSM adalah kurangnya ketersediaan dari informasi peta mengenai sebagian besar daerah di dunia dan munculnya alat navigasi portabel yang terjangkau.[12] OSM dianggap sebagai contoh utama dalam informasi geografis yang diberikan secara sukarela.

Pengembangan OSM diinspirasi oleh kesuksesan Wikipedia pengaruh dari peta *propritery* di UK dan tempat lain. Sejak diluncurkan hingga sekarang OSM telah berkembang hingga memiliki lebih dari dua juta pengguna yang teregistrasiyang dapat mengambil data menggunakan survey manual, piranti GPS, *aerial photography* dan sumber bebas lainnya. Data yang terdapat pada OSM berada dalam lisensi *Open Database License* sehingga data dari OSM dapat dengan bebas digunakan oleh semua orang.

Pada Tugas Akhir ini penulis menggunakan data yang tersedia pada OSM untuk membuat skenario lalu lintas berdasarkan peta Surabaya.

2.5 JOSM

JOSM (*Java OpenStreetMap Editor*) adalah sebuah alat untk menyuting data yang didapatkan dari OpenStreetMap. Aplikasi JOSM dapat diunduh pada alamat `https://josm.openstreetmap.de/`. Penulis menggunakan aplikasi ini untuk menyuting dan merapikan potongan peta yang diunduh dari OpenStreetMap.

2.6 AWK

AWK merupakan sebuah *Domain Specific Language* (DSL) yang didesain untuk *text processing* dan biasanya digunakan sebagai alat ekstraksi data dan *reporting*. AWK bersifat *data-driven* yang berisikan kumpulan perintah yang akan dijalanpan pada data tekstual baik secara langsung pada file atau digunakan sebagai bagian dari *pipeline*. Pada Tugas Akhir ini penulis menggunakan AWK untuk memproses data yang dihasilkan dari simulasi pada NS-2 dan mendapatkan analisis mengenai *packet delivery rate, end-to-end delay* dan lain - lain.

2.7 *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. NS-3 dikembangkan sejak tanggal 1 Juli 2006 dan sedang dikembangkan secara aktif sedangkan

pengembangan NS-2 tidak begitu aktif karena fokus pengembang sebagian besar beralih ke NS-3. Pada Tugas Akhir ini penulis memilih NS-2 karena protokol AOMDV yang menjadi dasar dari pengembangan protokol HM-AOMDV masih hanya tersedia pada NS-2.

Versi terbaru NS-2 adalah ns-2.35 yang dirilis pada tanggal 4 November 2011. Dalam membuat sebuah simulasi jaringan komputer NS-2 menggunakan dua buah bahasa pemrograman, yaitu C++ dan OTcl. Bahasa C++ digunakan untuk mengimplementasi bagian - 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 dijalankan pada sistem operasi Windows dengan menggunakan Cygwin.

2.7.1 Instalasi NS-2

Sebelum melakukan instalasi NS-2 hal pertama yang harus dilakukan adalah menginstall dependensi yang dibutuhkan. Salah satu dependensi dari NS-2 adalah GCC versi 4.3, tetapi jika pada distribusi yang digunakan tidak tersedia GCC versi 4.3 versi 4.4 dapat digunakan. Gambar 2.6 menunjukkan dependensi NS-2 beserta GCC 4.4 dan cara menginstallnya pada distribusi Debian dan turunannya.

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

Gambar 2.6: Perintah untuk menginstall dpendensi NS-2 pada distribusi Debian

Setelah semua dependensi terinstall hal selanjutnya yang dapat dilakukan adalah mengunduh *source code* NS-2. Saat proses pengunduhan telah selesai, lakukan ekstraksi *file* tarball seperti yang

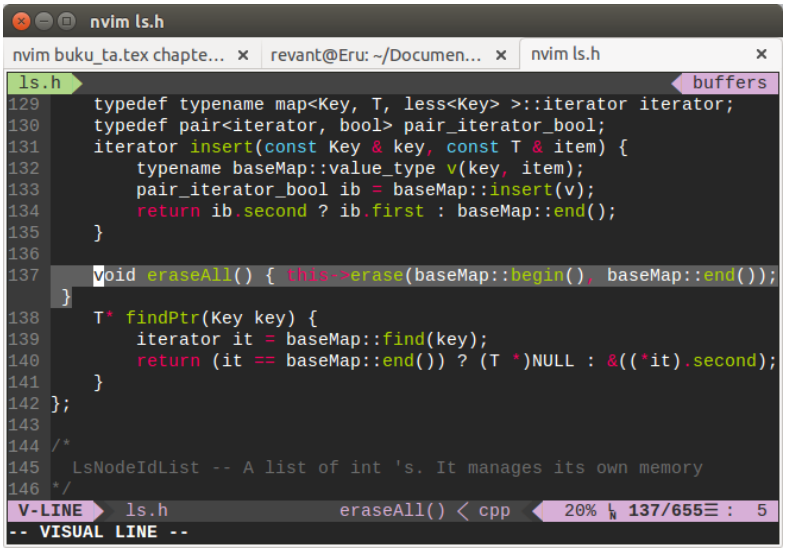
ditunjukkan pada gambar 2.7.

```
1 wget
   ↪ http://jaist.dl.sourceforge.net/project/nsnam/allinone/
   ↪ allinone-2.35/ns-allinone-2.35.tar.gz
   ↪ \\
2 tar -xvf ns-allinone-2.35.tar.gz
```

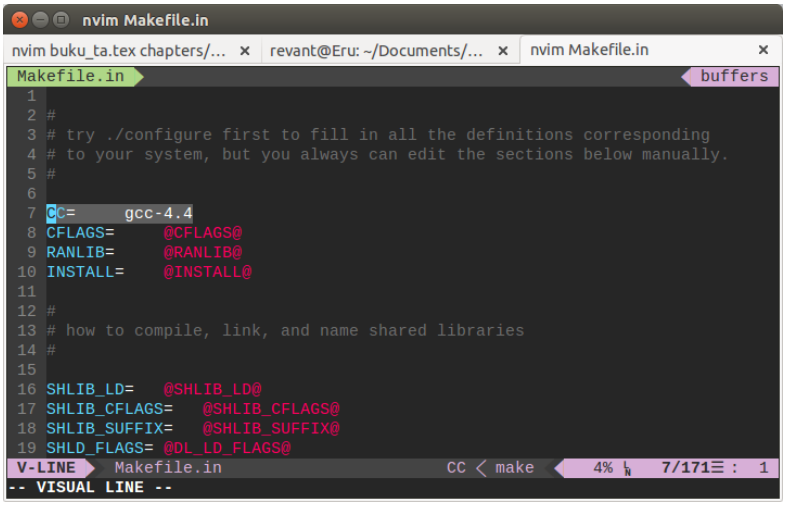
Gambar 2.7: Perintah untuk mengunduh dan mengekstrak NS-2

Navigasi menuju folder "linkstate" yang terdapat pada *ns-allinone-2.35/ns-2.35/linkstate*. Kemudian buka file yang bernama "ls.h" dan pergi ke baris 137 pada kode tersebut. Setelah itu ubah kata **"error"** menjadi **"this->error"**. *Screenshot* dari perubahan yang dilakukan pada file tersebut dapat dilihat pada gambar 2.8a.

Langkah terakhir yang harus dilakukan adalah memberitahu NS versi GCC yang akan digunakan. Untuk itu variabel CC pada *ns-allinone-2.35/otcl-1.14/Makefile.in* harus diubah dari `@CC@` menjadi `gcc-4.4`. Gambar 2.8b menunjukkan perubahan yang dilakukan.

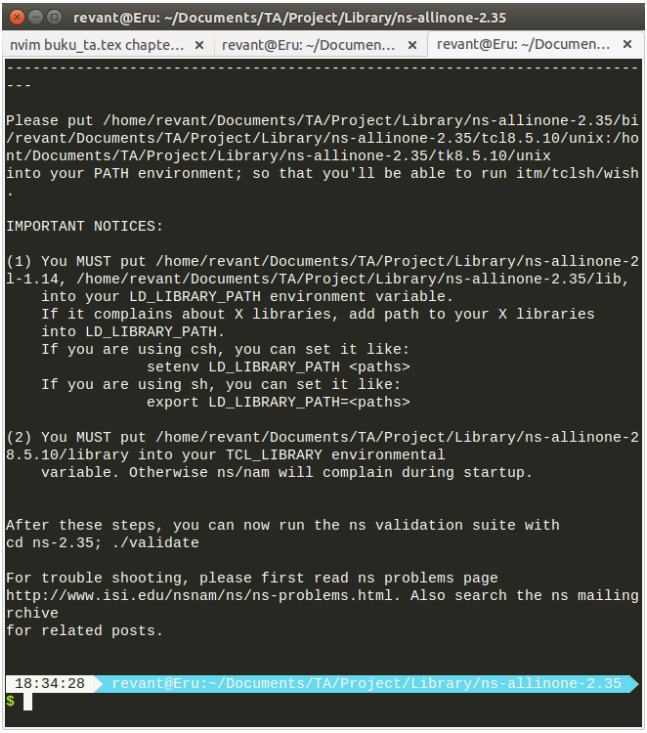


(a) Perubahan yang dilakukan pada ls.h



(b) Perubahan pada Makefile OTcl

Gambar 2.8: Setting pada Instalasi NS-2



Gambar 2.9: Hasil instalasi NS-2

Setelah semua tahap di atas selesai hal terakhir yang perlu dilakukan adalah menjalankan skrip instalasi NS-2. Untuk menjalankannya masukkan perintah *ns-allinone-2.35/./install* pada terminal dan tunggu hingga proses instalasi selesai. Gambar 2.9 menunjukkan *screenshot* dari hasil instalasi NS-2.

2.7.2 Penggunaan Skrip OTcl

OTcl merupakan ekstensi *object oriented* dari bahasa pemrograman Tcl. Bahasa ini dikembangkan oleh Wetherall [13] di MIT

sebagai bagian dari proyek VUsystem. Bahasa OTcl digunakan sebagai bahasa *scripting* pada NS-2 untuk mengatur lingkungan dan skenario simulasi.

Setiap *class* yang terdapat pada OTcl memiliki *binding* pada kode C++. Hal ini memungkinkan pembuatan skenario simulasi secara cepat dan tanpa perlu menggunakan bahasa C++ secara langsung. Gambar 2.10 memunjukkan contoh potongan kode OTcl pada NS-2 untuk melakukan *setting* lingkungan simulasi.

Pada baris pertama hingga kesembilan penulis menyimpan nilai setting lingkungan simulasi pada beberapa variabel. Kemudian pada baris kesebelas penulis menginstansiasi objek *simulator* yang merupakan objek yang akan menjalankan simulasi VANET. Setelah itu pada baris ketiga belas penulis menginstansiasi koneksi *wireless* yang akan digunakan oleh setiap *node*. Terakhir pada baris keempat belas hingga kedua puluh tujuh penulis memasukkan semua variabel pengaturan pada pengaturan *node*. Pengaturan ini akan membuat semua *node* yang diinstansiasi pada proses simulasi akan memiliki atribut sesuai dengan yang diinginkan oleh penulis.

2.7.3 NS-2 *Trace File*

Trace File merupakan *file* hasil dari simulasi sebuah skenario pada NS-2. Isi dari sebuah *trace file* adalah catatan mengenai paket yang dikirim dan diterima oleh setiap *node* dalam simulasi. Setiap jenis paket yang beredar pada jaringan memiliki pola - pola tersendiri sehingga dapat dibedakan satu sama lain dan membantu memudahkan analisis terhadap hasil simulasi. Pada tugas akhir ini penulis menggunakan peredaran paket komunikasi, HELLO, RREQ, RREP dan RRER dalam menganalisis hasil simulasi.

Pada gambar 2.11 ditunjukkan contoh *trace* yang didapat pada setiap jenis paket yang beredar di lingkungan simulasi. Gambar a hingga d menunjukkan contoh paket *routing controll* dari NS-2, sedangkan gambar e menunjukkan paket data komunikasi dari agen CBR *(Constant Bit Rate)*. Paket *routing controll* dan paket komuni-

```
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)      AOMDV;# routing protocol

set ns_ [new Simulator]

set chan_1_ [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) \
    -topoInstance $topo \
    -agentTrace ON \
    -routerTrace ON \
    -macTrace OFF \
    -movementTrace OFF \
    -channel $chan_1_
```

Gambar 2.10: Potongan kode pengaturan lingkungan simulasi VANET

```
r 120.001200134 _55_ RTR  --- 0 AOMDV 52 [0 ffffffff 119
 ↪  800] ------- [281:255 -1:255 30 0] [0x2 0 1 [282 0] [281
 ↪  4]] (REQUEST)
```

(a) Contoh paket RREQ

```
s 120.038438424 _282_ RTR  --- 0 AOMDV 52 [0 0 0 0] -------
 ↪  [282:255 281:255 30 153] [0x4 0 [282 2] 10.000000]
 ↪  (REPLY) [1 153]
```

(b) Contoh paket RREP

```
r 120.044360677 _217_ RTR  --- 0 AOMDV 44 [0 ffffffff 2f
 ↪  800] ------- [47:255 -1:255 1 0] [0x1 0 [47 2]
 ↪  4.000000] (HELLO) [0 0]
```

(c) Contoh paket HELLO

```
s 134.000000000 _155_ RTR  --- 0 AOMDV 32 [0 0 0 0] -------
 ↪  [155:255 -1:255 1 0] [0x8 1 [282 0] 0.000000] (ERROR)
 ↪  [0 0]
```

(d) Contoh paket RRER

```
r 134.473722701 _282_ AGT  --- 73 cbr 532 [13a 11a d3 800]
 ↪  ------- [281:0 282:0 23 282] [73] 8 0
```

(e) Contoh paket komunikasi data

Gambar 2.11: Contoh sampel tipe paket pada *trace file* NS-2

kasi data dapat dibedakan dengan melihat kata keempat pada baris deskripsi paket, pada paket *routing controll* kata yang tertulis pada kata keempat adalah RTR sedangkan pada paket komunikasi data yang tertulis adalah AGT. Perbedaan lain yang dapat dilihat pada contoh yang diberikan adalah status pengiriman dan penerimaan sebuah paket. Jika karakter pertama pada sebuah baris adalah r maka trace baris tersebut mendeskripsikan paket saat sedang diterima oleh sebuah *node*. Sebaliknya jika karakter pertama adalah s maka baris tersebut mendeskripsikan informasi saat paket sedang dikirim oleh sebuah *node*. Dengan mengetahui tipe paket dan pola yang terdapat pada *trace file* penulis dapat melakukan analisis hasil simulasi dari NS-2.

Halaman ini sengaja dikosongkan

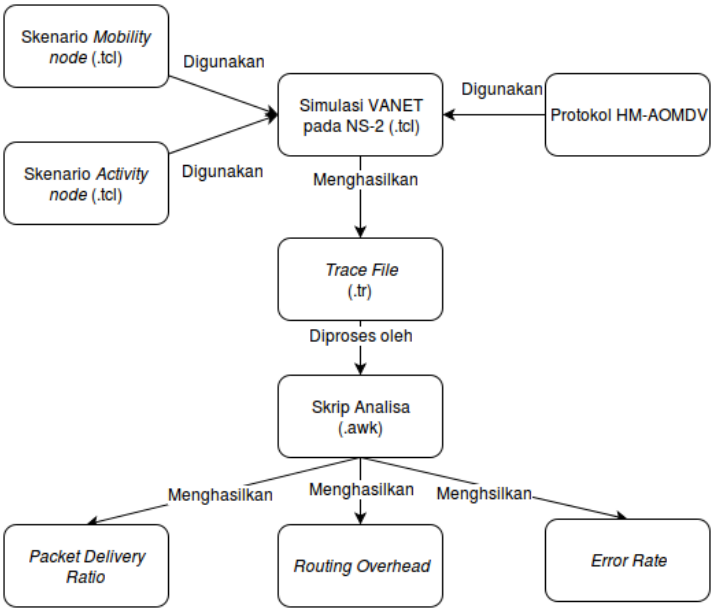
BAB 3

PERANCANGAN

Perancangan merupakan bagian yang sangat penting dari implementasi sistem. Bab ini secara khusus akan menjelaskan rancangan sistem yang dibuat pada Tugas Akhir. Bagian yang akan dijelaskan pada bab ini derawal dari deskripsi umum hingga perancangan skenario, alur dan implementasinya.

3.1 Deskripsi Umum

Pada Tugas Akhir ini penulis akan mengimplementasi protokol HM-AOMDV pada NS-2. Diagram dari rancangan simulasi dapat dilihat pada gambar 3.1.



Gambar 3.1: Diagram rancangan simulasi

Dalam Tugas Akhir ini peta skenario yang digunakan didasarkan pada peta riil lingkungan lalu lintas kota Surabaya. Peta tersebut diambil dari data yang terdapat pada OpenStreetMap, peta tersebut kemudian dirapikan menggunakan aplikasi JOSM. Setelah peta selesai dirapikan peta tersebut digunakan untuk melakukan simulasi lalu lintas menggunakan SUMO. Hasil simulasi tersebut kemudian digunakan bersama dengan protokol HM-AOMDV pada NS-2 untuk melakukan simulasi VANET. Hasil simulasi VANET tersebut kemudian dianalisis menggunakan skrip AWK untuk meendapatkan *packet delivery rate, routing overhead* dan *error rate* dari simulasi tersebut. Analisis tersebut dapat mengukur performa protokol HM-AOMDV dibandingkan dengan protokol AOMDV.

3.2 Perancangan Skenario Mobilitas

Perancangan skenario mobilitas dimulai dengan pemilihan daerah yang akan digunakan sebagai model untuk simulasi. Setelah mendapatkan daerah yang sesuai maka bagian peta tersebut diunduh dari OpenStreetMap degan menggunakan fitur *export*. Kemudian peta tersebut dirapikan menggunakan program JOSM. Beberapa hal yang harus dirapikan dari peta tersebut adalah pengaturan interval lampu lalu lintas dan menghapus jalan yang terputus sehingga peta tersebut menjadi daerah tertutup tanpa daerah yang terisolasi.

Sebelum melakukan konversi terhadap peta yang telah diproses dengan JOSM pertama - tama perlu dibuat sebuah *type file* yang mendeskripsikan batasan - batasan pada simulasi lalu lintas. Beberapa batasan yang dapat dideklarasikan adalah batas kecepatan pada sebuah tipe jalan dan batas kecepatan dari suatu tipe kendaraan. Setelah itu peta dikonversi berdasarkan *type file* yang telah dibuat menggunakan *tools* `netconvert` menjadi sebuah *file* berekstensi .net.xml. Hasil konversi tersebut digunakan untuk membuat *file* deskripsi pergerakan kendaraan menggunakan *tool* `randomTrips.py` dan `route2trips.py`.

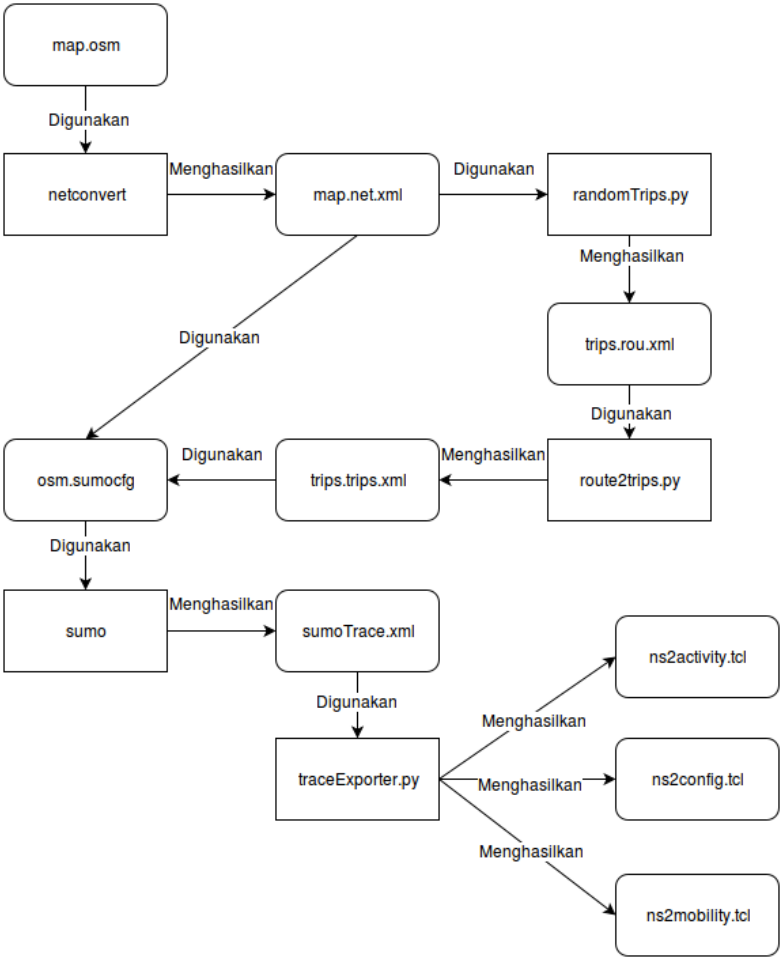
Simulasi lalu lintas kemudian dilakukan dengan menggunakan *file* peta yang telah dikonversi dan *file* pergerakan yang dihasilkan sebelumnya. Hasil dari simulasi dari SUMO adalah sebuah *file* dengan ekstensi .xml. Lalu hasil simulasi tersebut dikonversi menjadi *file mobility, activity* dan *configuration* yang dapat kompatibel dengan NS-2 menggunakan `traceExporter.py`. Ilustrasi alur pembuatan skenario mobilitas dapat dilihat pada gambar 3.2.

3.3 Perancangan Simulasi pada NS-2

Simulasi VANET pada NS-2 dilakukan dengan menggunakan skenario mobilitas dan digabungkan dengan skrip TCL yang berisikan konfigurasi mengenai lingkungan simulasi. Konfigurasi lingkungan simulasi VANET pada NS-2 dapat dilihat pada tabel 3.1.

Tabel 3.1: Parameter lingkungan simulasi

No.	Parameter	Spesifikasi
1	*Network simulator*	NS-2, 2.35
2	*Routing protocol*	HM-AOMDV dan AOMDV
3	Radius transmisi	250 m
4	Kecepatan kendaraan	11 m/s, 13 m/s, 15 m/s, 17 m/s dan 19 m/s
5	Deviasi kecepatan	10% dan 15%
6	Agen pengirim	*Constant Bit Rate* (CBR)
7	*Source / Destination*	Statis
8	*Packet Rate*	20 kb/s
9	Ukuran paket	512 bit
10	Protokol MAC	IEEE 802.11
11	Propagasi sinyal	*Two-ray ground*
12	Tipe kanal	*Wireless channel*



Gambar 3.2: Alur pembuatan skenario mobilitas

3.4 Perancangan Metrik Analisis

Berikut ini merupakan beberapa parameter yang dianalisis dalam Tugas Akhir ini:

3.4.1 *Packet Delivery Ratio* (PDR)

Packet delivery ratio merupakan perbandingan dari jumlah paket komunikasi yang dikirimkan dengan paket komunikasi yang diterima. *Packet delivery ratio* dihitung menggunakan persamaan 3.1, dimana $received$ adalah jumlah paket komunikasi yang diterima dan $sent$ adalah jumlah paket komunikasi yang dikirimkan.

$$PDR = \frac{Data_{received}}{Data_{sent}} \tag{3.1}$$

3.4.2 *Normalized Routing Overhead* (RO)

Routing overhead merupakan jumlah paket kontrol yang ditransmisikan per paket komunikasi yang terkirim ke tujuan selama simulasi terjadi. Paket kontrol yang dihitung adalah perbandingan antara jumlah *Route Request* (RREQ), *Route Reply* (RREP) dan *Route Error* (RRER) yang dikirim dengan jumlah paket komunikasi yang diterima. Rumus dari RO dapat dilihat pada persamaan 3.2.

$$RO = \frac{RREQ_{sent} + RREP_{sent} + RRER_{sent}}{Data_{received}} \tag{3.2}$$

3.4.3 *Error Rate* (ER)

Pada *routing overhead* paket kontrol yang dihitung adalah RREQ, RREP dan RRER tetapi pada *error rate* paket yang dihitung hanyalah RRER.

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

Rata-rata *End-to-End delay* mengindikasikan interupsi transmisi paket dari *node* asal ke tujuan. Total interupsi didapatkan dari akumulasi *delay-delay* kecil yang ada dalam jaringan. Total interupsi terdiri dari *delay* yang mungkin karena *buffer* pada *route discovery latency*, *delay* pada antrian *interface*, retransmisi *Media Access*

Control(MAC) *delay* dan waktu transfer serta *propagation delays*. Rata-rata E2E *delay* pada paket yang diterima bisa dihitung berdasarkan selisih waktu antara transmisi dan respon paket pada *Constant Bit Rate* (CBR) dan membaginya dengan jumlah total transimi CBR menggunakan persamaan 3.3. Semakin kecil *End-to-End delay* mengidikasikan performa yang lebih baik.

$$E2E = \frac{\sum_{i=1}^{recvnum} CBRRecvTime_i - CBRSentTime_i}{recvnum} \tag{3.3}$$

BAB 4

IMPLEMENTASI

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

4.1 Lingkungan Pembangunan Perangkat Lunak

Pembangunan perangkat lunak dilakukan pada lingkunan pengembangan sebagai berikut:

4.1.1 Lingkungan Perangkat Lunak

Adapun lingkungan perangkat lunak yang digunakan dalam pengembangan sistem adalah sebagai berikut:

- Sistem operasi berupa Linux Ubuntu 14.04 64-bit.
- SUMO versi 0.24.0 untuk mendesain skenario mobilitas VANET.
- JOSM versi 1.5 (8800 en) sebagai editor peta dari OpenStreetMap.
- ns2.35 sebagai untuk melakukan simulasi skenario VANET.

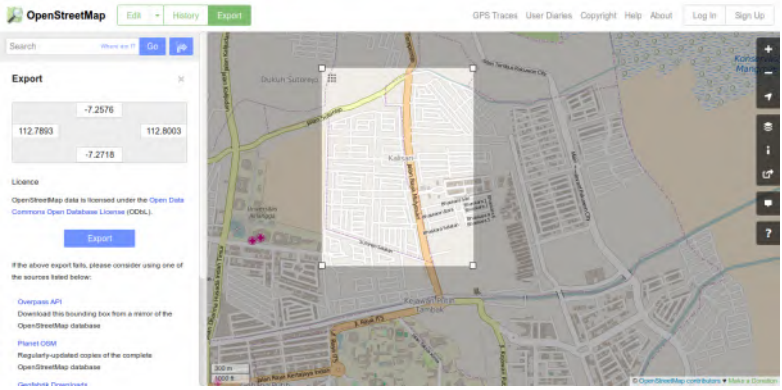
4.1.2 Lingkungan Perangkat Keras

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

- *Processor* Intel® Core™ i5-2410M CPU @ 2.30GHz × 4.
- 4 GB DDR3 RAM.
- Media penyimpanan sebesar 750 GB.

4.2 Implementasi Skenario Mobilitas

Dalam memgimplementasikan skenario mobilitas hal pertama yang harus dilakukan adalah mengekspor data dari OpenStreetMap seperti yang ditunjukkan oleh gamber 4.1.



Gambar 4.1: Proses ekspor peta dari OpenStreetMap

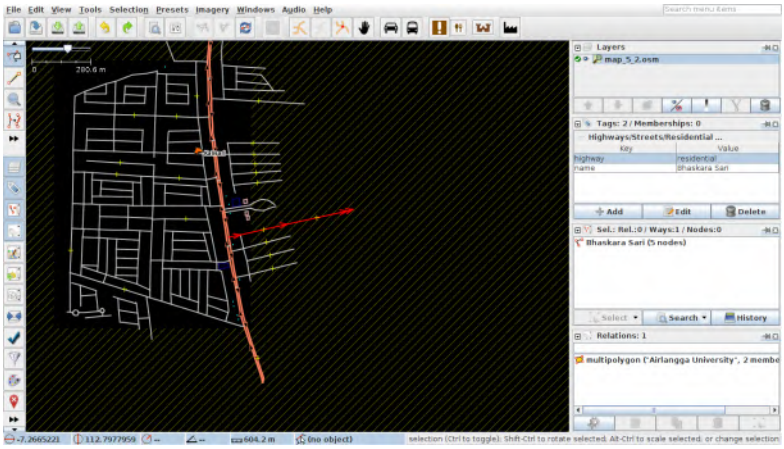
Peta yang telah di ekspor dari OpenStreetMap kemudian dibuka menggunakan JOSM untuk di*edit* lebih lanjut. Jalan yang tidak digunakan dan jalan yang terisolasi dihapus dari peta. Beberapa jalan baru juga ditambahkan agar tidak ada jalan yang buntu. *Screenshot* dari proses *editing* dapat dilihat pada gambar 4.2 dan 4.3.

Setelah proses *editing* dari peta selesai, peta tersebut kemudian dikonversi menjadi *file* dengan format .net.xml menggunakan `netconvert`. Perintah konversi dapat dilihat pada gambar 4.4.

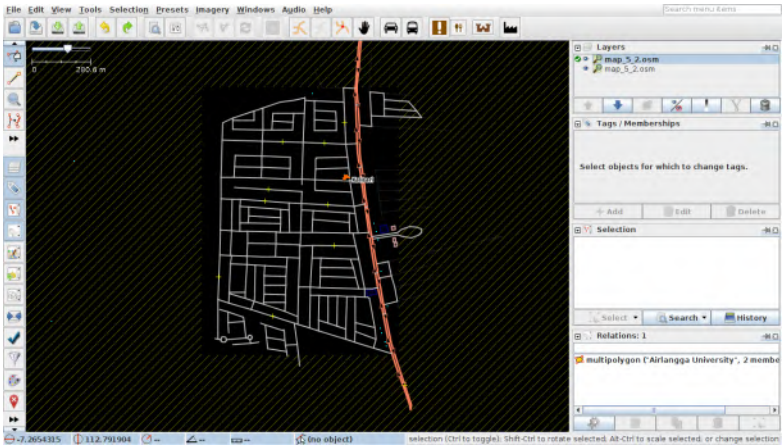
```
netconvert --remove-edges.by-vclass pedestrian,rail
 ↪  --try-join-tls --osm-files map.osm --output-file
 ↪  map.net.xml --remove-edges.isolated --type-files
 ↪  osmNetconvert.typ.xml
```

Gambar 4.4: Perintah untuk melakukan konversi *file* .osm ke .net.xml

Setelah peta selesai dikonversi tahap selanjutnya adalah membuat titik awal dan titik akhir dari setiap kendaraan yang masuk dalam



Gambar 4.2: Proses *editing* peta pada JOSM



Gambar 4.3: Hasil *editing* peta menggunakan JOSM

simulasi. Untuk melakukan itu digunakan *tool* `randomTrips.py`. Perintah yang digunakan pada `randomTrips.py` dapat dilihat pada gambar 4.5.

```
python $SUMO_HOME/tools/randomTrips.py -n map.net.xml
 ↪  --seed=42 --fringe-factor 5 -r map.passenger.rou.xml -e
 ↪   300 --vehicle-class passenger --vclass passenger
 ↪  --prefix veh --min-distance 300 --trip-attributes
 ↪   'departLane="best"'
```

Gambar 4.5: Perintah untuk membuat titik awal dan akhir untuk setiap kendaraan

File map.passenger.rou.xml yang merupakan *output* dari `randomTrips.py`. Untuk merubah standar deviasi kecepatan kendaraan, dalam tag **`<vType>`** perlu ditambahkan attribut `speedDev` yang berisikan nilai deviasi kecepatan kedaraan. Setelah atribut tersebut ditambahkan *file* tersebut kemudian diproses menggunakan `route2trips.py` untuk membuat jalur yang digunakan kendaraan untuk mencapai titik tujuannya. Gambar 4.6 menunjukkan perintah yang menggunakan `route2trips.py`.

```
python $SUMO_HOME/tools/route2trips.py map.passenger.rou.xml
 ↪   > map.passenger.trips.xml
```

Gambar 4.6: Perintah untuk membuat rute kendaraan

Setelah *file* peta dan trips tersedia maka *file* terakhir yang diperlukan adalah skrip skenario untuk SUMO yang berekstensi `.sumocfg`. Contoh *file* tersebut dapat dilihat pada gambar 4.7. *File* `.sumocfg` kemudian ditaruh pada tempat yang sesuai dengan *relative path* terhadap *file* yang telah dideskripsikan didalamnya. Untuk melihat tampilan grafis simulasi lalu lintas *file* .sumocfg dapat dibuka menggunakan `sumo-gui`. Cuplikan pergerakan dapat dilihat pada gambar 4.8.

Untuk mendapatkan hasil dari simulasi SUMO dalam format xml, saat simulasi perlu diberikan parameter `--fcd-output`. Se-

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/sumoConfigura

<input>
<net-file value="map.net.xml"/>
<route-files value="map.passenger.trips.xml"/>
</input>

<processing>
<ignore-route-errors value="true"/>
</processing>

<report>
<verbose value="true"/>
<no-step-log value="true"/>
</report>

</configuration>
```

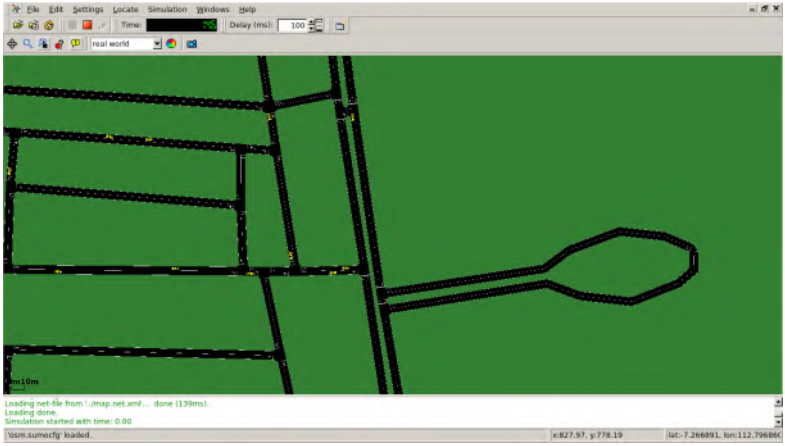
Gambar 4.7: Isi dari *file* .sumocfg

telah itu hasil simulasi dikonversi menjadi *file* skenario mobilitas NS-2 menggunakan `traceExporter.py`. Perintah untuk simulasi dan konversi dapt dilihat pada gambar 4.9.

4.3 Implementasi Protokol HM-AOMDV

Protokol HM-AOMDV merupakan turunan dari protokol AOMDV. Perubahan yang dilakukan pada protokol AODV antara lain adalah:

- Struktur data HELLO
- Struktur data RREP
- Penaganan paket HELLO
- Penanganan paket RREP



Gambar 4.8: Cuplikan simulasi pada sumo-gui

```
sumo -c osm.sumocfg --fcd-output sumoTrace.xml

python $SUMO_HOME/tools/traceExporter.py --fcd-input
 ↪  sumoTrace.xml --ns2mobility-output ns2mobility.tcl
 ↪  --ns2config-output ns2config.tcl --ns2activity-output
 ↪  ns2activity.tcl
```

Gambar 4.9: Perintah simulasi SUMO dan konversi ke *file* tcl

Kode implementasi dari protokol AOMDV pada NS-2 versi 2.35 berada pada direktori `ns-2.35/aomdv`. Di dalam direktori tersebut terdapat beberapa file yaitu `aomdv.cc`, `aomdv.h`, `aomdv_logs.cc`, `aomdv_logs.h`, `aomdv_packet.h`, `aomdv_rqueue.cc`, `aomdv_rqueue.h`, `aomdv_rtable.cc` dan `aomdv_rtable.h`. Dalam implementasi ini, *file* yang akan dimodifikasi adalah `aomdv_packet.h` untuk mengubah struktur paket *routing*, `aomdv.cc` dan `aomdv.h` untuk merubah perilaku *node* saat memproses sebuah paket, serta `aomdv_rtable.cc` dan `aomdv_rtable.h` untuk mengubah perilaku

dari *routing table* pada AOMDV.

Pada bagian ini penulis akan menjelaskan langkah - langkah dalam mengimplementasikan protokol HM-AOMDV degan menggunakan protokol AOMDV sebagai dasarnya.

4.3.1 Implementasi Struktur Data HELLO dan RREP

Pada implementasi protokol AODV maupun AOMDV di NS-2, paket HELLO dan RREP diimplementasikan sebagai satu buah *struct* yang sama. Untuk mengklasisifikasikan apakah paket tersebut merupakan paket HELLO ataupun RREP, sebuah *node* akan membandingkan *FLAG* yang terdapat pada paket tersebut. Gambar 4.10 menunjukkan bagaimana HELLO dan RREP diimplementasikan pada NS-2.

Berdasarkan tabel 2.1 data yang ditambahkan pada HELLO dan RREP masing - masing adalah *speed* dan *mobility factor*. Untuk menampung nilai tersebut maka dua buah variabel baru yaitu `rp_speed` dan `rp_factor` ditambahkan kedalam *struct* yang sudah disediakan. Kode implementasi HELLO dan RREP pada HM-AOMDV dapat dilihat pada lampiran A.5.

4.3.2 Modifikasi *Class* AOMDV

Seperti penjelasan yang diberikan pada subseksi 2.2.3 *mobility factor* dihitung berdasarkan nilai kecepatan *neighbor* yang didapatkan melalui HELLO dan kemudian di-*broadcast* kembali saat mengirimkan RREP. Selain itu dibutuhkan juga nilai Mf_{pre} dan N. Oleh karena itu hal pertama yang harus disiapkan adalah sebuah variabel pada setiap *node* yang akan menyimpan nilai Mf_{pre}, kecepatan dan N.

Untuk memiliki variabel yang terdapat pada semua *node* dan mudah untuk diakses saat pengolahan paket, maka variabel tersebut harus ditaruh kedalam *class* AOMDV. Definisi dari *class* AOMDV terdapat pada *file* aomdv.h. Pada bagian *protected* tambahkan tiga

```
struct hdr_aomdv_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;

    // used to compute route discovery latency
    // AOMDV code
    u_int32_t       rp_bcast_id; // Broadcast ID of the
 ↪  corresponding RREQ
    nsaddr_t        rp_first_hop;

    inline int size() {
        int sz = 0;
        sz = 6*sizeof(u_int32_t);
        // AOMDV code
        if (rp_type == AOMDVTYPE_RREP) {
            sz += sizeof(u_int32_t);    // rp_bcast_id
            sz += sizeof(nsaddr_t);     // rp_first_hop
        }
        assert (sz >= 0);
        return sz;
    }

};
```

Gambar 4.10: Implementasi *struct* HELLO dan RREP pada NS-2

buah variabel `int hello_counter`, `float mobility_factor` dan `float vehicle_speed`. Selain itu, diperlukan sebuah objek `MobileNode` untuk mendapatkan kecepatan dari kendaraan. Sehingga pada *class* AOMDV ditambahkan juga variabel `MobileNode* iNode` dan pada *file* `aomdv.h` ditambahkan juga header yang diperlukan untuk `MobileNode` dengan cara menambahkan baris `#include <mobilenode.h>` setelah *include guard*. Gambaran mengenai perubahan yang dilakukan pada *file* `aomdv.h` dapat dilihat pada gambar 4.11.

```
#ifndef __aomdv_h__
#define __aomdv_h__

//Another include
include <mobilenode.h> //MobileNode

//Another class definition
class AOMDV: public Agent{
    //Another method and attribute definition
    protected:
        int hello_counter;
        float mobility_factor;
        float vehicle_speed;
        MobileNode* iNode;
        float weight_factor;
};
```

Gambar 4.11: Gambaran mengenai perubahan yang dilakukan pada `aomdv.h`

4.3.3 Modifikasi Proses Pengiriman HELLO

Setiap *node* akan mengirimkan HELLO setiap HELLO_INTERVAL. *Method* yang menjalankan pengiriman paket HELLO adalah `void AOMDV::sendHello(void)`.

Sebelum mengirimkan kecepatan menggunakan HELLO, nilai kecepatan harus didapatkan terlebih dahulu. Untuk mendapatkan kecepatan kendaraan saat ini dibutuhkan sebuah objek `MobileNode` yang menunjuk ke *node* ini. *Pointer* yang telah didefinisikan pada `aomdv.h` dapat digunakan untuk mendapatkan objek tersebut dengan menggunakan kode `iNode = (MobileNode*) (Node::get_node_by_address(index))`. Kemudian kecepatan kendaraan bisa didapatkan dengan menggunakan `vehicle_speed = ((MobileNode *) iNode)->speed()`.

Selain untuk mengirim paket HELLO, *method* `void AOMDV::sendHello(void)` juga menandakan bahwa waktu sebanyak HELLO_INTERVAL telah berlalu. Sehingga pada method ini nilai dari Mf_{pre} dan N akan di-*reset*. Setelah itu nilai kecepatan kendaraan dapat dimasukkan ke dalam paket yang akan dikirim. Gambaran perubahan yang dilakukan dapat dilihat pada gambar 4.12. Sedangkan perubahan secara lengkap yang dilakukan pada *method* tersebut dapat dilihat pada lampiran A.6.

4.3.4 Modifikasi Proses Penerimaan HELLO

Method yang bertugas untuk mengatur proses penerimaan paket HELLO adalah `void AOMDV::recvHello (Packet *p)`. Pada HM-AOMDV, proses yang dilakukan pada saat penerimaan HELLO adalah menghitung nilai *mobility factor* dari *node* penerima. Rumus untuk menghitung *mobility factor* dapat dilihat pada persamaan 2.1.

Variabel - variabel yang diperlukan untuk menghitung nilai *mobility factor* telah disediakan pada paket HELLO dan *class* AOMDV. Implementasi dari *method* `AOMDV::recvHello` dapat dilihat pada lampiran A.7.

```
1  void
2  AOMDV::sendHello() {
3      //Allocate packet
4      Packet *p = Packet::alloc();
5      struct hdr_cmn *ch = HDR_CMN(p);
6      struct hdr_ip *ih = HDR_IP(p);
7      struct hdr_aomdv_reply *rh = HDR_AOMDV_REPLY(p);
8
9      //HM-AOMDV code
10     iNode = (MobileNode*)
   ↪   (Node::get_node_by_address(index));
11     vehicle_speed = ((MobileNode *) iNode)->speed();
12     hello_counter = 0;
13
14     rh->rp_speed = vehicle_speed;
15     mobility_factor = 0;
16     //----------------------------
17
18     //Set other parameter in packet and send it
19 }
```

Gambar 4.12: Gambaran perubahan pada *method* `AOMDV::sendHello`

4.3.5 Modifikasi Proses Penerimaan RREP

Modifikasi dari proses penerimaan RREP pada HM-AOMDV didasarkan pada *flowchart* yang terdapat pada gambar 2.5. *Pseudo-code* yang mengimplementasikan *flowchart* tersebut dapat dilihat pada gambar 4.13, sedangkan implementasi dalam kode C++ dapat dilihat dalam lampiran A.8 dan A.9.

4.4 Implementasi Metrik Analisis

Output dari simulasi skenario pada NS-2 adalah sebuah *tracefile* yang berisikan data berupa *plain text*. Dari data tersebut penulis

Input: $rrep$
$seqnumber \leftarrow$ sequence of $rrep$;
$Mf \leftarrow$ mobility factor of $rrep$;
$destination \leftarrow$ routing table of AOMDV;
$Mf_{last} \leftarrow$ last mobility factor of $rtable$;
$rt\_entry \leftarrow rt\_lookup(rtable, destination)$;
if $rt\_entry == 0$ **then**
    $rt\_entry \leftarrow rt\_add(rtable, destination)$;
end
$seqnumber_{last} \leftarrow$ sequence number of rt_entry;
if $seqnumber > seqnumber_{last}$ **then**
    rt_clear(rt_entry);
    rt_add$(rt\_entry, rrep)$;
    $Mf_{last} \leftarrow Mf$;
end
if *seqnumber* $==$ *seqnumber*$_{last}$ & $Mf >= Mf_{last}$ **then**
    rt_add$(rt\_entry, rrep)$;
    if $Mf > Mf_{last}$ **then**
        $Mf_{last} \leftarrow Mf$;
    end
end

Gambar 4.13: Pseudocode dalam memproses RREP yang diterima

dapat menganalisa performa dari *routing protocol* selama simulasi. Tiga buah metrik yang penulis analisa adalah *packet delivery ratio, normalized routing overhead* dan *error rate*.

4.4.1 Implementasi PDR

Pada buku Tugas Akhir ini bagian 2.7.3 telah ditunjukkan contoh dari struktur data *event* yang dicatat pada *tracefile* NS-2. Kemudian pada persamaan 3.1 telah dijelaskan rumus untuk menghitung PDR. Skrip awk untuk menghitung PDR berdasarkan kedua informasi tersebut dapat dilihat pada lampiran A.2.

Pada skrip tersebut penulis hanya mem-*filter* baris yang berisikan `AGT` karena kata tersebut menunjukkan *event* yang bersangkutan dengan paket komunikasi data. Dengan menggunakan karakter pertama dari baris yang telah di-*filter* jumlah dari paket yang dikirim dan diterima dapat dihitung. Setelah itu, nilai dari PDR dapat dihitung dengan menggunakan persamaan yang telah dijelaskan.

Contoh perintah untuk memanggil skrip tcl untuk menganalisis tracefile dan *output*nya dapat dilihat masing - masing pada gambar 4.14 dan 4.15.

```
awk -f PDR.awk hm-aomdv.tr
```

Gambar 4.14: Perintah untuk menjalankan skrip awk untuk menghitung PDR

```
cbr s:541 r:473, r/s Ratio:0.8743, f:2028
```

Gambar 4.15: *Output* dari skrip PDR.awk

4.4.2 Implementasi *Normalized Routing Overhead* dan *Error Rate*

Penghitungan dari *normalized routing overhead* didasarkan pada persamaan 3.2. Kode implementasi dari persamaan tersebut dapat dilihat pada lampiran A.3.

Analisis dimulai dengan menghitung jumlah paket komunikasi data yang diterima oleh *node* tujuan. Setelah itu dihitung jumlah paket RREQ, RREP dan RRER yang dikirim dan di=*forward*. Kemudian *normalized routing overhead* dapat dihitung dengan membandingkan kedua nilai tersebut. *Error rate* juga sekalian dihitung dengan mendapatkan jumlah RRER yang dikirim pada simulatsi tersebut.

Contoh perintah untuk menjalankan kode yang menghitung *normalized routing overhead* dapat dilihat pada gambar 4.16. Tampilan *output* yang diberikan dapat dilihat pada gambar 4.17.

```
awk -f overhead.awk hm-aomdv.tr
```

Gambar 4.16: Perintah untuk menjalankan skrip awk untuk menghitung *normalized routing overhead* dan *error rate*

```
Normalized Overhead: xxx
Error Rate: xxx
```

Gambar 4.17: *Output* dari skrip overhead.awk

4.4.3 Implementasi *End-to-End Delay* (E2E)

Perhitungan E2E didasarkan oleh rumus yang dijelaskan pada persamaan 3.3. Pada perhitungan E2E, kunci yang perlu diperhatikan dari *tracefile* adalah variabel *action* pada kolom pertama, waktu

pada kolom kedua, *layer* pada kolom ke 4, id paket pada kolom ke 6 dan tipe paket pada kolom ke 7. Kode implementasi dari *pseudo-code* tersebut dapat dilihat pada lampiran A.4.

Contoh perintah untuk memanggil skrip tcl untuk menganilisis *tracefile* dan *outputnya* dapat dilihat masing-masing pada gambar 4.18 dan 4.19.

```
awk -f e2e.awk hm_aomdv.tr
```

Gambar 4.18: Perintah untuk menjalankan skrip awk untuk menghitung E2E

```
Average delay: 0.0034
```

Gambar 4.19: *Output* dari skrip e2e.awk

4.5 Implementasi Simulasi pada NS-2

Untuk melakukan simulasi VANET menggunakan NS-2 hal pertama yang harus dilakukan adalah mendeskripsikan lingkungan simulasi pada sebuah *file* OTcl. *File* ini berisikan konfigurasi setiap *node* dan juga langkah langkah yang dilakukan selama simulasi. Contoh potongan pengaturan *node* dapat dilihat pada gambar 2.10.

Pengaturan yang dilakukan pada skenario tersebut antara lain lokasi *tracefile*, ukuran topografi, konfigurasi *node* dan konfigurasi pengiriman data. Kode implementasi skenario yang digunakan pada Tugas Akhir ini dapat dilihat pada lampiran A.1. Tabel 4.1 merupakan penjelasan dari bagian `node-config` dalam kode skenario simulasi.

Tabel 4.1: Penjelasan dari parameter node-config

Parameter	Value	Penjelasan
llType	LL	Menggunakan *link layer* standar
mactType	Mac/802_11	Menggunakan tipe MAC 802.11 karena komunikasi data bersifat wireless
ifqType	Queue/DropTail/PriQueue	Menggunakan *priority queue* sebagai antrian paket dan paket yang dihapus saat antrian penuh adalah paket yang paing baru
ifqLen	50	Jumlah maksimal paket pada antrian
antType	Antenna/Omni Antenna	Jenis antena yang digunakan adalah *omni antenna*
propType	Propagation/Two Ray Ground	Tipe propagasi sinyal wireless adalah *two ray ground*
phyType	Phy/Wireless Phy	Komunikasi menggunakan media *wireless*
topo Instance	$topo	Topologi yang digunakan daat menjalankan skenario
agentTrace	ON	Aktifkan pencatatan aktifitas dari agen *routing protocol*
router Trace	ON	Aktifkan pencatatan pada aktifitas *routing protocol*
mac Trace	OFF	Matikan pencatatan MAC *layer* pada *trace file*
movement Trace	OFF	Matikan pencatatan pergerakan *node*
channel	Channel/Wireless Channel	Channel komunikasi yang digunakan

Setelah semua *file mobility, activity* dan skenario telah siap, maka langkah selanjutnya adalah menjalankan skenario yang sudah dibuat menggunakan NS-2. Untuk menjalankan skenario simulasi masukkan perintah `ns <nama-file>`. Setelah simulasi selesai dijalankan maka hasil simulasi dapat dianalisis dari *tracefile* yang telah ditentukan pada *file* skenario. Isi dari *tracefile* adalah catatan menganai *event* yang dialami paket data yang beredar dalam simulasi, baik itu saat dikirim, diterima maupun saat di-*forward*.

Halaman ini sengaja dikosongkan

BAB 5

UJI COBA DAN EVALUASI

Pada bab ini akan dibahas mengenai uji coba dan evaluasi dari skenario simulasi yang telah dibuat.

5.1 Lingkungan Uji Coba

Uji coba dilakukan pada sebuah komputer dengan sistem operasi Linux yang telah terpasang NS-2 di dalamnya. Spesifikasi komputer yang digunakan dalam uji coba dapatdilihat pada tabel 5.1.

Tabel 5.1: Spesifikasi komputer yang digunakan

Komponen	Spesifikasi
CPU	*Processor* Intel® Core™ i5-2410M CPU @ 2.30GHz × 4
Sistem Operasi	Linux Ubuntu 14.04 64-bit
Memori	4 GB DDR3 RAM
Penyimpanan	750GB HDD

Pengujian dilakukan dengan menjalankan skenario yang telah dibuat pada NS-2. Kemudian tracefile dari simulasi tersbut dianalisis menggunakan skrip AWK yang telah dibuat sebelumnya.

5.2 Uji Coba Peta I

Peta I merupakan didasarkan oleh suatu lokasi di Surabaya. Hasil uji coba pada Peta I akan menunjukkan performa dari protokol HM-AOMDV dan AOMDV dalam menangani VANET pada topologi yang mendekati dunia nyata.

5.2.1 Parameter Uji Coba Peta I

Berikut ini adalah parameter yang digunakan untuk menjalankan simulasi:

Tabel 5.2: Parameter simulasi Peta I

No.	Parameter	Spesifikasi
1	Waktu simulasi	110 detik
2	Area simulasi	1600 m x 1800 m
3	Banyak kendaraan	120 - 230

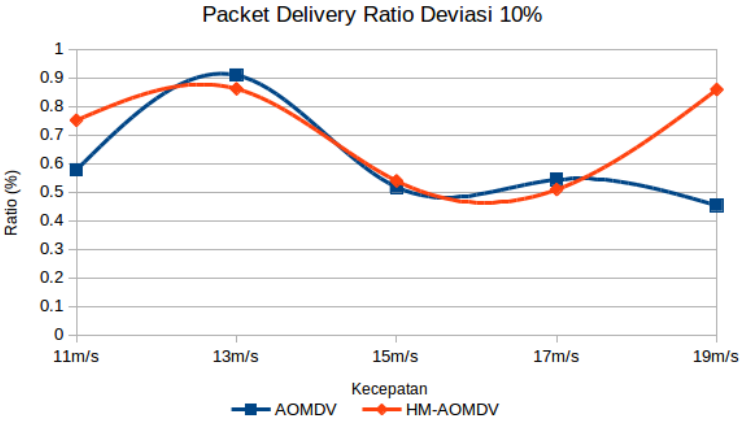
5.2.2 Hasil Uji Coba Peta I

Hasil analisis mengenai PDR, *normalized routing overhread, error rate* dan *end-to-end delay* dapat dilihat pada gambar 5.1, 5.2, 5.3 dan 5.4.

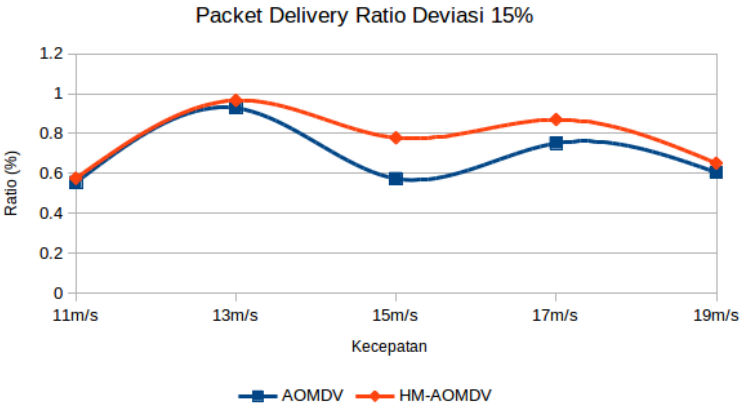
Berdasarkan hasil simulasi yang telah dilakukan, dapat dilihat bahwa PDR dari protokol HM-AOMDV cenderung lebih baik pada kecepatan dan deviasi yang sama jika dibandingkan dengan protokol AOMDV. Data yang menunjukkan performa protokol HM-AOMDV yang lebih rendah dibandingkan AOMDV adalah data pada kecepatan 13 m/s deviasi 10% dan kecepatan 17m/s deviasi 10%.

Pada kasus kecepatan 13 m/s deviasi 10% *error rate* dari protokol HM-AOMDV dan AOMDV bernilai sama, sedangkan nilai *normalized routing overherad* dari HM-AOMDV hanya lebih tinggi 0.074 poin. Hal ini berarti rendahnya PDR protokol HM-AOMDV dibandingkan protokol AOMDV pada kasus ini disebabkan oleh posisi putusnya jalur komunikasi. Dengan menggunakan kode pada lampiran A.10 untuk melihat rute yang dilalui oleh paket data, penulis dapat membuktikan bahwa penyebab PDR HM-AOMDV lebih rendah pada kasus ini adalah posisi putusnya jalur komunikasi.

Kasus pada kecepatan 17 m/s disebabkan oleh lebih tinggnya *error rate* pada HM-AOMDV sebanyak dua. Perbedaan *normalized*



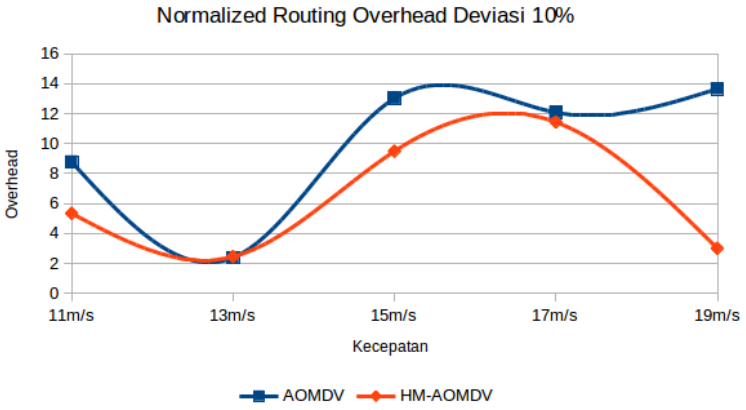
(a) Grafik PDR pada Deviasi 10%



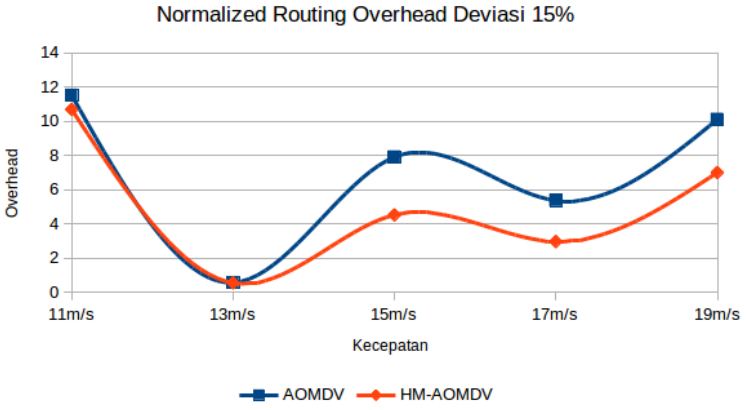
(b) Grafik PDR pada Deviasi 15%

Gambar 5.1: Grafik PDR dari Peta I

routing overhead pada kasus tersebut adalah sebanyak 0.647 poin lebih rendah dibandingkan AOMDV, juga berdasarkan dari *output*



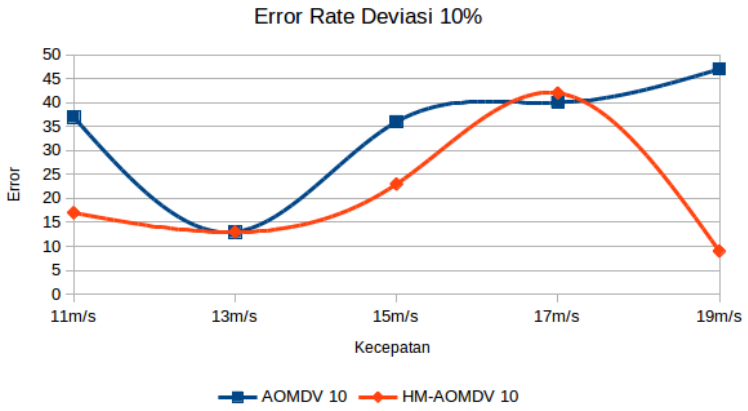
(a) *Overhead* pada deviasi 10%



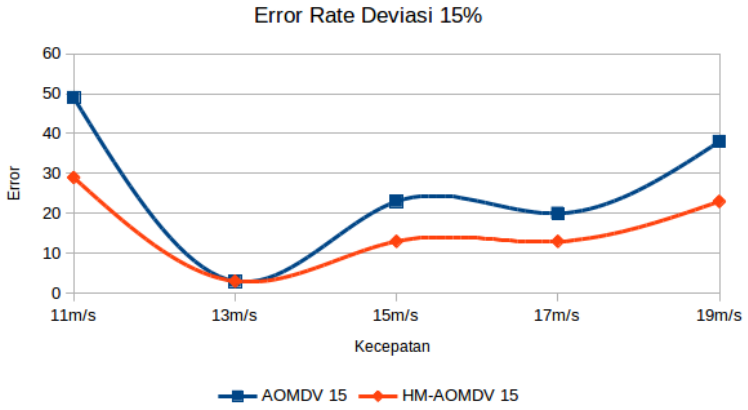
(b) *Overhead* pada deviasi 15%

Gambar 5.2: Grafik *normalized routing overhead* dari Peta I

kode pada lampiran A.10 dan data yang tercantum pada *mobility file*, *node* tempat terputusnya jaringan berjalan menjauh dari *next nop*. Hal ini menunjukkan bahwa arah dari *node* juga dapat diperhitungk-



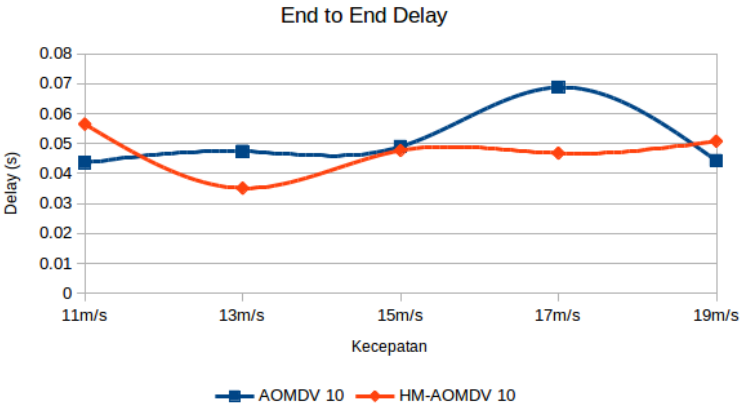
(a) Error pada deviasi 10%



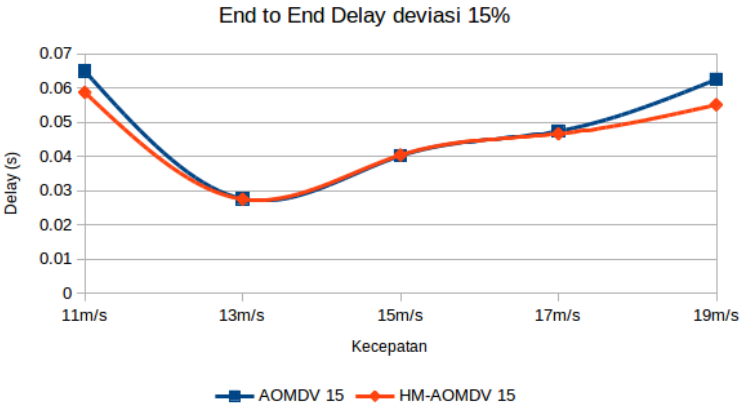
(b) Error pada deviasi 15%

Gambar 5.3: Grafik *error rate* dari Peta I

an untuk menghitung faktor pemilihan *next hop*.



(a) Delay pada deviasi 10%



(b) Delay pada deviasi 15%

Gambar 5.4: Grafik *end-to-end delay* dari Peta I

5.3 Uji Coba Peta II

Peta II merupakan peta grid yang dibangun berdasarkan peta grid dari SUMO.

5.3.1 Parameter Uji Coba Peta II

Berikut ini adalah parameter yang digunakan untuk menjalankan simulasi:

Tabel 5.3: Parameter simulasi Peta II

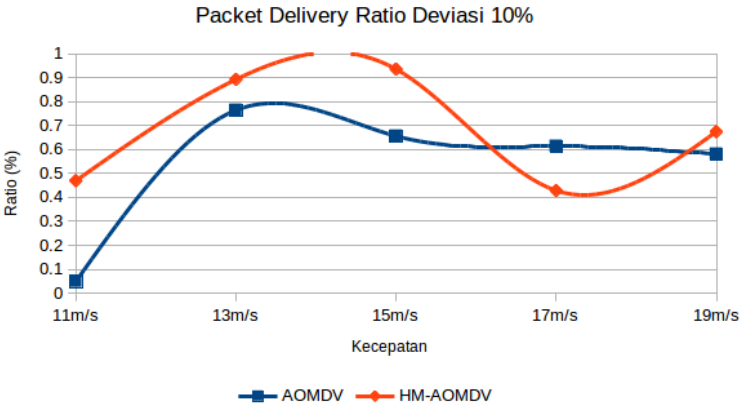
No.	Parameter	Spesifikasi
1	Waktu simulasi	110 detik
2	Area simulasi	1250 m x 1250 m
3	Banyak kendaraan	120 - 230
4	Jumlah persimpangan	6 x 6
5	Panjang antar persimpangan	250 m

5.3.2 Hasil Uji Coba Peta II

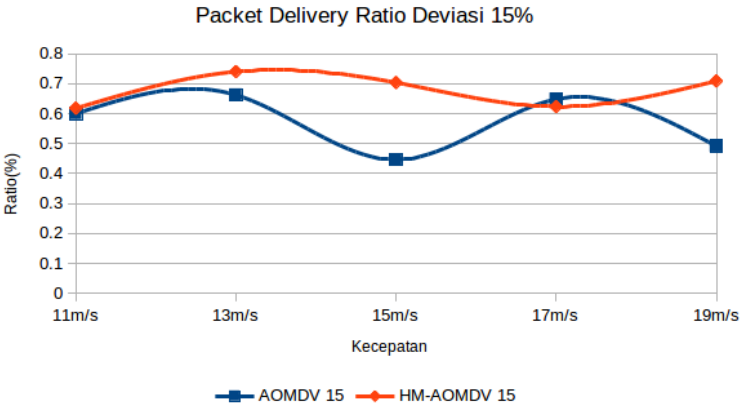
Hasil analisis mengenai PDR, *normalized routing overhread, error rate* dan *end-to-end delay* dapat dilihat pada gambar 5.5, 5.6, 5.7 dan 5.8.

Hasil simulasi yang diberikan oleh Peta II mengikuti tren yang mirip dengan Peta I, dimana PDR dari protokol HM-AOMDV cenderung lebih baik debandingkan dengan protokol AOMDV. Pada simulasi kali ini data yang menunjukkan performa protokol HM-AOMDV yang lebih rendah adalah pada kecepatan 17 m/s dengan deviasi 10% dan 15%.

Kasus pada kecepatan 17 m/s deviasi 10% mirip dengan kasus pada Peta I dengan kecepatan 13 m/s deviasi 10%. Pada kedua kasus tersebut protokol HM-AOMDV dan AOMDV memiliki *error rate* yang sama, tetapi protokol HM-AOMDV memiliki *normalized rou-*



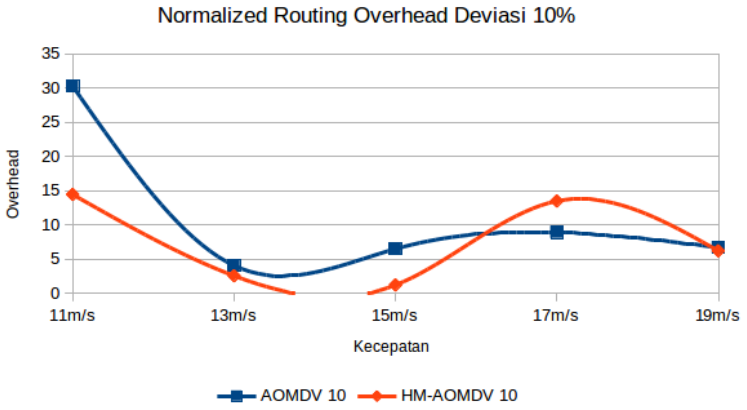
(a) Grafik PDR pada Deviasi 10%



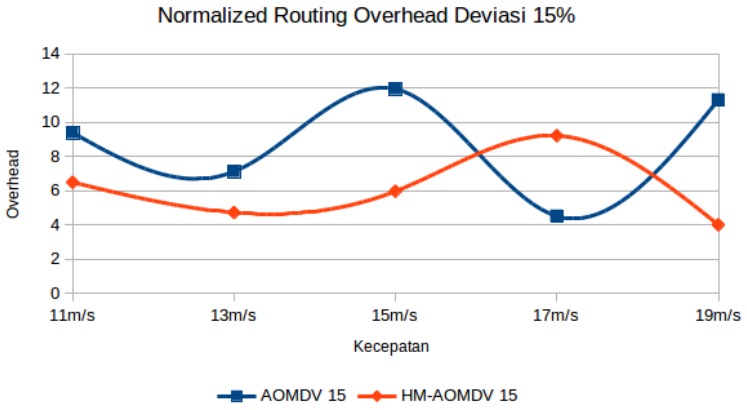
(b) Grafik PDR pada Deviasi 15%

Gambar 5.5: Grafik PDR dari Peta II

ting overhead yang lebih tinggi. Setelah diperiksa menggunakan skrip pada lampiran A.10 untuk melihat rute yang diambil oleh paket data, terlihat bahwa protokol HM-AOMDV menghabiskan banyak



(a) Overhead pada Deviasi 10%

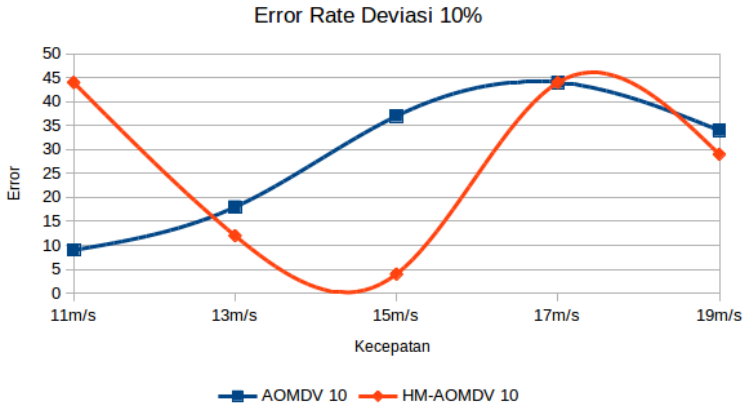


(b) Overhead pada Deviasi 15%

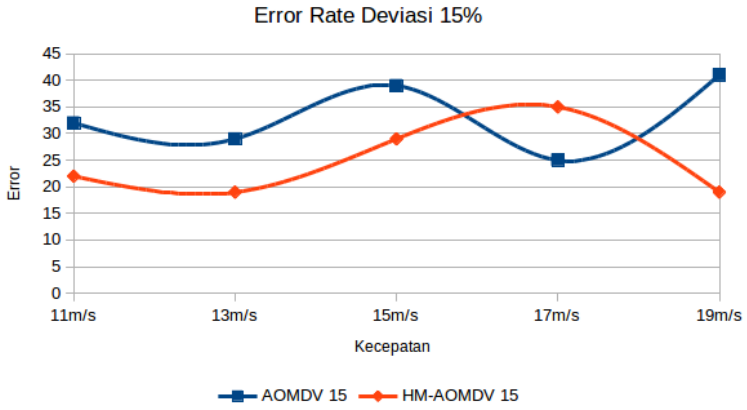
Gambar 5.6: Grafik *normalized routing overhead* dari Peta II

waktu untuk mencari rute alternatif yang masih valid. Oleh sebab itu lebih banyak paket yang akhirnya di-*drop* pada jaringan.

Pada kasus dengan kecepatan 17 m/s deviasi 15% penyebab ren-



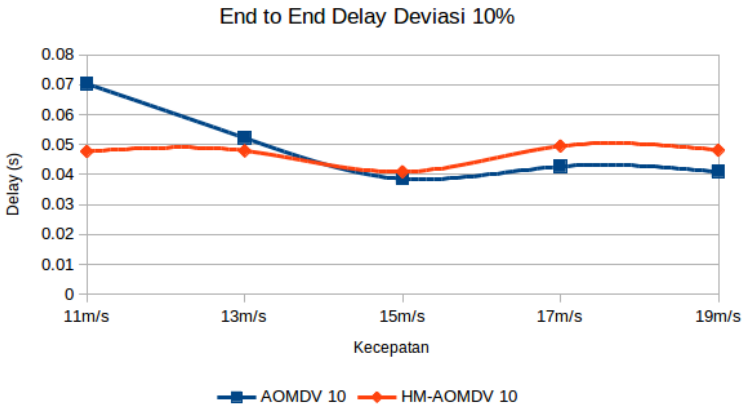
(a) Error pada Deviasi 10%



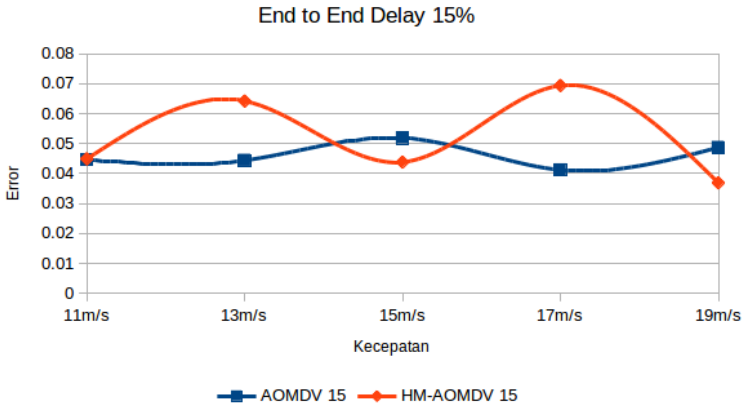
(b) Error pada Deviasi 15%

Gambar 5.7: Grafik *error rate* dari Peta II

dahnya PDR dapat dikorelasikan dengan lebih tinggnya *error rate*. Hal ini berarti protokol HM-AOMDV gagal mendapatkan jalur yang lebih stabil dari protokol AOMDV pada skenario tersebut. Se-



(a) Delay pada Deviasi 10%



(b) Delay pada Deviasi 15%

Gambar 5.8: Grafik *end-to-end delay* dari Peta II

telah dianalisis menggunakan kode pada lampiran A.10 dan *mobility file*, dapat dilihat bahwa protokol HM-AOMDV memilih *node* yang berjalan menjauh dari *node* selanjutnya. Hal ini menguatkan

lagi argumen yang diberikan pada percoban sebelumnya yaitu arah pergerakan dari *node* juga dapat digunakan untuk menghitung faktor pemilihan *next hop*.

A.1 Kode skenario NS-2

```
# ================
# Define options
# ================
set val(chan)    Channel/WirelessChannel;# channel type
set val(prop)    Propagation/TwoRayGround;# radio-propagation model
set val(netif)   Phy/WirelessPhy;# network interface type
set val(mac)     Mac/802_11;# MAC type
set val(ifq)     Queue/DropTail/PriQueue;# interface queue type
set val(ll)      LL;# link layer type
set val(ant)     Antenna/OmniAntenna;# antenna model
set val(ifqlen) 50;# max packet in ifq
set val(rp)      AOMDV;# routing protocol

set stopTime 280.0

set opt(config-path) [file dirname [info script]]
set opt(dirname) [lindex $argv 0]
set opt(output_file) [lindex $argv 1]
set opt(cf) $opt(config-path)
append opt(cf) "/"
append opt(cf) $opt(dirname)
append opt(cf) /ns2config.tcl;#contain total vehicle and grid size

source [file join $opt(cf)]
```

```
25 # ==============
26 # Main Program
27 # ==============
28 #
29 # Initialize Global Variables
30 #
31 set ns_ [new Simulator]
32 set tracefd [open $opt(output_file) w];#Trace file
33 $ns_ trace-all $tracefd
34
35 # set up topography object
36 set topo    [new Topography]
37 $topo load_flatgrid $opt(x) $opt(y)
38
39 #
40 # Create God
41 #
42 set all_node [expr {$opt(nn) + 2}]
43 create-god $all_node
44
45 # Configure node
46
47 set chan_1_ [new $val(chan)]
48 $ns_ node-config  -adhocRouting $val(rp) \
49     -llType $val(ll) \
50     -macType $val(mac) \
51     -ifqType $val(ifq) \
52     -ifqLen $val(ifqlen) \
53     -antType $val(ant) \
54     -propType $val(prop) \
55     -phyType $val(netif) \
56     -topoInstance $topo \
57     -agentTrace ON \
```

```
58      -routerTrace ON \
59      -macTrace OFF \
60      -movementTrace OFF \
61      -channel $chan_1_
62
63  for {set i 0} {$i < $opt(nn)} {incr i} {
64      set node_($i) [$ns_ node]
65      $node_($i) random-motion 0 ;# disable random
    ↪   motion
66  }
67
68  # Set source node
69  set source_ [$ns_ node]
70  $source_ random-motion 0
71  $source_ set X_ 36.37
72  $source_ set Y_ 1004.27
73  $source_ set Z_ 0
74  set udp_ [new Agent/UDP];# Using UDP transport
    ↪   layer
75  $ns_ attach-agent $source_ $udp_
76  set cbr_ [new Application/Traffic/CBR];# Constant
    ↪   bit rate sender
77  $cbr_ set packetSize_ 512
78  $cbr_ set rate_ 20kb
79  $cbr_ set maxpkts_ 20000
80  $cbr_ set random_ 1
81  $cbr_ attach-agent $udp_
82
83  # Set destination node
84  set destination_ [$ns_ node]
85  $destination_ random-motion 0
86  $destination_ set X_ 774.64
87  $destination_ set Y_ 694.61
```

```
88  $destination_ set Z_ 0
89  set null_ [new Agent/Null]
90  $ns_ attach-agent $destination_ $null_
91  $ns_ connect $udp_ $null_
92
93  $ns_ at 120.0 "$cbr_ start"
94  $ns_ at 230.0 "$cbr_ stop"
95
96  source [file join $opt(af)];# Activity file
97  source [file join $opt(mf)];# Mobility file
98
99  # Initiate other nodes
100 for {set i 0} {$i < $opt(nn) } {incr i} {
101     $ns_ at $stopTime "$node_($i) reset";
102 }
103
104 $ns_ at $stopTime "stop"
105 $ns_ at $stopTime "puts \"NS EXITING...\" ; $ns_
    ↪  halt"
106 proc stop {} {
107     global ns_ tracefd
108     $ns_ flush-trace
109     close $tracefd
110 }
111
112 puts "Starting Simulation..."
113 $ns_ run
```

A.2 Kode *awk* untuk menghitung *packet delivery ratio*

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

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

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

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

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

A.3 Kode *awk* untuk menghitung *normalized routing overhead* dan *error rate*

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

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

$0 ~/^s|f.* RTR.*REQUEST|REPLY|ERROR/ {
    routingPacket ++ ;
}

END {
    printf("Normalized Overhead: %.3f\n",
    routingPacket/recvLine);
}
```

A.4 Kode *awk* untuk menghitung *end-to-end delay*

```
BEGIN {
    total_delay = 0;
    recvPkts = 0;
    sentPkts[0] = 0;
}

$0 ~/^s.* AGT/{
    sentPkts[$6] = $2
}

$0 ~/^r.* AGT/{
    recvPkts ++ ;
    total_delay = total_delay + $2 - sentPkts[$6]
    delete sentPkts[$6]
}

END {
    printf("Average delay:%.4f\n",
    ↪ total_delay/recvPkts);
}
```

A.5 Kode implementasi HELLO dan RRER pada HM-AOMDV

```
struct hdr_aomdv_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;

    // used to compute route discovery latency
    // AOMDV code
    u_int32_t       rp_bcast_id; // Broadcast ID
    ↪ of the corresponding RREQ
    nsaddr_t        rp_first_hop;

    //HM-AOMDV code
    float rp_speed;
    float rp_factor;

    inline int size() {
        int sz = 0;
        sz = 6*sizeof(u_int32_t);
        // AOMDV code
        if (rp_type == AOMDVTYPE_RREP) {
```

```
26                 sz += sizeof(u_int32_t);    //
    ↪  rp_bcast_id
27                 sz += sizeof(nsaddr_t);     //
    ↪  rp_first_hop
28             }
29             assert (sz >= 0);
30             return sz;
31         }
32
33 };
```

A.6 Kode implementasi `AOMDV::sendHello`

```
1  void
2  AOMDV::sendHello() {
3      Packet *p = Packet::alloc();
4      struct hdr_cmn *ch = HDR_CMN(p);
5      struct hdr_ip *ih = HDR_IP(p);
6      struct hdr_aomdv_reply *rh =
   ↪   HDR_AOMDV_REPLY(p);
7
8  #ifdef DEBUG
9      fprintf(stderr, "sending Hello from %d at
   ↪   %.2f\n", index,
   ↪   Scheduler::instance().clock());
10 #endif // DEBUG
11
12     rh->rp_type = AOMDVTYPE_HELLO;
13     //rh->rp_flags = 0x00;
14     // AOMDV code
15     rh->rp_hop_count = 0;
16     rh->rp_dst = index;
17     rh->rp_dst_seqno = seqno;
18     rh->rp_lifetime = (1 + ALLOWED_HELLO_LOSS) *
   ↪   HELLO_INTERVAL;
19     //HM-AOMDV code
20     iNode = (MobileNode*)
   ↪   (Node::get_node_by_address(index));
21     vehicle_speed = ((MobileNode *)
   ↪   iNode)->speed();
22     hello_counter = 0;
23     double xpos_t, ypos_t, zpos_t;
24
25     rh->rp_speed = vehicle_speed;
```

```
26      mobility_factor = 0;
27      //-----------------------------
28
29      // ch->uid() = 0;
30      ch->ptype() = PT_AOMDV;
31      ch->size() = IP_HDR_LEN + rh->size();
32      ch->iface() = -2;
33      ch->error() = 0;
34      ch->addr_type() = NS_AF_NONE;
35      ch->prev_hop_ = index;          // AODV hack
36
37      ih->saddr() = index;
38      ih->daddr() = IP_BROADCAST;
39      ih->sport() = RT_PORT;
40      ih->dport() = RT_PORT;
41      ih->ttl_ = 1;
42
43      Scheduler::instance().schedule(target_, p,
    ↪  0.0);
44  }
```

A.7 Kode implementasi `AOMDV::recvHello`

```
1  void
2  AOMDV::recvHello(Packet *p) {
3      // AOMDV code
4      struct hdr_ip *ih = HDR_IP(p);
5      struct hdr_aomdv_reply *rp =
   ↪  HDR_AOMDV_REPLY(p);
6      AOMDV_Neighbor *nb;
7
8      nb = nb_lookup(rp->rp_dst);
9      if(nb == 0) {
10         nb_insert(rp->rp_dst);
11     }
12     else {
13         nb->nb_expire = CURRENT_TIME +
14         (1.5 * ALLOWED_HELLO_LOSS *
   ↪  HELLO_INTERVAL);
15     }
16
17     //HM-AOMDV code
18     float r = fabs(vehicle_speed - rp->rp_speed);
19     int counter = ++hello_counter;
20     if(counter == 0) counter = 1;
21
22     //If packet is received the same time when
   ↪  node send HELLO
23     float mf_pre = mobility_factor;
24     mobility_factor = (pow(1.0 + r, weight_factor)
   ↪  + (mf_pre * (float)(counter - 1))) / (float)
   ↪  counter;
25     //---------------------------------------
26     // AOMDV code
```

```
27      // Add a route to this neighbor
28      ih->daddr() = index;
29      rp->rp_src = ih->saddr();
30      rp->rp_first_hop = index;
31      recvReply(p);
32
33  }
```

A.8 Kode implementasi `AOMDV::recvReply`

```
1  void
2  AOMDV::recvReply(Packet *p) {
3      struct hdr_cmn *ch = HDR_CMN(p);
4      struct hdr_ip *ih = HDR_IP(p);
5      struct hdr_aomdv_reply *rp =
   ↪  HDR_AOMDV_REPLY(p);
6      aomdv_rt_entry *rt0, *rt;
7      AOMDVBroadcastID* b = NULL;
8      AOMDV_Path* forward_path = NULL;
9
10 #ifdef DEBUG
11     fprintf(stderr, "%d - %s: received a REPLY\n",
   ↪  index, __FUNCTION__);
12 #endif // DEBUG
13
14
15     /* If I receive a RREP with myself as source -
   ↪  drop packet (should not occur).
16 Comment: rp_dst is the source of the RREP, or
   ↪  rather the destination of the RREQ. */
17     if (rp->rp_dst == index) {
18         Packet::free(p);
19         return;
20     }
21
22     /*
23      *  Got a reply. So reset the "soft state"
   ↪  maintained for
24      *  route requests in the request table. We
   ↪  don't really have
```

```
25          *  have a separate request table. It is just
    ↪  a part of the
26          *  routing table itself.
27          */
28         // Note that rp_dst is the dest of the data
    ↪  packets, not the
29         // the dest of the reply, which is the src of
    ↪  the data packets.
30
31         rt = rtable.rt_lookup(rp->rp_dst);
32
33         /*
34          *  If I don't have a rt entry to this host...
    ↪  adding
35          */
36         if(rt == 0) {
37             rt = rtable.rt_add(rp->rp_dst);
38         }
39
40         /* If RREP contains more recent seqno for
    ↪  (RREQ) destination -
41            delete all old paths and add the new
    ↪  forward path to (RREQ) destination */
42         if (rt->rt_seqno < rp->rp_dst_seqno) {
43             rt->rt_seqno = rp->rp_dst_seqno;
44             rt->rt_advertised_hops = INFINITY;
45             rt->path_delete();
46             rt->rt_flags = RTF_UP;
47             /* Insert forward path to RREQ
    ↪  destination. */
```

```
48          forward_path = rt->path_insert(rp->rp_src,
    ↪ rp->rp_hop_count+1, CURRENT_TIME +
    ↪ rp->rp_lifetime, rp->rp_first_hop,
    ↪ rp->rp_speed_factor); //HM-AOMDV change
49          // CHANGE
50          rt->rt_last_hop_count =
    ↪ rt->path_get_max_hopcount();
51          // CHANGE
52      }
53      /* If the sequence number in the RREP is the
    ↪ same as for route entry but
54         with a smaller hop count - try to insert
    ↪ new forward path to (RREQ) dest. */
55      else if ( (rt->rt_seqno == rp->rp_dst_seqno)
    ↪ &&
56              //(rt->rt_advertised_hops >
    ↪ rp->rp_hop_count)
57              (rt->rt_mfactor <=
    ↪ rp->rp_speed_factor)) { //HM-AOMDV change
58
59          assert (rt->rt_flags == RTF_UP);
60          /* If the path already exists - increase
    ↪ path lifetime */
61          if ((forward_path =
    ↪ rt->disjoint_path_lookup(rp->rp_src,
    ↪ rp->rp_first_hop))) {
62              assert (forward_path->hopcount ==
    ↪ (rp->rp_hop_count+1));
63              forward_path->expire =
    ↪ max(forward_path->expire, CURRENT_TIME +
    ↪ rp->rp_lifetime);
64          }
```

```
65          /* If the path does not already exist,
    ↪ there is room for it and it
66                 does not differ too much in length - we
    ↪ add the path */
67          else if (
    ↪ rt->new_disjoint_path(rp->rp_src,
    ↪ rp->rp_first_hop) &&
68                      (rt->rt_num_paths_ <
    ↪ aomdv_max_paths_) &&
69                      ((rp->rp_hop_count+1) -
    ↪ rt->path_get_min_hopcount() <=
    ↪ aomdv_prim_alt_path_len_diff_)
70                      ) {
71                  /* Insert forward path to RREQ
    ↪ destination. */
72                  forward_path =
    ↪ rt->path_insert(rp->rp_src,
    ↪ rp->rp_hop_count+1, CURRENT_TIME +
    ↪ rp->rp_lifetime, rp->rp_first_hop,
    ↪ rp->rp_speed_factor);
73                  // CHANGE
74                  rt->rt_last_hop_count =
    ↪ rt->path_get_max_hopcount();
75                  // CHANGE
76          }
77          /* Path did not exist nor could it be
    ↪ added - just drop packet. */
78          else {
79              Packet::free(p);
80              return;
81          }
82      }
```

```
83      /* The received RREP did not contain more
    ↪   recent information
84         than route table - so drop packet */
85      else {
86          Packet::free(p);
87          return;
88      }
89      /* If route is up */
90      if (rt->rt_flags == RTF_UP) {
91          // Reset the soft state
92          rt->rt_req_timeout = 0.0;
93          rt->rt_req_last_ttl = 0;
94          rt->rt_req_cnt = 0;
95
96          if (ih->daddr() == index) {
97              // I am the RREP destination
98
99  #ifdef DYNAMIC_RREQ_RETRY_TIMEOUT // This macro
    ↪   does not seem to be set.
100             if (rp->rp_type == AOMDVTYPE_RREP) {
101                 rt->rt_disc_latency[rt-
    ↪   >hist_indx] = (CURRENT_TIME -
    ↪   rp->rp_timestamp)
102                         / (double)
    ↪   (rp->rp_hop_count+1);
103                 // increment indx for next time
104                 rt->hist_indx = (rt->hist_indx +
    ↪   1) % MAX_HISTORY;
105             }
106 #endif // DYNAMIC_RREQ_RETRY_TIMEOUT
107         }
108
109         /*
```

```
110             * Find out whether any buffered packet
    ↪ can benefit from the
111             * forward route.
112             */
113            Packet *buffered_pkt;
114            while ((buffered_pkt =
    ↪ rqueue.deque(rt->rt_dst))) {
115                if (rt && (rt->rt_flags == RTF_UP)) {
116                    forward(rt, buffered_pkt,
    ↪ NO_AOMDV_DELAY);
117                }
118            }
119
120        }
121        /* If I am the intended receipient of the RREP
    ↪ nothing more needs
122           to be done - so drop packet. */
123        if (ih->daddr() == index) {
124            Packet::free(p);
125            return;
126        }
127        /* If I am not the intended receipient of the
    ↪ RREP - check route
128           table for a path to the RREP dest (i.e. the
    ↪ RREQ source). */
129        rt0 = rtable.rt_lookup(ih->daddr());
130        b = id_get(ih->daddr(), rp->rp_bcast_id); //
    ↪ Check for <RREQ src IP, bcast ID> tuple
131
132    #ifdef AOMDV_NODE_DISJOINT_PATHS
133
134        if ( (rt0 == NULL) || (rt0->rt_flags !=
    ↪ RTF_UP) || (b == NULL) || (b->count) ) {
```

```
135             Packet::free(p);
136             return;
137         }
138 
139         b->count = 1;
140         AOMDV_Path *reverse_path = rt0->path_find();
141 
142         ch->addr_type() = AF_INET;
143         ch->next_hop_ = reverse_path->nexthop;
144         ch->xmit_failure_ = aomdv_rt_failed_callback;
145         ch->xmit_failure_data_ = (void*) this;
146 
147         // route advertisement
148         rp->rp_src = index;
149         if (rt->rt_advertised_hops == INFINITY)
150             rt->rt_advertised_hops =
    ↪    rt->path_get_max_hopcount();
151         rp->rp_hop_count = rt->rt_advertised_hops;
152         rp->rp_first_hop =
    ↪    (rt->path_find())->lasthop;
153 
154         reverse_path->expire = CURRENT_TIME +
    ↪    ACTIVE_ROUTE_TIMEOUT;
155 
156         // CHANGE
157         rt->rt_error = true;
158         // CHANGE
159         forward(rt0, p, NO_AOMDV_DELAY);
160         //   Scheduler::instance().schedule(target_,
    ↪    p, 0.);
161     #endif // AOMDV_NODE_DISJOINT_PATHS
162     #ifdef AOMDV_LINK_DISJOINT_PATHS
```

```
163     /* Drop the RREP packet if we do not have a
    ↪   path back to the source,
164        or the route is marked as down, or if we
    ↪   never received the original RREQ. */
165     if ( (rt0 == NULL) || (rt0->rt_flags !=
    ↪   RTF_UP) || (b == NULL) ) {
166         Packet::free(p);
167         return;
168     }
169     /* Make sure we don't answer along the same
    ↪   path twice in response
170        to a certain RREQ. Try to find an unused
    ↪   (reverse) path to forward the RREP. */
171     AOMDV_Path* reverse_path = NULL;
172     AOMDV_Path *r = rt0->rt_path_list.lh_first;
173     for(; r; r = r->path_link.le_next) {
174         if (b->reverse_path_lookup(r->nexthop,
    ↪   r->lasthop) == NULL) {
175             fprintf(stderr, "\tcycle cycle\n");
176             reverse_path = r;
177             break;
178         }
179     }
180     /* If an unused reverse path is found and the
    ↪   forward path (for
181        this RREP) has not already been replied -
    ↪   forward the RREP. */
182     if ( reverse_path &&
183             (b->forward_path_lookup(forward_path-
    ↪   >nexthop, forward_path->lasthop) == NULL) )
    ↪   {
184         assert (forward_path->nexthop ==
    ↪   rp->rp_src);
```

```
185         assert (forward_path->lasthop ==
    ↪  rp->rp_first_hop);
186         /* Mark the forward and reverse path used
    ↪  to answer this RREQ as used. */
187         b->reverse_path_insert(reverse_path-
    ↪  >nexthop,
    ↪  reverse_path->lasthop);
188         b->forward_path_insert(forward_path-
    ↪  >nexthop,
    ↪  forward_path->lasthop);
189
190         ch->addr_type() = AF_INET;
191         ch->next_hop_ = reverse_path->nexthop;
192         ch->xmit_failure_ =
    ↪  aomdv_rt_failed_callback;
193         ch->xmit_failure_data_ = (void*) this;
194
195         // route advertisement
196         if (rt->rt_advertised_hops == INFINITY)
197             rt->rt_advertised_hops =
    ↪  rt->path_get_max_hopcount();
198         rp->rp_hop_count =
    ↪  rt->rt_advertised_hops;
199         rp->rp_src = index;
200
201         reverse_path->expire = CURRENT_TIME +
    ↪  ACTIVE_ROUTE_TIMEOUT;
202
203         // CHANGE
204         rt->rt_error = true;
205         // CHANGE
206         forwardReply(rt0, p, NO_AOMDV_DELAY);  //
    ↪  CHANGE (previously used forward())
```

```
207             //
    ↪   Scheduler::instance().schedule(target_, p,
    ↪   0.);
208         }
209         else {
210             Packet::free(p);
211             return;
212         }
213     #endif // AOMDV_LINK_DISJOINT_PATHS
214     }
```

A.9 Kode implementasi `aomdv_rt_entry::path_insert`

```
1  AOMDV_Path*
2  aomdv_rt_entry::path_insert(nsaddr_t nexthop,
   ↪  u_int16_t hopcount, double expire_time,
   ↪  nsaddr_t lasthop, float mobility_factor) {
3      AOMDV_Path *path = new AOMDV_Path(nexthop,
   ↪  hopcount, expire_time, lasthop,
   ↪  mobility_factor);
4
5      assert(path);
6  #ifdef DEBUG
7      fprintf(stderr, "%s: (%d\t%d)\n",
   ↪  __FUNCTION__, path->nexthop, path->hopcount);
8  #endif // DEBUG
9
10     /*
11      * Insert path at the end of the list
12      */
13     AOMDV_Path *p = rt_path_list.lh_first;
14     if (p) {
15         for(; p->path_link.le_next &&
   ↪  p->mobility_factor >= path->mobility_factor;
   ↪  p = p->path_link.le_next)
16             /*Do nothing*/;
17         if(path->mobility_factor ==
   ↪  p->mobility_factor){
18             LIST_INSERT_AFTER(p, path,
   ↪  path_link);
19         }
20         else{
21             LIST_INSERT_BEFORE(p, path,
   ↪  path_link);
```

```
22              }
23          }
24          else {
25              LIST_INSERT_HEAD(&rt_path_list, path,
    ↪   path_link);
26          }
27          rt_num_paths_ += 1;
28
29          //HM-AOMDV code
30          rt_mfactor =
    ↪   rt_path_list.lh_first->mobility_factor;
31
32          return path;
33      }
```

A.10 Kode untuk mencetak rute yang dilalui oleh paket data

```
BEGIN {
    sentPkts[0] = 0;
    sendLine = 0;
}

$0 ~/^s.* AGT/{
    sentPkts[$6] = $6" : "$3" "
    sendLine ++ ;
}

$0 ~/^r.* AGT/{
    sentPkts[$6] = sentPkts[$6]" "$3" finish"
}

$0 ~/^f.* RTR.* cbr/{
    sentPkts[$6] = sentPkts[$6]""$3" "
}

END {
    for (i = 0; i < sendLine; i++)
        printf("%s\n", sentPkts[i]);
}
```

BAB 6

PENUTUP

Pada bab ini akan diberikan kesimpulan yang diambil selama pengerjaan Tugas Akhir ini serta saran-saran tentang pengembangan yang dapat dilakukan terhadap tugas akhir ini di masa yang akan datang.

6.1 Kesimpulan

Kesimpulan dan saran yang diperoleh pada uji coba dan evaluasi adalah sebagai berikut:

1. Kecepatan relatif antar kendaraan dapat digunakan sebagai faktor perkiraan stabilitas jaringan.
2. Lokasi terputusnya jaringan berpengaruh pada PDR. Jika jaringan terputus di tengah jalur maka paket yang sudah terlanjur berada pada jaringan bergantung pada *relay node* terakhir untuk memberikan jalur alternatif sebelum proses *route discovery* dijalankan kembali. Sebaliknya jika jaringan terputus di dekat *node* pengirim perbaikan jalur dapat lebih cepat dilakukan.
3. Arah pergerakan kendaraan berpengaruh pada stabilitas jaringan yang digunakan.
4. Fluktuasi performa protokol HM-AOMDV memiliki pola yang mirip dengan protokol AOMDV dengan rata - rata PDR 8,5% hingga 10% lebuh tinggi.

6.2 Saran

Saran yang dapat diberikan dari hasil pengujian ini adalah:

1. Diperlukan penelitian lebih lanjut untuk memodifikasi persamaan *mobility factor* untuk menggunakan arah laju kendaraan sebagai parameternya.

2. Urutan jalur alternatif pada *routing table* perlu diperbarui secara periodik agar penggantian jalur saat terjadi jaringan terputus dapat dilakuakan dengan lebih cepat.
3. Ditambahkan fungsi *local repair* sehingga *relay node* dapat menemukan jalur baru menuju penerima dan paket data yang sudah terlanjur berada pada jaringan tidak langsung terbuang.

DAFTAR PUSTAKA

[1] W. I. Lee, S. I. Chowdhury, G. Y. Kee, W. S. Baek, and J. Y. Pyun. Velocity aware multipath distance vector routing protocol for high mobility over vehicular ad-hoc networks. In *Multimedia and Ubiquitous Engineering (MUE), 2011 5th FTRA International Conference on*, pages 189--194, June 2011. doi: 10.1109/MUE.2011.42.

[2] Y. B. Wang, T. Y. Wu, W. T. Lee, and C. H. Ke. A novel geographic routing strategy over vanet. In *Advanced Information Networking and Applications Workshops (WAINA), 2010 IEEE 24th International Conference on*, pages 873--879, April 2010. doi: 10.1109/WAINA.2010.151.

[3] George Corser. VANET Introduction. `https://www.youtube.com/watch?v=DrH-1505-Mg`, May 2013. Accessed: 2016-06-28.

[4] C.K. Toh. *Ad Hoc Mobile Wireless Networks: Protocols and Systems*. Prentice Hall PTR, 2002. ISBN 9780130078179.

[5] Council of European Union. Directive 2010/40/EU of The European Parliament and of The Council of 7 July 2010, 2010. `http://http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2010:207:0001:0013:EN:PDF`.

[6] Christoph Sommer and Falko Dressler. *Vehicular Networking*. Cambridge University Press, 2015. ISBN 1107046718.

[7] Mahesh K. Marina and Samir R. Das. Ad hoc on-demand multipath distance vector routing. *SIGMOBILE Mob. Comput. Commun. Rev.*, 6(3):92--93, June 2002. ISSN 1559-1662. doi: 10.1145/581291.581305. URL `http://doi.acm.org/10.1145/581291.581305`.

[8] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc on-demand distance vector (aodv) routing. RFC 3561, RFC Editor, July 2003. http://www.rfc-editor.org/rfc/rfc3561.txt.

[9] Steven McCanne and Sally Floyd. ns Network Simulator. http://www.isi.edu/nsnam/ns/.

[10] OpenStreetMap. https://www.openstreetmap.org/. Accessed: 2016-04-28.

[11] Daniel Krajzewicz, Jakob Erdmann, Michael Behrisch, and Laura Bieker. Recent development and applications of SUMO - Simulation of Urban MObility. *International Journal On Advances in Systems and Measurements*, 5(3&4):128--138, December 2012.

[12] Mark Anderson. Global positioning tech inspires do-it-yourself mapping project. http://news.nationalgeographic.com/news/2006/10/061018-street-maps.html, October 2006. Accessed: 2016-04-16.

[13] David Wetherall. OTcl. http://otcl-tclcl.sourceforge.net/otcl/, 1997. Accessed: 2016-04-28.

BIODATA PENULIS

I Putu Pradnyana, biasa dipanggil Prad, dilahirkan di kota Mataram pada tanggal 27 September tahun 1994. Penulis adalah anak sulung dari dua bersaudara. Penulis menempuh pendidikan TK Pertiwi (1998-2000), SDN 5 Gianyar (2000-2006), SMPN 1 Gianyar (2006-2009), SMAN 1 Gianyar (2009-2012). Pada tahun 2012, penulis mengikuti SNMPTN Tulis dan diterima di Strata Satu Jurusan Teknik Informatika, Fakultas Teknologi Informasi, Institut Teknologi Sepuluh Nompember Surabaya angkatan 2012 yang terdaftar dengan NRP 5112100128. Di Jurusan Teknik Informatika ini, penulis mengambil bidang minat Arsitektur dan Jaringan Komputer (AJK). Selama menempuh kuliah, penulis juga pernah aktif sebagai anggota Departemen Riset dan Teknologi di Himpunan Mahasiswa Teknik Computer (HMTC). Penulis juga aktif sebagaianggota TPKH-ITS. Penulis dapat dihubungi melalui alamat e-mail prad.i@live.com.