TUGAS AKHIR - KI141502

IMPLEMENTASI *ADAPTIF FORWARDING NODE* PADA *AD-HOC ON DEMAND DISTANCE VECTOR* (AODV) UNTUK MENCEGAH *BROADCAST STORM PROBLEM* PADA VANETs

DWIYAN SATRIA UTAMA
NRP 05111440000045

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

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

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

(Halaman ini sengaja dikosongkan)

TUGAS AKHIR – KI141502

IMPLEMENTASI *ADAPTIF FORWARDING NODE* PADA *AD-HOC ON DEMAND DISTANCE VECTOR* (AODV) UNTUK MENCEGAH *BROADCAST STORM PROBLEM* PADA VANETS

DWIYAN SATRIA UTAMA
NRP 05111440000045

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

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

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

(Halaman ini sengaja dikosongkan)

UNDERGRADUATE THESES – KI141502

IMPLEMENTATIONS ADAPTIVE FORWARDING NODE IN AD-HOC ON DEMAND DISTANCE VECTOR (AODV) TO PREVENT BROADCAST STORM PROBLEM FOR VANETS

DWIYAN SATRIA UTAMA
NRP 05111440000045

First Advisor
Ir. Muchammad Husni, M.Kom.

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

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

(Halaman ini sengaja dikosongkan)

LEMBAR PENGESAHAN

IMPLEMENTASI *ADAPTIF FORWARDING NODE* PADA *AD-HOC ON DEMAND DISTANCE VECTOR* (AODV) UNTUK MENCEGAH *BROADCAST STORM PROBLEM*

TUGAS AKHIR

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

Oleh:
DWIYAN SATRIA UTAMA
NRP: 05111440000045

Disetujui oleh Pembimbing Tugas Akhir:

1. Ir. Muchammad Husni, M.Kom. .....
(NIP. 196002211984031001) (Pembimbing 1)

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

SURABAYA
JANUARI, 2019

(Halaman ini sengaja dikosongkan)

IMPLEMENTASI *ADAPTIF FORWARDING NODE* PADA *AD-HOC ON DEMAND DISTANCE VECTOR* (AODV) UNTUK MENCEGAH *BROADCAST STORM PROBLEM* PADA VANETS

Nama Mahasiswa	**: Dwiyan Satria Utama**
NRP	**: 05111440000045**
Departemen	**: Informatika FTIK-ITS**
Dosen Pembimbing 1	**: Ir. Muchammad Husni, M.Kom.**
Dosen Pembimbing 2	**: Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.**

Abstrak

Vehicular Ad-Hoc Networks (VANETs). *Vehicle Ad hoc Networks* (VANETs) merupakan perkembangan dari *Mobile Ad hoc Network* (MANET) dimana setiap node memiliki mobilitas yang tinggi dan terbatas pada pola pergerakannya. Ada banyak *routing protocol* yang dapat diimplementasikan pada VANETs, salah satunya adalah *Ad hoc On demand Distance Vector* (AODV).

AODV merupakan salah satu *routing protocol* yang termasuk dalam klasifikasi *reactive routing protocol*, sebuah protokol yang hanya akan membuat rute ketika *node* sumber membutuhkannya. AODV memiliki dua fase, yaitu *route discovery* dan *route maintenance. Route discovery* digunakan untuk meminta dan meneruskan informasi rute yang terdiri dari proses pengiriman *Route Request* (RREQ) dan *Route Reply* (RREP), sedangkan *route maintenance* digunakan untuk mengetahui informasi adanya kesalahan pada rute. Pada fase ini terdapat proses pengiriman *Route Error* (RERR).

Pada Tugas Akhir ini akan dilakukan pembatasan pada proses *forwarding node* yang adaptif pada *route discovery* berdasarkan level konektivitas *node* tetangga, yaitu dengan cara mengeliminasi jumlah *forwarding node* yang bertugas untuk mengirim ulang *(re-broadcast)* RREQ dengan batas *Threshold* yang menggunakan modifikasi average. Hal ini dilakukan agar dapat meningkatkan

kinerja protokol AODV untuk mencari rute yang stabil dengan cara memodifikasi beberapa bagian dari mekanisme pengiriman paket RREQ. Dari hasil uji coba, AODV yang dimodifikasi pada skenario grid berhasil meningkatkan nilai *Packet Delivery Ratio* (PDR) sebesar 2,04% untuk *node* 300. *Routing Overhead* (RO) mengalami penurunan nilai hingga 16,71%. Sedangkan pada skenario real tidak berhasil meningkatkan nilai *Packet Delivery Ratio* (PDR) sehingga mengalami penurunan hingga 9.05%, namun *Routing Overhead* (RO) mengalami penurunan nilai hingga 12,42%.

Kata kunci: VANETs, AODV, Adaptif Forwarding, Node Tetangga

IMPLEMENTATIONS ADAPTIVE FORWARDING NODE IN AD-HOC ON DEMAND DISTANCE VECTOR (AODV) TO PREVENT BROADCAST STORM PROBLEM FOR VANETS

Student's Name	**: Dwiyan Satria Utama**
Student's ID	**: 05111440000045**
Department	**: Informatics – FTIK ITS**
First Advisor	**: Ir. Muchammad Husni, M.Kom.**
Second Advisor	**: Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.**

Abstract

VANETs are an improvement of MANET which have high mobility node characteristic and limited movement pattern. There are many routing protocols that can be implemented on VANETs and one of them is AODV.

AODV is an example of reactive routing protocol classification, a protocol that will only create a route when the source node needs it. AODV have 2 phase which are route discovery and route maintenance. Route discovery is used for requesting and forwarding a route information that consist of Route Request (RREQ) and Route Reply (RREP), meanwhile route maintenance that consist of Route Error (RERR) is used for finding out an error information in route.

In this final project, there will be limitation on adaptive forwarding node on AODV routing protocol based on neighbour node connectivity level by eliminating number of one-hop node that eligible rebroadcast a RREQ with a Threshold using average modification. This is done to improve the performance of the AODV routing protocol to find a stable route by modifying some parts of the RREQ packet delivery mechanism. The evaluation shows that, in the grid scenario the value of Packet Delivery Ratio (PDR) has

decreased by 39,19% on node 60 and 150, but increased by 2,04% on node 300, the value of Routing Overhead (RO) has decreased by 16,71% and in the real scenario the value of Packet Delivery Ratio (PDR) has decreased by 9.05%, the value of Routing Overhead (RO) has decreased by 12,42%.

Keyword: VANETs, AODV, Adaptive Forwarding Node, Neighbor Node

KATA PENGANTAR

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

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

"Implementasi *Adaptif Forwarding Node* pada *Ad-hoc On Demand Distance Vector* (AODV) Untuk Mencegah *Broadcast Storm Problem* pada VANETs"

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

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

1. Allah SWT atas semua rahmat yang diberikan sehingga penulis dapat menyelesaikan Tugas Akhir ini.
2. Bapak Nur Basuki dan Ibu Marti Rustanti selaku kedua orangtua penulis atas segala dukungan berupa motivasi serta doa sehingga penulis dapat mengerjakan Tugas Akhir ini.
3. Bapak Dr. Eng. Radityo Anggoro, S.Kom., M.Sc., dan Bapak Ir. Muchammad Husni, M.Kom. selaku dosen pembimbing, atas arahan dan bantuannya dalam pengerjaan Tugas Akhir ini.
4. TC 14 yang membantu penulis dalam menjalani masa-masa perkuliahan di ITS.
5. Teman – teman SOLITS dan SORITIA yang selama ini sudah menyemangati Penulis dalam menjalankan kuliah di ITS.

6. Juga tidak lupa kepada semua pihak yang belum sempat disebutkan satu per satu yang telah membantu penulis dalam menyelesaikan Tugas Akhir ini.

Penulis telah berusaha sebaik-baiknya dalam menyusun tugas akhir ini, namun penulis mohon maaf apabila terdapat kekurangan, kesalahan maupun kelalaian yang telah penulis lakukan. Kritik dan saran yang membangun dapat disampaikan sebagai bahan perbaikan selanjutnya.

Surabaya, Januari 2019

Dwiyan Satria Utama

DAFTAR ISI

(Halaman ini sengaja dikosongkan)

DAFTAR GAMBAR

(Halaman ini sengaja dikosongkan)

DAFTAR TABEL

(Halaman ini sengaja dikosongkan)

BAB I
PENDAHULUAN

1.1 Latar Belakang

Dimasa kini kemajuan perangkat keras, perangkat lunak, dan teknologi komunikasi memberikan gebrakan baru untuk pengimplementasian berbagai macam jaringan yang dapat diimplementasikan di berbagai macam jenis lingkungan. Salah satu jaringan yang bertumbuh pesat dan mendapat perhatian cukup besar adalah *Vehicular Ad-Hoc Networks* (VANETs). *Vehicle Ad hoc Networks* (VANETs) merupakan perkembangan dari *Mobile Ad hoc Network* (MANET) dimana setiap node memiliki mobilitas yang tinggi yang menyebabkan perubahan dalam topologi jaringan sehingga kesulitan dalam mengaturnya. Topologi yang dinamis ini merupakan perbedaan mendasar antara VANET dan MANET. Tingkat mobilitas yang tinggi pada setiap node juga dapat menyebabkan adanya rute terputus dikarenakan node keluar dari jangkauan sinyal transmisi [1].

Didalam VANET *Routing protocol* terbagi menjadi dua model, yaitu *proactive* dan *reactive routing. Proactive routing* adalah protokol yang bekerja dengan cara membentuk tabel routing dan melakukan update setiap saat pada selang waktu tertentu tanpa memperhatikan beban jaringan, *bandwidth* dan ukuran jaringan. Sedangkan *reactive routing* adalah merupakan mekanisme routing yang membentuk tabel routing jika ada permintaan pengiriman data atau terjadinya perubahan rute dalam setiap jaringan. Contoh *proactive routing protocol* adalah *Destination-Sequenced Distance Vector* (DSDV), dan *Optimized Link State Routing protocol* (OLSR) sedangkan contoh *reactive routing protocol* adalah *Adhoc On-Demand Distance Vector* (AODV), dan *Dynamic Source Routing* (DSR).

Pencarian rute adalah salah satu mekanisme yang penting untuk mendukung mobilitas kinerja VANET. Saat proses pencarian rute, pemilihan rute yang stabil sangat diperlukan untuk memperpanjang waktu penggunaan rute. Pemilihan rute dengan

kemungkinan kecil tertuputus adalah salah satu cara yang dapat dilakukan [2].

Broadcast Storm Problem terjadi ketika sebuah *system network* terlalu banyak menerima *broadcast traffic* secara beruntun sehingga *system* mengalami *overload*. Ketika *node* yang berbeda mem-*broadcast* suatu data melalui suatu jaringan, dan ada *device* lain yang me-*rebroadcast* data tersebut melalui jaringan yang sama, hal ini dapat menyebabkan seluruh jaringan untuk *melt-down* dan gagal untuk melakukan komunikasi di jaringan tersebut.

Pada Tugas Akhir ini diusulkan suatu mekanisme *routing discovery* yang bersifat adaptif pada *reactive routing* AODV untuk memperoleh rute yang paling stabil serta mencegah *Broadcast Storm Problem* berdasarkan level konektivitas *one-hop* pada VANETs. Pencegahan *Broadcast Storm Problem* dapat dilakukan dengan membatasi jumlah *Forwarded Route Request* yang dikirimkan *node* tetangga. Hasil akhir yang diharapkan adalah mengetahui perbandingan kinerja antara AODV dan AODV yang telah dimodifikasi diukur berdasarkan performansi *Packet Delivery Ratio* (PDR), *Routing Overhead*, dan *Delivery Delay*.

1.2 Rumusan Masalah

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

1. Bagaimana membatasi jumlah *forwarding node* secara adaptif untuk mencegah *Broadcast Storm Problem* pada VANETs?
2. Bagaimana dampak pembatasan *forwarding node* terhadap performa protokol AODV secara keseluruhan di lingkungan yang dinamis?

1.3 Batasan Permasalahan

Batasan masalah pada Tugas Akhir ini adalah sebagai berikut:

1. Jaringan yang digunakan adalah jaringan *Vehicular Ad hoc Networks* (VANETs).
2. *Routing protocol* yang diujicobakan yaitu AODV.
3. Pembuatan skenario uji coba menggunakan *Simulation of Urban Mobility* (SUMO).
4. Simulasi pengujian jaringan menggunakan *Network Simulator 2* (NS-2).

1.4 Tujuan

Tujuan dari Tugas Akhir ini adalah sebagai berikut:

1. Mereduksi jumlah *forwarding node* secara adaptif untuk mencegah terjadinya *Broadcast Storm Problem*.
2. Mengurangi jumlah *control packet* yang di*broadcast* dalam jaringan.

1.5 Manfaat

Dengan dibuatnya Tugas Akhir ini akan memberikan sebuah manfaat dalam mengetahui dampak performa AODV dengan pembatasan *Threshold* dengan modifikasi average yang dilakukan.

1.6 Metodologi

Pembuatan Tugas Akhir ini dilakukan dengan menggunakan metodologi sebagai berikut:

1.6.1 Penyusunan Proposal Tugas Akhir

Proposal tugas akhir ini berisi tentang deskripsi pendahuluan dari tugas akhir yang akan dibuat. Pendahuluan ini terdiri atas hal yang menjadi latar belakang diajukannya usulan tugas akhir, rumusan masalah yang diangkat, batasan masalah

untuk tugas akhir, tujuan dari pembuatan tugas akhir, dan manfaat dari hasil pembuatan tugas akhir. Selain itu dijabarkan pula tinjauan pustaka yang digunakan sebagai referensi pendukung pembuatan tugas akhir. Sub bab metodologi berisi penjelasan mengenai tahapan penyusunan tugas akhir mulai dari penyusunan proposal hingga penyusunan buku tugas akhir. Terdapat pula sub bab jadwal kegiatan yang menjelaskan jadwal pengerjaan tugas akhir.tugas akhir ini, rumusan masalah, batasan masalah, dan tujuan dari pembuatan tugas akhir. Tinjauan pustaka digunakan sebagai referensi yang mendukung pembuatan tugas akhir dan metodologi berisi penjelasan mengenai tahapan penyusunan. Terakhir dijabarkan pula jadwal kegiatan dalam pembuatan tugas akhir ini.

1.6.2 Studi Literatur

Pada studi literatur ini, akan dipelajari sejumlah referensi yang diperlukan dalam melakukan implementasi yaitu mengenai NS-2 *Network Simulator*, *Vehicular Ad hoc Networks* (VANETs), dan *reactive routing protocol* AODV.

1.6.3 Analisis dan Desain Sistem

Pada tahap ini dilakukan analisis dari hasil percobaan modifikasi *adaptive forwarding node* AODV yang dibuat. Data yang dianalisis berasal dari perhitungan *Packet Delivery Ratio* (PDR), *Routing Overhead* (RO), dan *Delivery Delay* paket dari node ke node lainnya.

1.6.4 Implementasi Sistem

Pada tahap ini dilakukan perancangan model jaringan implementasi yang dibuat berupa modifikasi dari protokol AODV agar mendapatkan rute yang stabil.

1.6.5 Pengujian dan Evaluasi

Pengujian dilakukan dengan NS-2 *Network Simulator* dan akan menghasilkan *trace file*. Dari *trace file* tersebut akan dihitung *packet delivery ratio, routing overhead,* dan *delivery delay* untuk menguji performa *routing protocol* yang telah dimodifikasi.

1.6.6 Penyusunan Buku

Pada tahap ini dilakukan penyusunan buku yang menjelaskan seluruh konsep, teori dasar dari metode yang digunakan, implementasi, serta hasil yang telah dikerjakan sebagai dokumentasi dari pelaksanaan Tugas Akhir.

1.7 Sistematika Penulisan Laporan

Sistematika penulisan laporan Tugas Akhir adalah sebagai berikut:

1. Bab I. Pendahuluan
   Bab ini berisikan penjelasan mengenai latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, metodologi, dan sistematika penulisan dari pembuatan Tugas Akhir.
2. Bab II. Tinjauan Pustaka
   Bab ini berisi kajian teori atau penjelasan dari metode, algoritma, *library*, dan *tools* yang digunakan dalam penyusunan Tugas Akhir ini. Bab ini berisi tentang penjelasan singkat mengenai VANETs, AODV, NS2, OpenStreetMap, Java OpenStreetMap (JOSM), SUMO, dan AWK.
3. Bab III. Perancangan
   Bab ini berisi pembahasan mengenai perancangan skenario mobilitas *grid* dan *real*, perancangan simulasi pada NS2, perancangan modifikasi AODV, serta perancangan metrik

analisis (*Packet Delivery Ratio, Routing Overhead, Forwarded Route Request,* dan *End-to-End Delay*).

4. Bab IV. Implementasi
   Bab ini menjelaskan implementasi yang berbentuk kode sumber dari proses modifikasi protokol AODV, pembuatan simulasi pada NS2, SUMO, dan perhitungan metrik analisis.
5. Bab V. Uji Coba dan Evaluasi
   Bab ini berisikan hasil uji coba dan evaluasi dari implementasi yang telah dilakukan untuk menyelesaikan masalah yang dibahas pada Tugas Akhir.
6. Bab VI. Kesimpulan dan Saran
   Bab ini merupakan bab yang menyampaikan kesimpulan dari hasil uji coba yang dilakukan, masalah-masalah yang dialami pada proses pengerjaan Tugas Akhir, dan saran untuk pengembangan solusi ke depannya.
7. Daftar Pustaka
   Bab ini berisi daftar pustaka yang dijadikan literatur dalam Tugas Akhir.
8. Lampiran
   Dalam lampiran terdapat tabel-tabel data hasil uji coba dan kode sumber program secara keseluruhan.

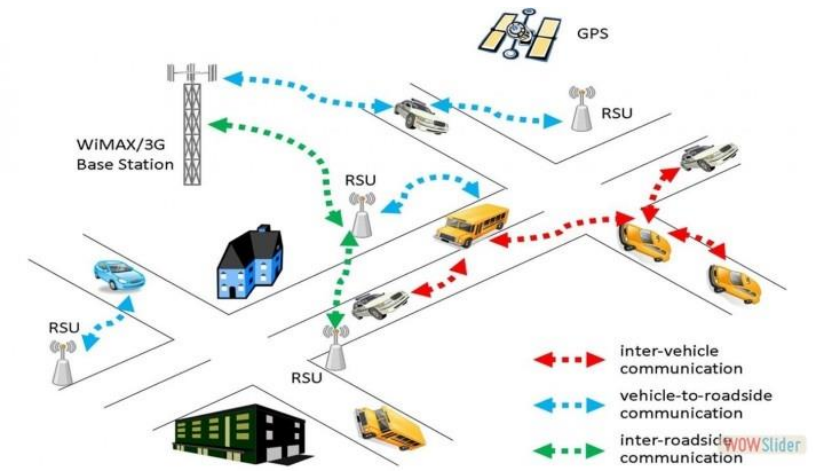
BAB II
TINJAUAN PUSTAKA

Bab ini berisi pembahasan mengenai teori-teori dasar atau penjelasan dari metode dan *tools* yang digunakan dalam Tugas Akhir.

2.1 VANETs

Vehicular Ad Hoc Network (VANET) merupakan pengembangan dari Mobile Ad Hoc Network (MANET) yang memungkinkan terjadinya komunikasi antar kendaraan sebagai pengembangan Intelligent Transport System (ITS) dengan tujuan meningkatkan keselamatan dan kenyamanan berkendara. Komunikasi Wireless ini meliputi komunikasi Inter-Vehicle Communication (IVC), Vehicle to Roadside (V2R), atau Roadside to Roadside (R2R). Setiap node pada VANET berlaku baik sebagai partisipan ataupun router pada jaringan, baik bagi node utama atau intermediate node yang berkomunikasi di dalam radius transmisinya. VANET merupakan jaringan yang self-organized, artinya jaringan ini tidak bergantung pada infrastruktur jaringan manapun. Ada beberapa node yang secara tetap berdiri sebagai Roadside Unit, yakni yang dapat memfasilitasi jaringan kendaraan dengan informasi data geografis ataupun akses internet [3].

(Halaman ini sengaja dikosongkan)



Gambar 2.1 Ilustrasi VANETs *[1]*

Dalam Tugas Akhir ini, penulis akan mengimplementasikan *routing protocol* AODV yang dimodifikasi dan menguji performa protokol tersebut pada lingkungan VANETs.

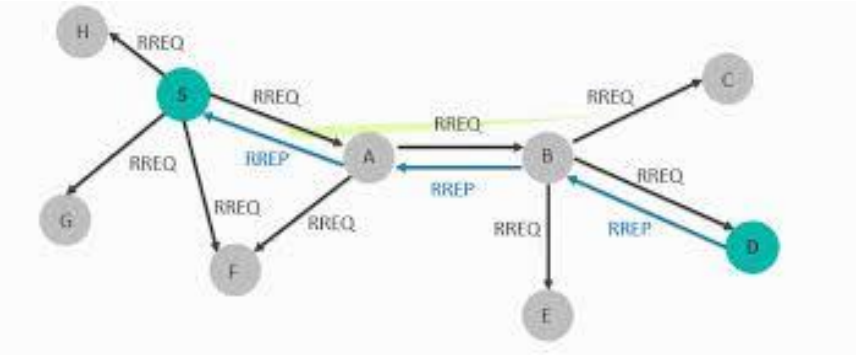
2.2 *Ad-hoc On demand Distance Vector* (AODV)

Ad-hoc On demand Distance Vector (AODV) merupakan salah satu *routing* protokol yang masuk dalam klasifikasi *reactive routing protocol*. Sebuah protokol yang hanya membuat satu rute saat dibutuhkan. AODV dikembangkan oleh C. E. Perkins, E.M. Belding-Royer dan S. Das pada RFC 3561.

Ciri terpenting dari AODV adalah menjaga *timer-based state* pada setiap *node* sesuai dengan tabel *routing* yang digunakan. Tabel *routing* akan kadaluarsa jika jarang digunakan. AODV memiliki dua tahapan yaitu *route discovery* dan r*oute maintenance*. *Route discovery* memiliki dua pesan yaitu berupa *Route Request* (RREQ)

dan *Route Reply* (RREP). Sedangkan *Route maintenance* berupa *Route Error* (RERR).

AODV merupakan sebuah metode *routing* pesan antar *node* yang memungkinkan *node-node* tersebut untuk melewatkan pesan melalui lingkungannya ke *node* yang tidak dapat dihubungi secara langsung. AODV melakukan ini dengan cara menemukan rute yang bisa dilalui oleh pesan. Selain itu AODV juga memastikan rute ini tidak mengandung perulangan (*loop*), menangani perubahan rute, dan membuat rute baru apabila terjadi *error* [4]. Ilustrasi pencarian rute oleh AODV dapat dilihat pada Gambar 2.2.



Gambar 2.2 Ilustrasi pencarian rute *routing protocol* AODV *[14]*

Pada setiap *node* yang menggunakan protokol AODV pasti memiliki sebuah *routing table* dengan *field* sebagai berikut:

- *Destination Sequence Number*: *sequence number* dari jalur komunikasi.
- *Destination Address:* berisi alamat dari *node* tujuan.
- *Lifetime*: waktu dalam milidetik yang diperlukan *node* untuk menerima RREP.
- *Hop Count*: jumlah *hop* yang harus dilakukan agar paket dapat mencapai *node* tujuan.
- *Next Hop*: alamat *node* yang akan meneruskan paket data.
- *Routing Flags*: status jalur. Terdapat tiga tipe status, yaitu *up* (valid), *down* (tidak valid) atau sedang diperbaiki.

Salah satu contoh proses *route discovery* dalam AODV, ilustrasi pada Gambar 2.2 menggambarkan bagaimana *source node*, yaitu *node* A mencari rute untuk menuju *destination node* yaitu *node* F. *Node* A akan membuat paket RREQ dan melakukan *broadcast* kepada semua *node* tetangganya (*neighbor node*). Jika *destination sequence number* yang terdapat pada paket RREQ sama atau lebih kecil dari yang ada pada *routing table* dan rute menuju *node* tujuan belum ditemukan, maka paket tersebut tidak akan dilanjutkan (*drop*). Jika *destination sequence number* pada RREQ lebih besar dibandingkan dengan yang terdapat pada *routing table*, maka *entry* pada *routing table* akan diperbarui dan paket tersebut akan diteruskan oleh *neighbor node* sekaligus membuat *reverse path* menuju *source node.* Paket RREQ akan diteruskan hingga mencapai *node* F. Kemudian, jika rute menuju *node* F sudah terbentuk di dalam *routing table* dan memiliki *routing flags* "*up*", maka *node* F akan mengirimkan paket RREP melalui rute tersebut menuju *node* .

Pada Tugas Akhir ini, penulis menggunakan *routing protocol* AODV yang akan diimplementasikan pada lingkungan VANETs dengan menggunakan beberapa skenario.

2.3 *Network Simulator-2* (NS-2)

Network Simulator (NS) adalah suatu *interpreter* yang berorientasi objek, dan *discrete event-driven* yang dikembangkan oleh University of California Berkeley dan USC ISI sebagai bagian dari projek *Virtual INternet Testbed* (VINT). NS yang banyak dikenal dengan NS-2 (versi 2) menjadi salah satu *tool* yang sangat berguna untuk menunjukkan simulasi jaringan melibatkan *Local Area Network* (LAN), *Wide Area Network* (WAN), tapi fungsi dari *tool* ini telah berkembang selama beberapa tahun belakangan untuk memasukkan jaringan nirkabel (*wireless*) dan juga jaringan *ad hoc* [5].

Pada Tugas Akhir ini, NS-2 digunakan untuk melakukan simulasi lingkungan VANETs menggunakan protokol AODV yang

sudah dimodifikasi. *Trace file* yang dihasilkan oleh NS-2 juga digunakan untuk mengukur peforma *routing* protokol AODV yang sudah dimodifikasi.

2.3.1 Instalasi

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

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

Gambar 2.3 Perintah untuk meng*install dependency* NS-2

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

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

Gambar 2.4 Baris kode yang diubah pada file ls.h

NS-2.

2.3.2 *Trace File*

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

Tabel 2.1 Detail Penjelasan *Trace File* AODV

Kolom ke-	Penjelasan	Isi

1	*Event*	s : *sent* r : *received* f : *forwarded* D : *dropped*
2	*Time*	Waktu terjadinya *event*
3	ID *Node*	_x_ : dari 0 hingga banyak *node* pada topologi
4	*Layer*	AGT : *application* RTR : *routing* LL : *link layer* IFQ : *packet queue* MAC : *MAC* PHY : *physical*
5	*Flag*	--- : Tidak ada
6	*Sequence Number*	Nomor paket
7	*Packet Type*	AODV : paket *routing* AODV cbr : berkas paket CBR (*Constant Bit Rate*) RTS : *Request To Send* yang dihasilkan MAC 802.11 CTS : *Clear To Send* yang dihasilkan MAC 802.11 ACK : MAC ACK ARP : Paket *link layer address resolution protocol*
8	Ukuran	Ukuran paket pada *layer* saat itu
9	Detail MAC	[a b c d] a : perkiraan waktu paket b : alamat penerima c : alamat penerima d : IP header
10	*Flag*	-------- : Tidak ada
11	*Detail* IP *source, destination,* dan *nexthop*	[a:b c:d e f] a : IP *source node* b : *port source node*

		c : IP *destination node* (jika -1 berarti *broadcast*) d : *port destination node* e : IP *header* ttl f : IP *nexthop* (jika 0 berarti *node* 0 atau *broadcast*)

2.4 OpenStreetMap

OpenStreetMap (OSM) merupakan sebuah proyek berbasis web untuk membuat peta dunia yang gratis dan terbuka, dibangun sepenuhnya oleh sukarelawan dengan melakukan survei menggunakan GPS, mendigitalisasi citra satelit dan mengumpulan serta membebaskan data geografis yang tersedia di publik. Melalui *Open Data Commons Open Database License 1.0*, kontributor OSM dapat memiliki, memodifikasi, dan membagikan data peta secara luas. Terdapat beragam jenis peta digital yang tersedia di internet, namun sebagian besar memiliki keterbatasan secara legal maupun teknis. Hal ini membuat masyarakat, pemerintah, peneliti dan akademisi, *inovator*, dan banyak pihak lainnya tidak dapat menggunakan data yang tersedia di dalam peta tersebut secara luas. Di sisi lain, baik peta dasar OSM maupun data yang tersedia di dalamnya dapat diunduh secara gratis dan terbuka, untuk kemudian digunakan untuk diredistribusikan kembali.

Di banyak tempat di dunia ini, terutama di daerah terpencil dan terbelakang secara ekonomi, tidak terdapat insentif komersil sama sekali bagi perusahaan pemetaan untuk mengembangkan data di tempat ini. OSM dapat menjadi jawaban di banyak tempat seperti ini, baik itu pengembangan ekonomi, tata kota, kontinjensi bencana, maupun untuk berbagai tujuan lainnya [6].

Pada Tugas Akhir ini, penulis menggunakan data yang tersedia pada OSM untuk membuat skenario lalu lintas berdasarkan peta daerah di Surabaya. Peta yang diambil lalu digunakan untuk simulasi skenario *rea*l VANETs.

2.5 *Java OpenStreetMap Editor* (JOSM)

Java OpenStreetMap Editor (JOSM) adalah aplikasi untuk menyunting data yang didapatkan dari OpenStreetMap [7].

Pada Tugas Akhir ini, penulis menggunakan aplikasi ini untuk menyunting dan merapikan peta yang diunduh dari OpenStreetMap yaitu dengan menghilangkan dan menyambungkan jalan yang ada. Penyuntingan juga dilakukan dengan menghilangkan gedung – gedung yang ada di peta.

2.6 *Simulation of Urban Mobility* (SUMO)

Simulation of Urban Mobility (SUMO) merupakan paket simulasi lalu lintas yang bersifat *open-source* dimana pengembangannya dimulai pada tahun 2001. Dan semenjak itu SUMO telah berubah menjadi sebuah simulasi lalu lintas dengan kelengkapan fitur dan pemodelannya termasuk kemampuan jalannya jaringan untuk membaca *format* yang berbeda.

SUMO juga memungkinkan untuk mendefinisikan kendaraan dengan sifat tertentu seperti panjang kendaraan, kecepatan maksimum, percepatan dan perlambatannya. SUMO juga menyediakan pilihan bagi pengguna menentukan rute acak untuk kendaraan. Ada juga pilihan yang tersedia untuk model sistem transportasi umum, dimana setiap kendaraan datang dan berangkat sesuai dengan jadwal [8].

Pada Tugas Akhir ini, penulis menggunakan SUMO untuk menghasilkan skenario VANETs, peta area simulasi, dan pergerakan *node* sehingga menyerupai keadaan lalu lintas yang sebenarnya. Untuk setiap skenario VANETs yang dibuat menggunakan SUMO, akan dihasilkan pergerakan *node* yang acak sehingga setiap skenario memiliki pergerakan yang berbeda.

2.7 AWK

AWK adalah bahasa pemrograman yang digunakan untuk melakukan *text processing* dan ekstraksi data [9]. AWK merupakan sebuah program filter untuk teks, seperti halnya perintah grep pada terminal linux. AWK dapat digunakan untuk mencari bentuk / model dalam sebuah berkas teks ke dalam bentuk teks lain. AWK dapat juga digunakan untuk melakukan proses aritmatika seperti yang dilakukan oleh perintah expr. AWK sama halnya seperti bahasa shell atau C yang memiliki karakteristik yaitu sebagai *tool* yang cocok untuk *jobs* juga sebagai pelengkap untuk *filter* standar.

Pada Tugas Akhir ini, AWK digunakan untuk membuat script menghitung *Packet Delivery Ratio* (PDR*), Routing Overhead* (RO), *Forwarded Route Request* (RREQ F), dan *End-to-End Delay* dari *trace file* NS2.

2.8 *Broadcast Storm Problem*

Broadcast Storm Problem terjadi ketika sebuah *system network* terlalu banyak menerima *broadcast traffic* secara beruntun sehingga *system* mengalami *overload*. Ketika *node* yang berbeda mem-*broadcast* suatu data melalui suatu jaringan, dan ada *device* lain yang me-*rebroadcast* data tersebut melalui jaringan yang sama, hal ini dapat menyebabkan seluruh jaringan untuk *melt-down* dan gagal untuk melakukan komunikasi di jaringan tersebut.

Broadcast Storm Problem memiliki indikasi sebagai berikut :

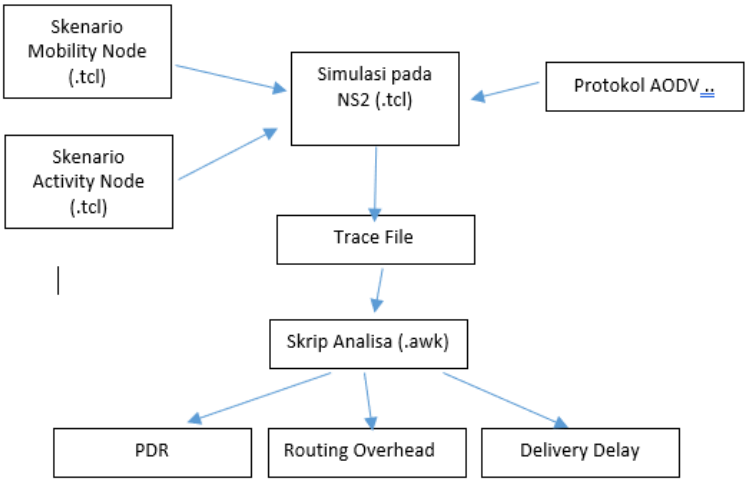
- Tiap-tiap *node* menerima paket yang sama beberapa kali dan saat *node* tersebut akan melakukan *re-broadcast, node-node* tetangganya telah memiliki paket tersebut. Sehingga proses *re-broadcast* menjadi tidak efektif.

BAB III
PERANCANGAN

Perancangan merupakan bagian penting dari pembuatan sistem secara teknis sehingga bab ini secara khusus menjelaskan perancangan sistem yang dibuat dalam Tugas Akhir. Berawal dari deskripsi umum sistem hingga perancangan skenario, alur dan implementasinya.

3.1 Deskripsi Umum

Pada Tugas Akhir ini akan diimplementasikan *routing protocol* AODV dengan memodifikasi pada bagian proses *route discovery* yang dijalankan pada simulator NS-2. Diagram dari rancangan simulasi dari AODV asli dan AODV modifikasi dapat dilihat pada



Gambar 3.1 Diagram Rancangan Simulasi AODV Modifikasi

Modifikasi diawali dengan mencari jumlah tetangga setiap *node* perantara (*one-hop node)*. Jumlah tetangga tiap-tiap *node* akan didapatkan dengan menggunakan HELLO *messages* yang terdapat pada AODV. Setelah jumlah tetangga tiap-tiap *node* didapatkan, maka modifikasi dilanjutkan untuk melakukan perhitungan nilai *threshold* yang dilakukan setiap *i* interval waktu dan penyeleksian *forwarding node* yang bertugas untuk *rebroadcast* paket *Route Request* (RREQ) yang akan diteruskan. Hal tersebut dilakukan dengan cara menyeleksi *node – node* mana saja yang mempunyai jumlah tetangga lebih dari nilai *threshold* yang sudah ditentukan. Jika *node* tersebut mempunyai jumlah tetangga lebih dari *threshold*, maka *node* tersebut akan meneruskan paket RREQ, namun jika sebaliknya, maka *node* tersebut tidak akan meneruskan paket RREQ atau paket akan di-*drop*.

Modifikasi yang telah dilakukan akan disimulasikan pada NS-2 dengan peta berbentuk *grid* dan peta *real* pada lingkungan lalu lintas di kota Surabaya. Pembuatan kedua peta tersebut menggunakan bantuan *tools* SUMO. Simulasi tersebut akan memberikan hasil *trace file* yang kemudian dianalisis menggunakan skrip AWK untuk mendapatkan *Packet Delivery Ratio* (PDR), *Routing Overhead* (RO), dan *Forwarded Route Request* (RREQ F), dan *End-to-End Delay* (E2E). Analisis tersebut dapat mengukur performa *routing protocol* AODV yang telah dimodifikasi dibandingkan dengan *routing protocol* AODV sebelum dimodifikasi. Analisis ini digunakan untuk mengukur tingkat reliabilitas pengiriman data antara protokol AODV dengan protokol AODV yang dimodifikasi. Daftar istilah yang sering digunakan pada buku Tugas Akhir ini dapat dilihat pada Tabel 3.1.

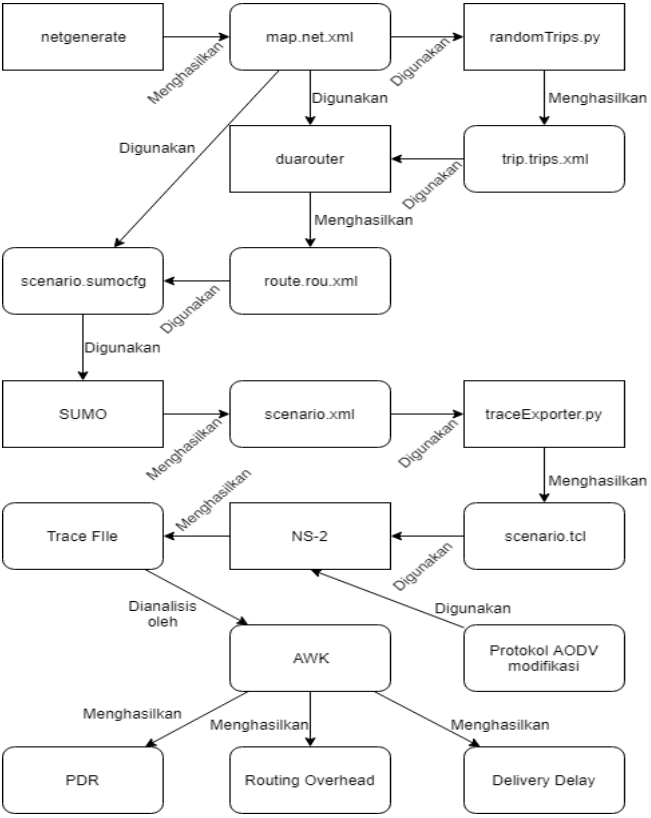
Tabel 3.1 Daftar Istilah

No.	Istilah	Penjelasan
1	AODV	Singkatan dari *Ad hoc On-demand Distance Vector*. Protokol yang digunakan pada Tugas Akhir ini.

No.	Istilah	Penjelasan
2	PDR	*Packet Delivery Ratio*. Salah satu metrik analisis yang diukur. Berupa rasio jumlah pengiriman paket yang terkirim.
3	E2E	*Average End-to-End Delay*. Jeda waktu yang diukur saat paket terkirim.
4	RO	*Routing Overhead*. Jumlah *control packet* yang terkirim
5	RREQ	*Route Request*. Paket *request* pada AODV yang dikirim untuk mendapatkan rute
6	RREP	*Route Reply*. Paket *reply* pada AODV yang dikirim ke *node* sumber melalui rute yang sudah terbuat.
7	*Threshold*	Batas nilai yang dijadikan sebagai acuan

3.2 Perancangan Skenario Mobilitas

Perancangan skenario mobilitas dimulai dengan membuat area simulasi, pergerakan *node*, dan implementasi pergerakan *node*. Dalam Tugas Akhir ini, terdapat dua macam area simulasi yang akan digunakan yaitu peta *grid* dan *real*. Peta *grid* yang dimaksud adalah bentuk jalan berpetak – petak sebagai contoh jalan berpotongan yang sederhana. Peta *grid* digunakan sebagai simulasi awal VANETs karena lebih stabil. Peta *grid* didapatkan dengan menentukan panjang dan jumlah petak area menggunakan SUMO. Sedangkan yang dimaksud peta *real* adalah peta asli / nyata yang digunakan sebagai area simulasi. Peta *real* didapatkan dengan mengambil daerah yang diinginkan sebagai area simulasi menggunakan OpenStreetMap. Pada Tugas Akhir ini, peta *real* yang diambil penulis adalah salah satu area di kota Surabaya.



Gambar 3.2.1 Alur perancangan skenario *grid*

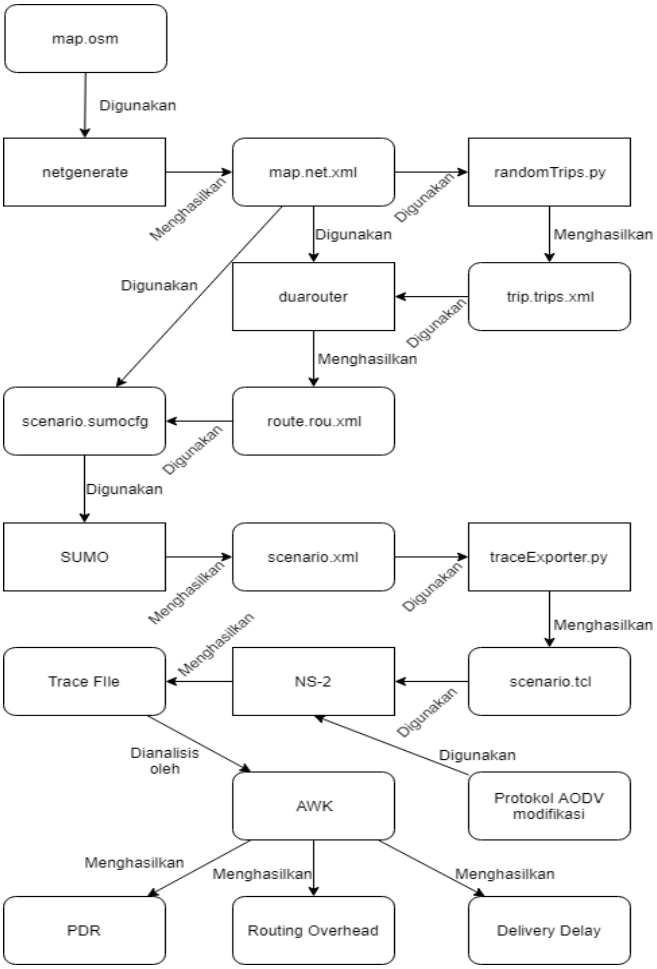
3.2.1 Perancangan Skenario *Grid*

Perancangan skenario mobilitas *grid* diawali dengan merancang luas area peta *grid* yang dibutuhkan. Luas area tersebut bisa didapatkan dengan cara menentukan terlebih dahulu jumlah titik persimpangan yang diinginkan, sehingga dari jumlah persimpangan tersebut dapat diketahui berapa banyak peta yang dibutuhkan. Dengan mengetahui jumlah petak yang dibutuhkan, dapat ditentukan panjang

tiap petak sehingga mendapatkan luas area yang dibutuhkan yaitu 1300 m x 1300 m. Dengan 4 titip persimpangan, maka akan didapatkan 9 petak dan panjang tiap peta adalah 400m.

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

Skenario mobilitas *grid* dihasilkan dengan menggabungkan *file* peta *grid* dan *file* pergerakan *node* yang telah dibuat. Penggabungan tersebut menghasilkan *file* dengan ekstensi .xml. Selanjutnya, untuk dapat diterapkan pada NS-2, *file* skenario mobilitas *grid* yang berekstensi .xml dikonversi ke dalam bentuk *file* .tcl. Konversi ini dilakukan menggunakan *tool* traceExporter.



3.2.2 Perancangan Skenario *Real*

Perancangan skenario mobilitas *real* diawali dengan memilih area yang akan dijadikan simulasi. Pada Tugas Akhir ini, digunakan peta dari OpenStreetMap untuk mengambil area yang dijadikan model simulasi. Setelah memilih area, dilakukan pengunduhan dengan

menggunakan fitur *export* yang telah disediakan oleh OpenStreetMap. Peta hasil *export* tersebut memiliki ekstensi .osm.

Setelah mendapatkan peta area yang akan dijadikan simulasi, peta tersebut dikonversi ke dalam bentuk *file* dengan ekstensi .net.xml menggunakan *tools* SUMO yaitu netconvert. Tahap berikutnya memiliki tahapan yang sama seperti merancang skenario *grid*, yaitu membuat pergerakan *node* menggunakan randomTrips dan duarouter. Kemudian dilakukan penggabungan *file* peta *real* yang sudah dikonversi ke dalam *file* dengan ekstensi .net.xml dan *file* pergerakan *node* yang sudah dibuat sebelumnya. Hasil dari penggabungan tersebut merupakan *file* skenario berkekstensi .xml. *File* yang dihasilkan tersebut dikonversi ke dalam bentuk *file* dengan ekstensi .tcl agar dapat diterapkan pada NS-2.

3.3 Perancangan Modifikasi *Routing Protocol* AODV

Protokol AODV yang diajukan pada Tugas Akhir ini merupakan modifikasi dari protokol AODV yang mengubah mekanisme *route discovery* pada protokol tersebut. Pada protokol AODV, mekanisme pencarian *node* untuk pengiriman ulang (*rebroadcast*) paket RREQ langsung dikirim begitu saja, maka pada AODV yang dimodifikasi ini akan ada proses penyeleksian *node*. Seleksi *node* dilakukan dengan cara *node* sumber mengetahui jumlah tetangga yang dimiliki *node* perantara (*one-hop node*). Setelah mengetahui jumlah tetangga yang dimiliki setiap node maka terdapat fungsi yang berjalan setiap *i* interval yang akan menghitung *threshold* berdasarkan average jumlah tetangga tiap node. Selanjutnya, dilakukan perbandingan menggunakan *threshold* yang didapatkan dari fungsi modifikasi untuk pengiriman RREQ. Apabila jumlah tetangga *one-hop node* tersebut lebih besar atau sama dengan dengan *threshold* yang sudah ditentukan, maka pengiriman RREQ akan dilanjutkan. Namun sebaliknya, apabila jumlahnya kurang dari *threshold*, maka paket akan di*drop*. Jika sudah mencapai *node* tujuan, *node* akan mengembalikannya dengan paket RREP. Rute yang dilalui paket RREP adalah rute dengan pembatasan *forwarding node* yang

sudah dilakukan sebelumnya. Rute tersebut tidak melakukan *rebroadcast* RREQ ke semua *node* dan paket RREQ hanya melewati *node* – *node* terpilih untuk sampai ke *node* tujuan. Karena beberapa paket RREQ di-drop, maka kemacetan dan tabrakan paket pada jaringan akan lebih rendah sehingga bisa menaikkan PDR.

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

Terdapat beberapa cara untuk dapat mengetahui jumlah tetangga yang ada di sekitar *node*. Pada Tugas Akhir ini, penghitungan jumlah tetangga dilakukan dengan memanfaatkan HELLO *messages* yang ada pada protokol AODV. HELLO *packets* akan mengirimkan HELLO *messages* secara terus menerus untuk memberikan informasi kepada *node* tersebut mengenai tetangga yang ada di sekitarnya. Setiap *node* yang menerima HELLO *messages* dari *node* lainnya, sudah dipastikan menjadi tetangga *node* tersebut. Dengan begitu, jumlah tetangga dapat dihitung dari setiap *node* yang mengirimkan HELLO *messages*. Modifikasi akan dilakukan dengan menambahkan counter jumlah tetangga tiap node pada fungsi `nb_insert()` yang digunakan untuk menambah node tetangga dan `nb_delete()` yang digunakan untuk mengurangi node tetangga pada file aodv.cc yang terletak di dalam *folder* ns-2.35/aodv.

3.3.2 Perancangan Perhitungan *Threshold*

Threshold akan ditentukan melalui perhitungan nilai average dari jumlah tetangga tiap node. Perhitungan threshold dilakukan setiap *i* interval waktu. Interval waktu yang digunakan adalah 5 detik, 10 detik dan 15 detik. Jumlah tetangga tiap node disimpan dalam bentuk *array* sehingga diperlukan proses *sorting array* terlebih dahulu. Setelah array di sorting, maka akan dicari nilai jumlah rata-rata tiap *node* tetangga atau disebut nilai average. Nilai average yang didapatkan akan digunakan sebagai *threshold*. *Pseudocode* untuk perhitungan *threshold* dapat dilihat pada Gambar 3.3.

```
for i=0 to node_count do
 for j=i+1 to node_count do
  if neighbour_count[i]>neighbour_count[j]
  then
   tmp=neighbour_count[i]
   neighbour_count[i]=neighbour_count[j]
   neighbour_count[j]=tmp
  endif
 endfor
endfor
```

```
for i=0 to node_count do
 for j=0 to node_count do
  if neighbour_count[i]==neighbour_count[j]
  then
   count_mode[i}++
  endif
 endfor
endfor

for i=0 to node_count
 if count_mode[i]>threshold then
  threshold=count_mode[i]
 endif
endfor
```

Gambar 3.3 *Pseudocode* perancangan perhitungan *Threshold*

3.3.3 Perancangan Pemilihan *Forwarding Node*

Setelah melakukan penghitungan jumlah *node* tetangga, akan dilakukan pemilihan *forwarding node*, yaitu *node* yang berhak untuk melakukan *rebroadcast* paket RREQ. Penentuan *forwarding node* dilakukan dengan cara membandingkan jumlah *node* tetangga dengan *threshold* atau batas nilai yang sudah ditentukan. Apabila jumlah tetangga yang dimiliki *node* tersebut melebihi *threshold*, maka *node* tersebut berhak melakukan *rebroadcast*, jika sebaliknya maka paket akan di-*drop*. Langkah tersebut dilakukan untuk mengurangi jumlah paket RREQ yang dikirim sehingga dapat mengurangi kemacetan yang terjadi pada jaringan. *Pseudocode* untuk pemilihan *forwarding node* dapat dilihat pada 4.

```
if count < threshold
      drop RREQ packet
end if
for i=0 to node_count do
 for j=i+1 to node_count do
  if neighbour_count[i]>neighbour_count[j]
  then
   tmp=neighbour_count[i]
   neighbour_count[i]=neighbour_count[j]
   neighbour_count[j]=tmp
  endif
 endfor
endfor
```

Gambar 3.4 Pseudocode Pemilihan *Forwarding Node*

3.4 Perancangan Simulasi pada NS-2

Simulasi VANETs pada NS-2 dilakukan dengan menggabungkan *file* skenario yang telah dibuat menggunakan SUMO dan *file* skrip dengan ekstensi .tcl yang berisikan konfigurasi lingkungan simulasi.

Kode yang diubah diantaranya adalah pencarian jumlah *node* tetangga pada *file* node.cc dan perbandingan dengan *threshold* pada *file* aodv.cc. Pada saat simulasi NS-2 dijalankan, maka *routing protocol* AODV akan menyeleksi *node* mana saja yang berhak melakukan *rebroadcast* paket RREQ.

3.5 Perancangan Metrik Analisis

Berikut ini merupakan parameter – parameter yang akan dianalisis pada Tugas Akhir ini untuk dapat membandingkan performa dari *routing protocol* AODV yang asli dengan AODV yang telah dimodifikasi:

3.5.1 *Packet Delivery Ratio* (PDR)

Packet delivery ratio merupakan perbandingan dari jumlah paket data yang dikirim dengan paket data yang diterima. PDR dapat menunjukkan keberhasilan paket yang dikirimkan. Semakin tinggi PDR artinya semakin berhasil pengiriman paket yang dilakukan Rumus untuk menghitung PDR dapat dilihat pada persamaan 3.1.

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

Keterangan:

PDR = *Packet Delivery Ratio*

received = banyak paket data yang diterima

sent = banyak paket data yang dikirimkan

3.5.2 *Average End-to-End Delay* (E2E)

Average End-to-End Delay dihitung berdasarkan rata-rata *delay* antara waktu paket data diterima dan waktu paket dikirimkan dalam satuan detik. Delay tiap paket didapat dari rentang waktu antara *node* asal saat mengirimkan paket dan *node* tujuan menerima paket. Delay tiap paket tersebut semua dijumlahkan dan dibagi dengan jumlah paket yang berhasil diterima, maka akan didapatkan rata – rata E2E, yang dapat dihitung dengan persamaan 3.2.

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

Keterangan:

E2E = *End-to-End Delay*

CBRRecvTime = Waktu *node* asal mengirimkan paket

CBRSentTime = Waktu *node* tujuan menerima paket

recvnum = Jumlah paket yang berhasil diterima

3.5.3 *Routing Overhead* (RO)

Routing Overhead adalah jumlah paket kontrol *routing* yang ditransmisikan per data paket ke *node* tujuan selama simulasi terjadi. *Routing Overhead* didapatkan dengan menjumlahkan semua paket kontrol *routing* yang ditransmisikan, baik itu paket *route request* (RREQ), *route reply* (RREP), maupun *route error* (RERR).. Perhitungan *Routing Overhead* dapat dilihat dengan persamaan 3.3.

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

3.5.4 *Forwarded Route Request* (RREQ F)

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

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

(Halaman ini sengaja dikosongkan)

BAB IV
IMPLEMENTASI

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

4.1 Implementasi Skenario Mobilitas

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

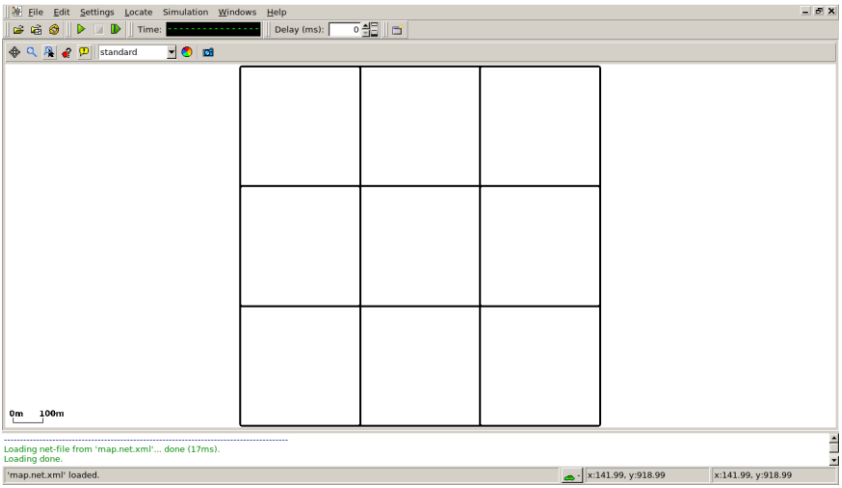
4.1.1 Skenario *Grid*

Dalam mengimplementasikan skenario *grid*, SUMO menyediakan *tools* untuk membuat peta *grid* yaitu netgenerate. Pada Tugas Akhir ini, penulis membuat peta *grid* dengan luas 1300 m x 1300 m yang terdiri dari titik persimpangan antara jalan vertikal dan jalan horisontal sebanyak 4 titik x 4 titik. Dengan jumlah titik persimpangan sebanyak 4 titik tersebut, makat terbentuk 9 buah petak. Sehingga untuk mencapai luas area sebesar 1300 m x 1300 m dibutuhkan luas per petak sebesar 400 m x 400 m. Berikut perintah netgenerate untuk membuat peta tersebut dengan kecepatan *default* kendaraan sebesar 20m/s dapat dilihat pada Gambar 4.1.

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

Gambar 4.1 Perintah netgenerate

Setelah itu akan didapat *file* peta berekstensi .xml. Gambar hasil peta yang telah dibuat dengan netgenerate dapat dilihat pada Gambar 4.2.



Gambar 4.2 Hasil *Generate* Peta *Grid*

Setelah peta terbentuk, maka dilakukan pembuatan *node* dan pergerakan *node* dengan menentukan titik awal dan titik akhir setiap *node* seacara random menggunakan *tools* randomTrips yang terdapat di SUMO. Perintah penggunaan *tools* randomTrips untuk membuat *node* sebanyak n *node* dengan pergerakannya dapat dilihat pada Gambar 4.3.

```
python $SUMO_HOME/tools/randomTrips.py -n
map.net.xml -e 58 -l --trip-
attributes="departLane=\"best\"
departSpeed=\"max\"
departPos=\"random_free\"" -o trip.trips.xml
```

Gambar 4.3 Perintah randomTrips

Selanjutnya dibuatkan rute yang digunakan kendaraan untuk mencapai tujuan dari *file* hasil sebelumnya menggunakan *tools* duarouter. Perintah penggunaan *tools* duarouter dapat dilihat pada Gambar 4.4.

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

Gambar 4.4 Perintah duarouter

Ketika menggunakan *tools* duarouter, SUMO memastikan bahwa jalur untuk *node-node* yang di*generate* tidak akan melenceng dari jalur peta yang sudah di*generate* menggunakan *tools* randomTrips. Selanjutnya untuk menjadikan peta dan pergerakan *node* yang telah di*generate* menjadi sebuah skenario dalam bentuk *file* berekstensi .xml, dibutuhkan sebuah *file* skrip dengan ekstensi .sumocfg untuk menggabungkan *file* peta dan rute pergerakan *node*. Isi dari *file* skrip .sumocfg dapat dilihat pada Gambar 4.5.

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

Gambar 4.5 File Skrip .sumocfg

File .sumocfg disimpan dalam direktori yang sama dengan *file* peta dan *file* rute pergerakan *node*. Untuk percobaan sebelum dikonversi, *file* .sumocfg dapat dibuka dengan menggunakan *tools* sumo-gui. Kemudian buat *file* skenario dalam bentuk *file* .xml dari sebuah *file* skrip berekstensi .sumocfg menggunakan *tools* SUMO. Perintah untuk menggunakan *tools* SUMO dapat dilihat pada Gambar 4.6.

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

Gambar 4.6 Perintah SUMO untuk membuat skenario .xml

File skenario berekstensi .xml selanjutnya dikonversi ke dalam bentuk *file* berekstensi .tcl agar dapat disimulasikan menggunakan NS-2. *Tools* yang digunakn untuk melakukan konversi

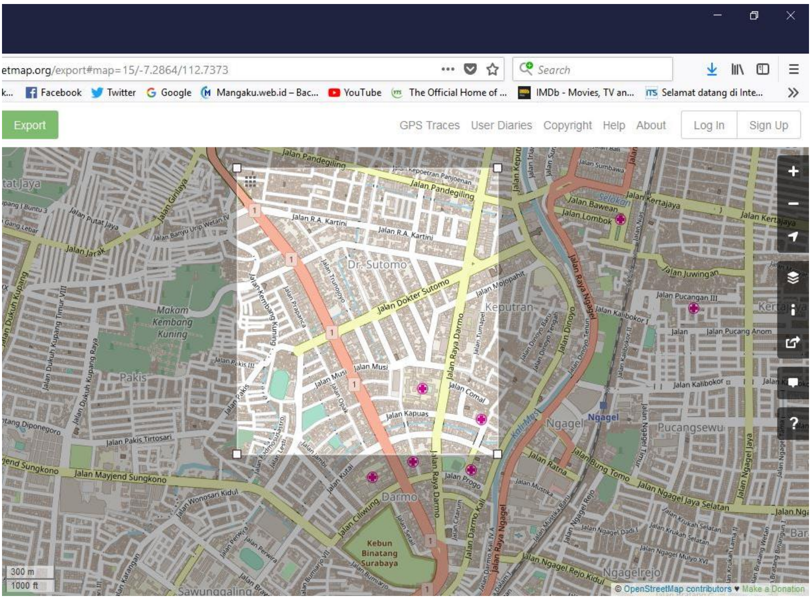
ini adalah traceExporter. Perintah untuk menggunakan traceExporter dapat dilihat pada Gambar 4.7.

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

Gambar 4.7 Perintah traceExporter

4.1.2 Skenario *Real*

Dalam mengimplementasikan skenario *real*, langkah pertama adalah menentukan area yang akan dijadikan area simulasi. Pada Tugas Akhir ini penulis mengambil area jalan sekitar Jl. Dr. Soetomo Surabaya. Setelah menentukan area simulasi, ekspor data peta dari OpenStreetMap seperti yang ditunjukkan pada Gambar 4.8.



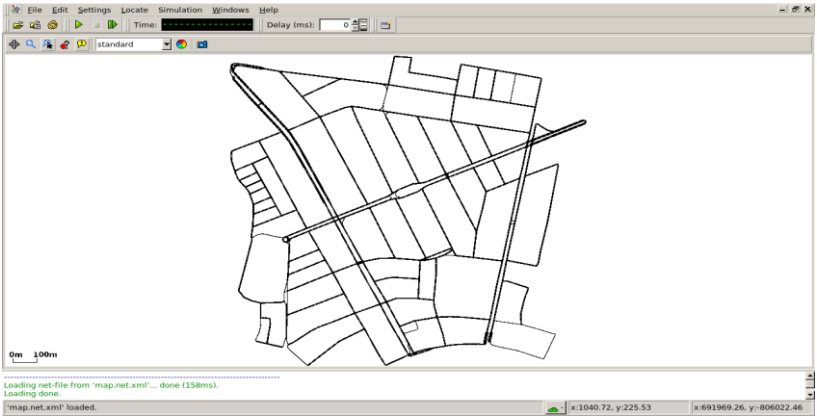
Gambar 4.8 Ekspor Peta dari OpenStreetMap

File hasil ekspor dari OpenStreetMap tersebut adalah *file* peta dengan ekstensi .osm. Kemudian konversi *file* .osm tersebut menjadi peta dalam bentuk *file* berekstensi .xml menggunakan *tools* netconvert dari SUMO. Perintah untuk menggunakan netconvert dapat dilihat pada Gambar 4.9.

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

Gambar 4.9 Perintah netconvert

Hasil konversi peta dari *file* berekstensi .osm menjadi *file* berekstensi .xml dapat dilihat menggunakan *tools* sumo-gui seperti yang ditunjukkan pada Gambar 4.10.



Gambar 4.10 Hasil Konversi Peta *Real*

Langkah selanjutnya sama dengan ketika membuat skenario mobilitas *grid*, yaitu membuat *node* asal dan *node* tujuan menggunakan *tool* randomTrips. Lalu membuat rute *node* untuk sampai ke tujuan menggunakan *tool* duarouter. Kemudian membuat *file* skenario berekstensi .xml menggunakan *tool* SUMO dengan bantuan *file* skrip berekstensi .sumocfg. Selanjutnya dilakukan konversi *file* skenario berekstensi .tcl untuk dapat disimulasikan pada

NS-2 menggunakan *tool* traceExporter. Perintah untuk menggunakan *tools* tersebut sama dengan ketika membuat skenario *grid* di atas.

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

Pada Tugas Akhir ini dilakukan modifikasi pada *routing protocol* AODV agar dapat mengurangi jumlah *forwarding node*, yaitu *node* yang bertugas untuk melakukan *rebroadcast* paket RREQ. Hal tersebut dilakukan dengan cara memilih *forwarding node* berdasarkan jumlah tetangga *node* tersebut dengan *threshold* yang dihitung berdasarkan fungsi yang berjalan setiap *i* interval yang membuat *threshold* bersifat adaptif, sehingga dapat dilihat peningkatan performa pada *routing* AODV yang telah dimodifikasi.

Implementasi modifikasi routing protocol AODV ini dibagi menjadi 3 bagian yaitu:

- Implementasi Penghitungan Jumlah *Node* Tetangga
- Implementasi Penghitungan *Threshold*
- Implementasi Pemilihan *Forwarding Node*

Kode implementasi dari routing protocol AODV pada NS-2 versi 2.35 berada pada direktori ns-2.35/aodv. Pada direktori tersebut terdapat beberapa file diantaranya seperti aodv.cc, aodv.h dan sebagainya. Pada Tugas Akhir ini, penulis memodifikasi *file* aodv.cc yang terdapat dalam *folder* ns-2.35/aodv untuk menghitung jumlah *node* tetangga, menghitung *threshold* dan menentukan forwarding node dan *file* aodv.h yang ada di dalam *folder* ns-2.35/aodv untuk mendaftarkan fungsi dan *timer* baru. Pada bagian ini penulis akan menjelaskan langkah – langkah dalam mengimplementasikan modifikasi *routing protocol* AODV untuk mengurangi jumlah *forwarding node* yang melakukan *rebroadcast*.

4.2.1 Implementasi Penghitungan Jumlah *Node* Tetangga

Langkah awal yang dilakukan untuk menghitung jumlah tetangga seperti yang telah dirancang pada subbab 3.3.1 adalah

dengan cara menangkap pesan HELLO *messages* dari *node* yang mengirimkannya.

Node yang mengirimkan HELLO *messages* sudah dapat dipastikan berada di sekitar *node* tersebut dan berada di *range* transmisi dari *node* yang sedang dieksekusi. Secara *default*, AODV menginformasikan siapa saja *node* tetangga yang ada di sekitar *node* tersebut. Terdapat fungsi nb_insert dan nb_delete yang digunakan untuk menambah atau menghapus node tetangga yang mendekat atau menjauh. Pada tugas akhir ini, penulis memanfaatkan kedua fungsi tersebut dengan tambahan modifikasi counter setiap penambahan atau pengurangan node. Kedua fungsi tersebut terdapat pada kode sumber aodv.cc yang terdapat dalam folder ns2.3.5/aodv. Untuk potongan kode tersebut bisa dilihat pada Gambar 4.11. Kode selengkapnya dapat dilihat di lampiran A1.

```
void AODV::nb_insert(nsaddr_t id)
{
  count_neighbour[index]+=1;

  AODV_Neighbor *nb = new
AODV_Neighbor(id);

  assert(nb);
  nb->nb_expire = CURRENT_TIME +
                  (1.5 *
ALLOWED_HELLO_LOSS * HELLO_INTERVAL);
  LIST_INSERT_HEAD(&nbhead, nb, nb_link);
  seqno += 2; // set of neighbors changed
  assert((seqno % 2) == 0);
}
```

```
void AODV::nb_delete(nsaddr_t id)
{
  count_neighbour[index]-=1;
  AODV_Neighbor *nb = nbhead.lh_first;
  log_link_del(id);
  seqno += 2; // Set of neighbors changed
  assert((seqno % 2) == 0);

  for (; nb; nb = nb->nb_link.le_next)
  {
    if (nb->nb_addr == id)
    {
      LIST_REMOVE(nb, nb_link);
      delete nb;
      break;
    }
  }
  handle_link_failure(id);
}
```

Gambar 4.11 Potongan modifikasi Kode Fungsi nb_insert() dan nb_delete()

4.2.2 Implementasi Perhitungan Threshold

Pada tahap selanjutnya setelah dapat mengetahui jumlah tetangga, langkah yang harus dilakukan selanjutnya adalah menghitung threshold secara adaptif untuk digunakan sebagai pembatasan *forwarding node* untuk melanjutkan paket RREQ.

Penghitungan ini dilakukan pada fungsi `countThreshold()` yang terletak pada kode sumber aodv.cc yang terletak pada ns2.35/aodv. Jumlah tetangga yang sudah ditemukan, disimpan dalam variabel *count_neighbour*. Jumlah tetangga kemudian akan dicari nilai average. nilai average dari jumlah tetangga akan dibuat sebagai acuan untuk Threshold. Fungsi ini berjalan setiap

interval yang ditentukan. Interval yang digunakan dalam uji coba adalah 5 detik, 10 detik dan 15 detik. Potongan kode untuk proses seleksi *forwarding node* dapat dilihat pada Gambar 4.12

```
for(int i=0;i<nodes_count;i++){
 for(int j=(i+1);j<nodes_count;j++){

if(count_neighbour[i]>count_neighbour[j]){
     int tmp;
     tmp=count_neighbour[i];
     count_neighbour[i]=count_neighbour[j];
     count_neighbour[j]=tmp;
    }
  }
}
for(int i=0;i<nodes_count;i++){
 count_mode[i]=0;
  for(int j=0;j<nodes_count;j++){

if(count_neighbour[i]==count_neighbour[j]){
      count_mode[i]++;
     }
  }
}

for(int i=0;i<nodes_count;i++){
  if(count_mode[i]>th ){
    th=count_mode[i];
  }
}
```

Gambar 4.12 *Potongan Kode Perhitun*gan Threshold

4.2.3 Implementasi Pemilihan *Forwarding Node*

Pada tahap selanjutnya setelah dapat mengetahui jumlah tetangga dan *Threshold* , langkah yang harus dilakukan selanjutnya adalah pemilihan *forwarding node* untuk melanjutkan paket RREQ.

Penghitungan ini dilakukan pada fungsi `recvRequest()` yang terletak pada kode sumber aodv.cc yang terletak pada ns2.35/aodv.. Apabila jumlah tetangga kurang dari *threshold* yang ditentukan dan bukan *node* sumber, maka paket RREQ akan di*drop*. Potongan kode untuk proses seleksi *forwarding node* dapat dilihat pada Gambar 4.13

```
if (count < threshold && rq->rq_dst != 
index){
     Packet::free(p);
     return;
}
```

Gambar 4.13 Potongan Kode Penyeleksian *Forwarding Node*

Dengan proses penyeleksian *node* mana saja yang dapat melanjutkan paket RREQ sudah dapat mengurangi jumlah paket *routing overhead* untuk paket RREQ. Kode implementasi ini diletakkan pada lampiran A.3 Kode Fungsi nb_remove().

4.3 Implementasi Simulasi pada NS-2

Implementasi simulasi VANETs diawali dengan pendeskripsian lingkungan simulasi pada sebuah *file* tcl. *File* ini berisikan konfigurasi setiap *node* dan langkah-langkah yang dilakukan selama simulasi. Potongan konfigurasi lingkungan simulasi dapat dilihat pada Gambar 4.14

```
set val(chan)    Channel/WirelessChannel
set val(prop)    Propagation/TwoRayGround
set val(netif)   Phy/WirelessPhy
set val(mac)     Mac/802_11
set val(ifq)     Queue/DropTail/PriQueue
set val(ll)      LL
set val(ant)     Antenna/OmniAntenna
set opt(x)       1500
set opt(y)       1500
set val(ifqlen)  1000
set val(nn)      60
set val(seed)    1.0
set val(adhocRouting) AODV
set val(stop)    200
set val(cp)      "cbr60.txt"
set val(sc)      "scenario1.txt"
```

Gambar 4.14 Implementasi Simulasi NS-2

Pada konfigurasi dilakukan pemanggilan terhadap *file traffic* yang berisikan konfigurasi *node* asal, *node* tujuan, pengiriman paket, serta *file* skenario yang berisi pergerakan *node* yang telah di*generate* oleh SUMO. Kode implementasi pada NS-2 dapat dilihat pada lampiran A.4 Kode Skenario NS-2.

Konfigurasi untuk *file traffic* bisa dilakukan dengan membuat *file* berekstensi .txt untuk menyimpan konfigurasi tersebut. Pada *file* konfigurasi lingkungan simulasi, *file traffic* tersebut dimasukkan agar dibaca sebagai *file traffic*. Potongan konfigurasi *file traffic* dapat dilihat pada Gambar 4.15.

```
set udp_(0) [new Agent/UDP]
$ns_ attach-agent $node_(58) $udp_(0)
set null_(0) [new Agent/Null]
$ns_ attach-agent $node_(59) $null_(0)
set cbr_(0) [new Application/Traffic/CBR]
$cbr_(0) set packetSize_ 512
$cbr_(0) set interval_ 1
$cbr_(0) set random_ 1
$cbr_(0) set maxpkts_ 10000
$cbr_(0) attach-agent $udp_(0)
$ns_ connect $udp_(0) $null_(0)
$ns_ at 2.5568388786897245 "$cbr_(0) start"
```

Gambar 4.15 Implementasi Simulasi File Traffic

Pada konfigurasi tersebut, ditentukan *node* sumber dan *node* tujuan pengiriman paket. Pengiriman dimulai pada detik ke- 2.55. Implementasi konfigurasi *file traffic* untuk simulasi pada NS-2 dapat dilihat pada lampiran A.5 Kode Konfigurasi *Traffic*

4.4 Implementasi Metrik Analisis

Simulasi yang telah dijalankan oleh NS-2 menghasilkan sebuah *trace file* yang berisikan data mengenai apa saja yang terjadi selama simulasi dalam bentuk *file* berekstensi .tr. Dari data *trace file* tersebut, dapat dilakukan analisis performa *routing protocol* dengan mengukur beberapa metrik. Pada Tugas Akhir ini, metrik yang akan dianalisis adalah PDR, E2E, dan RO, dan Forwarded Route Request (RREQ F).

4.4.1 Implementasi *Packet Delivery Ratio* (PDR)

Pada subbab 2.3.2 telah ditunjukkan contoh struktur data *event* yang dicatat dalam *trace file* oleh NS-2. Kemudian, pada

persamaan 3.1 telah dijelaskan bagaimana menghitung PDR. Skrip awk untuk menghitung PDR berdasarkan kedua informasi tersebut dapat dilihat pada lampiran A.6 Kode Skrip AWK *Packet Delivery Ratio*.

PDR didapatkan dengan cara menghitung setiap baris terjadinya *event* pengiriman dan penerimaan paket data yang dikirim melalui agen pada *trace file*. Skrip menyaring setiap baris yang mengandung *string* AGT karena kata kunci tersebut menunjukan *event* yang berhubungan dengan paket komunikasi data. Penghitungan dilakukan dengan menjumlahkan paket yang dikirimkan dan paket yang diterima dengan menggunakan karakter pada kolom pertama sebagai *filter*. Kolom pertama menunjukan event yang terjadi dari sebuah paket. Setelah itu nilai PDR dihitung dengan cara persamaan 3.1. Pseudocode untuk menghitung PDR dapat dilihat pada Gambar 4.16.

```
sent = 0
received = 0
for i = 1 to the number of rows
     if in a row contains "s" and AGT then
          sent++
     else if in a row contains "r" and AGT then
          received++
     end if
pdr = received / sent
```

Gambar 4.16 Pseudocode untuk Menghitung PDR

Contoh perintah pengeksekusian skrip awk untuk menganalisis *trace file* adalah `awk -f pdr.awk tracefile.tr`.

4.4.2 Implementasi *Average End-to-End Delay (E2E)*

Skrip awk untuk menghitung E2E dapat dilihat pada lampiran A.7 Kode Skrip AWK Rata-Rata *End-to-End Delay*.

Dalam perhitungan E2E, langkah yang digunakan untuk mendapatkan E2E hampir sama dengan ketika mencari PDR, hanya saja yang perlu diperhatikan adalah waktu dari sebuah *event* yang tercatat pada kolom ke-2 dengan *filter event* pada kolom ke-4 adalah layer AGT dan *event* pada kolom pertama guna membedakan paket dikirim atau diterima. Setelah seluruh baris yang memenuhi didapatkan, akan dihitung *delay* dari paket dengan mengurangi waktu dari paket diterima dengan waktu dari paket dikirim dengan syarat memiliki *id* paket yang sama.

Setelah mendapatkan *delay* paket, langkah selanjutnya adalah dengan mencari rata-rata dari *delay* tersebut dengan menjumlahkan semua *delay* paket dan membaginya dengan jumlah paket. *Pseudocode* untuk menghitung rata-rata E2E dapat dilihat pada Gambar 4.17.

```
sum_delay = 0
counter = 0

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

Gambar 4.17 Pseudocode untuk Perhitungan Rata-Rata E2E

Contoh perintah pengeksekusian skrip awk untuk menganalisis *trace file* adalah `awk –f e2e.awk tracefile.tr`.

4.4.3 Implementasi *Routing Overhead* (RO)

Seperti yang telah dijelaskan sebelumnya, *routing overhead* merupakan jumlah dari paket kontrol *routing* baik itu RREQ, RREP, maupun RERR. Dengan begitu, untuk mendapatkan RO yang perlu dilakukan adalah menjumlahkan tiap paket dengan *filter event sent* pada kolom pertama dan *event layer* RTR pada kolom ke-4. Perhitungan RO telah dijelaskan pada persamaan 3.3. Skrip AWK untuk menghitung RO dapat dilihat pada lampiran A.8 Kode Skrip AWK *Routing Overhead*. *Pseudocode* untuk menghitung RO dapat dilihat pada Gambar 4.18.

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

Gambar 4.18 Pseudocode untuk Perhitungan *Routing Overhead*

Contoh perintah pengeksekusian skrip awk untuk menganalisis *trace file* adalah `awk -f ro.awk scenario1.tr`.

4.4.4 Implementasi *Forwarded Route Request* (RREQ F)

Forwarded Route Request (RREQ F) adalah jumlah paket *control route request* yang diteruskan per-data paket ke *node* tujuan selama simulasi terjadi. Dengan begitu, untuk mendapatkan RREQ F yang perlu dilakukan adalah menjumlahkan tiap paket dengan *filter event sent* pada kolom pertama, *event layer* RTR pada kolom ke-4, dan *event layer route request* pada kolom-25. Penyaringan juga dilakukan pada kolom ke-3 yang menunjukan *node* id. Selama *node* bukan *node* sumber, maka akan terus dilakukan penjumlahan baris yang terdapat pada *file* dengan ekstensi .tr. Perhitungan RREQ F telah

dijelaskan pada persamaan 3.4. Skrip awk untuk menghitung RREQ F dapat dilihat pada lampiran

A.9 Kode Skrip *AWK Forwarded Route Request* . Pseudocode untuk menghitung RREQ F dapat dilihat pada Gambar 4.19

```
rreqf = 0
for i = 1 to the number of rows
     if packet is AODV and RREQ and not source
node then
        rreqf++
     end if
```

Gambar 4.19 Pseudocode Perhitungan *Forwarded Route Request*

Contoh perintah pengeksekusian skrip awk untuk menganalisis *trace file* adalah `awk -f rreqf.awk scenario1.tr.`

(Halaman ini sengaja dikosongkan)

BAB V
UJICOBA DAN EVALUASI

Pada bab ini akan dilakukan tahap ujicoba dan evaluasi sesuai dengan rancangan dan implementasi. Dari hasil yang didapatkan setelah melakukan uji coba, akan dilakukan evaluasi sehingga dapat ditarik kesimpulan pada bab selanjutnya.

5.1 Lingkungan Uji Coba

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

Tabel 5.1 Spesifikasi Perangkat yang Digunakan

Komponen	**Spesifikasi**
CPU	Intel(R) Core(TM) i5-4200U 1.60 GHz
Sistem Operasi	Xubuntu 16.04 LTS
Linux Kernel	Linux kernel 4.4
Memori	8.0 GB

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

- SUMO versi 0.25.0 untuk pembuatan skenario mobilitas VANETs.
- JOSM versi 10301 untuk penyuntingan peta OpenStreetMap.
- NS-2 versi 2.35 untuk simulasi skenario VANETs.

Parameter lingkungan uji coba yang digunakan pada NS-2 dapat diihat pada Tabel 5.2. Pengujian dilakukan dengan menjalankan skenario yang disimulasikan pada NS-2. Dari simulasi tersebut dihasilkan sebuah *trace file* dengan ekstensi .tr yang akan dianalisis dengan bantuan skrip awk untuk mendapatkan PDR, E2E, RO, dan RREQ F menggunakan kode yang terdapat pada lampiran A.6 Kode Skrip AWK *Packet Delivery Ratio*, A.7 Kode Skrip AWK Rata-Rata

End-to-End Delay, A.8 Kode Skrip AWK *Routing Overhead*, dan A.9 Kode Skrip Awk *Forwarded Route Request*.

Tabel 5.2 Lingkungan Uji Coba

No.	Parameter	Spesifikasi
1	Network simulator	NS-2.35
2	Routing protocol	AODV
3	Waktu simulasi	200 detik
4	Area simulasi	1500 m x 1500 m
5	Jumlah *Node*	60, 150, 300
6	Radius transmisi	400m
7	Kecepatan maksimum	20 m/s
8	Protokol MAC	IEEE 802.11p
9	Model Propagasi	*Two-ray ground*

5.2 Hasil Uji Coba

5.2.1 Hasil Uji Coba Skenario *Grid*

Pengujian pada skenario *grid* digunakan untuk melihat perbandingan PDR, RO, RREQ F, dan E2E antara *routing protocol* AODV asli dan AODV yang telah dimodifikasi dalam pemilihan *node* yang dapat menerima paket *route request*.

Pengambilan data uji PDR, RO, RREQ F, dan E2E pada skenario *grid* dilakukan sebanyak 10 kali dengan skenario mobilitas *random* pada peta *grid* dengan luas area 1300 m x 1300 m dan *node* sebanyak 60 untuk lingkungan yang jarang, 150 *node* untuk lingkungan yang sedang, dan 300 *node* untuk lingkungan yang padat dilakukan pada kecepatan standar yaitu 20 m/s. Hasil analisis dapat dilihat pada Tabel 5.3, Tabel 5.4, Tabel 5.5, dan Tabel 5.6.

Tabel 5.3 Hasil Perhitungan PDR Skenario Grid

Jumlah *Node*	AODV Asli	AODV Modifikasi	Perbedaan
60	0,85937	0,80765	-0,05172
150	0,88319	0,86713	-0,01606
300	0,86558	0,86735	+0,00177

Tabel 5.4 Hasil Perhitungan E2E Skenario Grid

Jumlah *Node*	AODV Asli	AODV Modifikasi	Perbedaan
60	471,2205	507,263	-36,0425
150	523,1213	576,07911	-52,95781
300	290,29838	380,25793	-89.95955

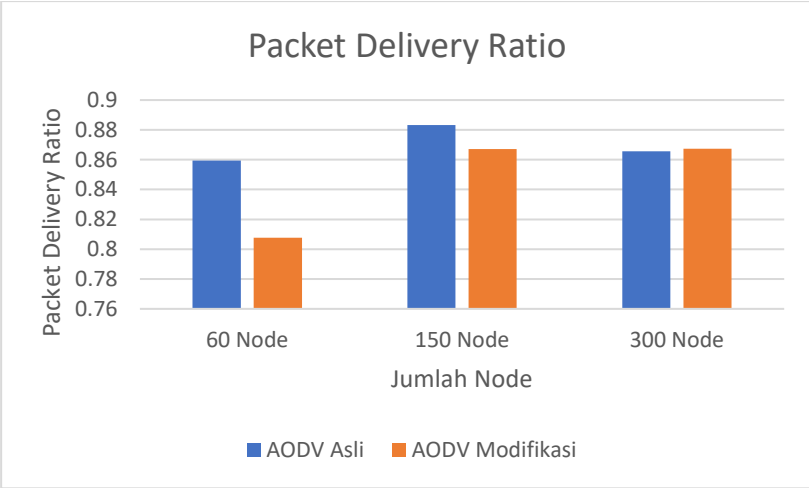
Tabel 5.5 Hasil Perhitungan RO Skenario Grid

Jumlah *Node*	AODV Asli	AODV Modifikasi	Perbedaan
60	20112	14677	+5435
150	84185	69987	+14198
300	71452	66980	+4472

Tabel 5.6 Hasil Perhitungan RREQ F Skenario Grid

Jumlah *Node*	AODV Asli	AODV Modifikasi	Perbedaan
60	2099,5	1999,2	+100,3
150	9179,3	7724,4	+1454,9
300	7107	5784,7	+1322,3

Dari data di atas, dibuat grafik yang merepresentasikan hasil perhitungan PDR, RO, RREQ F, dan E2E yang ditunjukkan pada Gambar 5.1, Gambar 5.2, Gambar 5.3, dan Gambar 5.4



Gambar 5.1 Grafik PDR Skenario *Grid*

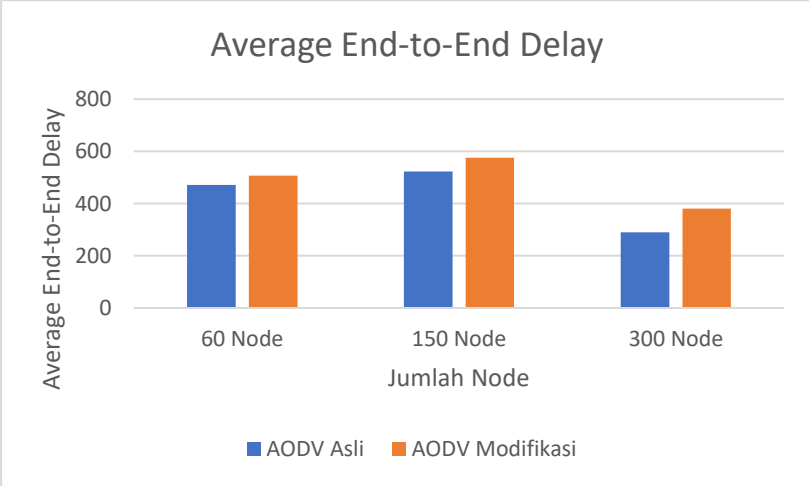
Berdasarkan grafik pada Gambar 5.1, dalam data tersebut dapat kita lihat *routing protocol* AODV asli dan AODV yang telah dimodifikasi. Pada lingkungan yang jarang dengan *node* berjumlah 60 dan dibandingkan dengan hasil PDR AODV asli, AODV modifikasi dengan menghasilkan perbedaan selisih PDR sebesar 0,05172, dimana terjadi penurunan sebesar 60,2 %. Berdasarkan hal tersebut, routing protocol AODV modifikasi tidak berhasil mengungguli *routing protocol* AODV asli.

Pada lingkungan yang sedang dengan *node* berjumlah 150 dan dibandingkan dengan hasil PDR AODV asli, AODV modifikasi menghasilkan perbedaan selisih PDR sebesar 0,01606, dimana terjadi penurunan sebesar 18,18 %. Berdasarkan hal tersebut, routing protocol AODV modifikasi tidak berhasil mengungguli *routing protocol* AODV asli.

Pada lingkungan yang padat dengan *node* berjumlah 300 dan dibandingkan dengan hasil PDR AODV asli, AODV modifikasi menghasilkan perbedaan selisih PDR sebesar 0,00177, dimana terjadi kenaikan sebesar 2,04 %. Berdasarkan hal tersebut, routing protocol

AODV modifikasi berhasil mengungguli *routing protocol* AODV asli.

Berdasarkan hasil simulasi pada ketiga lingkungan tersebut, dapat dilihat bahwa dengan menggunakan AODV modifikasi pada lingkungan yang padat dengan *node* berjumlah 300 menghasilkan PDR yang lebih baik daripada menggunakan AODV asli, AODV modifikasi pada lingkungan yang jarang dengan *node* berjumlah 60 dan AODV modifikasi pada lingkungan yang sedang dengan *node* berjumlah 150 menghasilkan PDR yang tidak dapat mengungguli AODV asli. Nilai kenaikan PDR pada skenario grid dengan menggunakan AODV modifikikasi pada *node* yang berjumlah 300 adalah sebesar 2,04%. Sedangkan nilai penurunan PDR pada scenario grid dengan menggunakan AODV modifikasi pada *node* yang berjumlah 60 dan 150 adalah sebesar 39.19%.



Gambar 5.2 Grafik E2E Skenario *Grid*

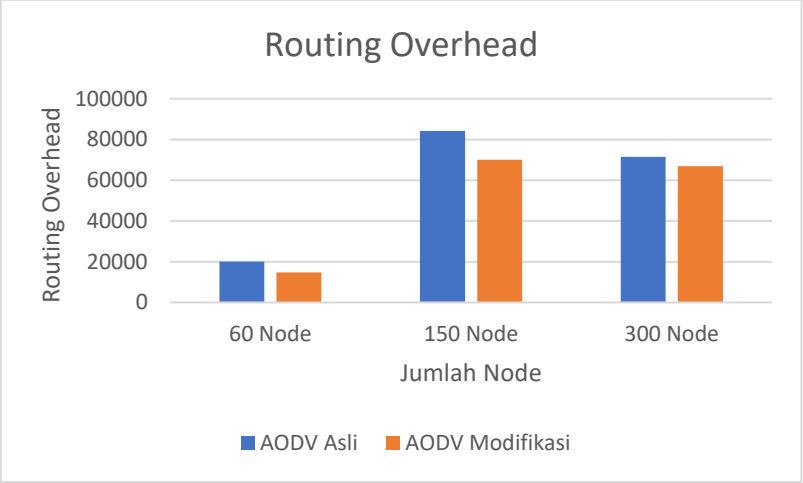
Berdasarkan grafik pada Gambar 5.2, dalam data tersebut dapat kita lihat rata-rata E2E antara *routing protocol* AODV asli dan AODV yang telah dimodifikasi. Pada lingkungan yang jarang dengan jumlah *node* 60 dan dibandingkan dengan AODV asli, AODV

modifikasi menghasilkan perbedaan selilish *end-to-end delay* sebesar 36,0425 ms dimana AODV asli unggul dalam hal ini. Dimana terjadi kenaikan sebesar 7,1%.

Pada lingkungan yang sedang dengan jumlah 150 *node* dan dibandingkan dengan AODV asli, AODV modifikasi menghasilkan perbedaan selilish *end-to-end delay* sebesar 52,95781 ms dimana AODV asli unggul dalam hal ini. Dimana terjadi kenaikan sebesar 9,1%.

Pada lingkungan yang padat dengan jumlah 300 *node* dan dibandingkan dengan AODV asli, AODV modifikasi menghasilkan perbedaan selilish *end-to-end delay* sebesar 89,95955 ms dimana AODV asli unggul dalam hal ini. Dimana terjadi kenaikan 23,65%.

Dari hasil tersebut, pada lingkungan yang jarang, sedang, maupun padat AODV asli lebih unggul dalam hal E2E. Routing protocol AODV asli lebih unggul daripada *routing protocol* AODV yang telah dimodifikasi. Hasil kenaikan E2E adalah sebesar 13,28% .



Gambar 5.3 Grafik Routing Overhead Skenario *Grid*

Berdasarkan grafik pada Gambar 5.3, dalam data tersebut dapat kita lihat *routing protocol* AODV asli dan AODV yang telah dimodifikasi. Pada lingkungan yang jarang dengan jumlah 60 *node*

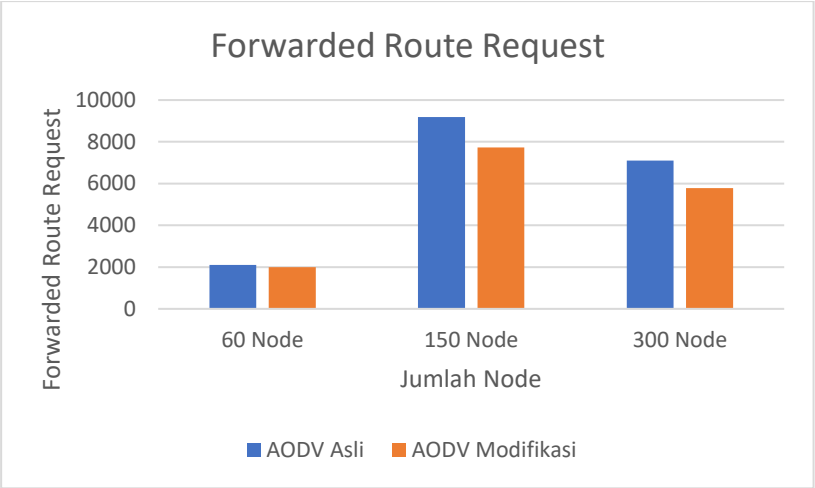
dan dibandingkan dengan AODV asli, AODV modifikasi menghasilkan perbedaan selisih RO sebesar 5435, dimana terjadi penurunan RO sebesar 27,02%. Berdasarkan hasil tersebut, maka *routing protocol* AODV yang telah dimodifikasi lebih unggul dalam hal RO karena menghasilkan RO yang lebih rendah dari AODV asli.

Pada lingkungan yang sedang dengan jumlah 150 *node* dan dibandingkan dengan AODV asli, AODV modifikasi menghasilkan perbedaan selisih RO sebesar 14198, dimana terjadi penurunan RO sebesar 16,86%. Berdasarkan hasil tersebut, maka *routing protocol* AODV yang telah dimodifikasi lebih unggul dalam hal RO karena menghasilkan RO yang lebih rendah dari AODV asli.

Pada lingkungan yang padat dengan jumlah 300 *node* dan dibandingkan dengan AODV asli, AODV modifikasi menghasilkan perbedaan selisih RO sebesar 4472. dimana terjadi penurunan RO sebesar 6,25%. Berdasarkan hasil tersebut, maka *routing protocol* AODV yang telah dimodifikasi lebih unggul dalam hal RO tersebut karena menghasilkan RO yang lebih rendah dari AODV asli.

Dari hasil ketiga lingkungan tersebut dapat dilihat bahwa *routing protocol* AODV yang telah di modifikasi pada lingkungan dengan *node* yang jarang, sedang, maupun padat menghasilkan RO yang lebih sedikit daripada *routing protocol* AODV asli. AODV modifikasi mengungguli AODV asli dengan nilai penurunan RO pada AODV yang dimodifikasi adalah sebesar 16,71%. Dapat dilihat pula bahwa AODV yang telah dimodifikasi menghasilkan RO yang lebih bagus atau dalam hal ini lebih rendah daripada AODV asli.

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



Gambar 5.4 Grafik *Forwarded Route Request* Skenario *Grid*

Berdasarkan grafik pada Gambar 5.4, dalam data tersebut dapat kita lihat *routing protocol* AODV asli dan AODV yang telah dimodifikasi. Pada lingkungan yang jarang dengan jumlah 60 *node* dan dibandingkan dengan AODV asli, AODV modifikasi menghasilkan perbedaan selisih *forwarded route request* sebesar 100,3, dimana terjadi penurunan RREQ F sebesar 4,77%. Dari hasil tersebut *routing protocol* AODV modifikasi lebih unggul dalam hal RREQ F karena menghasilkan RREQ F yang lebih rendah dari *routing protocol* AODV asli.

Pada lingkungan yang sedang dengan jumlah 150 *node* dan dibandingkan dengan AODV asli, AODV modifikasi menghasilkan perbedaan selisih *forwarded route request* sebesar 1454,9, dimana terjadi penurunan RREQ F sebesar 15,84%. Dari hasil tersebut *routing protocol* AODV modifikasi lebih unggul dalam hal RREQ F karena menghasilkan RREQ F yang lebih rendah dari *routing protocol* AODV asli.

Pada lingkungan yang padat dengan jumlah 300 *node* dan dibandingkan dengan AODV asli, AODV modifikasi menghasilkan perbedaan selisih *forwarded route request* sebesar 1322,3, dimana

terjadi penurunan RREQ F sebesar 18,6%. Dari hasil tersebut *routing protocol* AODV modifikasi lebih unggul dalam hal RREQ F karena menghasilkan RREQ F yang lebih rendah dari *routing protocol* AODV asli.

Dari hasil ketiga lingkungan tersebut dapat dilihat bahwa *routing protocol* AODV yang telah di modifikasi pada lingkungan dengan *node* yang jarang, sedang, maupun padat menghasilkan RO yang lebih rendah daripada *routing protocol* AODV asli. AODV modifikasi tidak dapat mengungguli AODV asli, AODV modifikasi mengalami penurunan RREQ F dengan nilai adalah sebesar 13,07%. Dapat dilihat pula bahwa AODV yang telah dimodifikasi dapat menghasilkan RREQ F yang lebih bagus atau dalam hal ini lebih rendah daripada RREQ F asli.

5.2.2 Hasil Uji Coba Skenario *Real*

Pengujian pada skenario *grid* digunakan untuk melihat perbandingan PDR, RO, RREQ F, dan E2E antara *routing protocol* AODV asli dan AODV yang telah dimodifikasi dalam pemilihan *node* yang dapat menerima paket *route request*.

Pengambilan data uji PDR, RO, RREQ F, dan E2E pada skenario *grid* dilakukan sebanyak 10 kali dengan skenario mobilitas *random* pada peta *grid* dengan luas area 1300 m x 1300 m dan *node* sebanyak 60 untuk lingkungan yang jarang, 150 *node* untuk lingkungan yang sedang, dan 300 *node* untuk lingkungan yang padat dilakukan pada kecepatan standar yaitu 20 m/s. Hasil analisis dapat dilihat pada Tabel 5.7, Tabel 5.8, Tabel 5.9, dan Tabel 5.10.

Tabel 5.7 Hasil Perhitungan PDR pada Skenario Real

Jumlah *Node*	AODV Asli	AODV Modifikasi	Perbedaan
60	0,84357	0,80287	-0,0407
150	0,91164	0,83673	-0,07491
300	0,85961	0,73815	-0,12146

Tabel 5.8 Hasil Perhitungan E2E pada Skenario Real

Jumlah *Node*	AODV Asli	AODV Modifikasi	Perbedaan
60	554,47328	572,15112	-17,67784
150	254,56383	239,64175	+14,92208
300	180,11612	285,46415	-105,34803

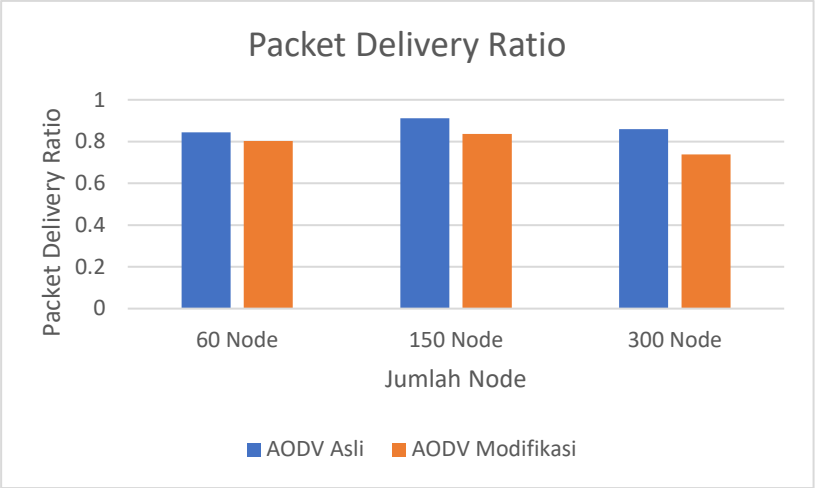
Tabel 5.9 Hasil Perhitungan RO pada Skenario Real

Jumlah *Node*	AODV Asli	AODV Modifikasi	Perbedaan
60	19519	14697	+4822
150	42658	39695	+2963
300	66389	62655	+3734

Tabel 5.10 Hasil Perhitungan RREQ F pada Skenario Real

Jumlah *Node*	AODV Asli	AODV Modifikasi	Perbedaan
60	2257,1	2068,1	+189
150	1920,9	2098,5	-177,6
300	3255,375	2041,25	+1214,125

Dari data di atas, dibuat grafik yang merepresentasikan hasil perhitungan PDR, E2E, RO, dan RREQ F yang ditunjukkan pada Gambar 5.5, Gambar 5.6, Gambar 5.7, dan Gambar 5.8



Gambar 5.5 Grafik PDR Skenario *Real*

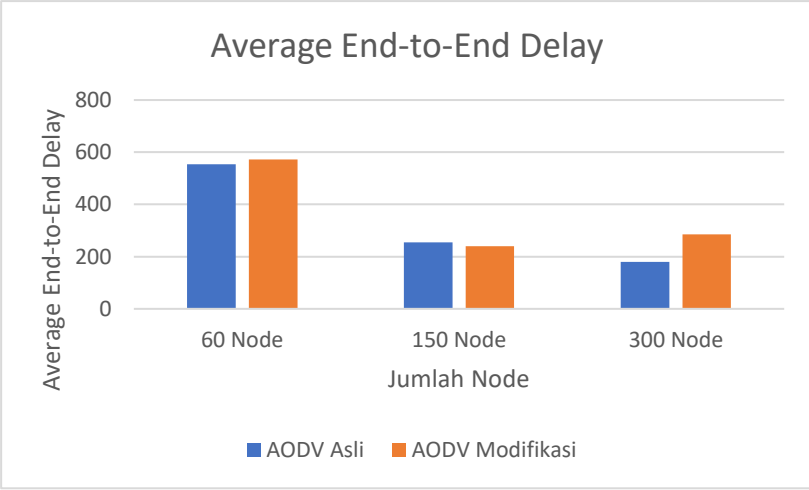
Berdasarkan grafik pada Gambar 5.5, dalam data tersebut dapat kita lihat *routing protocol* AODV asli dan AODV yang telah dimodifikasi. Pada lingkungan yang jarang dengan *node* berjumlah 60 dan dibandingkan dengan hasil PDR AODV asli, AODV modifikasi dengan menghasilkan perbedaan selisih PDR sebesar 0,0407, dimana terjadi penurunan sebesar 4,82%. Berdasarkan hal tersebut, routing protocol AODV modifikasi tidak berhasil mengungguli *routing protocol* AODV asli.

Pada lingkungan yang sedang dengan *node* berjumlah 150 dan dibandingkan dengan hasil PDR AODV asli, AODV modifikasi menghasilkan perbedaan selisih PDR sebesar 0.07491, dimana terjadi penurunan sebesar 8,21%. Berdasarkan hal tersebut, routing protocol AODV modifikasi tidak berhasil mengungguli *routing protocol* AODV asli.

Pada lingkungan yang padat dengan *node* berjumlah 300 dan dibandingkan dengan hasil PDR AODV asli, AODV modifikasi menghasilkan perbedaan selisih PDR sebesar 0,12146, dimana terjadi penurunan sebesar 14,12%. Berdasarkan hal tersebut, routing protocol

AODV modifikasi tidak berhasil mengungguli *routing protocol* AODV asli.

Berdasarkan hasil simulasi pada ketiga lingkungan tersebut, dapat dilihat bahwa dengan menggunakan AODV modifikasi pada lingkungan yang jarang dengan *node* berjumlah 60, AODV modifikasi pada lingkungan yang sedang dengan *node* berjumlah 150, dan AODV modifikasi pada lingkungan yang padat dengan *node* berjumlah 300 menghasilkan PDR yang tidak dapat mengungguli AODV asli. Nilai rata-rata penurunan PDR pada scenario real dengan menggunakan AODV modifikasi pada *node* yang berjumlah 60, 150, dan 300 adalah sebesar 9.05%.



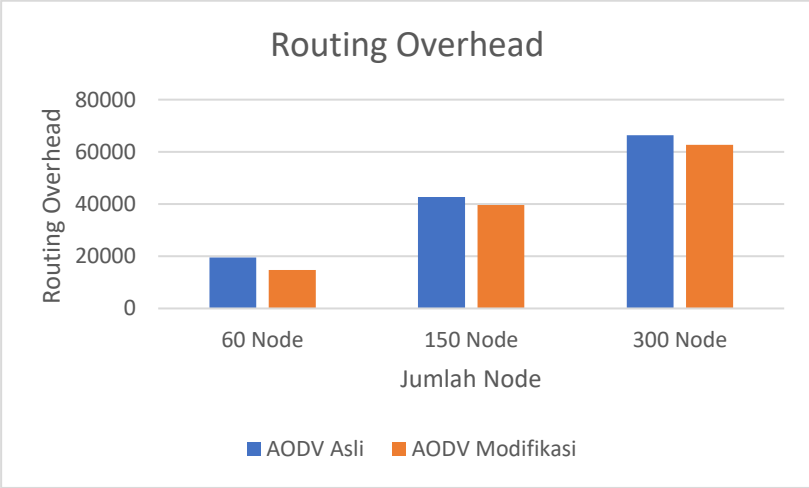
Gambar 5.6 Grafik E2E pada Skenario *Real*

Berdasarkan grafik pada Gambar 5.6, dalam data tersebut dapat kita lihat rata-rata E2E antara *routing protocol* AODV asli dan AODV yang telah dimodifikasi. Pada lingkungan yang jarang dengan jumlah *node* 60 dan dibandingkan dengan AODV asli, AODV modifikasi menghasilkan perbedaan selilish *end-to-end delay* sebesar 17,67784 ms dimana AODV asli unggul dalam hal ini. Dimana terjadi kenaikan sebesar 3,08%.

Pada lingkungan yang sedang dengan jumlah 150 *node* dan dibandingkan dengan AODV asli, AODV modifikasi menghasilkan perbedaan selilish *end-to-end delay* sebesar 14,92208 ms dimana AODV modifikasi unggul dalam hal ini. Dimana terjadi penurunan sebesar 5,86%.

Pada lingkungan yang padat dengan jumlah 300 *node* dan dibandingkan dengan AODV asli, AODV modifikasi menghasilkan perbedaan selilish *end-to-end delay* sebesar 105,34803 ms dimana AODV asli unggul dalam hal ini. Dimana terjadi kenaikan 36,9%.

Dari hasil tersebut, pada lingkungan yang jarang dengan *node* berjumlah 60 dan padat dengan *node* berjumlah 300 AODV asli lebih unggul dalam hal E2E. Sedangkan pada lingkungan yang sedang dengan *node* berjumlah 150 AODV modifikasi lebih unggul dalam hal E2E. Hasil kenaikan E2E pada *node* 60 dan pada *node* 300 adalah sebesar 19,99% . Sedangka hasil penurunan E2E pada *node* 150 adalah sebesar 5,86%



Gambar 5.7 Grafik RO Skenario *Real*

Berdasarkan pada grafik Gambar 5.7, dalam data tersebut dapat kita lihat *routing protocol* AODV asli dan AODV yang telah

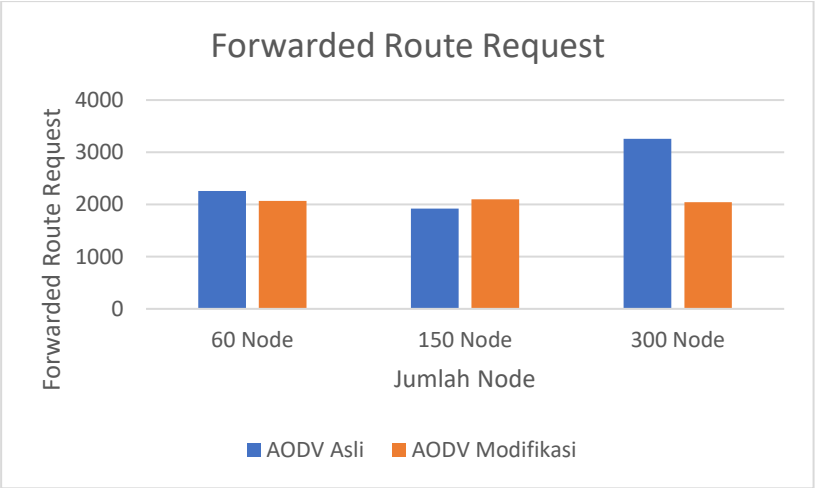
dimodifikasi. Pada lingkungan yang jarang dengan jumlah 60 *node* dan dibandingkan dengan AODV asli, AODV modifikasi menghasilkan perbedaan selisih RO sebesar 4822, dimana terjadi penurunan RO sebesar 24,7%. Berdasarkan hasil tersebut, maka *routing protocol* AODV yang telah dimodifikasi lebih unggul dalam hal RO karena menghasilkan RO yang lebih rendah dari AODV asli.

Pada lingkungan yang sedang dengan jumlah 150 *node* dan dibandingkan dengan AODV asli, AODV modifikasi menghasilkan perbedaan selisih RO sebesar 2963, dimana terjadi penurunan RO sebesar 6,94%. Berdasarkan hasil tersebut, maka *routing protocol* AODV yang telah dimodifikasi lebih unggul dalam hal RO karena menghasilkan RO yang lebih rendah dari AODV asli.

Pada lingkungan yang padat dengan jumlah 300 *node* dan dibandingkan dengan AODV asli, AODV modifikasi menghasilkan perbedaan selisih RO sebesar 3734. dimana terjadi penurunan RO sebesar 5,62%. Berdasarkan hasil tersebut, maka *routing protocol* AODV yang telah dimodifikasi lebih unggul dalam hal RO tersebut karena menghasilkan RO yang lebih rendah dari AODV asli.

Dari hasil ketiga lingkungan tersebut dapat dilihat bahwa *routing protocol* AODV yang telah di modifikasi pada lingkungan dengan *node* yang jarang, sedang, maupun padat menghasilkan RO yang lebih sedikit daripada *routing protocol* AODV asli. AODV modifikasi mengungguli AODV asli dengan nilai penurunan RO pada AODV yang dimodifikasi adalah sebesar 12,42%. Dapat dilihat pula bahwa AODV yang telah dimodifikasi menghasilkan RO yang lebih bagus atau dalam hal ini lebih rendah daripada AODV asli.

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



Gambar 5.8 Grafik RREQ F Skenario *Real*

Berdasarkan grafik pada Gambar 5.8, dalam data tersebut dapat kita lihat *routing protocol* AODV asli dan AODV yang telah dimodifikasi. Pada lingkungan yang jarang dengan jumlah 60 *node* dan dibandingkan dengan AODV asli, AODV modifikasi menghasilkan perbedaan selisih *forwarded route request* sebesar 189, dimana terjadi penurunan RREQ F sebesar 8,37%. Dari hasil tersebut *routing protocol* AODV modifikasi lebih unggul dalam hal RREQ F karena menghasilkan RREQ F yang lebih rendah dari *routing protocol* AODV asli.

Pada lingkungan yang sedang dengan jumlah 150 *node* dan dibandingkan dengan AODV asli, AODV modifikasi menghasilkan perbedaan selisih *forwarded route request* sebesar 177,6, dimana terjadi kenaikan RREQ F sebesar 8,46%. Dari hasil tersebut *routing protocol* AODV asli lebih unggul dalam hal RREQ F karena menghasilkan RREQ F yang lebih rendah dari *routing protocol* AODV yang telah dimodifikasi.

Pada lingkungan yang padat dengan jumlah 300 *node* dan dibandingkan dengan AODV asli, AODV modifikasi menghasilkan perbedaan selisih *forwarded route request* sebesar 1214,125, dimana

terjadi penurunan RREQ F sebesar 37,29%. Dari hasil tersebut *routing protocol* AODV modifikasi lebih unggul dalam hal RREQ F karena menghasilkan RREQ F yang lebih rendah dari *routing protocol* AODV asli.

Dari hasil ketiga lingkungan tersebut dapat dilihat bahwa *routing protocol* AODV yang telah di modifikasi pada lingkungan yang jarang dengan *node* 60 dan pada lingkungan yang padat dengan *node* 300 menghasilkan RREQ F yang lebih rendah daripada *routing protocol* AODV asli. AODV modifikasi pada lingkungan tersebut dapat mengungguli AODV asli. Sedangkan pada lingkungan yang sedang dengan *node* 150 menghasilkan RREQ F yang lebih besar daripada *routing protocol* AODV asli. AODV modifikasi mengalami penurunan RREQ F pada *node* 60 dan pada *node* 300 dengan nilai sebesar 22,83%. Sedangkan pada *node* 150 AODV modifikasi mengalami penurunan RREQ F sebesar 8,46%. Dapat dilihat pula bahwa AODV yang telah dimodifikasi pada *node* 60 dan pada *node* 300 dapat menghasilkan RREQ F yang lebih bagus atau dalam hal ini lebih rendah daripada RREQ F asli. Namun tidak berhasil menghasilkan RREQ F yang lebih bagus pada *node* 150.

5.3 Evaluasi

Secara keseluruhan, AODV modifikasi pembatasan *threshold* dengan nilai average tidak dapat mengungguli AODV asli dalam hal *Packet Delivery Ratio* dan *End-to-end delay*. Karena AODV modifikasi menghasilkan nilai yang lebih rendah daripada AODV asli dalam *Packet Delivery Ratio* dan *End-to-end delay*.

Namun untuk *Routing Overhead* dan *Forwarded Route Request* AODV modifikasi lebih unggul dalam hal tersebut, karena dapat menghasilkan nilai yang lebih rendah daripada AODV asli.

Penurunan nilai *Forwarded Route Request* juga mengindikasikan bahwa berhasilnya AODV modifikasi dalam membatasi *re-broadcast* yang menyebabkan berhasilnya AODV modifikasi untuk mencegah *Broadcast Storm Problem*.

5.3.1 Uji Coba Skenario *Real* dengan Perubahan Kecepatan Maksimum

Pengujian pada skenario *grid* digunakan untuk melihat perbandingan PDR, RO, RREQ F, dan E2E antara *routing protocol* AODV asli dan AODV yang telah dimodifikasi dalam pemilihan *node* yang dapat menerima paket *route request*.

Pengambilan data uji PDR, RO, RREQ F, dan E2E pada skenario *grid* dilakukan sebanyak 10 kali dengan skenario mobilitas *random* pada peta *grid* dengan luas area 1300 m x 1300 m dan *node* sebanyak 60 dilakukan pada kecepatan 10 m/s. Dan dengan *node* 300 dilakukan pada kecepatan 5 m/s.

Tabel 5.11 Hasil Perhitungan PDR pada Skenario Real

Jumlah *Node*	AODV Asli	AODV Modifikasi	Perbedaan
60	0,75961	0,63815	-0,12146
300	0,81164	0,73673	-0,07491

Tabel 5.12 Hasil Perhitungan E2E pada Skenario Real

Jumlah *Node*	AODV Asli	AODV Modifikasi	Perbedaan
60	80,11612	185,46415	-105,34803
300	139,64175	154,56383	-14,92208

Tabel 5.13 Hasil Perhitungan RO pada Skenario Real

Jumlah *Node*	AODV Asli	AODV Modifikasi	Perbedaan
60	56389	52655	+3734
300	32658	29695	+2963

Tabel 5.14 Hasil Perhitungan RREQ F pada Skenario Real

Jumlah *Node*	**AODV Asli**	**AODV Modifikasi**	**Perbedaan**
60	2255,375	1041,25	+1214,125
300	1098,5	920,9	+177,6

BAB VI
KESIMPULAN DAN SARAN

Pada Bab ini akan diberikan kesimpulan yang diperoleh dari Tugas Akhir yang telah dikerjakan dan saran tentang pengembangan dari Tugas Akhir ini yang dapat dilakukan di masa yang akan datang.

5.4 Kesimpulan

Kesimpulan yang diperoleh pada uji coba dan evaluasi Tugas Akhir ini adalah sebagai berikut:

1. Pada VANETs, AODV modifikasi sudah berhasil membatasi jumlah *forwarding node* karena dapat mengurangi jumlah *Forwarded Route Request* sehingga dapat mencegah *Broadcast Storm Problem*, yang dimana *Broadcast Storm Problem* terjadi karena tiap-tiap *node* menerima paket yang sama beberapa kali.
2. Dampak pengimplementasian *adaptive forwarding node* terhadap performa protokol AODV adalah penurunan *Packet Delivery Ratio* (PDR) sebesar 9,05% dan penurunan *Routing Overhead* (RO) sebesar 14,56%.

5.5 Saran

Saran yang dapat diberikan dari hasil uji coba dan evaluasi adalah sebagai berikut:

1. Perlunya penambahan uji coba yang dilakukan untuk mendapatkan hasil yang lebih akurat.
2. Menambahkan aspek lain untuk melakukan pembatasan *forwarding node* agar didapatkan peningkatan performa AODV yang lebih signifikan

(Halaman ini sengaja dikosongkan)

DAFTAR PUSTAKA

[1] R. G. Engoulou, M. Bellaiche, S. Pierre dan A. Quintero, "VANET Security Surveys," *Computer Communication*, vol. 44, p. 2, 2014.

[2] J. Harri, F. Filali dan C. Bonnet, "Mobility Models for Vehicular Ad Hoc Network: A Survey and Taxonomy," IEEE, Florida, 2009.

[3] R. Brendha dan V. S. J. Prakash, "A Survey on Routing Protocols for Vehicular Ad hoc Networks," IEEE, Coimbatore, 2017.

[4] R. F. Sari dan A. Syarif, "Analisis Kinerja Protokol Routing Ad Hoc On-Demand Distance Vector (AODV) pada Jaringan Ad Hoc," p. 22, October 2010.

[5] P. Meeneghan dan D. Delaney, "An Introduction to NS Nam and OTcl scripting," April 2004.

[6] "OpenStreetMap," [Online]. Available: https://www.openstreetmap.org/. [Diakses 10 Desember 2018].

[7] "JOSM," [Online]. Available: https://josm.openstreetmap.de/. [Diakses 10 Desember 2018].

[8] D. Krajzewics, J. Erdmann, M. Behrisch dan L. Bieker, "Recent Development and Application of SUMO," *International Journal On Advances in Systems and Measurements*, p. 128, December 2012.

[9] "AWK," [Online]. Available: http://tldp.org/LDP/abs/html/awk.html. [Diakses 10 12 2018].

[10] "VANET - Vehicle Ad hoc Network," [Online]. Available: http://comp.ist.utl.pt/~rmr/WSN/CaseStudies2007-no/WSN_Transportation/. [Diakses 10 Desember 2018].

[11] A. Rhim dan Z. Dziong, “Routing Based on Link Expiration Time for MANET Performance Improvement,” IEEE, Kuala Lumpur, 2009.

[12] P. Jacquet, P. Muhlethaler, T. Clausen, A. Laouiti, A. Qayyum dan L. Viennot, “Optimized Link State Routing Protocol for Ad hoc Networks,” IEEE, Lahore, 2001.

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

[14] M. Iqbal, M. Shafiq, H. Attaullah, J.-G. Choi, K. Akram dan X. Wang, “Design and Analysis of a Novel Hybrid Wireless Mesh Network Routing Protocol,” p. 22, January 2014.

[15] A. Virgono, L. V. Yovita dan A. V. Hutauruk, “IMULASI DAN ANALISIS PERBANDINGAN PERFORMANSI ROUTING PROTOCOL AODV & DSR PADA VEHICULAR AD HOC NETWORK (VANET),” *e-Proceeding of Engineering*, p. 795, 1 April 2016.

LAMPIRAN

A.1 Kode Fungsi CountThreshold()

```
void CountThreshold::handle(Event *)
{
  double now =
Scheduler::instance().clock(); // get the
time
  double interval = 5.0;

  int run = now / interval;
  if (masuk[run]==0) masuk[run]=0;

  if(now > 0.000000 && masuk[run]==0)
  {
    masuk[run]=100;
    for(int i=0;i<nodes_count;i++)
    {
     for(int j=(i+1);j<nodes_count;j++)
      {

if(count_neighbour[i]>count_neighbour[j])
        {
         int tmp;
         tmp=count_neighbour[i];

count_neighbour[i]=count_neighbour[j];
         count_neighbour[j]=tmp;
        }
      }
    }
    for(int i=0;i<nodes_count;i++)
```

```
    {
     count_mode[i]=0;
      for(int j=0;j<nodes_count;j++)
      {

if(count_neighbour[i]==count_neighbour[j])
         {
          count_mode[i]++;
         }
      }
    }
    for(int i=0;i<nodes_count;i++)
    {
      if(count_mode[i]>th )
      {
        th=count_mode[i];
      }
    }


    old_th = th;
  }


  Scheduler::instance().schedule(this,
&intr, interval);
}
```

A.2 Kode Fungsi nb_insert()

```
void AODV::nb_insert(nsaddr_t id)
{
  count_neighbour[index]+=1;
  AODV_Neighbor *nb = new AODV_Neighbor(id);

  assert(nb);
  nb->nb_expire = CURRENT_TIME +
                  (1.5 * ALLOWED_HELLO_LOSS *
HELLO_INTERVAL);
  LIST_INSERT_HEAD(&nbhead, nb, nb_link);
  seqno += 2; // set of neighbors changed
  assert((seqno % 2) == 0);
}
```

A.3 Kode Fungsi nb_remove()

```
void AODV::nb_delete(nsaddr_t id)
{
  count_neighbour[index]-=1;

  AODV_Neighbor *nb = nbhead.lh_first;

  log_link_del(id);
  seqno += 2; // Set of neighbors changed
  assert((seqno % 2) == 0);

  for (; nb; nb = nb->nb_link.le_next)
  {
    if (nb->nb_addr == id)
    {
      LIST_REMOVE(nb, nb_link);
      delete nb;
      break;
    }
  }

  handle_link_failure(id);
}
```

```
  if (id_lookup(rq->rq_src, rq-
>rq_bcast_id)){
      #ifdef DEBUG
      fprintf(stderr, "%s: discarding
request\n", __FUNCTION__);
      #endif // DEBUG

      Packet::free(p);
      return;
  }

  if (count < 45 && rq->rq_dst != index){
     Packet::free(p);
     return;
 }

 id_insert(rq->rq_src, rq->rq_bcast_id);

 aodv_rt_entry *rt0; // rt0 is the reverse
route

 rt0 = rtable.rt_lookup(rq->rq_src);
 if (rt0 == 0){
      rt0 = rtable.rt_add(rq->rq_src);
 }
 rt0->rt_expire = max(rt0->rt_expire,
(CURRENT_TIME +
 REV_ROUTE_LIFE));

 if ((rq->rq_src_seqno > rt0->rt_seqno) ||
 ((rq->rq_src_seqno == rt0->rt_seqno) &&
 (rq->rq_hop_count < rt0->rt_hops))){
     rt_update(rt0, rq->rq_src_seqno, rq-
>rq_hop_count, ih->saddr(),
               max(rt0->rt_expire,
(CURRENT_TIME + REV_ROUTE_LIFE)));
```

```
    if (rt0->rt_req_timeout > 0.0)
       {
         rt0->rt_req_cnt = 0;
         rt0->rt_req_timeout = 0.0;
         rt0->rt_req_last_ttl = rq-
>rq_hop_count;
         rt0->rt_expire = CURRENT_TIME +
ACTIVE_ROUTE_TIMEOUT;
       }

       assert(rt0->rt_flags == RTF_UP);
       Packet *buffered_pkt;
       while ((buffered_pkt =
rqueue.deque(rt0->rt_dst)))
       {
         if (rt0 && (rt0->rt_flags == RTF_UP))
         {
           assert(rt0->rt_hops != INFINITY2);
           forward(rt0, buffered_pkt,
NO_DELAY);
         }
       }
     }
     rt = rtable.rt_lookup(rq->rq_dst);
     if (rq->rq_dst == index)
     {

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

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

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

      Packet::free(p);
    }
    else if (rt && (rt->rt_hops != INFINITY2)
&&
             (rt->rt_seqno >= rq-
>rq_dst_seqno))
    {

     assert(rq->rq_dst == rt->rt_dst);
      sendReply(rq->rq_src,
                rt->rt_hops + 1,
                rq->rq_dst,
                rt->rt_seqno,
                (u_int32_t)(rt->rt_expire -
CURRENT_TIME),
                rq->rq_timestamp);
      rt->pc_insert(rt0->rt_nexthop); //
nexthop to RREQ source
      rt0->pc_insert(rt->rt_nexthop); //
nexthop to RREQ destination

  #ifdef RREQ_GRAT_RREP
```

```
   sendReply(rq->rq_dst,
                 rq->rq_hop_count,
                 rq->rq_src,
                 rq->rq_src_seqno,
                 (u_int32_t)(rt->rt_expire -
CURRENT_TIME),
                 //             rt->rt_expire
- CURRENT_TIME,
                 rq->rq_timestamp);
  #endif
     Packet::free(p);
    else
    {
      ih->saddr() = index;
      ih->daddr() = IP_BROADCAST;
      rq->rq_hop_count += 1;
        if (rt)
        rq->rq_dst_seqno = max(rt->rt_seqno,
rq->rq_dst_seqno);
      forward((aodv_rt_entry *)0, p, DELAY);
    }

}
```

A.4 Kode Skenario NS-2

```
set val(chan)   Channel/WirelessChannel;
set val(prop)   Propagation/TwoRayGround;
set val(netif)  Phy/WirelessPhy;
set val(mac)    Mac/802_11;
set val(ifq)    Queue/DropTail/PriQueue;
set val(ll)     LL;
set val(ant)    Antenna/OmniAntenna;
set opt(x)      1300;
set opt(y)      1300;
set val(ifqlen) 1000;
set val(nn)     60;
set val(seed)   1.0;
set val(adhocRouting)   AODV;
set val(stop)   200;
set val(cp)      "cbr1.txt";
set val(sc)      "scen1modif.txt";

set ns_          [new Simulator]

# setup topography object

set topo   [new Topography]

# create trace object for ns and nam

set tracefd     [open scenario1.tr w]
set namtrace    [open scenario1.nam w]

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

```
# Create God
set god_ [create-god $val(nn)]

#global node setting
$ns_ node-config -adhocRouting
$val(adhocRouting) \
                  -llType $val(ll) \
                  -macType $val(mac) \
                  -ifqType $val(ifq) \
                  -ifqLen $val(ifqlen) \
                  -antType $val(ant) \
                  -propType $val(prop) \
                  -phyType $val(netif) \
                  -channelType $val(chan)
\
                  -topoInstance $topo \
                  -agentTrace ON \
                  -routerTrace ON \
                  -macTrace ON \
                  -movementTrace ON \

# 802.11p default parameters
Phy/WirelessPhy set   RXThresh_ 5.57189e-
11 ; #400m
Phy/WirelessPhy set   CSThresh_ 5.57189e-
11 ; #400m

#  Create the specified number of nodes
[$val(nn)] and "attach" them
#  to the channel.
for {set i 0} {$i < $val(nn)} {incr i} {
           set node_($i) [$ns_ node]
           $node_($i) random-motion 0 ;#
disable random motion
}
```

```
# Define node movement model
puts "Loading connection pattern..."
source $val(cp)

# Define traffic model
puts "Loading scenario file..."
source $val(sc)

# Define node initial position in nam

for {set i 0} {$i < $val(nn)} {incr i} {

    # 20 defines the node size in nam,
must adjust it according to your scenario
    # The function must be called after
mobility model is defined

    $ns_ initial_node_pos $node_($i) 20
}

# Tell nodes when the simulation ends
for {set i 0} {$i < $val(nn) } {incr i} {
    $ns_ at $val(stop).0 "$node_($i)
reset";
}

#$ns_ at  $val(stop)  "stop"
$ns_ at  $val(stop).0002 "puts \"NS
EXITING...\" ; $ns_ halt"

puts $tracefd "M 0.0 nn $val(nn) x
$opt(x) y $opt(y) rp $val(adhocRouting)"
puts $tracefd "M 0.0 sc $val(sc) cp
$val(cp) seed $val(seed)"
puts $tracefd "M 0.0 prop $val(prop) ant
$val(ant)"

puts "Starting Simulation..."
$ns_ run
```

A.5 Kode Konfigurasi *Traffic*

```
set udp_(0) [new Agent/UDP]
$ns_ attach-agent $node_(58) $udp_(0)
set null_(0) [new Agent/Null]
$ns_ attach-agent $node_(59) $null_(0)
set cbr_(0) [new Application/Traffic/CBR]
$cbr_(0) set packetSize_ 512
$cbr_(0) set interval_ 1
$cbr_(0) set random_ 1
$cbr_(0) set maxpkts_ 10000
$cbr_(0) attach-agent $udp_(0)
$ns_ connect $udp_(0) $null_(0)
$ns_ at 2.5568388786897245 "$cbr_(0) start"
```

A.6 Kode Skrip AWK *Packet Delivery Ratio*

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

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

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

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

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

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

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

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

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

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

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

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

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

A.8 Kode Skrip AWK *Routing Overhead*

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

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

A.9 Kode Skrip AWK *Forwarded Route Request*

```
BEGIN {
      rt_forward = 0;
}
{
            if (($1 == "s") && ($4 ==
"RTR") && ($7 == "AODV") && ($25 ==
"(REQUEST)") && ($3 != "_58_")){
                  rt_forward++;
            }
      }
}
END {
      printf "Forwarded Route Request\t=
%d \n", rt_forward;
}
```

(Halaman ini sengaja dikosongkan)

BIODATA PENULIS

Dwiyan Satria Utama, lahir pada tanggal 5 juni 1996 di Surakarta. Penulis adalah anak kedua dari dua bersaudara. Penulis menempuh pendidikan sekolah dasar di SD Muhammadiyah 2 Surakarta lalu melanjutkan pendidikan sekolah menengah pertama di SMPN 1 Surakarta dan penulis menempuh pendidikan menengah atas di SMA Negeri 3 Surakarta. Selanjutnya penulis melanjutkan pendidikan sarjana di Departemen Informatika, Fakultas Teknologi Informasi dan Komunikasi, Institut Teknologi Sepuluh Nopember Surabaya.

Dalam menyelesaikan pendidikan S1, penulis mengambil bidang minat Arsitektur dan Jaringan Komputer (AJK). Sebagai mahasiswa, penulis berperan aktif dalam beberapa organisasi kampus seperti staf Dalam Negeri Himpunan Mahasiswa Teknik-Computer (HMTC) ITS. Selain itu, penulis juga menjadi koordinator NLC region Surakarta dan sekitarnya pada SCHEMATICS 2015 dan staf NST pada acara SCHEMATICS 2016. Penulis pernah melakukan kerja praktik di Politeknik Kesehatan Kemenkes Surakarta periode Juli – Agustus 2017 dan membuat aplikasi berbasis web untuk database video dari Library Integrated Onlince Service (LIOS). Penulis dapat dihubungi melalui nomor *handphone*: 081329501220 atau *email*: dwiyansatria@yahoo.co.id