Institut
Teknologi
Sepuluh Nopember

TUGAS AKHIR – IF184802

IMPLEMENTASI SELEKSI RUTE BERDASARKAN KONDISI ENERGI JALUR DALAM AD-HOC ON DEMAND DISTANCE VECTOR (AODV) DALAM MANET

ACHMAD HANIF PRADIPTA
NRP 05111640000154

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

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

Departemen Teknik Informatika
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2020

TUGAS AKHIR – IF184802

IMPLEMENTASI SELEKSI RUTE BERDASARKAN KONDISI ENERGI JALUR DALAM *AD-HOC ON DEMAND DISTANCE VECTOR (AODV)* DALAM MANET

ACHMAD HANIF PRADIPTA
NRP 05111640000154

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

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

Departemen Teknik Informatika
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2020

(Halaman ini sengaja dikosongkan)

UNDERGRADUATE THESES – IF184802

IMPLEMENTATION OF ROUTE SELECTION BASED ON THE CONDITION OF ENERGY ROUTES IN AD-HOC ON DEMAND DISTANCE VECTOR (AODV) ON MANET

Achmad Hanif Pradipta
NRP 05111640000154

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

Second Advisor
Ir. F.X. Arunanto, M. Sc.

Department of Informatics Engineering
Faculty of Intelligent Electrical and Informatics Technology
Sepuluh Nopember Institute of Technology
Surabaya 2020

(Halaman ini sengaja dikosongkan)

LEMBAR PENGESAHAN

IMPLEMENTASI SELEKSI RUTE BERDASARKAN KONDISI ENERGI JALUR DALAM AD-HOC ON DEMAND DISTANCE VECTOR (AODV) PADA MANET

TUGAS AKHIR

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

Oleh:
ACHMAD HANIF PRADIPTA
NRP: 05111640000154

Disetujui oleh Pembimbing Tugas Akhir:

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

2. Ir. F.X. Arunanto, M. Sc.
(NIP. 195701011983031004) (Pembimbing 2)

SURABAYA
JUNI, 2020

(Halaman ini sengaja dikosongkan)

IMPLEMENTASI SELEKSI RUTE BERDASARKAN KONDISI ENERGI JALUR DALAM AD-HOC ON DEMAND DISTANCE VECTOR (AODV) PADA MANET

Nama Mahasiswa	**: Achmad Hanif Pradipta**
NRP	**: 05111640000154**
Departemen	**: Teknik Informatika FTEIC-ITS**
Dosen Pembimbing 1	**: Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.**
Dosen Pembimbing 2	**: Ir. F.X. Arunanto, M. Sc.**

Abstrak

Mobile Ad-hoc Network atau MANET merupakan sebuah jaringan yang tidak menggunakan infrastruktur permanen. Hal ini disebabkan karena tingkat mobilitas *node* yang tinggi menyebabkan MANET tidak memerlukan infrastruktur yang tetap seperti layaknya topologi jaringan yang lain.Secara singkat MANET merupakan sistem jaringan dimana setiap *node* tidak hanya berperan sebagai *host* atau perangkat pengguna saja, akan tetapi setiap *node* berperan sebagai *router* yang dapat meneruskan paket kepada *node* lain yang menjadi tujuan pengiriman paket tersebut.Dalam MANET terdapat banyak *routing protocol* yang dapat diterapkan. Salah satunya adalah AODV, yaitu *Ad hoc On demand Distance Vector*.

AODV merupakan *routing protocol* yang termasuk ke dalam *reactive routing protocol*, yang hanya meminta rute ketika dibutuhkan. AODV memiliki dua fase, yaitu *route discovery* dan *route maintenance. Route discovery* digunakan untuk meminta dan meneruskan informasi rute, sedangkan *route maintenance* digunakan untuk mengetahui informasi adanya kesalahan pada rute.

Modifikasi yang dilakukan dengan melakukan penghitungan energi rata-rata dan jalur energi dimana dalam algoritma ini area node dibagi menjadi 3 bagian yaitu *zona dalam,zona tengah* dan

zona luar yang bertujuan untuk mengurangi pengiriman paket secara terus-menerus dan mengurangi jumlah hope di route discovery,kemudian dilakukan seleksi lagi berdasarkan threshold kecepatan,arah dan congestion level dengan *Total Weight of the Route* (TWR). Kemudian akan dilakukan seleksi forwarding node berdasarkan kekuatan sinyal dan pemilihan rute RREP berdasarkan rata-rata energi pada RREQ.

Dari hasil simulasi, AODV yang dimodifikasi pada skenario *grid* berhasil meningkatkan nilai rata-rata *Packet Delivery Ratio* (PDR) hingga 28,47% dan menurunkan *end-to-end delay* hingga 67,75%. Dan pada skenario *real* berhasil meningkatkan nilai rata-rata *Packet Delivery Ratio* (PDR) hingga 35,24% dan menurunkan *end-to-end delay* hingga 60,21%.

Kata kunci: AODV, MANET, TWR

IMPLEMENTATION OF ROUTE SELECTION BASED ON THE CONDITION OF ENERGY ROUTES IN AD-HOC ON DEMAND DISTANCE VECTOR (AODV) ON MANET

Student's Name	**: Achmad Hanif Pradipta**
Student's ID	**: 05111640000154**
Department	**: Informatics Engineering – FTEIC ITS**
First Advisor	**: Dr.Eng. Radityo Anggoro, S. Kom., M.Sc.**
Second Advisor	**: Ir. F.X. Arunanto, M. Sc.**

Abstract

Mobile Ad-hoc Network or MANET is a network that does not use permanent infrastructure. This is because the high level of node mobility causes MANET not to need a fixed infrastructure like other network topologies. Briefly MANET is a network system where each node does not only act as a host or user device, but each node acts as a router can forward packets to other nodes to which the packet is sent. In MANET there are many routing protocols that can be applied. One of them is AODV, which is Ad hoc On demand Distance Vector. AODV is a routing protocol which is included in the reactive routing protocol which only requests routes when required. AODV has two phases route which are discovery and route maintenance. Route discovery is used to request and forward route information, while route maintenance is used to find out information on the route's error.

AODV is a routing protocol that is included in the reactive routing protocol, which only requests routes when needed. AODV has two phases, namely route discovery and route maintenance. Route discovery is used to request and forward route information, while route maintenance is used to find information about errors on the route.

Modifications are made by calculating the average energy and energy path where in this algorithm the node area is divided into 3 parts, namely the inner zone, the middle zone and the outer zone which aims to reduce continuous packet delivery and reduce the amount of hope in route discovery, then do the selection again based on the speed threshold, direction and congestion level with Total Weight of the Route (TWR). Then the forwarding node selection will be based on signal strength and RREP route selection based on the average energy in the RREQ.

Based on the result of simulation, the modified AODV in the grid scenario could increase the average value of the Packet Delivery Ratio (PDR) to 28,47% and reduce end-to-end delay to 67,75, while in the real scenario the average value of Packet Delivery Ratio (PDR) could increase to 35.24% and reduce end-to-end delay value to 60.21%.

Keyword: MANET, AODV, TWR

KATA PENGANTAR

Puji syukur penulis panjatkan ke hadirat Allah SWT, atas rahmat, barokah, dan ridho-Nya sehingga penulis dapat menyelesaikan Laporan Tugas Akhir yang berjudul

"IMPLEMENTASI SELEKSI RUTE BERDASARKAN KONDISI ENERGI JALUR DALAM AD-HOC ON DEMAND DISTANCE VECTOR (AODV) PADA MANET".

Laporan Tugas Akhir ini ditulis dalam rangka memenuhi persyaratan memperoleh Gelar Sarjana Komputer pada Bidang Studi Arsitektur dan Jaringan Komputer Program Studi S-1 Departemen Teknik Informatika Fakultas Teknologi Elektro dan Informatika Cerdas Institut Teknologi Sepuluh Nopember.

Penulis menyadari bahwa laporan Tugas Akhir ini dapat diselesaikan berkat support dan bantuan dari berbagai pihak. Oleh karena itu, penulis berterimakasih kepada semua pihak yang secara langsung dan tidak langsung memberikan kontribusi dalam penyelesaian laporan ini.

1. Keluarga penulis atas segala dukungan berupa motivasi, doa, moral, dan material sehingga penulis tetap semangat untuk menyelesaikan Tugas Akhir ini.
2. Bapak Dr. Eng. Radityo Anggoro, S.Kom., M.Sc., dan Bapak Ir. F.X. Arunanto, M.Sc. selaku dosen pembimbing penulis atas nasihat, arahan, bimbingan dan bantuannya sehingga penulis dapat menyeleksaikan Tugas Akhir ini.
3. Teman-teman dekat penulis di kontrakan (Fahrizal,Ilham,Farhan) yang selalu memberikan semangat, hiburan, dan menjadi tempat bertukar pikiran, pendapat serta menemani penulis sehari-hari di selama penulis berkuliah di Departemen Informatika ITS.
4. Teman satu topik penulis (Irham,Fahrizal).

5. Teman-teman Discord Darari Garis Keras.
6. Teman teman HMTC Kreasi dan HMTC Garang yang sudah memberikan kesibukan lebih untuk penulis semasa kuliah di Informatika ITS.
7. Pihak lain yang tidak dapat disebutkan satu per satu.

Seperti kata pepatah *Tiada gading yang tak retak*, dalam penyusunan laporan ini penulis menyadari tentunya masih banyak terdapat kekurangan, kesalahan dan kekhilafan karena keterbatasan kemampuan penulis, untuk itu sebelumnya penulis mohon maaf yang sebesar-besarnya. Penulis juga mengharapkan kritik dan saran dari semua pihak demi perbaikan yang bersifat membangun atas laporan ini.

Akhirnya dengan segala kerendahan hati penulis mengucapkan terima kasih dan semoga laporan ini dapat bermanfaat bagi penulis maupun kita bersama.

Surabaya, Juni 2020

ACHMAD HANIF PRADIPTA

DAFTAR ISI

(Halaman ini sengaja dikosongkan)

DAFTAR GAMBAR

DAFTAR TABEL

(Halaman ini sengaja dikosongkan)

BAB I
PENDAHULUAN

1.1 Latar Belakang

Di dalam beberapa tahun ini perkembangan mobile computing sedang berkembang dengan cepat,dikarenakan murahnya perangkat dan ketersediaan jaringan dan perangkat wireless.Hal ini membuka kesempatan untuk mengembangkan penelitian mengenai *Mobile Adhoc Network* (MANET).*Mobile Adhoc Network* (MANET) merupakan teknologi jaringan yang tersusun dari kumpulan ad-hoc node yang berbeda-beda atau sejenis dan setiap node pada MANET terhubung dengan node lain menggunakan koneksi wireless,akan tetapi jika salah satu node diluar jangkauan transmisi maka membutuhkan node lain untuk meneruskan pesan.

Permasalahan yang ada pada MANET adalah topologi jaringan yang sering berubah atau bersifat dinamis,Oleh karena itu protokol routing sangat diperlukan untuk menentukan rute atau jalur pada jaringan MANET.Dalam Mobile Adhoc Network (MANET) selama proses routing,konsumsi energi tetap menjadi hambatan karena node seluler memiliki baterai yang terbatas.

Tugas Akhir ini mengusulkan suatu metode pemilihan rute dengan energi dan Total Weight of the route (TWR) menjadi salah satu parameter utama untuk meningkatkan masa pakai jaringan dan meminimalkan kerusakan pada saat pengiriman. Hasil akhir yang diharapkan adalah mengetahui perbandingan kinerja antara AODV dan AODV yang telah dimodifikasi yang diukur berdasarkan performansi *Packet Delivery Ratio* (PDR), *Routing Overhead*, dan *End-to-end Delay*.

1.2 Rumusan Masalah

Tugas Akhir ini mengangkat beberapa rumusan masalah sebagai berikut:

1. Bagaimana melakukan seleksi rute berdasarkan kondisi energi jalur pada AODV?
2. Bagaimana dampak perubahan yang dilakukan terhadap performa AODV diukur berdasarkan Packet Delivery Ratio, End-to-end Delay, dan Routing Overhead?

1.3 Batasan Permasalahan

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

1. Jaringan yang digunakan adalah jaringan *Mobile Ad hoc Networks* (MANETs).
2. Simulasi pengujian jaringan menggunakan *Network Simulator 2* (NS-2) dengan bahasa pemrograman C++.
3. *Routing Protocol* yang diujicobakan yaitu AODV

1.4 Tujuan

Tujuan dari Tugas Akhir ini adalah sebagai berikut:

1. Meningkatkan kinerja AODV dengan cara melakukan seleksi rute dengan kondisi energi jalur di MANET.
2. Menganalisa perfoma AODV yang telah dimodifikasi berdasarkan matriks Packet Delivery Ratio, End-to-end Delay, dan Routing Overhead.

1.5 Manfaat

Dengan dibuatnya Tugas Akhir ini akan memberikan sebuah manfaat berupa analisa dalam mengetahui dampak AODV terhadap modifikasi 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*, *Mobile Ad hoc Network* (MANET), dan *reactive routing protocol* AODV.

1.6.3 Analisis dan Desain Sistem

Pada tahap ini dilakukan analisis dari hasil percobaan modifikasi AODV yang dibuat. Data yang dianalisis berasal dari

perhitungan Jumlah Energi rata-rata dan Pemilihan jalur protocol serta perhitungan *Packet Delivery Ratio*, *Routing Overhead*, dan *End-to-End Delay* paket dari *node* ke *node* lainnya.

1.6.4 Implementasi Sistem

Implementasi merupakan tahap untuk membangun metode yang telah diajukan pada proposal Tugas Akhir. Pada tahap ini dilakukan implementasi yang dibuat berupa modifikasi dari protocol AODV menggunakan NS-2 sebagai *simulator*, Bahasa C/C++ sebagai bahasa pemrograman.

1.6.5 Pengujian dan Evaluasi

Pengujian dilakukan dengan NS-2 *Network Simulator* dan akan menghasilkan *Packet Delivery Ratio,End-to-end Delay* dan *Routing Overhead*.Dari beberapa parameter tersebut akan dibandingkan dengan AODV tradiosional 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
   Pada bagian ini berisi penjelasan mengenai latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat,

metodologi, dan sistematika penulisan dari pembuatan Tugas Akhir.

2. Bab II. Tinjauan Pustaka
   Pada bagian ini berisi kajian teori atau penjelasan dari metode, algoritma, *library*, dan *tools* yang digunakan dalam penyusunan Tugas Akhir ini.
3. Bab III. Perancangan
   Pada bagian ini berisi pembahasan mengenai perancangan beberapa skenario antara lain modifikasi, pengujian, dan perhitungan metrik analisis.
4. Bab IV. Implementasi
   Pada bagian ini menjelaskan implementasi yang berbentuk kode program dari proses modifikasi, proses pengujian. Dan perhitungan metrik analisis.
5. Bab V. Uji Coba dan Evaluasi
   Pada bagian ini berisi hasil uji coba dan evaluasi dari implementasi yang telah dilakukan untuk menyelesaikan masalah yang dibahas pada Tugas Akhir.
6. Bab VI. Kesimpulan dan Saran
   Pada bagian ini berisi kesimpulan dari hasil uji coba yang dilakukan, masalah yang dialami pada proses pengerjaan Tugas Akhir, dan saran untuk pengembangan dari solusi yang ada.
7. Daftar Pustaka
   Pada bagian ini berisi daftar pustaka yang dijadikan literatur dalam pengerjaan Tugas Akhir.
8. Lampiran
   Pada bagian ini terdapat tabel-tabel yang berisi data hasil uji coba dan beberapa kode program.

(Halaman ini sengaja dikosongkan)

BAB II
TINJAUAN PUSTAKA

Bab ini berisi penjelasan mengenai teori-teori dasar yang berkaitan dengan pengimplementasian perangkat lunak dan penunjangnya. Penjelasan ini bertujuan untuk memberikan gambaran secara umum terhadap *routing protocol*, tools, serta definisi yang digunakan dalam pembuatan Tugas Akhir.

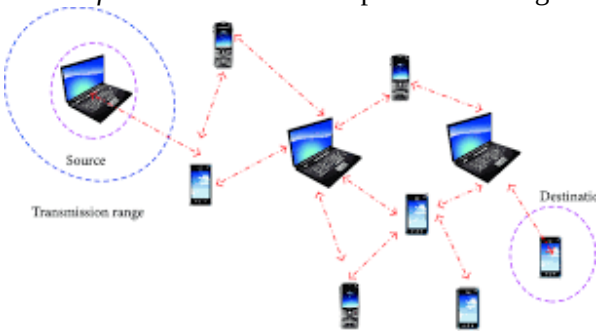
2.1 *Mobile Ad-hoc Networks* (MANETs)

Mobile Ad hoc Network(MANET) merupakan sebuah jaringan yang terbentuk dari beberapa *node* yang bergerak bebas dan dinamis. MANET memungkinkan terjadinya komunikasi jaringan tanpa bergantung pada ketersedian infrastruktur jaringan yang tetap [1]. Setiap *node* dalam jaringan MANET dapat bertindak sebagai host dan *route*r. Setiap *node* dapat saling melakukan komunikasi antara yang satu dengan yang lainnya tanpa adanya access point. Perangkat di jaringan MANET harus mampu mendeteksi keberadaan perangkat lain dan melakukan pengaturan yang diperlukan untuk melakukan komunikasi dan berbagi data. Pada MANET memungkinkan perangkat untuk mempertahankan koneksi ke jaringan serta dengan mudah menambahkan dan menghapus perangkat pada jaringan. Karena pergerakan *node* yang dinamis, topologi jaringan dapat berubah dengan cepat dan tak terduga dari waktu ke waktu. Jaringan MANET bersifat desentralisasi, di mana organisasi jaringan dan pengiriman pesan harus dijalankan oleh *node* sendiri. MANET memiliki beberapa karakteristik yaitu :

- Topologi yang dinamis : *Node* pada MANET dapat bergerak secara bebas dan berpindah-pindah kemana saja. Topologi jaringan yang bisanya multihop dapat berubah secara acak dan tidak terpola.
- Keterbatasan *bandwidth* : *Link* pada jaringan nirkabel memiliki kapasitas rendah daripada jaringan kabel. Selain itu, *throughput* komunikasi nirkabel seringkali lebih

rendah dari tingkat transmisi maksimum radio. Hal ini dapat menyebabkan terjadinya *congestion*(kemacetan).

- Keterbatasan energi : Karena bersifat *mobile*, semua *node* pada MANET dapat dipastikan sangat mengandalkan baterai sebagai sumber energi. Sehingga diperlukan desain sistem untuk optimasi energi.
- Keterbatasan Keamanan : Karena jaringan nirkabel pada umumnya sangat rentan terhadap ancaman keamanan. Beberapa ancaman seperti *eavesdroping*, *spoofing* dan *denial of service* harus lebih diperhatikan dengan cermat.



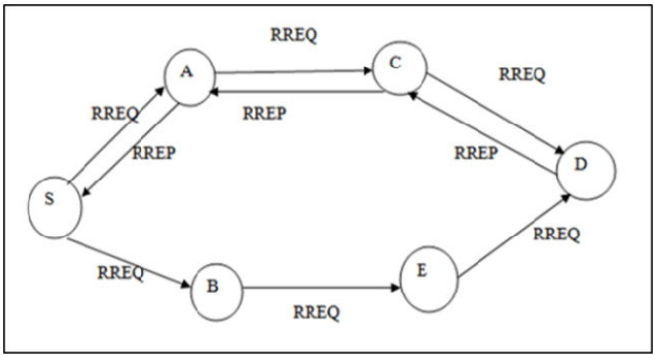
Gambar 2.1 Jaringan MANETs [2]

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

Ad-hoc On-demand Distance Vector (AODV) adalah salah satu *routing* protokol yang dirancang untuk jaringan *ad-hoc mobile* dan termasuk dalam klasifikasi *reactive routing protocol*. Hal itu berarti protocol tersebut hanya membuat sebuah rute antara node hanya saat dibutuhkan. AODV menggunakan table routing satu *entry* untuk setiap tujuan, tanpa menggunakan routing. AODV dikembangkan oleh C. E. Perkins, E.M. Belding-Royer dan S. Das yang didefinisikan dalam RFC 3561.

Ada dua tahapan dalam AODV 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 mempercayakan pada tabel *routing* untuk menyebarkan *Route Reply* kembali ke sumber dan mengarahkan paket menuju tujuan. Ciri utama dari AODV adalah menjaga *timer-based state* pada setiap *node* sesuai dengan penggunaan tabel *routing*.

AODV adalah 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* [3]. Ilustrasi pencarian rute oleh AODV dapat dilihat pada Kesalahan! Sumber referensi tidak d itemukan. dan struktur paket RREQ dapat dilihat pada Tabel 2.1.



Gambar 2.2 Ilustrasi pecarian rute routing protocol AODV [17]

Tabel 2.1 Struktur Paket RREQ

source_addr	*source_sequence_#*	*broadcast_id*
dest_addr	*dest_sequence_#*	*hop_cnt*

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

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

Sebagai contoh proses *route discovery* dalam AODV, ilustrasi pada Gambar 2.2 menggambarkan bagaimana *source node*, yaitu *node* dengan label "S" mencari rute untuk menuju *destination node* yaitu *node* dengan label "D". *Node* S 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* D. Kemudian, jika rute menuju *node* D sudah terbentuk di dalam *routing table* dan memiliki *routing flags* "*up*", maka *node* D akan mengirimkan paket RREP melalui rute tersebut menuju *node.*

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

Network Simulator-2(NS-2) adalah suatu tool yang berguna untuk melakukan simulasi jaringan dengan melibatkan Local Area Network(LAN), Wide Area Network(WAN), dan beberapa perkembangan terbaru telah menambakan jaringan nirkabel dan juga jaringan adhoc. NS-2 menggunakan dua bahasa pemrograman yaitu C++ dan Object-oriented *open*(OTCL). Pada NS-2, bahasa pemrogaman C++ dapat digunakan dalam hal pengaturan mekanisme internal(backend) dari objek simulasi yaitu pengaturan protokol yang digunakan saat melakukan simulasi, dan bahasa pemrograman OTCL mendefinisikan lingkungan simulasi eksternal(frontend) untuk perakitan dan konfigurasi objek. Proses simulasi pada NS-2 akan memberikan output berupa *file* NAM dan trace *file*.

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 automake
autoconf 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

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

2.3.2 *Trace File*

Trace file merupakan *file* hasil simulasi yang dilakukan oleh NS-2 dan berisikan informasi detail pengiriman paket data. *Trace file* digunakan untuk menganalisis performa *routing protocol* yang disimulasikan. Detail penjelasan *trace file* ditunjukkan pada Tabel 2.2

Tabel 2.2 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 *Total Weigth of Route (TWR)*

Total Weigth of Route (TWR) merupakan bobot sebuah route dinilai dari kecepatan,percepatan,dan juga jarak dari node ke node selanjutnya.

Formula TWR dirancang dengan mempertimbangkan beberapa faktor berikut :

- Kecepatan dan Percepatan

  Semakin besar perbedaan kecepatan dan akselerasi antar dua kendaraan, maka semakin besar pula kemungkinan dua kendaraan tersebut saling berjauhan. Asumsi tersebut berimplikasi pada TWR dalam bentuk pelebaran nilai TWR. Alasannya adalah jika kedua kendaraan saling berjauhan, maka semakin besar

kemungkinan terjadinya kerusakan koneksi antar kendaraan tersebut. Jika kedua kendaraan tersebut berada di dalam radius komunikasi dan bergerak dengan kecepatan dan akselerasi yang relatif sama antara satu dengan yang lainnya, maka bisa diasumsikan kedua kendaraan tersebut akan berada pada radius komunikasi dalam durasi yang lebih lama.

- Arah kendaraan

  Jika dua kendaraan yang bergerak dengan arah yang relatif sama, maka kedua kendaraan tersebut akan berada dalam radius komunikasi dalam durasi yang lebih lama.

- Kualitas hubungan antar kendaraan

  Kualitas hubungan yang dimaksud adalah apakah *node* yang mengirimkan paket dan *node* penerimanya masih berada dalam radius komunikasi. Dalam lingkungan MANET, terdapat banyak *neighbor node* dan halangan lainnya yang dapat mempengaruhi kualitas hubungan. Kualitas hubungan didefinisikan dalam indeks stabilitas (*stability index*).

Formula dari indeks stabilitas dapat dilihat pada persamaan dibawah:

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

dimana,

i_x, i_y : Koordinat dari kendaraan i
j_x, j_y : Koordinat dari kendaraan j
r : Jarak transmisi maksimum

Nilai indeks stabilitas yang paling baik adalah 1. Indeks stabilitas dengan nilai 1 dihasilkan ketika kendaraan i dan j memiliki

vektor pergerakan yang sama. Nilai indeks stabilitas paling buruk adalah 0. Indeks stabilitas dengan nilai 0 didapatkan ketika kendaraan i dan j memiliki jarak yang lebih jauh dari jarak transmisi maksimum. Indeks stabilitas dengan nilai 0 mengimplikasikan kendaraan i dan j berada di luar radius komunikasi dari masing – masing kendaraan. Kemudian nilai kualitas hubungan dihitung dari rumus pada Persamaan dibawah:

$$Q = \frac{1}{s_{ij}}$$

Berdasarkan faktor – faktor di atas, rumus TWR di ekspresikan dengan Persamaan:

$$TWR = f_s \times |S_n - S_d| + f_a \times |A_n - A_d| + f_d \times |\theta_n - \theta_d| + f_q \times Q$$

dimana,
S_n, A_n, θ_n : Kecepatan, akselerasi dan arah *next-hop node*
S_d, A_d, θ_d : Kecepatan, akselerasi dan arah *node* tujuan
f_s : Faktor pengali kecepatan
f_a : Faktor pengali akselerasi
f_d : Faktor pengali arah
f_q : Faktor pengali kualitas hubungan
Q : Kualitas hubungan antara *node* sumber dengan *next-hop node*.

Nilai TWR ditentukan oleh perbedaan kecepatan, akselerasi dan arah dari *next-hop node* dan *node* tujuan, serta kualitas hubungan antara *node* sumber dengan *next-hop node*. Formula TWR mengimplikasikan *next-hop node* yang baik adalah node yang memiliki nilai TWR yang rendah karena memiliki kecepatan, akselerasi dan arah yang relatif sama dengan *node* tujuan, serta memiliki kualitas hubungan yang relatif bagus dengan *node* sumber.

2.5 AWK

AWK adalah bahasa pemrograman yang digunakan untuk melakukan *text processing* dan ekstraksi data [4]. 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.

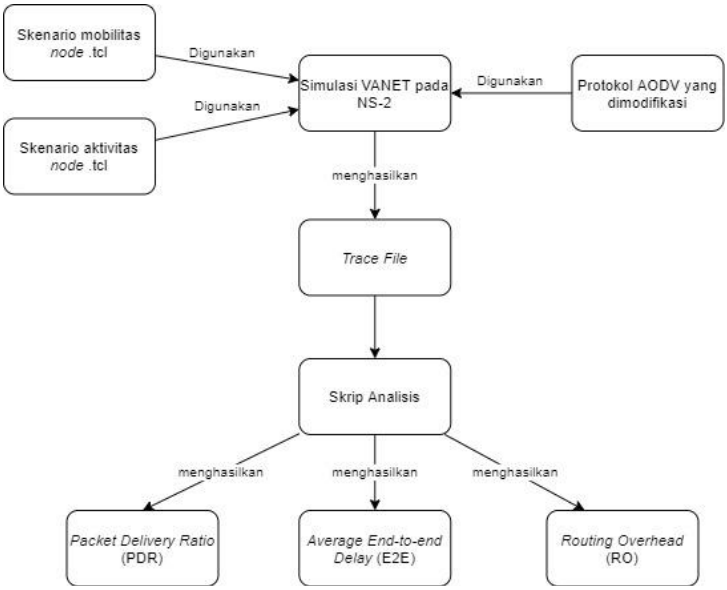
Pada Tugas Akhir ini, AWK digunakan untuk membuat script menghitung *Packet Delivery Ratio (*PDR*), End-to-end Delay*, dan *Routing Overhead* (RO) dari *trace file* NS-2.

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.



Gambar 3.1 Diagram Alur Rancangan Simulasi

Modifikasi yang dilakukan dengan melakukan penghitungan energi rata-rata dan jalur energi dimana dalam algoritma ini area node dibagi menjadi 3 bagian yaitu zona dalam,zona tengah dan zona luar yang bertujuan untuk mengurangi pengiriman paket secara terus-menerus dan mengurangi jumlah hope di route discovery dan hanya zona tengah saja yang dipakai untuk mencari energi rata-rata,kemudian dilakukan seleksi lagi berdasarkan threshold kecepatan,arah dan congestion level dengan *Total Weight of the Route* (TWR).Setelahnya akan dilakukan route request (RREQ) untuk mencari energi rata-rata,saat RREQ sampai di destination tidak akan langsung merespon melainkan akan menunggu RREQ lain selama waktu yang ditentukan,lalu destination akan mereply RREQ berdasarkan rata-rata energi tertinggi dan akan tersimpan di routing table,jikalau terdapat RREQ yang lebih tinggi maka RREQ tersebut akan digunakan dan akan melakukan route reply (RREP) untuk diteruskan ke source node.

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), *End-to-end Delay (*E2E), dan *Routing Overhead* (RO). Analisis tersebut dapat mengukur performa *routing protocol* AODV yang telah dimodifikasi dibandingkan dengan 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.

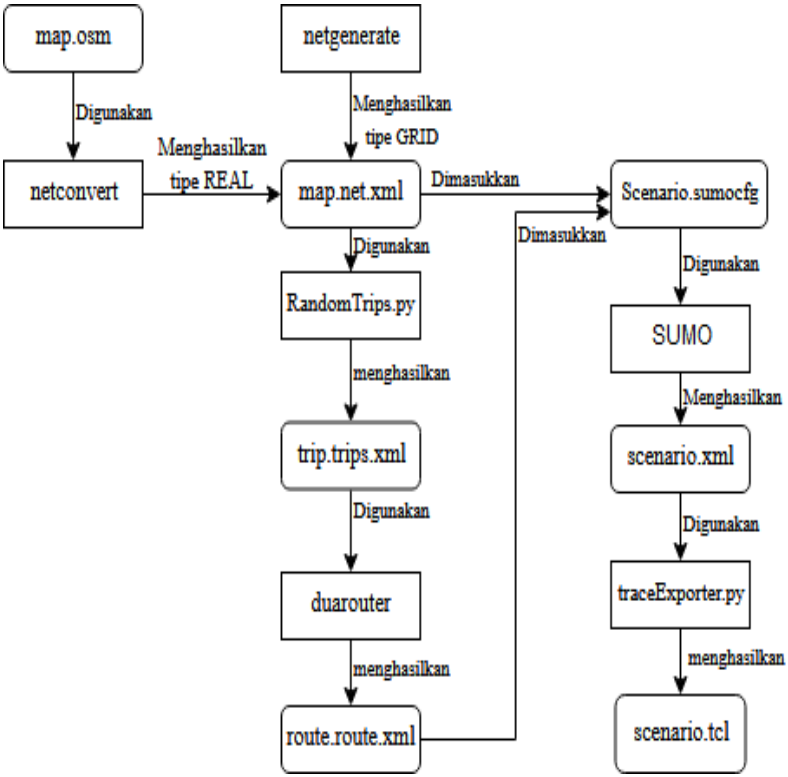
Tabel 3.1 Daftar Istilah

No.	Istilah	Penjelasan
1	AODV	Singkatan dari *Ad hoc On-demand Distance Vector*. Protokol yang digunakan pada Tugas Akhir ini.
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
8	*TWR*	*Total Weigth of Route (TWR)* merupakan bobot sebuah route dinilai dari kecepatan,percepatan,dan juga jarak dari node ke node selanjutnya.

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 Alur perancangan skenario *Grid* dan *Real*

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 berukuran 1100 m x 1100 m. Dengan 9 titik persimpangan, maka akan didapatkan jumlah 81 petak dan panjang tiap petak adalah 200 m.

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

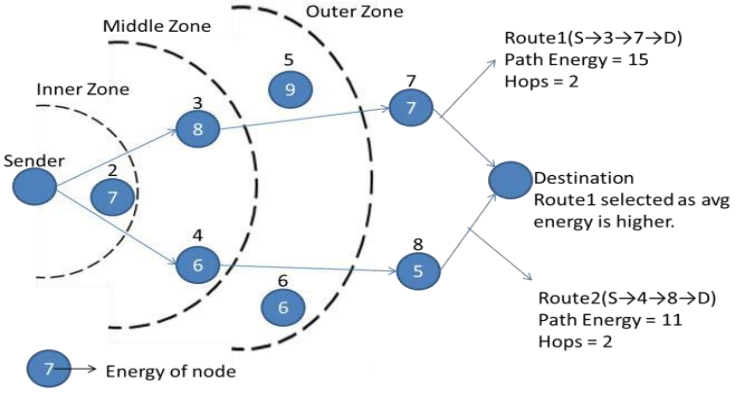
Setelah mendapatkan peta area yang akan dijadikan simulasi, peta tersebut dikonversi ke dalam bentuk *file* dengan ekstensi .net.xml menggunakan *tools* SUMO yaitu netconvert. Tahap berikutnya memiliki tahapan yang sama seperti merancang skenario *grid*, yaitu membuat pergerakan *node* menggunakan randomTrips 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

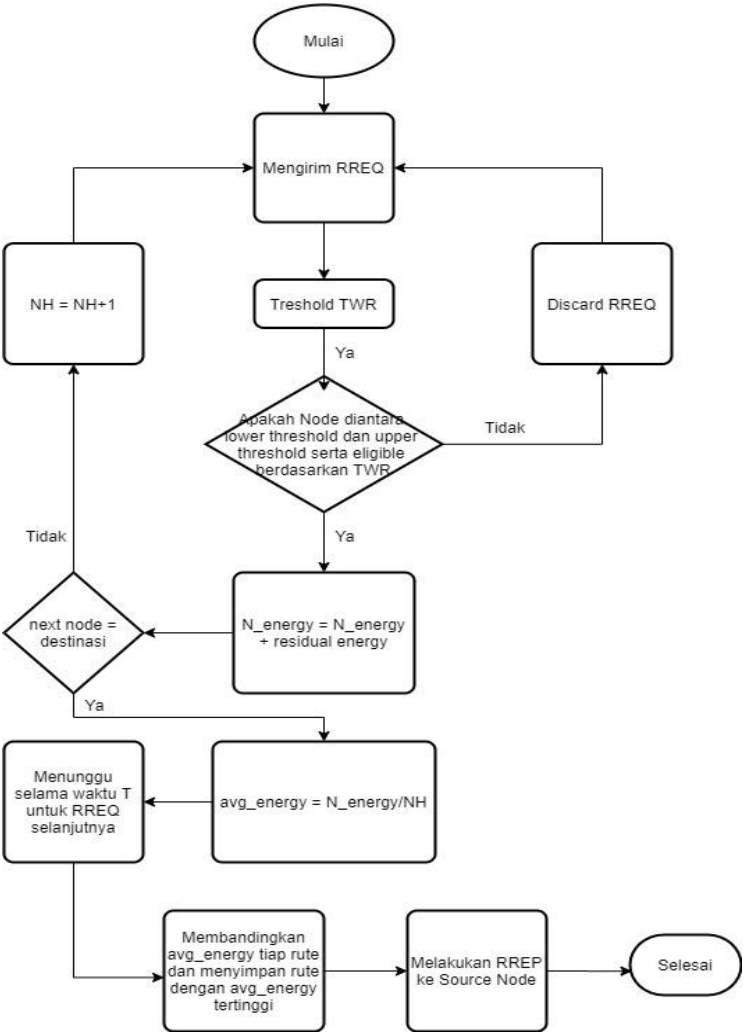
Protokol AODV yang diajukan pada Tugas Akhir ini merupakan modifikasi dari protokol AODV yang mengubah mekanisme route discovery pada protokol tersebut. Pada protokol AODV, route request (RREQ) disebarkan (broadcast) ke seluruh jaringan untuk mencari node destinasi. Lalu, setiap node akan mentransmisikan ulang (rebroadcast) paket RREQ sampai dengan node destinasi. Ketika, node tujuan menerima RREQ maka node akan mengirim kembali paket route reply (RREP) ke node sumber melalui rute terdekat atau memiliki jumlah hop minimum. Karena jumlah hop minimum digunakan sebagai pembanding untuk pemilihan rute, maka node dengan energi residu kecil dapat masuk kedalam jalur dan menyebabkan broken link dikarenakan habisnya energi pada node tersebut.Oleh karena itu, Modifikasi yang dilakukan dengan melakukan penghitungan energi rata-rata dan jalur energi dimana dalam algoritma ini area node dibagi menjadi 3 bagian yaitu zona dalam,zona tengah dan zona luar yang bertujuan untuk mengurangi pengiriman paket secara terus-menerus dan mengurangi jumlah hope di route discovery dan hanya zona tengah saja yang dipakai untuk mencari energi rata-rata, ,kemudian sebelum dilakukan Route Request (RREQ) oleh Sender (Source Node) akan dilakukan seleksi berdasarkan threshold kecepatan,arah dan congestion level dengan

Total Weight of the Route (TWR). Kemudian akan dilakukan seleksi forwarding node berdasarkan kekuatan sinyal dan list node eligible berdasarkan penjumlahan nilai TWR pada zona tengah (Middle Zone) serta pemilihan rute RREP berdasarkan rata-rata energi pada RREQ.,saat RREQ sampai di destination tidak akan langsung merespon melainkan akan menunggu RREQ lain selama waktu yang ditentukan,lalu destination akan mereply RREQ berdasarkan rata-rata energi tertinggi dan akan tersimpan di routing table,jikalau terdapat RREQ yang lebih tinggi maka RREQ tersebut akan digunakan dan akan melakukan route reply (RREP) untuk diteruskan ke source node. jika *destination node* menerima RREQ setelah mengirim RREP dan rata-rata energi pada RREQ yang datang lebih besar dari pada rata-rata RREQ yang tersimpan, maka destination node akan mengirim RREP kembali yang akan digunakan sebagai backup rute apabila terjadi kegagalan saat proses route discovery.

Gambar 3.3 dan Gambar 3.4 berikut ini merupakan *Ilustrasi* dan *flowchart* dari keseluruhan proses modifikasi AODV tersebut:



***Gambar 3.3** Ilustrasi Modifikasi [18]*



Gambar 3.4 *Flowchart Proses Modifikasi*

3.3.1 Perancangan Total Weight of the Route (TWR)

Dalam Tugas Akhir ini,simulasi dilakukan dengan *node* sumber dan *node* tujuan berada dalam posisi diam(*stationary node*)sehingga perlu dilakukan penyesuaian terhadap formula TWR.Perhitungan TWR terdiri atas empat faktor,yaitu kecepatan,akselerasi,arah dan kualitas hubungan.
Faktor kecepatan,akselerasi dan arah didapat dari selisih nilai yang dimiiliki oleh *next-hop node* terhadap *node* tujuan.*Node* tujuan yang berada dalam posisi diam menyebabkan faktor arah menjadi tidak relevan.Alaasnannya adalah *node* tujuan tidak memiliki arah.Agar tidak terjadi pengurangan bobo pada TWR,maka faktor aarah dubah menjadi jarak antara *next-hop node* dengan *node* tujuan.koordinat dari *node* tujuan dapat diketahui karena node tersebut adalah *stationary node.*Berikut adalah formula TWR:

$$TWR = f_s \times |S_n - S_d| + f_a \times |A_n - A_d| + f_d \times |\theta_n - \theta_d| + f_q \times Q \quad \ldots(3.1)$$

dimana,

S_n, A_n, θ_n : Kecepatan, akselerasi dan arah *next-hop node*
S_d, A_d, θ_d : Kecepatan, akselerasi dan arah *node* tujuan
f_s : Faktor pengali kecepatan
f_a : Faktor pengali akselerasi
f_d : Faktor pengali arah
f_q : Faktor pengali kualitas hubungan
Q : Kualitas hubungan antara *node* sumber dengan *next-hop node*.

Selain Formula TWR,formula indeks stabilitas juga perlu diubah karena jika hasil fungsi minimum antara jarak dan radius transmisi maksimum menghasilkan nilai dari radius trasmisi maksimum,akan terjadi pembagian dengan nol.

Berikut adalah perubahan formula indeks stabilitas:

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

Perhitungan future TWR membutuhkan prediksi nilai kecepatan dan posisi yang akan datang.Berikut adalah beberapa rumus fisika gerak lurus yang akan digunakan dalam menghitung nilai prediksi:

- Perhitungan akselerasi

...(3.3)

$$\alpha = \frac{\Delta v}{\Delta t}$$

dimana,
Δv: *Selisih kecepatan sekarang dan kecepatan sebelumnya*
Δt: *Selisih waktu*

- Perhitungan kecepatan yang akan datang

$$v' = v + \alpha \times t \quad \ldots(3.4)$$

dimana,
v : *Kecepatan sekarang*
α : *Akselerasi*
t : *Waktu (dalam detik)*

- Perhitungan posisi yang akan datang

$$x' = x + v_0 t + \tfrac{1}{2}\alpha t^2 \quad \ldots(3.5)$$

$$y' = y + v_0 t + \frac{1}{2}\alpha t^2 \quad \text{...(3.6)}$$

dimana,
x,y : Koordinat sekarang
v_0 : Kecepatan sekarang
a : Akselerasi
t : $Waktu\ (dalam\ detik)$

Nilai waktu yang digunakan pada rumus-rumus diatas adalah 3 detik.Nilai kecepatan dan akselerasi didapatkan dari paket HELLO yang dikirimkan oleh *neighbor node* secara periodik.

3.3.1.1 Perancangan Pemilihan *Forwarding node*

Setelah melakukan penghitungan Nilai *Total Weight of the Route* (TWR) maka selanjutnya akan dilakukan pemilihan *forwarding node*,yaitu *node* yang berhak melakukan *rebroadcast* paket RREQ.Pemilihan *forwarding node* dipilih berdasarkan penerimaan kekuatan sinyal.Apabila *node* saat ini mempunyai kekuatan sinyal yang lebih besar dari lower *threshold* dan lebih kecil dari *upper threshold* dan *node* tersebut terdapat pada list *node* yang eligible untuk meneruskan paket RREQ berdasarkan perhitungan TWR, maka *node* tersebut dapat meneruskan paket RREQ, lalu tambahkan energi RREQ tersebut dengan energi yang dimiliki oleh *node* tersebut. Namun jika tidak, *node* tersebut tidak dapat meneruskan paket RREQ

Gambar 3.5 berikut ini menjelaskan *pseudocode* untuk pemilihan Forwarding Node

```
if (current node's signal > lower threshold
&& current node's signal < upper treshold &&
current node are in the list of eligible node)
then
RREQ_energy = RREQ_energy + node's energy
forward RREQ
else
discard RREQ
end if
```

Gambar 3.5 Pseudocode Perhitungan *Forwarding Node*

3.3.1.2 Perancangan Pemilihan Rute Route Reply (RREP)

Setelah melakukan penghitungan Nilai TWR dan melakukan pemilihan *Forwarding node,*langkah selanjutnya adalah melakukan pemilihan rute RREP berdasarkan pengiriman paket RREQ dari *source node* ke *destination node.* Jika *destination node* menerima RREQ, ia akan menunggu selama waktu tertentu apakah ada RREQ lain yang datang. Jika waktu tunggu habis, *destination node* akan me-reply dengan cara mengirim RREP melalui rute dengan rata-rata energi tertinggi. Kemudian, jika *destination node* menerima RREQ setelah mengirim RREP maka ia akan mengecek apakah rata-rata energi pada RREQ yang datang lebih besar dari pada rata-rata RREQ yang tersimpan. Jika rata-rata energi pada RREQ yang datang lebih besar dari pada rata-rata RREQ yang tersimpan, maka destination node akan mengirim RREP kembali yang akan digunakan sebagai backup rute apabila terjadi kegagalan saat proses route discovery.

Gambar 3.6 berikut ini menjelaskan *pseudocode* untuk Pemilihan Rute RREP

```
cur_residual_energy = rq->rq_energy / hop_count
wait for another T second for another RREP
if (cur_residual_energy > max_residual_energy) then
    max_residual_energy = cur_residual energy
end if
```

```
if waiting time expires then
   send RREP via highest avg energy
end if
if another RREQ received by source node and waiting
time expires then
   if cur_residual_Energy > max_residual_energy then
    send RREP for backup route in case of failure
   end if
end if
```

Gambar 3.6 Pseudocode Pemilihan Rute RREP

3.4 Perancangan Simulasi pada NS-2

Simulasi MANET 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 nilai Total Weight of the Route (TWR),pemilihan Forwarding node dan pemilihan rute Route Reply (RREP) pada file aodv.cc serta perubahan struktur paket RREQ pada file aodv_packet.h. Pada saat simulasi NS-2 dijalankan, maka *routing protocol* AODV akan melakukan seleksi node berdasarkan nilai TWR dan kekuatan sinyal,serta memilih rute RREP berdasarkan rata-rata energi tertinggi yang dimiliki oleh RREQ tersebut.

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 3.7.

$$PDR = \frac{received}{sent} x\ 100\ \% \qquad \ldots\ldots (3.7)$$

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 3.8.

$$E2E = \frac{\sum_{m=1}^{recvnum} CBRRecvTime - CBRSentTime}{recvnum} \qquad \ldots\ldots (3.8)$$

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 3.9.

$$RO = \sum_{m=1}^{sentnum} packet\ sent \qquad \ldots\ldots\ldots\ldots \quad (3.9)$$

Keterangan:

Packet sent = Jumlah kontrol paket RREQ, RREP, dan RRER yang dikirimkan

sentnum = Jumlah total control paket yang terkirim

(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.

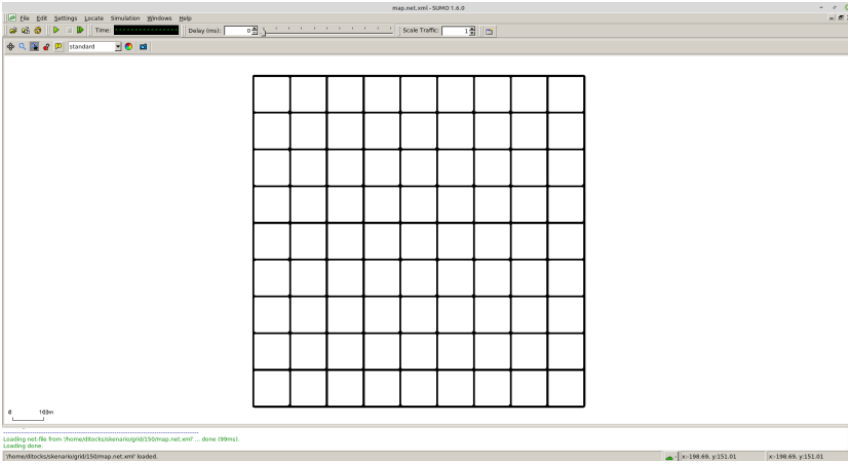
4.1.1 Skenario *Grid*

Dalam mengimplementasikan skenario *grid*, SUMO menyediakan *tools* untuk membuat peta *grid* yaitu netgenerate. Pada Tugas Akhir ini, penulis membuat peta *grid* dengan luas 1100 m x 1100 m yang terdiri dari titik persimpangan antara jalan vertikal dan jalan horisontal sebanyak 10 titik x 10 titik. Dengan jumlah titik persimpangan sebanyak 64 titik tersebut (8x8 persimpangan), maka terbentuk 81 buah petak. Sehingga untuk mencapai luas area sebesar 1100 m x 1100 m dibutuhkan luas per petak sebesar 120 m x 120 m. Berikut perintah netgenerate untuk membuat peta tersebut dengan kecepatan *default* kendaraan sebesar 20 m/s dapat dilihat pada Gambar 4.1.

```
netgenerate --grid --grid.number=10 --
grid.length=120 --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 48 -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

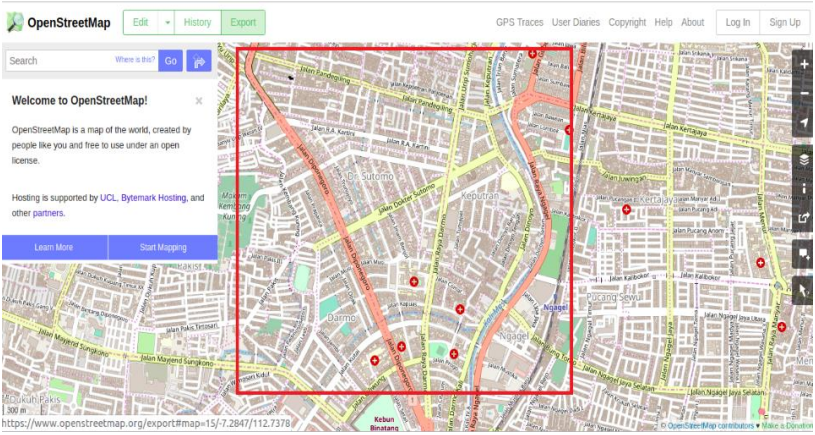
File skenario berekstensi .xml selanjutnya dikonversi ke dalam bentuk *file* berekstensi .tcl agar dapat disimulasikan menggunakan NS-2. *Tools* yang digunakn untuk melakukan konversi ini adalah traceExporter. Perintah untuk menggunakan traceExporter dapat dilihat pada Gambar 4.7.

```
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. Area simulasi ditandai di dalam kotak berwarna merah. Setelah menentukan area simulasi, ekspor data peta dari OpenStreetMap seperti yang ditunjukkan pada Gambar 4.8.



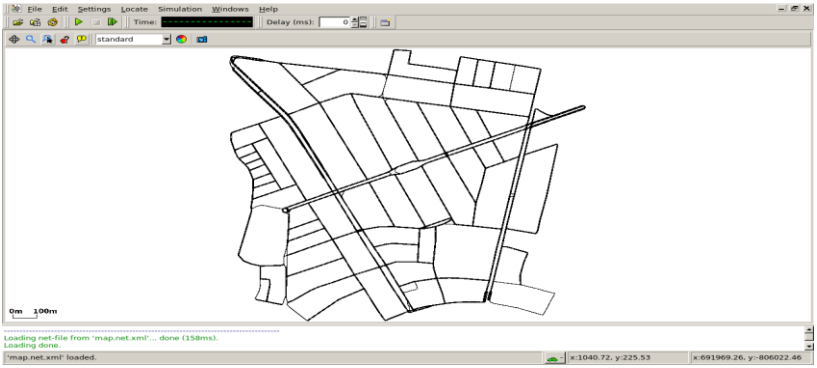
Gambar 4.8 Ekspor Peta dari OpenStreetMap

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 Nilai Total Weight of the Route (TWR),*Forwarding Node* dan Rute RREP

Pada Tugas Akhir ini dilakukan modifikasi pada *routingprotocol* AODV agar dapat melakukan pemilihan rute RREP berdasarkan rata-rata residu energi terbesar.Hal tersebut dilakukan dengan cara menghitung nilai *Total Weight of the Route* setelahnya dilakukan pemilihan *forwarding node* berdasarkan kekuatan sinyal,Setelahnya akan dilakukan RREP berdasarkan nilai rata-rata residu energi terbesar, sehingga dapat dilihat peningkatan performa pada *routing* AODV yang telah dimodifikasi.

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

- Implementasi Penghitungan Nilai *Total Weight of the Route*
- Implementasi Pemilihan *Forwarding Node*
- Implementasi Pemilihan Rute RREP

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 nilai *Total Weight of the Route*,menentukan forwarding *node* dan menentukan rute RREP dan *file* aodv.h yang ada di dalam *folder* ns-2.35/aodv untuk melakukan perubahan struktur paket RREQ. Pada bagian ini penulis akan menjelaskan langkah – langkah dalam mengimplementasikan modifikasi *routing protocol* AODV.

4.2.1 Implementasi Penghitungan Nilai *Total Weight of the Route* dan *Future TWR*

Perhitungan TWR dimulai dengan menghitung jarak antara *next- hop node* dengan *node* tujuan. Kemudian menghitung kualitas hubungan antara *node* pengirim dan *next-hop node* dengan formula indeks stabilitas yang dinormalisasi. Selanjutnya kecepatan dan

akselerasi dari *next-hop node* masing-masing dikurangkan dengan kecepatan dan akselerasi dari *node* tujuan. Tetapi karena *node* tujuan diam, kecepatan dan akselerasi dari *next-hop node* masing-masing dikurangi nol. Kemudian kecepatan, akselerasi, jarak, dan kualitas hubungan dikalikan dengan faktor pengali dan dijumlahkan sehingga menghasilkan nilai TWR.

Fungsi tersebut terdapat pada kode sumber aodv.cc yang terdapat dalam folder ns2.35/aodv. Untuk potongan kode modifikasi penghitungan Total Weight of the Route dapat dilihat pada **Gambar 4.11** berikut.

Kode selengkapnya dapat dilihat di lampiran **A1**.

```
1. // TWR Calculation
2. // Calculate distance between next-hop and dst
3. double nb_distance;
4. nb_distance = sqrt(pow((nb->nb_x - xDst), 2) +
   pow((nb->nb_y - yDst), 2));
5.
6. // radius between this node and neighbor
7. // minimum radius -> min(√(i x – j x )^2 + (i y
   – j y )^2 ; r)
8. double radius = std::min(
9. sqrt(pow((nb->nb_x - posX), 2) + pow((nb->nb_y -
   posY), 2)), (double) maxTxRange);
10.
11.double quality = 1.0 / (1.0 - (radius /
   ((double) maxTxRange + 1.0)));
12.
13.double modSpeed = fSpeed * nb->nb_speed;
14.double modAccel = fAccel * nb->nb_accel;
15.double modDistance = fDistance * nb_distance;
16.double modQuality = fQuality * quality;
17.
18.// TWR = f s × |S n – S d | + f a × |A n – A d |
   + f d × |θ n – θ d | + f q × Q
19.double TWR = modSpeed + modAccel + modDistance +
   modQuality;
```

Gambar 4.11 Potongan Kode Penghitungan Nilai *Total Weight of the Route*

Setelah melakukan perhitungan Setelah melakukan perhitungan TWR, dilakukan perhitungan TWR yang akan datang (*future* TWR) dengan memanfaatkan masi yang sudah didapatkan sebelumnya dan prediksi nilai dengan menggunakan beberapa rumus fisika gerak lurus yang sudah diran- cang. Kecepatan *next-hop node* yang akan datang menggunakan persamaan 3.4, koordinat x dan y masing-masing dihitung dengan persamaan 3.5 dan 3.6. Implementasi dapat dilihat pada Gambar 4.12.

```
20. // Calculate future TWR for next [timeModifier]
    seconds
21. // Future speed v' = v + a × t
22. double nb_speedFuture = nb->nb_speed + (nb-
    >nb_accel * timeModifier);
23.
24. // Future position will be used to calc future
    direction
25. // Formula: x' = x + v0 t × 0.5at^2
26. // Future neighbor position77
27. double nb_xFuture = nb->nb_x + (nb->nb_speed *
    timeModifier) + (0.5 * nb->nb_accel *
    timeModifier * timeModifier);
28. double nb_yFuture = nb->nb_y + (nb->nb_speed *
    timeModifier) + (0.5 * nb->nb_accel *
    timeModifier * timeModifier);
29.
30. // Future this_node position
31. double iXFuture = posX + (iSpeed * timeModifier)
    + (0.5 * iAccel * timeModifier * timeModifier);
32. double iYFuture = posY + (iSpeed * timeModifier)
    + (0.5 * iAccel * timeModifier * timeModifier
33. // Calculate future distance between next-hop
    and dst
34. // Future radius between this node and neighbor:
    both using future values
35. // Calculate future TWR
```

Gambar 4.12 Potongan Kode Penghitungan *Future TWR*

Setelah TWR dan future TWR didapatkan,kedua nilai tersebut disimpan ke dalam sebuah vector yang akan digunakan untuk mengklasifikasikan nilai TWR untuk dilanjutkan ke pengiriman paket RREQ

4.2.2 Implementasi Pemilihan *Forwarding Node*

Di tahap selanjutnya, setelah menghitung nilai Total Weight of the Route, langkah yang harus dilakukan selanjutnya adalah pemilihan *forwarding node* berdasarkan penerimaan kekuatan sinyal serta list node eligible berdasarkan penghitungan TWR untuk melanjutkan paket RREQ.

Potongan kode untuk proses pemilihan Forwarding Node dapat dilihat pada **Gambar 4.13** berikut.

Kode selengkapnya dapat dilihat pada lampiran **A2.**

```
1. // If list is null, then drop packet
2.
3. if(rq->rq_eligible_nodes == NULL) {
4.                Packet::free(p);
5.                return;
6.         }
7. else if (rq->rq_dst != index) {
8.         isEligible = false;
9.
10.for (u_int32_t i = 0;
11.i < rq ->nodes_list_len; ++i) {
12.
13.// I'm on the list, so I'm eligible for this
   packet
14.if (rq->rq_eligible_nodes[i] == index) {
15.isEligible = true;
16.
17.//break;
18.//modified
19.//modif sinyal
20.
21. signal_strength = p->txinfo_.RxPr;
   //RECEIEVE SIGNAL
22. double sinyal;
23. if (signal_strength == 0) {
24.           sinyal = -200;}
25.                }
```

```
29.
30. else {
31. sinyal= 10*log10(signal_strength)+30;
32.      }
33. if (sinyal >= -75 && sinyal <= -30) {
34. rq->rq_energy_count =
35. rq->rq_energy_count + iEnergy;  //penambahan
    energi
36. double energy_avg =
37. rq->rq_energy_count /
38. rq->rq_hop_count;
39.
40.  #ifdef DEBUG
41.  FILE *fp;
42.  fp = fopen("debug.txt", "a");
43.  fprintf(fp, "\n fungsi recvrequest: node %d
    energy: %f, hc: %d, avg energy: %f, sinyal:
    %f", index, rq->rq_energy_count, rq-
    >rq_hop_count, energy_avg, sinyal);
44.           fclose(fp);
45.           #endif
46.         }
47.  else {
48.   Packet::free(p); //drop paket
49.   #ifdef DEBUG
50.   FILE *fp;
51.   fp = fopen("debug.txt", "a");
52.  fprintf(fp, "\n fungsi recvrequest: node %d
    didrop, sinyal: %f", index, sinyal);
53.   fclose(fp);
54.   #endif
```

Gambar 4.13 *Potongan Kode Penghitungan forwarding note*

4.2.3 Implementasi Pemilihan Rute RREP

Pada tahap selanjutnya setelah menghitung nilai Total Weight of the Route (TWR) dan *forwarding node*, langkah yang harus dilakukan selanjutnya adalah pemilihan rute RREP berdasarkan nilai rata-rata residu energi tertinggi serta RREP kembali yang akan digunakan sebagai backup rute apabila terjadi kegagalan saat proses

route discovery..Potongan kode untuk Pemilihan Rute RREP dapat dilihat pada Gambar 4.14

```
1. //modifikasi pemilihan RREP
2. if(rq->rq_dst == index) {
3.   counts++;
4.   double mytime;
5.   double energy_selected = rq->rq_energy_count /
   rq->rq_hop_count; //avg energy = jml energi /
   hop count
6.   double diff; //beda waktu antara rreq ke-n
   dengan rreq ke 1
7.   // rreq pertama yg diterima node tujuan
8.   if (counts == 1) {
9.     last_rreq_on_dest =
   Scheduler::instance().clock();
10.    mytime = Scheduler::instance().clock();
11.    energy_select = energy_selected;
12.    energi_terbesar = rq->rq_src;
13.    diff = mytime - last_rreq_on_dest;
14.    seqno = max(seqno, rq->rq_dst_seqno)+1;
15.    if (seqno%2) seqno++;
16.    sendReply(energi_terbesar, // IP Destination
17.    1, // Hop Count
18.    index, // Dest IP Address
19.    seqno, // Dest Sequence Num
20.    MY_ROUTE_TIMEOUT, // Lifetime
21.    rq->rq_timestamp); // timestamp*/
22.    #ifdef DEBUG
23.          FILE *fp2;
24.          fp2 = fopen("debug.txt", "a");
25.          fprintf(fp2, "destination node
   menerima RREQ pertama dari node %d, ttl
   energy: %f avg energy: %f, hop count: %d,
   last rreq: %f, diff: %f",
26.            rq->rq_src, rq-
   >rq_energy_count, energy_selected, rq-
   >rq_hop_count, last_rreq_on_dest, diff);
27.          fclose(fp);
28.          #endif
29.        }
```

```
30. //rreq berikutnya
31.   else if (counts != 1) {
32.     mytime = Scheduler::instance().clock();
33.     double temp = energy_selected;
34.     //pemilihan energi, jika energi sekarang
    lebih besar drpd sebelumnya, update
35.     if (temp > energy_select) {
36.       isHigher = true;
37.       energy_select = temp;
38.       energi_terbesar = rq->rq_src;
39.     }
40.     diff = mytime - last_rreq_on_dest;
41.     FILE *fp2;
42.     fp2 = fopen("debug.txt", "a");
43.     fprintf(fp2, "\n destination node menerima
    RREQ ke-%d dari node %d, ttl energy: %f avg
    energy: %f, hop count: %d, last rreq: %f, diff:
    %f, energy select: %f",
44.       counts, rq->rq_src, rq->rq_energy_count,
    energy_selected, rq->rq_hop_count,
    last_rreq_on_dest, diff, energy_select);
45.     fclose(fp);
46.   }
47. //pengiriman route reply, lewat energi yang
    terbesar
48.   if (rreq_already_sent == false && diff >=
    0.002) {
49.     rreq_already_sent = true;
50.     #ifdef DEBUG
51.     fprintf(stderr, "%d - %s: destination
    sending reply\n", index, __FUNCTION__);
52.     #endif // DEBUG
53.
54.     // Just to be safe, I use the max. Somebody
    may have
55.     // incremented the dst seqno.
56.     seqno = max(seqno, rq->rq_dst_seqno)+1;
57.     if (seqno%2) seqno++;
58.
59.
```

```
60. sendReply(energi_terbesar, // IP Destination
61.     1, // Hop Count
62.     index, // Dest IP Address
63.     seqno, // Dest Sequence Num
64.     MY_ROUTE_TIMEOUT, // Lifetime
65.     rq->rq_timestamp); // timestamp
66.     #ifdef DEBUG
67.       FILE *fp2;
68.       fp2 = fopen("debug.txt", "a");
69.       fprintf(fp2, "\n mengirim route reply ke
    node %d, energi: %f", energi_terbesar,
    energy_select);
70.       fclose(fp2);
71.     #endif
72.       Packet::free(p);
73.   }
74.   //rreq berikutnya buat jika terdapat error
    route discovery
75.   if (isHigher == true && rreq_already_sent ==
    true) {
76.     seqno = max(seqno, rq->rq_dst_seqno)+1;
77.     if (seqno%2) seqno++;
78.
79.     sendReply(energi_terbesar, // IP Destination
80.     1, // Hop Count
81.     index, // Dest IP Address
82.     seqno, // Dest Sequence Num
83.     MY_ROUTE_TIMEOUT, // Lifetime
84.     rq->rq_timestamp); // timestamp
85.     #ifdef DEBUG
86.       FILE *fp2;
87.       fp2 = fopen("debug.txt", "a");
88.       fprintf(fp2, "\n mengirim route reply ke
    node %d, energi: %f", energi_terbesar,
    energy_select);
89.       fclose(fp2);
90.     #endif
91. delete[] rq->rq_eligible_nodes;
92.     rq->rq_eligible_nodes = NULL;
93.       Packet::free(p);
94.   }
```

Gambar 4.14 Potongan Kode Pemilihan Rute RREP

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 4.15**. Kode selengkapnya di lampiran **A4**

```
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)       700
set opt(y)       700
set val(ifqlen)  1000
set val(nn)      200
set val(seed)    1.0
set val(adhocRouting) AODV
set val(stop)    200
set val(cp)      "cbr200.tcl"
```

Gambar 4.15 Implementasi Simulasi 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 4.16**.

```
set udp_(0) [new Agent/UDP]
$ns_ attach-agent $node_(198) $udp_(0)
set null_(0) [new Agent/Null]
$ns_ attach-agent $node_(199) $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_ 1000
$cbr_(0) attach-agent $udp_(0)
$ns_ connect $udp_(0) $null_(0)
$ns_ at 0 "$cbr_(0) start"
$ns_ at 200.0000000000000000 "$cbr_(0) stop"
```

Gambar 4.16 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*.

4.3.1 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 Packet Delivery Ratio (PDR), End-to-end Delay (E2E), dan Routing Overhead (RO).

4.3.2 Implementasi *Packet Delivery Ratio* (PDR)

Pada subbab 0 telah ditunjukkan contoh struktur data *event* yang dicatat dalam *trace file* oleh NS-2. Kemudian, pada persamaan

3.7 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 3.7. Pseudocode untuk menghitung PDR dapat dilihat pada Gambar 4.17.

```
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 4.17 Pseudocode untuk Perhitungan PDR

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

4.3.3 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 4.18.

```
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 4.18 Pseudocode untuk Perhitungan E2E

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

4.3.4 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 4.19.

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

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

Gambar 4.19 Pseudocode Perhitungan Routing Overhead

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™ i7-6700HQ CPU @ 2.70GHz
Sistem Operasi	Ubuntu 18.04.2 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, dan RO, 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*, dan A.8 Kode Skrip AWK *Routing Overhead*.

Tabel 5.2 Lingkungan Uji Coba

No.	Parameter	Spesifikasi
1	Network simulator	NS-2.35
2	*Routing protocol*	AODV dan AODV Modifikasi
3	Waktu simulasi	200 detik
4	Area simulasi	1100 m x 1100 m
5	Jumlah *Node*	50, 100, 150, 200
6	Radius transmisi	400m
7	Kecepatan maksimum	20 m/s
8	Paket Interval	1 paket/detik
9	Protokol MAC	IEEE 802.11p
10	Model Propagasi	*Two-ray ground*

5.2 Hasil Uji Coba

Hasil uji coba menggunakan *node* sumber dan *node* tujuan yang diletakkan secara statis. Hasil dapat dilihat sebagai berikut:

5.2.1 Hasil Uji Coba Skenario *Grid*

Pengujian pada skenario *grid* digunakan untuk melihat perbandingan PDR, E2E, dan RO antara *routing protocol* AODV asli dan AODV yang telah dimodifikasi.

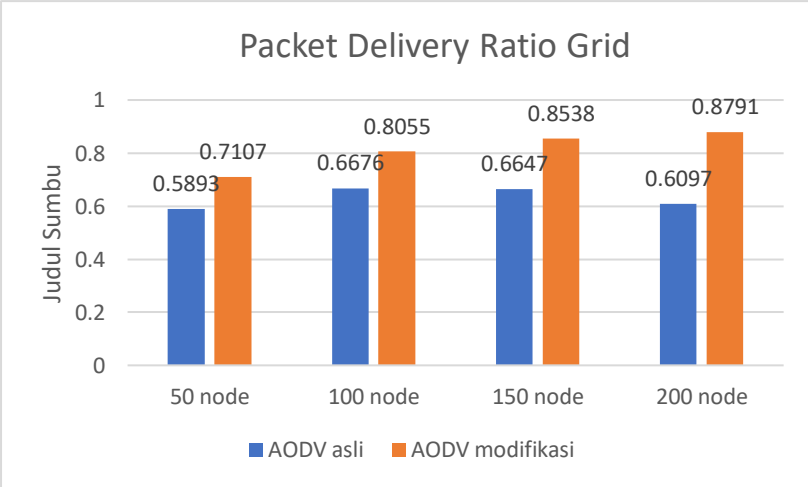
Pengambilan data uji pada skenario *grid* dilakukan sebanyak 10 kali dengan skenario mobilitas *random* pada peta *grid* dengan luas area 1100 m x 1100m m dan *node* sebanyak 50 untuk lingkungan jarang, 100 dan 150 *node* untuk lingkungan yang sedang, dan 200 *node* untuk lingkungan padat dilakukan pada kecepatan standar yaitu 20 m/s.

Tabel 5.3 berikut ini merupakan rata-rata hasil pengambilan data *Packet Delivery Ratio* (PDR) pada skenario *grid* dengan jumlah *node* 50, 100, 150, dan 200.

Hasil pengambilan data rata-rata untuk Packet Delivery Ratio pada skenario *grid* dengan jumlah *node* 50, 100, 150, dan 200 dapat dilihat pada **Gambar 5.1**.

Tabel 5.3 Hasil Rata-rata PDR Skenario *Grid*

Jumlah *Node*	**AODV Asli**	**AODV Modifikasi**	**Perbedaan**
50	0.5893	0.7107	+ 0.1214
100	0.6676	0.8055	+ 0.1379
150	0.6647	0.8538	+ 0.1891
200	0.6097	0.8791	+ 0.2694



Gambar 5.1 Grafik Packet Delivery Ratio Skenario Grid

Berdasarkan grafik pada **Gambar 5.1** di atas, terlihat bahwa performa *routing protocol* AODV yang telah dimodifikasi mengungguli AODV asli dari segi *packet delivery ratio* baik di lingkungan yang jarang, sedang, maupun padat. Pada lingkungan

yang jarang dengan jumlah 50 node, AODV yang telah dimodifikasi unggul dari AODV asli dengan selisih 0.1214 atau naik menjadi sekitar 20,60%. Pada lingkungan yang sedang dengan jumlah 100 node, AODV yang telah dimodifikasi unggul dari AODV asli dengan selisih 0,1379 atau naik menjadi sekitar 20.66%. Pada lingkungan yang sedang dengan jumlah 150 node, AODV yang telah dimodifikasi unggul dari AODV asli dengan selisih 0.1891 atau naik menjadi sekitar 28.45%. Pada lingkungan yang padat dengan jumlah 200 node, AODV yang telah dimodifikasi unggul dari AODV asli dengan selisih 0,2694 atau naik menjadi sekitar 44,19%.

Dapat dilihat rata-rata terjadi peningkatan PDR sebesar 28,47%. Jika ketiga lingkungan tersebut dibandingkan dapat dilihat bahwa dengan jumlah *node* yang lebih sedikit atau pada lingkungan dengan *node* yang jarang, akan dihasilkan nilai PDR yang lebih tinggi daripada di lingkungan dengan jumlah *node* yang sedang maupun yang padat baik untuk *routing protocol* AODV asli. Dapat dilihat pula bahwa AODV yang telah dimodifikasi menghasilkan PDR yang lebih baik daripada AODV asli dengan jumlah selisih PDR yang cukup signifikan yang menjadi semakin unggul seiring banyaknya jumlah node (makin padatnya lingkungan).

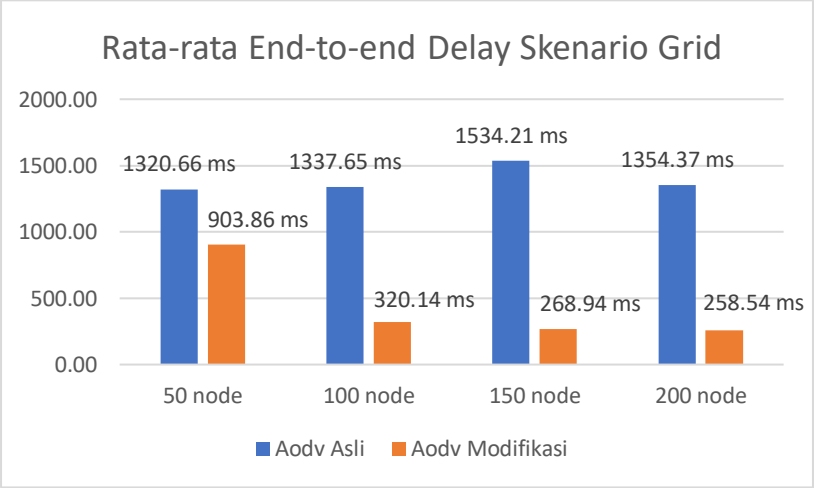
Peningkatan PDR tersebut disebabkan karena tingkat kepadatan node yang bertambah,Seiring dengan bertambahnya kepadatan node, maka akan semakin banyak pula kemungkinan rute yang dapat dilalui. Pada metode modifikasi setelah node tujuan mengirim RREP melalui rata-rata energi tertinggi, node tujuan akan mengirim RREP kembali yang nantinya akan dijadikan rute cadangan apabila sewaktu-waktu rute yang ada terputus. Hal ini dapat menyebabkan bertambahnya packet yang dapat diterima oleh node tujuan dikarenakan akan langsung mengambil rute cadangan sebagai rute yang dilalui paket, dan tidak melakukan proses route discovery dari awal. Serta metode modifikasi yang memilih forwarding node yang eligible berdasarkan faktor Total Weight of Route (TWR), pengukuran kekuatan sinyal,dan pemilihan rute RREP berdasarkan

rata-rata energi rute yang tertinggi sehingga rute yang dipilih lebih stabil yang memungkinkan lebih banyak paket yang dapat diterima dan mengurangi jumlah packet loss.

Hasil pengambilan data rata-rata untuk *end-to-end delay* (E2E) pada skenario *grid* dengan jumlah *node* 50, 100, 150, dan 200 dapat dilihat pada **Gambar 5.2**.

Tabel 5.4 Hasil Rata-rata E2E Skenario Grid

Jumlah Node	AODV Asli	AODV Modifikasi	Perbedaan
50	1320.66 ms	903.86 ms	-416.80 ms
100	1337.65 ms	320.14 ms	-1017.51 ms
150	1534.21 ms	268.94 ms	-1265.27 ms
200	1354.37 ms	258.54 ms	-1095.83 ms



Gambar 5.2 Grafik End-to-end Delay Skenario Grid

Berdasarkan grafik pada **Gambar 5.2** di atas, terlihat bahwa performa *routing protocol* AODV yang telah dimodifikasi mengungguli AODV asli dari segi *end-to-end delay* baik di lingkungan yang jarang, sedang, maupun padat. Pada lingkungan yang jarang dengan jumlah 50 node, terjadi perbedaan selisih *end-to-end delay* sebesar 416.80 ms antara *routing protocol* AODV asli dengan *routing protocol* AODV yang telah dimodifikasi atau mengalami penurunan sebesar 31,56%, dimana *routing protocol* AODV yang telah dimodifikasi unggul dalam hal *end-to-end delay* tersebut. Sedangkan pada lingkungan sedang dengan jumlah 100 node, terjadi perbedaan selisih *end-to-end delay* sebesar 1017,51 ms antara *routing protocol* AODV asli dengan *routing protocol* AODV yang telah dimodifikasi atau mengalami penurunan sebesar 76,07%, dimana *routing protocol* AODV yang dimodifikasi lebih unggul dalam hal *end-to-end delay* tersebut. Pada pada lingkungan sedang dengan jumlah 150 node, terjadi perbedaan selisih *end-to-end delay* sebesar 1265,27 ms antara *routing protocol* AODV asli dengan *routing protocol* AODV yang telah dimodifikasi atau mengalami penurunan sebesar 82,47%, dimana *routing protocol* AODV yang dimodifikasi lebih unggul dalam hal *end-to-end delay* tersebut. Pada lingkungan yang padat dengan jumlah 200 node, terjadi perbedaan selisih *end-to-end delay* sebesar 1095,83 ms antara ro*uting protocol* AODV asli dengan *routing protocol* AODV yang telah dimodifikasi atau mengalami penurunan sebesar 80,91%, dimana *routing protocol* AODV yang telah dimodifikasi jauh lebih unggul dalam hal *end-to-end delay* tersebut.

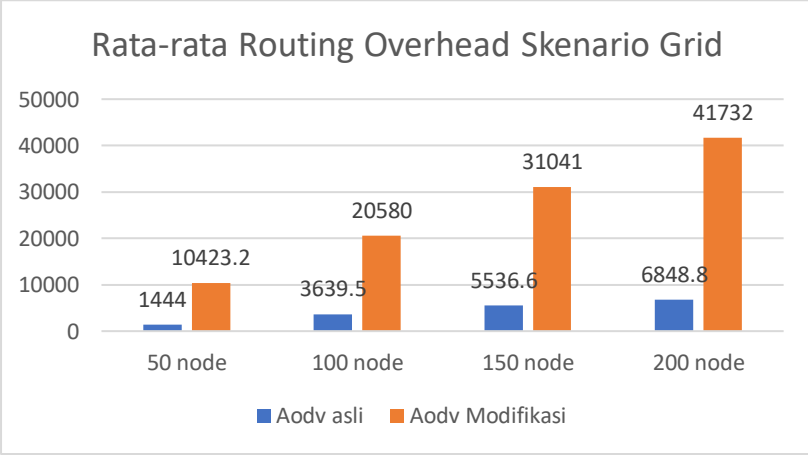
Jika membandingkan antara ketiga lingkungan tersebut memang pada lingkungan yang jarang, sedang, maupun padat, routing *protocol* AODV yang telah dimodifikasi selalu lebih unggul dalam hal *end-to-end delay*. Dapat dilihat bahwa dengan jumlah node jarang, sedang dan padat menghasilkan *end-to-end delay* yang lebih baik dan dapat dilihat pula bahwa AODV yang telah dimodifikasi menghasilkan *end-to end delay* yang lebih baik daripada AODV asli dengan jumlah selisih *end-to-end delay* yang cukup signifikan. Hal ini

dapat terjadi karena dengan AODV modifikasi hanya melewati node dengan nilai TWR dan kekuatan sinyal yang memenuhi kriteria sebagai relay node yang menyebabkan paket-paket akan lebih sedikit mengalami adanya delay dalam pengiriman. Hasil rata – rata E2E tidak dapat dianalisis karena terjadi fluktuasi dan tidak stabil. Hal ini dikarenakan waktu *delay* tergantung dari rata – rata waktu paket yang terkirim. Semakin banyak paket yang terkirim, maka semakin beragam *delay*nya.

Untuk hasil pengambilan data *routing overhead* pada skenario *grid* 50 *node*, 100 *node*, 150 *node* dan 200 *node* dapat dilihat pada **Gambar 5.3**.

Tabel 5.5 Hasil Rata-rata Routing Overhead Skenario Grid

Jumlah Node	AODV Asli	AODV Modifikasi	Perbedaan
50	1444	10423.2	8979.2
100	3639.5	20580	16940.5
150	5536.6	31041	25504.4
200	6848.8	41732	34883.2



Gambar 5.3 Grafik Routing Overhead Skenario Grid

Berdasarkan grafik pada **Gambar 5.3** dapat dilihat bahwa rata-rata *routing overhead* AODV asli mengungguli AODV modifikasi dengan perbedaan yang cukup signifikan. Pada lingkungan yang jarang dengan jumlah 50 node, menghasilkan perbedaan selisih *routing overhead* sebesar 8979,2 dimana *routing protocol* AODV asli unggul dalam hal *routing overhead* tersebut karena menghasilkan *routing overhead* yang lebih rendah dari routing AODV modifikasi. Pada lingkungan yang sedang dengan jumlah 100 node, menghasilkan perbedaan selisih *routing overhead* sebesar 16940,5 dimana *routing protocol* AODV asli unggul dalam hal *routing overhead* daripada AODV yang telah dimodifikasi. Pada lingkungan yang sedang dengan jumlah 150 node, menghasilkan perbedaan selisih *routing overhead* sebesar 25504,4 dimana *routing protocol* AODV asli unggul dalam hal *routing overhead* daripada AODV yang telah dimodifikasi. Pada lingkungan yang padat dengan jumlah 200 node, menghasilkan perbedaan selisih *routing overhead* sebesar 34883,2 dimana *routing protocol* AODV asli unggul dalam hal *routing overhead* tersebut.

Dapat dilihat bahwa kenaikan *routing overhead* AODV modifikasi adalah sekitar 4-5 kali lipat jika dibandingkan dengan AODV asli. Jika dibandingkan antara ketiga lingkungan tersebut dapat dilihat bahwa selisih routing overhead akan menjadi lebar seiring dengan peningkatan jumlah *node*.

5.2.2 Hasil Uji Coba Skenario *Real*

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

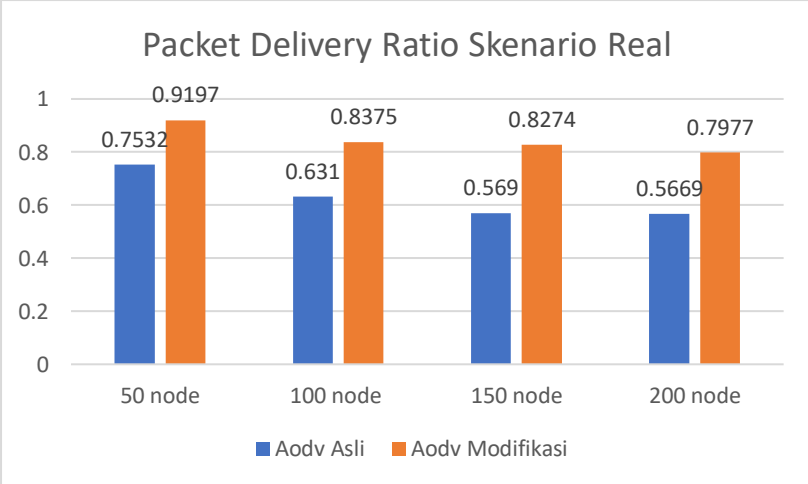
Pengambilan data uji PDR, E2E, dan RO pada skenario *real* dilakukan sebanyak 10 kali dengan skenario mobilitas *random* pada peta *real* dengan luas area 1100 m x 1100 m dengan *range transmisi* 250 meter dan *node* sebanyak 50 untuk lingkungan yang jarang, 100

dan 150 *node* untuk lingkungan yang sedang, dan 200 *node* untuk lingkungan yang padat dilakukan pada kecepatan standar yaitu 20 m/s. Untuk hasil analisis Skenario dengan peta *Real* dengan *node* 50, 100, 150 dan 200 dapat dilihat pada Tabel 5.9, Tabel 5.10, dan Tabel 5.11.

Tabel 5.6 berikut ini merupakan rata-rata hasil pengambilan data *Packet Delivery Ratio* (PDR) pada skenario *real* dengan jumlah *node* 50, 100, 150, dan 200.

Tabel 5.6 Hasil Rata-rata PDR Skenario Real

Jumlah *Node*	**AODV Asli**	**AODV Modifikasi**	**Perbedaan**
50	0.7532	0.9197	0.1665
100	0.631	0.8375	0.2065
150	0.569	0.8274	0.2584
200	0.5669	0.7977	0.2308



Gambar 5.4 Grafik Packet Delivery Ratio Skenario Real

Berdasarkan grafik pada **Gambar 5.4** di atas, terlihat bahwa performa *routing protocol* AODV yang telah dimodifikasi mengungguli AODV asli dari segi *packet delivery ratio* baik di lingkungan yang jarang, sedang, maupun padat. Pada lingkungan yang jarang dengan jumlah 50 node, AODV yang telah dimodifikasi unggul dari AODV asli dengan selisih 0,1665 atau naik menjadi sekitar 22,11%. Pada lingkungan yang sedang dengan jumlah 100 node, AODV yang telah dimodifikasi unggul dari AODV asli dengan selisih 0,2065 atau naik menjadi sekitar 32,73%. Pada lingkungan yang sedang dengan jumlah 150 node, AODV yang telah dimodifikasi unggul dari AODV asli dengan selisih 0,2584 atau naik menjadi sekitar 45,41%. Pada lingkungan yang padat dengan jumlah 200 node, AODV yang telah dimodifikasi unggul dari AODV asli dengan selisih 0,2308 atau naik menjadi sekitar 40,71%.

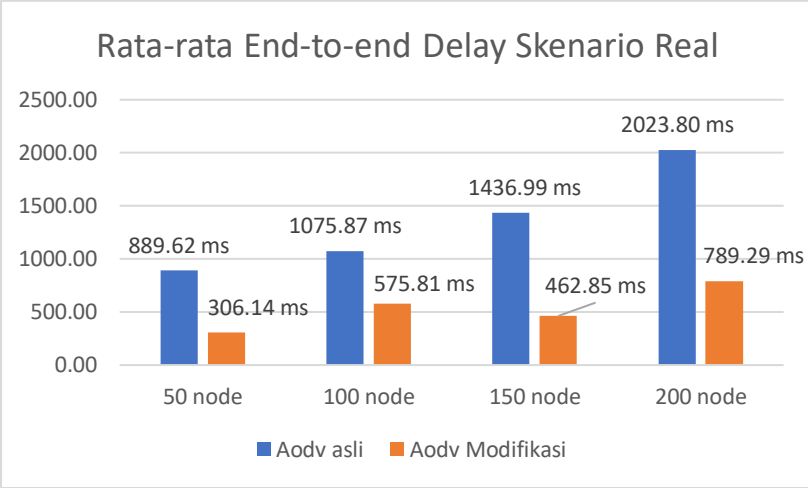
Dapat dilihat rata-rata terjadi peningkatan PDR sebesar 35,24%. Jika ketiga lingkungan tersebut dibandingkan dapat dilihat bahwa dengan jumlah *node* yang lebih sedikit atau pada lingkungan dengan *node* yang jarang, akan dihasilkan nilai PDR yang lebih bagus daripada di lingkungan dengan jumlah *node* yang sedang maupun yang padat baik untuk *routing protocol* AODV asli maupun *routing protocol* AODV yang telah dimodifikasi. Dapat dilihat pula bahwa AODV yang telah dimodifikasi menghasilkan PDR yang lebih bagus daripada AODV asli dengan jumlah selisih PDR yang cukup signifikan yang menjadi semakin unggul seiring banyaknya jumlah *node* (makin padatnya lingkungan).

Hasil pengambilan data rata-rata untuk *end-to-end delay* (E2E) pada skenario *real* dengan jumlah *node* 50, 100, 150, dan 200 dapat dilihat pada **Gambar 5.5**.

Tabel 5.7 Hasil Rata-rata E2E Skenario Real

Jumlah *Node*	**AODV Asli**	**AODV Modifikasi**	**Perbedaan**
50	889.62 ms	306.14 ms	-583.49 ms

100	1075.87 ms	575.81 ms	-500.06 ms
150	1436.99 ms	462.85 ms	-974.14 ms
200	2023.80 ms	789.29 ms	-1234.51 ms



Gambar 5.5 Grafik End-to-end Delay Skenario Real

Berdasarkan grafik pada **Gambar 5.5** di atas, terlihat bahwa performa *routing protocol* AODV yang telah dimodifikasi mengungguli AODV asli dari segi *end-to-end delay* di lingkungan yang sedang dan padat. Pada lingkungan yang jarang dengan jumlah 50 node, terjadi perbedaan selisih *end-to-end delay* sebesar 583,49 ms antara *routing protocol* AODV asli dengan *routing protocol* AODV yang telah dimodifikasi atau mengalami penurunan sebesar 65,59%, dimana *routing protocol* AODV modifikasi unggul dalam hal *end-to-end delay* tersebut. Sedangkan pada lingkungan sedang dengan jumlah 100 node, terjadi perbedaan selisih *end-to-end delay* sebesar 500,06 ms antara *routing protocol* AODV asli dengan *routing protocol* AODV yang telah dimodifikasi atau mengalami penurunan sebesar 46,68%, dimana *routing protocol* AODV yang dimodifikasi lebih unggul dalam hal *end-to-end delay* tersebut. Pada pada

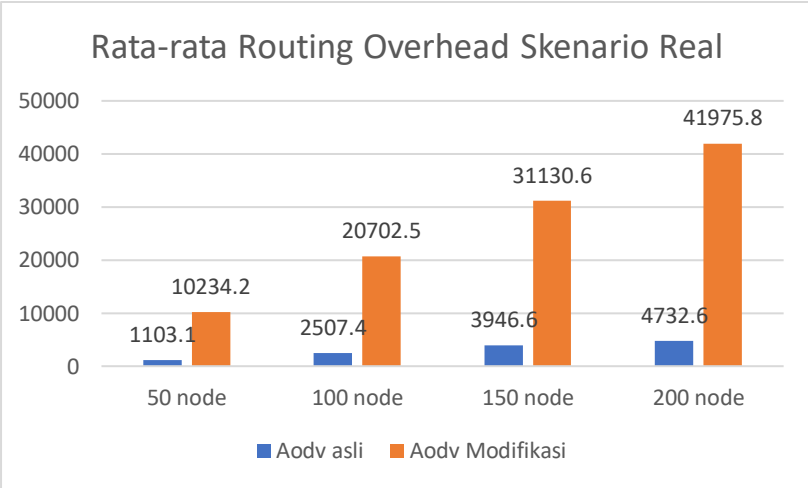
lingkungan sedang dengan jumlah 150 node, terjadi perbedaan selisih *end-to-end delay* sebesar 974,14 ms antara *routing protocol* AODV asli dengan *routing protocol* AODV yang telah dimodifikasi atau mengalami penurunan sebesar 67,79%, dimana *routing protocol* AODV yang dimodifikasi lebih unggul dalam hal *end-to-end delay* tersebut. Pada lingkungan yang padat dengan jumlah 200 node, terjadi perbedaan selisih *end-to-end delay* sebesar 1234,51 ms antara ro*uting protocol* AODV asli dengan *routing protocol* AODV yang telah dimodifikasi atau mengalami penurunan sebesar 61,00%, dimana *routing protocol* AODV yang telah dimodifikasi jauh lebih unggul dalam hal *end-to-end delay* tersebut.

Jika membandingkan antara ketiga lingkungan tersebut memang pada lingkungan yang jarang, sedang, maupun padat, routing *protocol* AODV yang telah dimodifikasi selalu lebih unggul dalam hal *end-to-end delay*. Dapat dilihat bahwa dengan jumlah node sedang dan padat menghasilkan *end-to-end delay* yang lebih baik dan dapat dilihat pula bahwa AODV yang telah dimodifikasi menghasilkan *end-to end delay* yang lebih baik daripada AODV asli dengan jumlah selisih *end-to-end delay* yang cukup signifikan.Hal ini dapat terjadi karena dengan AODV modifikasi hanya melewati node dengan nilai TWR dan kekuatan sinyal yang memenuhi kriteria sebagai relay node yang menyebabkan paket-paket akan lebih sedikit mengalami adanya delay dalam pengiriman. Hasil rata – rata E2E tidak dapat dianalisis karena terjadi fluktuasi dan tidak stabil. Hal ini dikarenakan waktu *delay* tergantung dari rata – rata waktu paket yang terkirim. Semakin banyak paket yang terkirim, maka semakin beragam *delay*nya.

Untuk hasil pengambilan data *routing overhead* pada skenario *real* 50 *node*, 100 *node*, 150 *node* dan 200 *node* dapat dilihat pada **Gambar 5.6**.

Tabel 5.8 Hasil Rata-rata Routing Overhead Skenario Real

Jumlah Node	AODV Asli	AODV Modifikasi	Perbedaan
50	1103.1	10234.2	9131.1
100	2507.4	20702.5	18195.1
150	3946.6	31130.6	27184
200	4732.6	41975.8	37243.2



Gambar 5.6 Grafik Routing Overhead Skenario Real

Berdasarkan grafik pada **Gambar 5.6** dapat dilihat bahwa rata-rata *routing overhead* AODV asli mengungguli AODV modifikasi dengan perbedaan yang cukup signifikan. Pada lingkungan yang jarang dengan jumlah 50 node, menghasilkan perbedaan selisih *routing overhead* sebesar 9131,1 dimana *routing protocol* AODV asli unggul dalam hal *routing overhead* tersebut karena menghasilkan *routing overhead* yang lebih rendah dari routing AODV modifikasi. Pada lingkungan yang sedang dengan jumlah 100 node, menghasilkan perbedaan selisih *routing overhead* sebesar 18195,1 dimana *routing*

protocol AODV asli unggul dalam hal *routing overhead* daripada AODV yang telah dimodifikasi. Pada lingkungan yang sedang dengan jumlah 150 node, menghasilkan perbedaan selisih *routing overhead* sebesar 27184 dimana *routing protocol* AODV asli unggul dalam hal *routing overhead* daripada AODV yang telah dimodifikasi. Pada lingkungan yang padat dengan jumlah 200 node, menghasilkan perbedaan selisih *routing overhead* sebesar 37423,2 dimana *routing protocol* AODV asli unggul dalam hal *routing overhead* tersebut.

Dapat dilihat bahwa kenaikan *routing overhead* AODV modifikasi adalah sekitar 6-9 kali lipat jika dibandingkan dengan AODV asli. Jika dibandingkan antara ketiga lingkungan tersebut dapat dilihat bahwa selisih routing overhead akan menjadi lebar seiring dengan peningkatan jumlah *node*.

BAB VI
KESIMPULAN DAN SARAN

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

6.1 Kesimpulan

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

1. Dengan melakukan seleksi rute berdasarkan kondisi energi jalur menggunakan metode penjumlahan nilai *Total Weight of the Route* (TWR) dan pemilihan *forwarding node* berdasarkan penerimaan kekuatan sinyal serta pemilihan rute RREP yang sudah terimplementasikan maka dapat meningkatkan performa pada *routing protocol* AODV
2. Dampak modifikasi AODV terhadap performa protokol AODV pada skenario *grid* adalah rata – rata kenaikan *Packet Delivery Ratio* sebesar 28,47%, rata – rata penurunan *End-to-end Delay* sebesar 67,75%, namun dari sisi *routing overhead* meningkat 6-9 kali lipat dibanding AODV asli.

6.2 Saran

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

1. Menambahkan aspek lain untuk mempercepat proses route discovery seperti arah, kecepatan,kepadatan node dan aspek lainnya.
2. Lebih banyak uji coba yang dilakukan untuk mendapatkan hasil yang lebih akurat.

(Halaman ini sengaja dikosongkan)

DAFTAR PUSTAKA

[1] M. L. Raja dan C. D. S. S. Baboo, "An Overview of MANET: Applications, Attacks and Challenges," *International Journal of Computer Science and Mobile Computing,* vol. 3, no. 1, pp. 408-417, 2014.

[2] A. Veerasamy, S. R. Madane, S. K dan A. Sivaraman, "Angle and Context Free Grammar Based Precarious Node Detection and Secure Data Transmission in MANETs," 2016.

[3] 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.

[4] "AWK," [Online]. Available: http://tldp.org/LDP/abs/html/awk.html. [Diakses 10 01 2017].

[5] "VANET - Vehicle Ad hoc Network," [Online]. Available: http://comp.ist.utl.pt/~rmr/WSN/CaseStudies2007-no/WSN_Transportation/. [Diakses 15 November 2017].

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

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

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

[9] "OpenStreetMap," [Online]. Available: https://www.openstreetmap.org/. [Diakses 15 November 2017].

[10] "JOSM," [Online]. Available: https://josm.openstreetmap.de/. [Diakses 15 November 2017].

[11] 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.

[12] 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.

[13] Y. Feng , B. Zhang, S. Chai, L. Cui dan Q. Li, "An Optimized AODV Protocol based on Clustering for WSNs," *6th International Conference on Computer Science and Network Technology (ICCSNT)*, 2017.

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

[15] X. Shen, Y. Wu, Z. Xu dan X. Lind, "AODV-PNT: An improved version of AODV routing protocol with predicting node trend in VANET," dalam *The 7th IEEE/International Conference on Advanced Infocomm Technology*, Fuzhou, China , 2014.

[16] S. Yousefi, M. S. Mousavi dan M. Fathi, "Vehicular Ad Hoc Networks (VANETs): Challenges and Perspectives," dalam *2006 6th International Conference on ITS Telecommunications*, Chengdu, China, 2006.

[17] A. Koujalagi, "Considerable Detection of Black Hole Attack and Analyzing its Performance on AODV Routing Protocol in MANET (Mobile Ad Hoc Network)," *American Journal of Computer Science and Information Technology*, 2018.

[18] M. K. Riaz , F. Yangyu dan I. Akhtar, "Energy Aware Path Selection based Efficient AODV for MANETs," 2019.

LAMPIRAN

A.1. Kode fungsi *Total Weight of the Route (TWR)*

```
// TWR list
std::vector<TWRContainer> listTWR;
std::vector<nsaddr_t> eligibleAddrList;
nsaddr_t *eligibleNodes;

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

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

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

u_int32_t fSpeed = 15;
u_int32_t fAccel = 10;
u_int32_t fDistance = 20;
u_int32_t fQuality = 50;

u_int32_t timeModifier = 3;
u_int32_t maxTxRange = 250; // ns2 default range
(based on Pt_)
u_int32_t thresholdW = 100;

// Dst fixed position (STA dst). real scenario: we can
find exact loc via GPS
double xDst = 25.0;
```

```
double yDst = 25.0;

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

for(; nb; nb = nb->nb_link.le_next) {
// TWR Calculation

// Calculate distance between next-hop and dst
double nb_distance;
nb_distance = sqrt(pow((nb->nb_x - xDst), 2) +
pow((nb->nb_y - yDst), 2));

// radius between this node and neighbor
// minimum radius -> min(√(i x − j x )^2 + (i y − j y
)^2 ; r)
double radius = std::min(
sqrt(pow((nb->nb_x - posX), 2) + pow((nb->nb_y -
posY), 2)), (double) maxTxRange);

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

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

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

// Calculate future TWR for next [timeModifier]
seconds
// Future speed v' = v + a × t
double nb_speedFuture = nb->nb_speed + (nb->nb_accel *
timeModifier);

// Future position will be used to calc future
direction
// Formula: x' = x + v0 t × 0.5at^2
// Future neighbor position77
```

```
double nb_xFuture = nb->nb_x + (nb->nb_speed *
timeModifier) + (0.5 * nb->nb_accel * timeModifier *
timeModifier);
double nb_yFuture = nb->nb_y + (nb->nb_speed *
timeModifier) + (0.5 * nb->nb_accel * timeModifier *
timeModifier);

// Future this_node position
double iXFuture = posX + (iSpeed * timeModifier) +
(0.5 * iAccel * timeModifier * timeModifier);
double iYFuture = posY + (iSpeed * timeModifier) +
(0.5 * iAccel * timeModifier * timeModifier);

// Calculate future distance between next-hop and dst
double futureDistance;
futureDistance = sqrt(pow((nb_xFuture - xDst), 2) +
pow((nb_yFuture - yDst), 2));

// Future radius between this node and neighbor: both
using future values
double futureRadius = std::min(
sqrt(pow((nb_xFuture - iXFuture), 2) + pow((nb_yFuture
- iYFuture), 2)), (double) maxTxRange);

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

// Calculate future TWR
modSpeed = fSpeed * nb_speedFuture;
modAccel = fAccel * nb->nb_accel;
modDistance = fDistance * futureDistance;
modQuality = fQuality * futureQuality;
double futureTWR = modSpeed + modAccel + modDistance +
modQuality;

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

```
listTWR.push_back(container);
}

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

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

for (u_int32_t i = 0; i < listTWR.size(); i++) {
// TWR yang akan datang dikatakan "lebih baik" apabila
nilainya lebih
// rendah dari TWR sekarang. TWR yang akan datang
dikatakan "lebih
// buruk" apabila nilainya lebih besar dari TWR
sekarang.
if (listTWR[i].futureTWR < listTWR[i].TWR) {
listTWR[i].isFutureBetter = true;
}

// Jika ΔTWR lebih kecil dari threshold W, maka node
tersebut
// dianggap sebagai "stable".
// Jika ΔTWR lebih besar dari threshold W, maka node
tersebut
// dianggap sebagai "unstable".
double deltaTWR = fabs(listTWR[i].TWR -
listTWR[i].futureTWR);
if (deltaTWR < thresholdW) {
listTWR[i].isStable = true;
}

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

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

// replace rq_eligible_nodes with the re-computed list
rq->nodes_list_len = eligibleAddrList.size();
eligibleNodes = new nsaddr_t[rq->nodes_list_len];
for (u_int32_t i = 0; i < rq->nodes_list_len; i++) {
eligibleNodes[i] = eligibleAddrList[i];
}
rq->rq_eligible_nodes = eligibleNodes;
}
```

A.2. Kode fungsi *Forwarding Note*

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

//modifikasi buat cari energi rata2 terbesar
pd node tujuan
  if (id_lookup(rq->rq_src, rq->rq_bcast_id)
&& rq->rq_dst != index) {

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

       Packet::free(p);
       return;
       }

// If list is null, then drop packet
       if(rq->rq_eligible_nodes == NULL) {
              Packet::free(p);
              return;
       }
       else if (rq->rq_dst != index) {
              isEligible = false;
for (u_int32_t i = 0; i <rq>nodes_list_len;
++i) {
// I'm on the list, so I'm eligible for this
packet
```

```
if (rq->rq_eligible_nodes[i] == index) {
       isEligible = true;
        //break;
        //modified
        //modif sinyal
        signal_strength = p->txinfo_.RxPr;
//RECIEVE SIGNAL
        double sinyal;
        if (signal_strength == 0) {
          sinyal = -200;
        }
        else {
    sinyal = 10*log10(signal_strength) + 30;
        }
    if (sinyal >= -75 && sinyal <= -30) {
rq->rq_energy_count = rq->rq_energy_count +
iEnergy;
//buat nambahin energi
double energy_avg = rq->rq_energy_count /
rq->rq_hop_count;
          #ifdef DEBUG
          FILE *fp;
          fp = fopen("debug.txt", "a");
          fprintf(fp, "\n fungsi
recvrequest: node %d energy: %f, hc: %d, avg
energy: %f, sinyal: %f", index, rq-
>rq_energy_count, rq->rq_hop_count,
energy_avg, sinyal);
          fclose(fp);
          #endif
        }
        else {
```

```
Packet::free(p); //buat drop paket
           #ifdef DEBUG
           FILE *fp;
           fp = fopen("debug.txt", "a");
           fprintf(fp, "\n fungsi
recvrequest: node %d didrop, sinyal: %f",
index, sinyal);
           fclose(fp);
           #endif
         }
                        }
                   }
       }

// If I'm not in eligible list, then drop
packet
if (!isEligible) {
       Packet::free(p);
       return;
}
// If I'm eligible, then re-compute to
create new eligible list
else if (rq->rq_dst != index) {
/*
```

A.3. Kode fungsi recvRequest()

```
AODV::recvRequest(Packet *p) {
// struct hdr_cmn *ch = HDR_CMN(p);
struct hdr_ip *ih = HDR_IP(p);
struct hdr_aodv_request *rq =
HDR_AODV_REQUEST(p);
aodv_rt_entry *rt;
bool isEligible = true;
MobileNode *iNode;
iNode = (MobileNode *)
(Node::get_node_by_address(index));
iEnergy = iNode->energy_model()->energy();
//modif, buat panggil energi

  #ifdef DEBUG
    FILE *fp;
    fp = fopen("debug.txt", "a");
    fprintf(fp, "\n%.6f  %s: in node %d ,
packet from node %d to node destination %d
",  CURRENT_TIME, __FUNCTION__, index, rq-
>rq_src, rq->rq_dst);
    fclose(fp);
   #endif // DEBUG

/*
* Drop if:
*      - I'm the source
*      - I recently heard this request.
*      - I'm not in the list of eligible
nodes
*/

       if(rq->rq_src == index) {
       #ifdef DEBUG
```

```
        fprintf(stderr, "%s: got my own
REQUEST\n", __FUNCTION__);
        #endif // DEBUG
        Packet::free(p);
        return;
        }

        //modifikasi buat cari energi rata2
terbesar pd node tujuan
  if (id_lookup(rq->rq_src, rq->rq_bcast_id)
&& rq->rq_dst != index) {

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

        Packet::free(p);
        return;
        }

        // If list is null, then drop packet
        if(rq->rq_eligible_nodes == NULL) {
                Packet::free(p);
                return;
        }
        else if (rq->rq_dst != index) {
                isEligible = false;
                for (u_int32_t i = 0; i < rq-
>nodes_list_len; ++i) {
                        // I'm on the list, so
I'm eligible for this packet
                        if (rq-
>rq_eligible_nodes[i] == index) {
                                isEligible =
true;
```

```
                                //break;
        //modified
        //modif sinyal
        signal_strength = p->txinfo_.RxPr;
//RECIEVE SIGNAL
        double sinyal;
        if (signal_strength == 0) {
          sinyal = -200;
        }
        else {
          sinyal = 10*log10(signal_strength)
+ 30;
        }
        if (sinyal >= -75 && sinyal <= -30)
{
          rq->rq_energy_count = rq-
>rq_energy_count + iEnergy; //buat nambahin
energi
          double energy_avg = rq-
>rq_energy_count / rq->rq_hop_count;
          #ifdef DEBUG
          FILE *fp;
          fp = fopen("debug.txt", "a");
          fprintf(fp, "\n fungsi
recvrequest: node %d energy: %f, hc: %d, avg
energy: %f, sinyal: %f", index, rq-
>rq_energy_count, rq->rq_hop_count,
energy_avg, sinyal);
          fclose(fp);
          #endif
        }
        else {
          Packet::free(p); //buat drop paket
          #ifdef DEBUG
          FILE *fp;
          fp = fopen("debug.txt", "a");
```

```
            fprintf(fp, "\n fungsi
recvrequest: node %d didrop, sinyal: %f",
index, sinyal);
            fclose(fp);
            #endif
          }
                        }
                  }
        }

// If I'm not in eligible list, then drop
packet
if (!isEligible) {
        Packet::free(p);
        return;
}
// If I'm eligible, then re-compute to
create new eligible list
else if (rq->rq_dst != index) {
/*
* AODV-PNT re-compute TWR
*/

// TWR list
std::vector<TWRContainer> listTWR;
std::vector<nsaddr_t> eligibleAddrList;
nsaddr_t *eligibleNodes;

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

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

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

u_int32_t fSpeed = 15;
u_int32_t fAccel = 10;
u_int32_t fDistance = 20;
u_int32_t fQuality = 50;

u_int32_t timeModifier = 3;
u_int32_t maxTxRange = 250; // ns2 default
range (based on Pt_)
u_int32_t thresholdW = 100;

// Dst fixed position (STA dst). real
scenario: we can find exact loc via GPS
double xDst = 25.0;
double yDst = 25.0;

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

for(; nb; nb = nb->nb_link.le_next) {
// TWR Calculation
```

```
// Calculate distance between next-hop and
dst
double nb_distance;
nb_distance = sqrt(pow((nb->nb_x - xDst), 2)
+ pow((nb->nb_y - yDst), 2));

// radius between this node and neighbor
// minimum radius -> min(√(i x − j x )^2 +
(i y − j y )^2 ; r)
double radius = std::min(
sqrt(pow((nb->nb_x - posX), 2) + pow((nb-
>nb_y - posY), 2)), (double) maxTxRange);

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

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

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

// Calculate future TWR for next
[timeModifier] seconds
// Future speed v' = v + a × t
double nb_speedFuture = nb->nb_speed + (nb-
>nb_accel * timeModifier);

// Future position will be used to calc
future direction
// Formula: x' = x + v0 t × 0.5at^2
```

```
// Future neighbor position77
double nb_xFuture = nb->nb_x + (nb->nb_speed
* timeModifier) + (0.5 * nb->nb_accel *
timeModifier * timeModifier);
double nb_yFuture = nb->nb_y + (nb->nb_speed
* timeModifier) + (0.5 * nb->nb_accel *
timeModifier * timeModifier);

// Future this_node position
double iXFuture = posX + (iSpeed *
timeModifier) + (0.5 * iAccel * timeModifier
* timeModifier);
double iYFuture = posY + (iSpeed *
timeModifier) + (0.5 * iAccel * timeModifier
* timeModifier);

// Calculate future distance between next-
hop and dst
double futureDistance;
futureDistance = sqrt(pow((nb_xFuture -
xDst), 2) + pow((nb_yFuture - yDst), 2));

// Future radius between this node and
neighbor: both using future values
double futureRadius = std::min(
sqrt(pow((nb_xFuture - iXFuture), 2) +
pow((nb_yFuture - iYFuture), 2)), (double)
maxTxRange);

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

// Calculate future TWR
modSpeed = fSpeed * nb_speedFuture;
modAccel = fAccel * nb->nb_accel;
```

```
modDistance = fDistance * futureDistance;
modQuality = fQuality * futureQuality;
double futureTWR = modSpeed + modAccel +
modDistance + modQuality;

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

listTWR.push_back(container);
}

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

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

for (u_int32_t i = 0; i < listTWR.size();
i++) {
// TWR yang akan datang dikatakan "lebih
baik" apabila nilainya lebih
// rendah dari TWR sekarang. TWR yang akan
datang dikatakan "lebih
// buruk" apabila nilainya lebih besar dari
TWR sekarang.
if (listTWR[i].futureTWR < listTWR[i].TWR) {
listTWR[i].isFutureBetter = true;
```

```
}

// Jika ΔTWR lebih kecil dari threshold W,
maka node tersebut
// dianggap sebagai “stable”.
// Jika ΔTWR lebih besar dari threshold W,
maka node tersebut
// dianggap sebagai “unstable”.
double deltaTWR = fabs(listTWR[i].TWR -
listTWR[i].futureTWR);
if (deltaTWR < thresholdW) {
listTWR[i].isStable = true;
}

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

```
else if (!listTWR[i].isOptimal &&
listTWR[i].isStable &&
!listTWR[i].isFutureBetter) {
eligibleAddrList.push_back(listTWR[i].addres
s);
}
}

// replace rq_eligible_nodes with the re-
computed list
rq->nodes_list_len =
eligibleAddrList.size();
eligibleNodes = new nsaddr_t[rq-
>nodes_list_len];
for (u_int32_t i = 0; i < rq-
>nodes_list_len; i++) {
eligibleNodes[i] = eligibleAddrList[i];
}
rq->rq_eligible_nodes = eligibleNodes;
}

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

/*
* We are either going to forward the REQUEST
or generate a
* REPLY. Before we do anything, we make sure
that the REVERSE
* route is in the route table.
*/
aodv_rt_entry *rt0; // rt0 is the reverse
route
```

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

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

if ( (rq->rq_src_seqno > rt0->rt_seqno ) ||
((rq->rq_src_seqno == rt0->rt_seqno) && (rq-
>rq_hop_count < rt0->rt_hops)) ) {
// If we have a fresher seq no. or lesser
#hops for the
// same seq no., update the rt entry. Else
don't bother.
rt_update(rt0, rq->rq_src_seqno, rq-
>rq_hop_count, ih->saddr(), max(rt0-
>rt_expire, (CURRENT_TIME +
REV_ROUTE_LIFE)));
if (rt0->rt_req_timeout > 0.0) {
// Reset the soft state and
// Set expiry time to CURRENT_TIME +
ACTIVE_ROUTE_TIMEOUT
// This is because route is used in the
forward direction,80

// but only sources get benefited by this
change
rt0->rt_req_cnt = 0;
rt0->rt_req_timeout = 0.0;
rt0->rt_req_last_ttl = rq->rq_hop_count;
rt0->rt_expire = CURRENT_TIME +
ACTIVE_ROUTE_TIMEOUT;
}
```

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

/*
* We have taken care of the reverse route
stuff.
* Now see whether we can send a route reply.
*/

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

// First check if I am the destination ..

//modifikasi pemilihan RREP
if(rq->rq_dst == index) {
  //rrep pertama, main route
  counts++;
  double mytime;
```

```
  double energy_selected = rq-
>rq_energy_count / rq->rq_hop_count; //avg
energy = jml energi / hop count
  double diff; //beda waktu antara rreq ke-n
dengan rreq ke 1
  bool isHigher = false; //cek rrep buat
backup apakah lebih tinggi dari stored value
  // rreq pertama yg diterima node tujuan
  if (counts == 1) {
    last_rreq_on_dest =
Scheduler::instance().clock();
    mytime = Scheduler::instance().clock();
    energy_select = energy_selected;
    energi_terbesar = rq->rq_src;
    diff = mytime - last_rreq_on_dest;
    #ifdef DEBUG
    FILE *fp2;
    fp2 = fopen("debug.txt", "a");
    fprintf(fp2, "destination node menerima
RREQ pertama dari node %d, ttl energy: %f
avg energy: %f, hop count: %d, last rreq:
%f, diff: %f",
      rq->rq_src, rq->rq_energy_count,
energy_selected, rq->rq_hop_count,
last_rreq_on_dest, diff);
    fclose(fp);
    #endif
  }
  //rreq berikutnya
  else if (counts != 1) {
    mytime = Scheduler::instance().clock();
    double temp = energy_selected;
    //pemilihan energi, jika energi sekarang
lebih besar drpd sebelumnya, update
    if (temp > energy_select) {
      isHigher = true;
```

```
      energy_select = temp;
      energi_terbesar = rq->rq_src;
    }
    diff = mytime - last_rreq_on_dest;
    FILE *fp2;
    fp2 = fopen("debug.txt", "a");
    fprintf(fp2, "\n destination node
menerima RREQ ke-%d dari node %d, ttl
energy: %f avg energy: %f, hop count: %d,
last rreq: %f, diff: %f, energy select: %f",
      counts, rq->rq_src, rq-
>rq_energy_count, energy_selected, rq-
>rq_hop_count, last_rreq_on_dest, diff,
energy_select);
    fclose(fp);
  }
  //pengiriman route reply, lewat energi yg
terbesar
  if (rreq_already_sent == false && diff >=
0.002) {
    rreq_already_sent = true;
    #ifdef DEBUG
    fprintf(stderr, "%d - %s: destination
sending reply\n", index, __FUNCTION__);
    #endif // DEBUG

    // Just to be safe, I use the max.
Somebody may have
    // incremented the dst seqno.
    seqno = max(seqno, rq->rq_dst_seqno)+1;
    if (seqno%2) seqno++;

    sendReply(energi_terbesar, // IP
Destination
    1, // Hop Count
    index, // Dest IP Address
```

```
    seqno, // Dest Sequence Num
    MY_ROUTE_TIMEOUT, // Lifetime
    rq->rq_timestamp); // timestamp
    #ifdef DEBUG
      FILE *fp2;
      fp2 = fopen("debug.txt", "a");
      fprintf(fp2, "\n mengirim route reply
ke node %d, energi: %f", energi_terbesar,
energy_select);
      fclose(fp2);
    #endif
      Packet::free(p);
  }
  //rreq berikutnya buat klo ada error pas
route discovery
  if (isHigher == true && rreq_already_sent
== true) {
    seqno = max(seqno, rq->rq_dst_seqno)+1;
    if (seqno%2) seqno++;

    sendReply(energi_terbesar, // IP
Destination
    1, // Hop Count
    index, // Dest IP Address
    seqno, // Dest Sequence Num
    MY_ROUTE_TIMEOUT, // Lifetime
    rq->rq_timestamp); // timestamp
    #ifdef DEBUG
      FILE *fp2;
      fp2 = fopen("debug.txt", "a");
      fprintf(fp2, "\n mengirim route reply
ke node %d, energi: %f", energi_terbesar,
energy_select);
      fclose(fp2);
    #endif
       delete[] rq->rq_eligible_nodes;
```

```
    rq->rq_eligible_nodes = NULL;
      Packet::free(p);
  }
}

// I am not the destination, but I may have
a fresh enough route.

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

//assert (rt->rt_flags == RTF_UP);
assert(rq->rq_dst == rt->rt_dst);
//assert ((rt->rt_seqno%2) == 0); // is the
seqno even?
sendReply(rq->rq_src,
rt->rt_hops + 1,
rq->rq_dst,
rt->rt_seqno,
(u_int32_t) (rt->rt_expire - CURRENT_TIME),
// rt->rt_expire - CURRENT_TIME,
rq->rq_timestamp);
// Insert nexthops to RREQ source and RREQ
destination in the
// precursor lists of destination and source
respectively
rt->pc_insert(rt0->rt_nexthop); // nexthop
to RREQ source
rt0->pc_insert(rt->rt_nexthop); // nexthop
to RREQ destination

#ifdef RREQ_GRAT_RREP

sendReply(rq->rq_dst,
rq->rq_hop_count,
rq->rq_src,
```

```
rq->rq_src_seqno,
(u_int32_t) (rt->rt_expire - CURRENT_TIME),
// rt->rt_expire - CURRENT_TIME,
rq->rq_timestamp);
#endif

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

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

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

ih->saddr() = index;
ih->daddr() = IP_BROADCAST;
rq->rq_hop_count += 1;
// Maximum sequence number seen en route
if (rt) rq->rq_dst_seqno = max(rt->rt_seqno,
rq->rq_dst_seqno);
forward((aodv_rt_entry*) 0, p, DELAY);

}

}
```

A.4 Kode 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)      700;
set opt(y)      700;
set val(ifqlen) 1000;
set val(nn)     200;
set val(seed)   1.0;
set val(adhocRouting)   AODV;
set val(stop)   200;
set val(cp)     "cbr200.tcl";
set val(sc)     "scenarioreal.tcl";

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 ; #250m
Phy/WirelessPhy set  CSThresh_ 5.57189e-
11 ; #250m

#  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_(198) $udp_(0)
set null_(0) [new Agent/Null]
$ns_ attach-agent $node_(199) $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_ 1000
$cbr_(0) attach-agent $udp_(0)
$ns_ connect $udp_(0) $null_(0)
$ns_ at 0 "$cbr_(0) start"
$ns_ at 200.0000000000000000 "$cbr_(0)
stop"
```

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;
}
```

BIODATA PENULIS

Achmad Hanif Pradipta, lahir di Surabaya, 6 Februari 1999. Penulis adalah anak pertama dari dua bersaudara. Jenjang Pendidikan Penulis diawali dengan menyelesaikan pendidikan sekolah dasar di SD YAPENKA Cipete kemudian penulis melanjutkan pendidikan sekolah menengah pertama di SMPN 68 JAKARTA. Setelah SMP, penulis melanjutkan pendidikan menengah atas di SMA Negeri 6 Jakarta. Selepas SMA, penulis melanjutkan pendidikan sarjana di Departemen Teknik Informatika, Fakultas Teknologi Elektro dan Informatika Cerdas, Institut Teknologi Sepuluh Nopember Surabaya.

Dalam menyelesaikan pendidikan S1, penulis mengambil bidang minat Arsitektur dan Jaringan Komputer (AJK). Selain itu, Penulis menyelesaikan kerja praktik di Telekomunikasi Indonesia Kebon Sirih Jakarta Barat pada Juni – Agustus 2019 dan membuat dashboard sistem monitoring berbasis *web*.

Sebagai mahasiswa, penulis berperan aktif dalam berbagai organisasi di tingkat departemen dan beberapa organisasi dan kepanitiaan kampus seperti staf dan staf ahli departemen Hubungan Luar Himpunan Mahasiswa Teknik-Computer Informatika (HMTC) ITS.Selain itu, penulis juga menjadi staf dan staf ahli REEVA (Revolutionary Entertainment and Expo with Various Arts) SCHEMATICS.Penulis dapat dihubungi melalui surel: pradiptahanif@gmail.com.