Institut
Teknologi
Sepuluh Nopember

TUGAS AKHIR – IF184802

IMPLEMENTASI K-MEANS CLUSTERING DALAM ADHOC ON-DEMAND DISTANCE VECTOR DENGAN ANT COLONY OPTIMIZATION UNTUK MENINGKATKAN PERFORMA MOBILE NODE PADA WIRELESS ADHOC NETWORK

ALCREDO SIMANJUNTAK
NRP 05111640000045

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

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

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

(Halaman ini sengaja dikosongkan)

TUGAS AKHIR – IF184802

IMPLEMENTASI K-MEANS CLUSTERING DALAM ADHOC ON-DEMAND DISTANCE VECTOR DENGAN ANT COLONY OPTIMIZATION UNTUK MENINGKATKAN PERFORMA MOBILE NODE PADA WIRELESS ADHOC NETWORK

ALCREDO SIMANJUNTAK
NRP 05111640000045

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

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

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

(Halaman ini sengaja dikosongkan)

UNDERGRADUATE THESIS – IF184802

IMPLEMENTATION OF K-MEANS CLUSTERING IN ADHOC ON-DEMAND DISTANCE VECTOR WITH ANT COLONY OPTIMIZATION TO ENHANCE MOBILE NODE PERFORMANCE ON WIRELESS ADHOC NETWORK

ALCREDO SIMANJUNTAK
NRP 05111640000045

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

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

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 K-MEANS CLUSTERING DALAM ADHOC ON-DEMAND DISTANCE VECTOR DENGAN ANT COLONY OPTIMIZATION UNTUK MENINGKATKAN PERFORMA MOBILE NODE PADA WIRELESS ADHOC NETWORK

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:
ALCREDO SIMANJUNTAK
NRP: 05111640000045

Disetujui oleh Pembimbing Tugas Akhir:

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

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

SURABAYA
JANUARI, 2020

(Halaman ini sengaja dikosongkan)

IMPLEMENTASI K-MEANS CLUSTERING DALAM ADHOC ON-DEMAND DISTANCE VECTOR DENGAN ANT COLONY OPTIMIZATION UNTUK MENINGKATKAN PERFORMA MOBILE NODE PADA WIRELESS ADHOC NETWORK

Nama Mahasiswa : Alcredo Simanjuntak
NRP : 05111640000045
Departemen : Teknik Informatika
Dosen Pembimbing 1 : Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.
Dosen Pembimbing 2 : Prof. Ir. Supeno Djanali, M.Sc., Ph.D.

Abstrak

Adhoc On-Demand Distance Vector(AODV) adalah salah satu proses *routing* yang dapat diimplementasikan pada *mobile adhoc network*(MANET). AODV memiliki dua fase, yaitu *route discovery* dan *route maintenance*. Dimana *route discovery* akan dimulai ketika suatu *node* akan mengirimkan suatu paket, fase ini terdiri dari proses pengiriman *Route Request*(RREQ) dan *Route Reply*(RREP). *Route maintenance* digunakan untuk melakukan pemeliharaan *route* dengan mengetahui adanya masalah didalam suatu *route*, didalam fase ini terdapat proses pengiriman *Route Error*(RERR).

Pada AODV, *route* yang dipilih adalah *route* dengan jumlah hop terkecil tanpa memperhatikan faktor-faktor esensial lainnya yang mungkin bisa sangat mempengaruhi dalam pemilihan suatu *route*. Ada beberapa faktor lain yang bisa mempengaruhi pemilihan *route* selain jumlah hop antara lain seperti posisi, energi, kepadatan, dan kekuatan sinyal. Penggunaan faktor-faktor tambahan tersebut akan meningkatkan akurasi didalam pemilihan *route*.

Pada Tugas Akhir ini diusulkan suatu algoritma *routing* sebagai modifikasi dari AODV yaitu dinamakan KMeans-AODV-ACO. Algoritma ini akan membagi *node-node* kedalam beberapa *cluster* dan kemudian akan dilakukan pemilihan *cluster head*, dimana *route discovery* dilakukan hanya oleh *node* yang bertindak sebagai

cluster head. Didalam proses *route discovery* juga akan dilakukan perhitungan faktor-faktor lain yaitu Received *Signal Strength Metric*(RSSM), *Congestion Metric*(CM), *Residual Energy Metric*(REM), dan Hop-Count *Metric*(HCM) yang akan menjadi pertimbangan didalam pemilihan *route*.

Kata kunci: MANET, AODV, K-Means, ACO

IMPLEMENTATION OF K-MEANS CLUSTERING IN ADHOC ON-DEMAND DISTANCE VECTOR WITH ANT COLONY OPTIMIZATION TO ENHANCE MOBILE NODE PERFORMANCE ON WIRELESS ADHOC NETWORK

Student's Name	**: Alcredo Simanjuntak**
Student's ID	**: 05111640000045**
Department	**: Informatics Engineering**
First Advisor	**: Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.**
Second Advisor	**: Prof. Ir. Supeno Djanali, M.Sc., Ph.D.**

Abstract

Adhoc On-Demand Distance Vector is one of the routing process which can be implemented in mobile adhoc network(MANET). AODV has two phases, namely route discovery and route maintenance. Route discovery will started when a node will send a packet, this phase consist of sending of route request(RREQ) and route reply(RREP). Route maintenance used for doing route maintenance by knowing there is a problem within a route, in this phase, there is a process for sending route error(RRER).

In AODV, elected route is a route with the smallest number of hop without considering other essential factor which might be impactful in a route election. There are some other factors which might be impactful route election besides hop count such as energy, congestion, and signal strength. The using of that additional factors will enhance accuracy in route election.

Consequently, In this thesis, routing algorithm as a modification of AODV called K-Means-AODV-ACO is proposed. This algorithm will group nodes into some clusters and the cluster head selection will be made, where route discovery will be made by cluster head only. In this route discovery other factors will also be calculated namely Received Signal Strength Metric(RSSM), Congestion Metric(CM), Residual Energy Metric(REM), and Hop-

Count Metric(HCM) which will be the considerations in route election.

Keyword: MANET, AODV, K-Means, ACO

KATA PENGANTAR

Puji syukur kepada Allah Yang Maha Esa atas segala karunia dan rahmat-Nya penulis dapat menyelesaikan Tugas Akhir yang berjudul ***"Implementasi K-Means Clustering dalam Adhoc On-Demand Distance Vector dengan Ant Colony Optimization untuk Meningkatkan Performa Mobile Node pada Wireless Adhoc Network*".**

Harapan penulis semoga apa yang tertulis di dalam buku Tugas Akhir ini dapat bermanfaat bagi pengembangan ilmu pengetahuan saat ini.

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

1. Allah SWT atas semua rahmat yang diberikan sehingga penulis dapat menyelesaikan Tugas Akhir ini.
2. Keluarga penulis (Bapak Mahadin Simanjuntak, Ibu Junna Roida Naibaho, Daniel Tigor Sulyvandi Simanjuntak, dan George Hans Christian Simanjuntak) yang selalu memberikan dukungan baik berupa doa, moral, dan materi kepada penulis, sehingga penulis dapat menyelesaikan Tugas Akhir ini.
3. Bapak Dr. Eng. Radityo Anggoro, S.Kom., M.Sc. dan Bapak Prof. Ir. Supeno Djanali, M.Sc., Ph.D. selaku dosen pembimbing, atas arahan dan bantuannya dalam pengerjaan Tugas Akhir ini.
4. Angkatan 2016 Informatika ITS (TC16), yang sudah memberikan saya dukungan moral selama pengerjaan Tugas Akhir ini.
5. Admin laboratorium Arsitektur dan Jaringan Komputer (AJK) Informatika ITS yang sudah membantu dan menyediakan fasilitas selama penulis mengerjakan Tugas Akhir ini.

6. Angkatan 24 SMAN 2 Asrama Yayasan Soposurung Balige (INVICTUSION), yang sudah memberikan dukungan baik berupa doa dan moral dalam pengerjaan Tugas Akhir ini.
7. Serta semua pihak yang telah turut membantu penulis dalam menyelesaikan Tugas Akhir ini.

Penulis menyadari bahwa masih terdapat kekurangan, kesalahan, maupun kelalaian yang telah penulis lakukan. Oleh karena itu, saran dan kritik yang membangun sangat dibutuhkan untuk penyempurnaan Tugas Akhir ini.

Surabaya, Januari 2020

Alcredo Simanjuntak

DAFTAR ISI

(Halaman ini sengaja dikosongkan)

DAFTAR GAMBAR

DAFTAR TABEL

(Halaman ini sengaja dikosongkan)

BAB I PENDAHULUAN

1.1 Latar Belakang

Teknologi komunikasi nirkabel pada dewasa ini telah berkembang secara pesat, hal ini dapat dilihat dari keinginan pengguna sebagai *endpoint* untuk menggunakan alat komunikasi dimanapun dan kapanpun dikarenakan oleh konektivitas nirkabel yang memberikan kebebasan pergerakan kepada penggunanya.

Mobile adhoc network(MANET) merupakan salah satu jenis jaringan yang tidak memiliki infrastruktur yang tetap dan tidak memiliki *access point*. Keunggulan teknologi ini salah satunya adalah memungkinkan komunikasi dapat terjadi ketika terjadi bencana alam atau beberapa hal lain yang menyebabkan infrastruktur untuk komunikasi rusak sehingga tidak bisa digunakan untuk berkomunikasi. Tugas Akhir ini menggunakan sebuah *routing* yang termasuk dalam jenis *reactive routing* dimana proses *routing* akan dilakukan ketika akan mengirimkan sebuah paket. *Adhoc On-Demand Distance Vector*(AODV) adalah metode *routing* yang akan digunakan pada Tugas Akhir ini.

Beberapa observasi menunjukkan seringkali terjadi putusnya hubungan antar *node* yang menyebabkan paket tidak terkirim. Permasalahan ini timbul disebabkan beberapa faktor antara lain mobilitas *node*, penggunaan energi, media nirkabel, ataupun beberapa faktor lainnya. AODV didalam proses *route discovery* hanyalah memperhatikan jumlah hop ketika menentukan *route* yang akan dipilih, penggunaan faktor tunggal tersebut menyebabkan *route* tersebut kurang handal dan sering terjadi error didalamnya. Modifikasi yang signifikan terhadap *routing* protokol AODV sangat diperlukan sehingga proses *route discovery* menjadi lebih optimal yang merujuk pada ditemukannya *route* yang lebih optimal juga.

Pada Tugas Akhir ini akan diterapkan salah satu metode klasifikasi yakni *K-Means Clustering* untuk melakukan pemilihan *forwarding node* berdasarkan posisi sehingga node yang tidak terpilih tidak terlibat dalam proses *route discovery*, metode ini

diharapkan dapat mengatasi keterbatasan dari sisi *energy*. Selanjutnya akan dilakukan optimasi untuk memilih *route* terbaik dari sisi *signal strength, energy, hop count*, dan *congestion* dengan menerapkan metode *Ant Colony Optimization* yakni penggunaan nilai *pheromone* dimana route dengan nilai *pheromone* tertinggi akan digunakan untuk mengirimkan paket.

1.2 Rumusan Masalah

Tugas Akhir ini mengangkat beberapa rumusan masalah sebagai berikut:

1. Bagaimana melakukan pemilihan *forwarding node* dengan metode *K-Means Clustering* ?
2. Bagaimana melakukan optimasi pemilihan *route* terbaik berdasarkan *signal strength, energy, congestion, dan hop count* pada protokol *Adhoc On-Demand Distance Vector* dengan menggunakan metode *Ant Colony Optimization* ?

1.3 Batasan Permasalahan

Permasalahan yang dibahas pada Tugas Akhir inimemiliki 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. Pembuatan skenario uji coba menggunakan *Simulation of Urban Mobility* (SUMO).

1.4 Tujuan

Tujuan dari Tugas Akhir ini adalah sebagai berikut:

1. Mereduksi jumlah *forwarding node* yang bertanggung jawab untuk meneruskan pesan *Route Request* (RREQ) di lingkungan MANETs.

2. Mencari *route* terbaik dari *source node* menuju ke *destination node* berdasarkan beberapa faktor tambahan di lingkungan MANETs.

1.5 Manfaat

Manfaat yang diperoleh dari pengerjaan Tugas Akhir ini adalah mengetahui pengaruh dari pengurangan jumlah *forwarding node* dan penambahan faktor lain dalam memilih *route* terbaik di lingkungan MANETs.

1.6 Metodologi

Pembuatan Tugas Akhir ini dilakukan dengan menggunakan metodologi sebagai berikut:

1.6.1 Penyusunan Proposal Tugas Akhir

Tahapan awal dari Tugas Akhir ini adalah penyusunan Proposal Tugas Akhir. Proposal Tugas Akhir berisi pendahuluan, deskripsi dan gagasan metode-metode yang dibuat dalam Tugas Akhir ini. Pendahuluan ini terdiri atas hal yang menjadi latar belakang diajukannya Tugas Akhir, rumusan masalah yang diangkat, batasan masalah untuk Tugas Akhir, dan manfaat dari hasil pembuatan Tugas Akhir ini. Selain itu dijabarkan pula tinjauan pustaka yang digunakan sebagai referensi pendukung pembuatan Tugas Akhir. Terdapat pula sub bab jadwal kegiatan yang menjelaskan jadwal pengerjaan Tugas Akhir.

1.6.2 Studi Literatur

Pada tahap ini, dipelajari sejumlah referensi yang diperlukan dalam melakukan implementasi yaitu mengenai MANETs, AODV, *Network Simulator* 2 (NS-2), *OpenStreetMap*, *Java OpenStreetMap* (JOSM), *Simulation of Urban Mobility* (SUMO), dan AWK.

1.6.3 Analisis dan Desain Sistem

Pada tahap ini dilakukan analisis dari hasil percobaan modifikasi AODV yang dibuat. Data yang dianalis berasal dari penghitungan *Throughput, Packet Delivery Ratio* (PDR), rata-rata *End-to-End Delay* (E2E) dari paket yang dikirim oleh *node* satu ke *node* lainnya, *Routing Overhead*, dan *Forwarded Route Request*. Hal ini bertujuan untuk merumuskan solusi yang tepat untuk konfigurasi AODV yang dimodifikasi dalam lingkungan topologi MANET. Setelah selesai diaplikasikan pada MANET, dilakukan simulasi yang dilakukan pada MANETs dengan bantuan SUMO.

1.6.4 Implementasi Sistem

Implementasi merupakan tahap untuk membangun metodemetode yang telah diajukan pada proposal Tugas Akhir. Pada tahap ini dilakukan implementasi menggunakan NS-2 sebagai *simulator*, Bahasa C/C++ sebagai bahasa pemrograman, dan SUMO sebagai *tool* untuk uji coba dan mengimplentasikan desain sistem yang telah dirancang.

1.6.5 Pengujian dan Evaluasi

Pada tahap ini dilakukan pengujian menggunakan SUMO, sebuah *traffic generator* untuk membuat simulasi keadaan topologi yang diujikan. Hasil dari SUMO tersebut akan dijalankan pada NS-2 yang akan menghasilkan *trace file*. *Throughput, Packet Delivery Ratio* (PDR), rata-rata *End-to-End Delay* (E2E) dari paket yang dikirim oleh *node* satu ke *node* lainnya, *Routing Overhead*, dan *Forwarded Route Request* akan dihitung dari *trace file* tersebut untuk menguji performa AODV 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

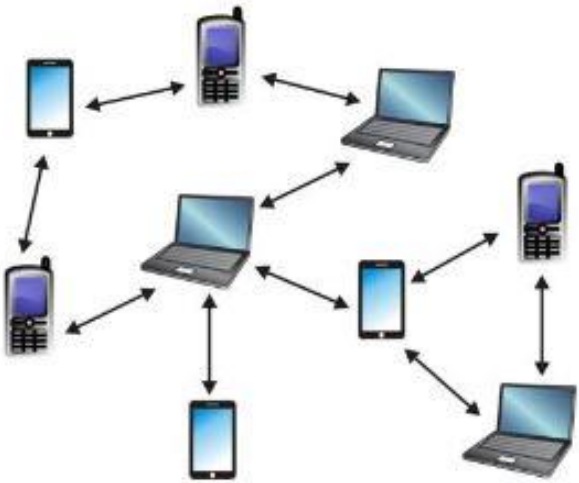
Bab ini berisi penjelasan mengenai teori-teori dasar yang berkaitan dengan pengimplementasian perangkat lunak dan penunjangnya. Penjelasan ini bertujuan untuk memberikan gambaran secara umum terhadap *routing protocol*, tools, serta definisi yang digunakan dalam pembuatan Tugas Akhir.

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

Ad-hoc On-demand Distance Vector(AODV) adalah salah satu *routing protocol* yang dapat digunakan pada adhoc *mobile* network. Algoritma *Bellman-Ford distance vector* merupakan dasar algoritma yang digunakan pada *routing protocol* ini, namun telah termodifikasi sehingga dapat berjalan pada lingkungan yang *mobile*. AODV termasuk didalam kategori *reactive protocol*, dimana *route* akan dibuat ketika dibutuhkan saja.

AODV merupakan *routing protocol* yang dinilai sangat sesuai jika digunakan di dalam MANETs, dimana *bandwidth* yang sangat

terbatas dapat teratasi dengan baik dengan menggunakan *routing protocol* AODV. Beberapa keunggulan *routing protocol* ini antara lain memastikan bahwa *node* yang tidak berada pada *route* aktif tidak menyimpan informasi tentang *route* tersebut dan penggunaan *sequence number* yang disisipkan pada setiap paket oleh setiap *node* yang mengirimkan paket tersebut yang bertujuan untuk menggantikan cached *route*s sekaligus memastikan bahwa *route*s yang ada pada *routing table* adalah *route*s yang terbaik berdasarkan *sequence number* dan *timestamps*.

Didalam prosesnya, AODV terbagi atas dua fase utama yaitu *Route Discovery* dan *Route Maintenance*. Pada setiap fase, *routing protocol* ini melibatkan beberapa jenis paket yang disebut *control message*s.

2.2.1 Control Messages

Control Messages adalah paket yang digunakan untuk mengatur proses *route discovery* dan *route maintenance*. Ada 4 jenis *control message*s antara lain *Route Request*(RREQ), *Route Reply*(RREP), *Route* Error(RERR), dan *Hello Message* [2].

2.2.1.1 *Route Request*(RREQ)

RREQ adalah jenis paket yang digunakan ketika *source node* ingin terhubung dengan *destination node*, dimana *source node* tersebut tidak memiliki informasi *route* untuk menuju *destination node*. *Source node* akan melakukan *broadcast* untuk RREQ terhadap *node* yang disekitarnya. RREQ memiliki beberapa *fields* seperti yang ditunjukan pada Tabel 2.1.

Source Address
Request ID
Source Sequence No
Destination Address
Destination Sequence No
Hop Count

Tabel 2.1 Format RREQ

Request ID akan bertambah setiap kali *source node* mengirimkan RREQ baru, sehingga dapat dipastikan bahwa setiap RREQ yang dikirim adalah unik dengan diidentifikasi dengan *source address* dan *request* id. *Node* yang menerima RREQ dengan identitas parameter yang sama akan membuang RREQ tersebut, sedangkan yang diterima, *hop count* dan *sequence number* RREQ tersebut akan dibandingkan dengan informasi yang ada di dalam *node* tersebut. Jika *node* tersebut tidak memiliki informasi tentang *destination node* ataupun memiliki informasi tentang *destination node* namun tidak yang terbaru maka *node* tersebut akan melakukan *rebroadcast* RREQ dengan menaikkan *hop count*, sedangkan jika *node* tersebut memiliki informasi *route* dengan *sequence number* lebih besar atau sama dengan *sequence number* pada RREQ yang diterima, RREP *message* akan dikirimkan menuju ke *source*.

2.2.1.2 *Route Reply*(RREP)

RREP adalah jenis paket yang digunakan ketika *node* tersebut merupakan *destination node* atau *node* tersebut memiliki informasi *route* menuju *destination node*, sehingga *node* tersebut akan melakukan *unicast* RREP menuju *source node*. RREP memiliki beberapa *fields* seperti yang ditunjukan pada Tabel 2.2.

Source Address
Destination Address
Destination Sequence No
Hop Count
Life Time

Tabel 2.2 Format RREP

Unicast dapat dilakukan karena setiap *node* yang melakukan *forwarding* RREQ menyimpan *route* untuk kembali ke *source node* atau disebut juga dengan proses *Reverse Path Setup*(RPS). Sedangkan dalam pengiriman RREP secara *unicast* setiap *node*

yang berada pada *route* tersebut menyimpan informasi darimana *node* tersebut mendapatkan paket RREP atau yang disebut juga dengan proses *Forward Path Setup*(FPS).

2.2.1.3 *Route* Error (RERR)

RERR adalah jenis paket yang digunakan untuk melakukan pengawasan terhadap beberapa *node* yang bertindak sebagai tetangga dari suatu *node*. Ketika suatu *route* tidak bisa digunakan karena adanya masalah pada suatu *node* di dalam *route* tersebut, maka RERR akan dikirimkan untuk memberikan informasi bahwa *route* tersebut telah *invalid*. Tujuan pengiriman RERR adalah untuk mencegah pengiriman paket kembali melalui *route* tersebut dan sehingga perlu dilakukan proses *route discovery* kembali.

2.2.1.4 *Hello Messages*

Setiap *node* dapat mengetahui tetangganya dengan mengirimkan local *broadcast* atau juga yang disebut dengan *Hello Messages*. *Node* tetangga adalah *node* yang diasumsikan dapat berkomunikasi secara langsung dengan suatu *node* tanpa melalui suatu intermediate *node*. *Hello Message* akan dikirimkan secara periodik, namun *forward* terhadap paket ini tidak dilakukan karena *broadcast* dilakukan dengan *Time to Live* (TTL) sama dengan 1. *Node* yang mendapatkan paket ini akan melakukan pembaharuan lifetime akan informasi *node* tetangga tersebut pada *routing table*. Penggunaan *Hello Messages* ini bertujuan agar *node* tersebut mengetahui bahwa *link* terhadap *node* tetangganya masih tetap ada.

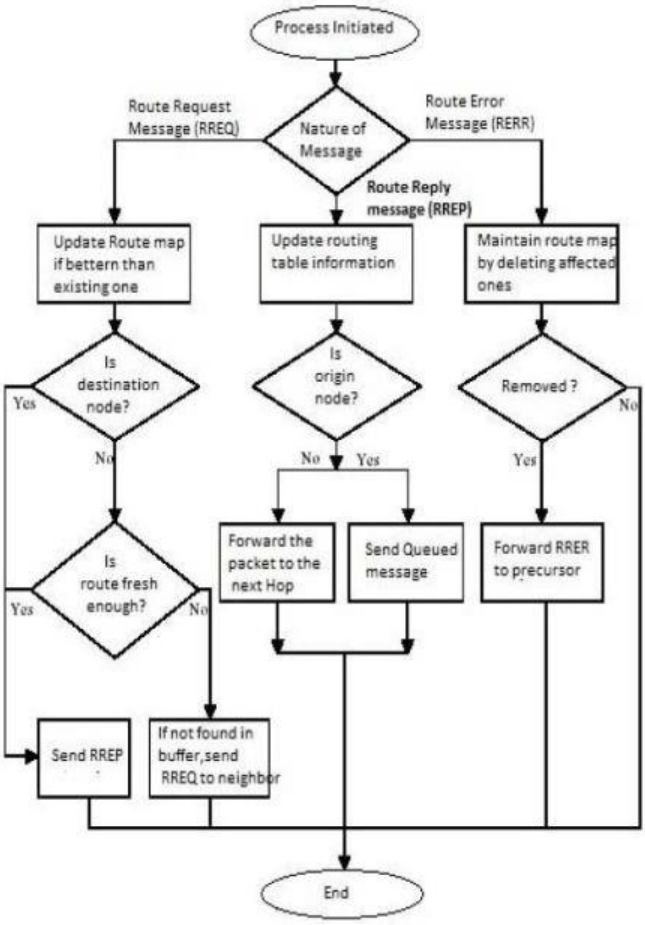
2.2.2 *Route Discovery*

Route discovery adalah proses pencarian suatu *route* menuju *destination* oleh *source node*. Dalam prosesnya *node* akan menggunakan RREQ *message* dan RREP *message* untuk mencari *route* yang diinginkan oleh *source node*. Ketika suatu *node* mengirimkan RREQ, *node* penerima mungkin saja mendapatkan RREQ yang sama kembali dari *node* tetangganya, sehingga untuk mencegah *infinite cycles* setiap *node* memiliki *buffer*, yang berisi daftar RREQ yang telah dilakukan *broadcast*. Sebelum RREQ

dikirim, *buffer* akan selalu diperiksa untuk memastikan RREQ yang sama tidak akan dikirimkan lagi.

Setiap intermediate *node* akan melakukan penambahan *sequence number* untuk memastikan bahwa *route* akan selalu diperbaharui dengan informasi terbaru. Pada setiap RREQ *message* akan dimasukan *source node sequence number* dan *destination node sequence number*. Gambar 2.1 akan menjelaskan bagaimana suatu *node* melakukan proses terhadap suatu paket *message* yang diterimanya.

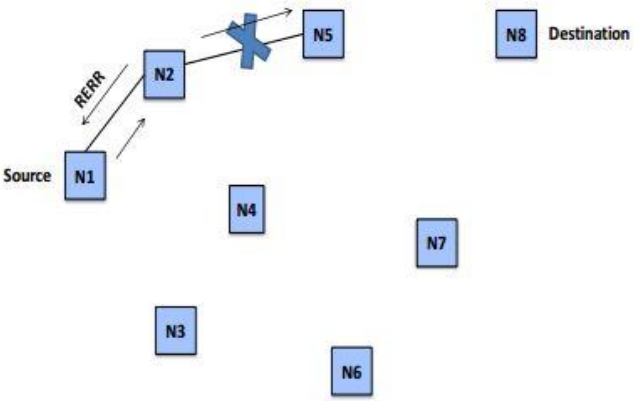
Proses *Route Discovery* akan hanya akan dilakukan ketika suatu *source node* akan mengirimkan sebuah paket menuju *destination node*, namun disisi lain tidak tersedianya informasi mengenai *route* tersebut menuju *destination node*. Suatu *route* akan ditemukan ketika RREP diterima oleh *source node* dari *destination node*. *Source node* mungkin saja menerima banyak RREP dari *destination node* dengan *route* yang berbeda-beda, *node* tersebut hanya akan melakukan pembaharuan pada *routing table* jikalau RREP mempunyai *sequence number* yang lebih besar dari yang ada pada *routing table source node.*



Gambar 2.2 Flowchart Pemrosesan Control Message

2.2.3 *Route Maintenance*

Route maintenance adalah sebuah proses yang bertujuan untuk mengecek keadaan *route* yang ada. Jika ada koneksi terputus antara 2 *node*, maka RRER dikirimkan kebeberapa *node* tujuan.



Gambar 2.3 Proses Pengiriman RERR

Pada Gambar 2.2 dapat dilihat ketika *link* daripada *node* N2 dan N5 tidak lagi tersambung, maka *node* N2 mengirimkan RERR secara *unicast* menuju *source node* yakni N1. Ketika sebuah *node* menerima RERR, *node* tersebut akan mengecek pengirim dari RERR tersebut untuk memastikan bahwa *node* pengirim adalah *next hop* dari salah satu daftar *node* yang ada pada RERR. Jika pengirim adalah *node* yang menjadi *next hop* dari salah satu daftar *node* pada RERR, maka *route* yang ada pada *routing table* akan dihapus dan RERR akan dikirimkan kembali menuju *source node*. Ketika *source node* mendapatkan RERR, *source node* akan memulai untuk melakukan proses *route discovery* kembali.

2.3 *K-Means Clustering*

K-Means Clustering merupakan teknik *clustering* yang bertujuan untuk melakukan partisi terhadap n objek kedalam k *cluster* berdasarkan jarak terdekat ke *centroid cluster* [3]. Berikut ini adalah langkah-langkah implementasi *K-Means Clustering*, yaitu :

- Langkah 1 : Penentuan k *centroid* awal secara acak
- Langkah 2 : Perhitungan *Euclidean Distance* dari masing-masing objek terhadap setiap *centroid* yang ada dan

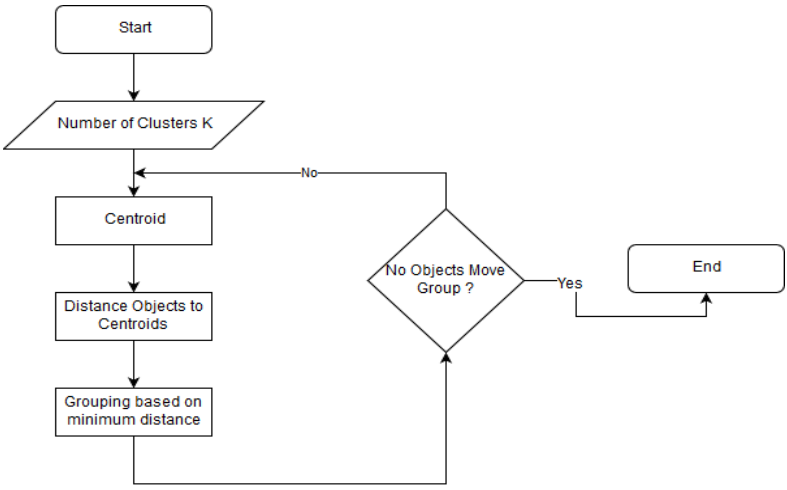
kemudian *cluster* setiap objek ditentukan berdasarkan jarak terdekat ke *centroid* yang ada.

$$\textit{Euclidean Distance} = \sqrt{(x_1 - x_2)^2 - (y_1 - y_2)^2}$$

Dimana x dan y adalah properti koordinat dari masing-masing objek yang ada.

- Langkah 3 : Dalam situasi setiap objek telah dibandingkan dengan situasi objek tersebut sebelumnya dan masing-masing *cluster* membentuk sebuah *centroid* kembali
- Langkah 4 : Jika posisi dari *centroid* berubah, maka kembali ke langkah 2 untuk mendapatkan *cluster* yang efisien. Jika posisi *centroid* tidak berubah maka proses *clustering* selesai. Posisi *centroid* terakhir dari setiap *cluster* akan digunakan untuk menentukan *Cluster Head*(CH). Proses pemilihan *cluster head* berdasarkan *node* yang terdekat pada *centroid* terakhir.

Gambar 2.2 akan menjelaskan alur bekerjanya proses *K-Means Clustering*.



Gambar 2.4 Flowchart *K-Means Clustering*

2.4 *Ant Colony Optimization* (ACO)

Ant Colony Optimization(ACO) merupakan bagian dari swarm intelligence dimana setiap metode didasari oleh tingkah laku beberapa hewan untuk melakukan proses simulasi. ACO adalah metode yang didasari oleh tingkah laku sekumpulan semut untuk mencari jalur terpendek antara sarangnya dan sumber makanan dengan menggunakan *pheromone*. Semut akan meninggalkan jejak dari *pheromone* sepanjang mereka bergerak dari awal hingga akhir dan akan bergerak melalui tempat-tempat yang memiliki intensitas *pheromone* yang tinggi untuk mencari sumber makanan. Pheromone dengan tingkat yang lebih tinggi diterima ketika sampai pada jalur-jalur terpendek lebih cepat, sehingga sekumpulan semut tersebut mendapat jalur terpendek menuju tujuannya.

2.5 *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*.

2.5.1 Instalasi

NS-2 membutuhkan beberapa *package* yang harus ada sebelum proses intalasi dilakukan. Dalam proses install dependency yang dibutuhkan dapat dilakukan dengan perintah yang ditunjukkan pada Gambar 2.4.

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

Gambar 2.5 Perintah untuk Menginstall Dependency NS-2

Selanjutnya, ekstrak *package* NS-2 dan lakukan perubahan pada baris kode ke-137 pada *file*s ls.h di folder *link*state menjadi seperti pada Gambar 2.5.

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

Gambar 2.6 Baris Kode yang Diubah Pada File ls.h

Proses instalasi dilakukan dengan menjalankan perintah ./install pada folder NS-2 yang telah diekstrak sebelumnya.

2.5.2 *Trace File*

Trace file merupakan *file* yang didapatkan setelah melakukan simulasi dengan menggunakan NS-2. File ini berisikan informasi tentang detail pengiriman setiap paket data. *Trace file* dapat digunakan untuk menganalisis performa *routing protocol* yang ingin disimulasikan. Detail informasi yang berada pada trace *file* ditunjukkan pada Tabel 2.3.

Kolom ke-	Penjelasan	Isi
1	Event	s : sent r : received f : *forward*ed D : dropped
2	Time	Waktu pada saat event terjadi
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 *head*er
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 *head*er ttl f : IP nexthop (jika 0 berarti *node* 0 atau *broadcast*)

Tabel 2.3 Detail Informasi *Trace* File AODV

2.6 OpenStreetMap(OSM)

OpenStreetMap(OSM) adalah sebuah proyek berbasis web untuk membuat peta dunia dan dapat diakses secara gratis. OSM dibangun oleh sukarelawan dengan melakukan survei menggunakan GPS dan mendigitalisasi citra satelit serta juga mengumpulkan data geografis yang tersedia di publik. Open Data Commons Commons Open Database License 1.0 adalah sarana dimana kontributor OSM dapat memiliki, memodifikasi, dan membagikan data geografis secara luas [4].

Pembuatan OSM dilakukan karena terdapat beragam jenis peta digital di internet , namun sebagian besar terbatas akibat aspek legal dan teknis. Hal ini menjadi dasar utama untuk pembentukan OSM. OSM ini dapat diunduh secara gratis dan kemudia dapat juga diredistribusikan kembali secara luas.

Di dalam OSM terdapat berbagai jenis atribut data yang bisa digunakan untuk berbagai macam kepentingan. Data tersebut dapat berupa jalan raya, stasiun, bandara, sekolah, kantor, dan berbagai jenis lainnya yang tersebar diseluruh dunia. Data tersebut secara bebas dapat digunakan untuk beberapa kepentingan berbeda.

2.7 Java OpenStreetMap(JOSM)

Java OpenStreetMap(JOSM) adalah aplikasi yang dapat digunakan untuk melakukan penyuntingan terhadap data yang didapatkan dari OpenStreetMap [5]. Penyuntingan ini dapat berupa merapikan data yang didapatkan dari OSM, melakukan filtering data, penghapusan objek yang tidak digunakan, dan beberapa fitur lainnya.

2.8 Simulation of Urban Mobilitiy(SUMO)

Simulation of Urban Mobility(SUMO) merupakan paket simulasi lalu lintas yang bersifat *open-source*. SUMO telah berkembang sehingga fitur-fitur didalamnya semakin lebih lengkap dapat dilihat dari kemampuannya dalam pemodelan dan jalannya jaringan untuk membaca format yang berbeda.

SUMO juga memungkinkan untuk mendefinisikan atribut tertentu pada kendaraan seperti panjang kendaraan, kecepatan

maksimum, percepatan dan perlambatannya. SUMO juga memberikan pilihan bagi pengguna untuk menentukan rute acak untuk kendaraan, disamping itu terdapat juga pilihan untuk transportasi umum dimana setiap kendaraan bergerak sesuai dengan jadwal tertentu.

SUMO terdiri dari beberapa tools yang dapat membantu pembuatan simulasi lalu lintas pada tahap-tahap yang berbeda. Penjelasan mengenai tools tersebut dapat dilihat pada Tabel 2.4.

Tool	Keterangan
netgenerate	Membuat peta seperti *grid*, spider, dan bahkan random network. Sebelum melakukan proses ini, pengguna dapat menentukan kecepatan maksimum dan membuat traffic light pada peta. Hasil dari proses ini akan menghasilkan *file* dengan ekstension .net.xml.
netconvert	Program CLI yang berfungsi untuk melakukan konversi dari peta seperti OpenStreetMap menjadi format native SUMO.
randomTrips.py	Membuat rute acak yang akan dilalui oleh kendaraan dalam simulasi
dua*route*r	Melakukan perhitungan rute berdasarkan definisi yang diberikan dan memperbaiki kerusakan rute.
sumo	Program yang melakukan simulasi lalu lintas berdasarkan data yang didapatkan dari netgenereate atau netconvert dari randomTrips.py. Hasil simulasi merupakan *file* hasil *export* untuk dikonversi menjadi format lain.
sumo-gui	GUI untuk melihat simulasi yang dilakukan oleh SUMO dengan grafis

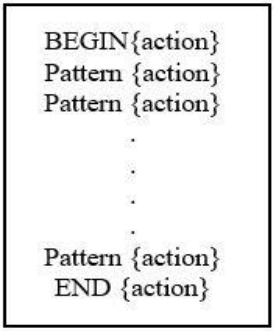
traceExporter.py	Melakukan konversi output dari sumo menjadi format yang dapat digunakan pada simulator lain.

Tabel 2.4 Tools pada Simulation of Urban Mobility(SUMO)

2.9 AWK

AWK adalah bahasa pemrograman yang digunakan untuk melakukan *text processing* dan ekstraksi data. AWK juga bisa digunakan untuk melakukan filtering terhadap suatu teks seperti halnya perintah grep pada terminal sistem operasi linux. Disamping itu, AWK juga dapat melakukan proses aritmatika seperti yang dilakukan oleh perintah expr.

Dalam penggunaanya AWK pattern dapat langsung diketikkan dalam terminal bersamaan dengan text sebagai paramater, selain itu AWK *script* dapat dibuatkan kedalam suatu *file* dengan beberapa pattern tertentu. Berikut ini adalah struktur dari *file* AWK ditunjukkan pada Gambar 2.3.



Gambar 2.7 Struktur File AWK

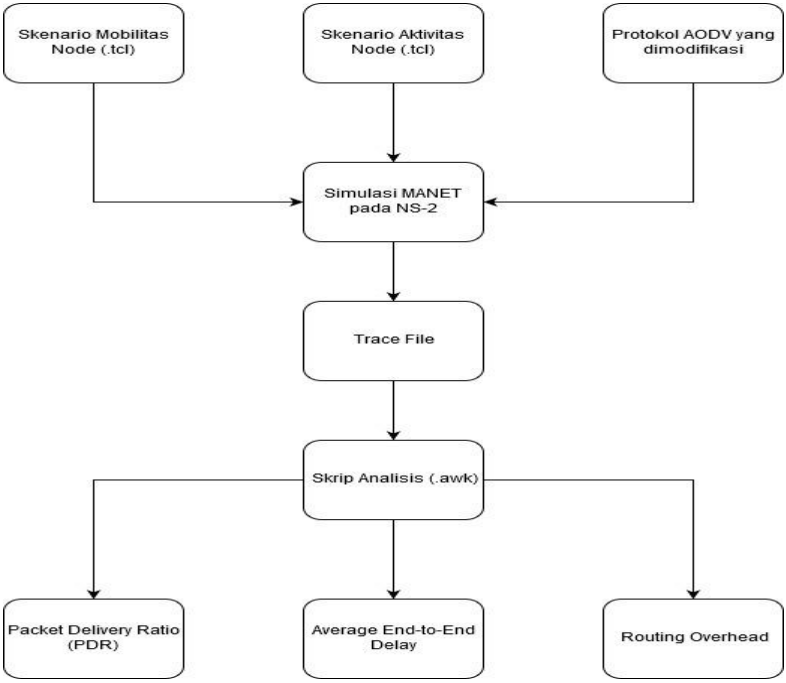
(Halaman ini sengaja dikosongkan)

BAB III PERANCANGAN

Perancangan merupakan bagian yang penting dari implementasi sistem sehingga bab ini secara khusus menjelaskan perancangan sistem yang dibuat pada Tugas Akhir. Bagian yang dijelaskan pada bab ini berawal dari deskripsi umum hingga perancangan implementasi.

3.1 Deskripsi Umum

Pada Tugas Akhir ini akan diimplementasikan sebuah *routing protocol* yakni AODV dengan modifikasi pada proses *route discovery* dan kemudian disimulasikan dengan menggunakan simulator NS-2. Proses simulasi AODV modifikasi dapat dilihat pada diagram yang ditunjukkan pada Gambar 3.1.



Gambar 3.1 Diagram perencanaan simulasi AODV modifikasi

Secara umum alur simulasi antara AODV asli dan AODV modifikasi tidak ada perbedaan. Berikut ini adalah beberapa istilah yang sering digunakan pada Tugas Akhir ini seperti pada Tabel 3.1.

No	Istilah	Penjelasan
1	AODV	Adhoc On-demand Distance Vector. Protocol yang akan digunakan pada Tugas Akhir ini.
2	PDR	Packet Delivery Ratio. Rasio jumlah pengiriman paket yang terkirim
3	E2E	Average End-to-End Delay. Jeda waktu rata-rata yang diukur saat paket terkirim hingga sampai pada tujuan
4	RO	*Routing* Over*head*. Jumlah *control message* yang terkirim.
5	RREQ	*Route Request*. Paket *request* pada AODV yang dikirim untuk mencari *route*
6	RREP	*Route Reply*. Paket reply pada AODV yang dikirim ke *node* sumber melalui *route* yang sudah dibuat.
7	RERR	*Route* Error. Paket error pada AODV yang dikirim untuk memberitahukan *node* pada suatu *route* bahwa *route* tersebut tidak bisa digunakan lagi.
8	ACO	*Ant Colony Optimization*. Bagian dari swarm intelligence yaitu suatu algoritma yang didasari pada tingkah laku semut untuk mencari makanan yakni menggunakan *pheromone*.
9	RSSM	*Residual Signal* Strengh *Metric*. Tingkat kekuatan signal dari paket yang diterima oleh suatu *node*
10	REM	*Residual Energy Metric*. Tingkat energy pada suatu *node*

11	HCM	*Hop Count Metric*. Jumlah *hop count* suatu paket hingga sampai pada suatu *node*
12	CM	*Congestion Metric*. Jumlah row pada *routing table* suatu *node*
13	PC	Pheromone Count. Tingkat *pheromone* value yang didapatkan dari beberapa parameter dengan rumus $\frac{RSSM\ x\ REM}{HCM\ x\ CM}$.

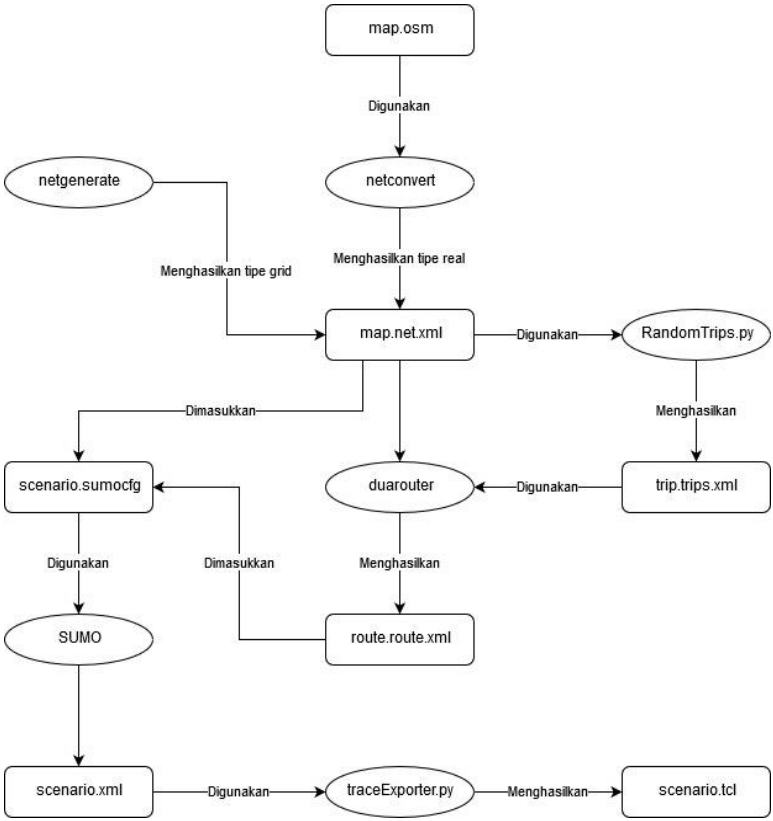
Tabel 3.1 Daftar Istilah

Modifikasi yang diberikan pada AODV di Tugas Akhir ini terbagi atas 2 bagian dimana yang pertama dilakukan *clustering* menggunakan metode *K-Means Clustering*, proses *clustering* ini dilakukan diawal simulasi pada detik t, sehingga data yang digunakan untuk melakukan *clustering* harus diperoleh sebelum detik t. Dalam melakukan *clustering* ini posisi *node* secara dua dimensi adalah parameter utama untuk melakukan *clustering* ini. Hasil dari proses *clustering* ini akan menghasilkan n *cluster* dengan *cluster head* dan *cluster gateway* untuk setiap *cluster*. Proses ini akan menghasilkan sebuah rule bahwa hanya *node* yang bertindak sebagai *cluster head* atau *cluster gateway* saja yang bisa melakukan pengiriman RREQ. Modifikasi bagian yang kedua yaitu menerapkan ACO pada proses *route discovery*. Dalam proses ini akan dilakukan perhitungan PC untuk setiap paket yang tiba pada suatu *node*. PC akan dihitung menggunakan sebuah rumus dengan beberapa parameter. Pada saat paket sampai pada *destination node*, nilai PC akan dikirim kembali menuju *source node* dengan disisipkan pada paket reply. Proses perbandingan nilai PC yang ada pada *destination node* dengan nilai PC paket yang tiba bertujuan untuk mencari *route* yang lebih baik dengan nilai PC yang lebih besar.

3.2 Perancangan Skenario Mobilitas

Tugas Akhir ini akan disimulasikan dengan menggunakan 2 skenario yakni skenario *grid* dan skenario *real*. Skenario *grid* akan

menggunakan peta *grid* dimana akan dibentuk jalan dengan beberapa petak yang saling berhimpitan. Peta *grid* ini akan digunakan sebagai simulasi awal MANETs karena lebih stabil. Peta *grid* ini dibentuk dengan menentukan panjang petak dan jumlah petak area menggunakan SUMO. Skenario *real* menggunakan peta *real* yang akan difungsikan sebagai area simulasi. Peta *real* diperoleh dengan mengambil suatu daerah yang akan digunakan sebagai area simulasi menggunakan OpenStreetmap. Gambar 3.2 menggambarkan secara umum alur simulasi pada Tugas Akhir ini.



Gambar 3.2 Alur Perancangan Skenario

3.2.1 Perancangan Skenario Grid

Perancangan skenario mobilitas *grid* diawali dengan merancang luas area peta *grid* yang dibutuhkan. Luas area tersebutbisa 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 300 m x 300 m. Dengan 4 titik persimpangan, maka akan didapatkan 9 petak dan panjang tiap peta adalah 300 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 dua*route*r. Skenario mobilitas *grid* dihasilkan dengan menggabungkan *file* peta *grid* dan *file* pergerakan *node* yang telah dibuat. Penggabungan tersebut menghasilkan *file* dengan ekstensi .xml. Selanjutnya, untuk dapat diterapkan pada NS-2, *file* skenario mobilitas *grid* yang berekstensi .xml dikonversi ke dalam bentuk *file* .tcl. Konversi ini dilakukan menggunakan tool traceExporter.

3.2.2 Perancangan Skenario Real

Perancangan skenario mobilitas *real* diawali dengan memilih area yang akan dijadikan simulasi. Pada Tugas Akhir ini, digunakan peta dari OpenStreetMap untuk mengambil area yang dijadikan model simulasi. Setelah memilih area, dilakukan pengunduhan dengan menggunakan fitur *export* yang telah disediakan oleh OpenStreetMap. Peta hasil *export* tersebut memiliki ekstensi .osm. Setelah mendapatkan peta area yang akan dijadikan simulasi, peta tersebut dikonversi ke dalam bentuk *file* dengan ekstensi .net.xml menggunakan tools SUMO yaitu

netconvert. Tahap berikutnya memiliki tahapan yang sama seperti merancang skenario *grid*, yaitu membuat pergerakan *node* menggunakan randomTrips dan dua*route*r. 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 berekstensi .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

AODV sebagai *routing protocol* yang digunakan dalam Tugas Akhir ini akan dimodifikasi dengan penambahan *K-Means Clustering* dan ACO. Penambahan metode-metode tersebut akan mengakibatkan adanya modifikasi yang signifikan terhadap AODV asli. Pada AODV asli akan dilakukan modifikasi, yaitu seperti pada *control message*, proses pengiriman *control message*, proses penerimaan *control message*, dan struktur class yang ada pada *routing protocol* ini.

3.3.1 Perancangan *K-Means Clustering*

Proses penerapan *K-Means Clustering* dimulai dengan melakukan pengumpulan data yang diperlukan untuk melakukan *clustering*. Data yang diperlukan untuk melakukan *clustering* adalah posisi dua dimensi dari seluruh *node* yang berada di daerah simulasi. Penerimaan setiap paket RREQ dan *Hello Messages* akan diikuti dengan proses pengambilan data posisi dari *nodes positions table* yang telah disisipkan pada setiap paket tersebut. Data tersebut akan ditampung ke dalam *nodes positions table* yang ada pada *node*. Tabel 3.2 menjelaskan variabel yang diperlukan didalam *nodes positions table*.

No	Variabel	Tipe	Keterangan
1	*nodes*_position_x	double, array	posisi absis *node*
2	*nodes*_position_y	double, array	posisi ordinat *node*

3	*nodes_timestamp*	double, array	waktu data diperoleh

Tabel 3.2 *Node*s Positions Table

*Node*s *positions table* merupakan tabel yang berisi data posisi dua dimensi seluruh *node* yang berada pada daerah simulasi. Proses pemindahan data melalui suatu paket yang dikirimkan terhadap suatu *node* akan memperhatikan *timestamp* untuk dapat memastikan bahwa data yang dimiliki oleh suatu *node* adalah data yang paling baru. Dalam setiap pemrosesan paket, data posisi *node* yang ada pada paket tersebut perlu juga dipastikan adalah data yang terbaru, karena paket tersebut mungkin saja akan dikirimkan menuju *node* yang lainnya, oleh karena itu paket tersebut harus juga melakukan pemindahan data dari suatu *node* ke dalam paket tersebut dengan memperhatikan *timestamp*. Proses pemindahan data ini dapat dilihat pada *pseudocode* yang ada pada Gambar 3.3.

```
for i=0 to number_of_nodes do
  if (paket. positions_table.timestamp[i] > node. positions_table.timestamp[i])
    then
      node. positions_table.x[i] = paket. positions_table.x[i]
      node. positions_table.y[i] = paket. positions_table.y[i]
  else if (paket. positions_table.timestamp[i] < node. positions_table.timestamp[i])
    then
      paket. positions_table.x[i] = node. positions_table.x[i]
      paket. positions_table.y[i] = node. positions_table.y[i]
  endif
endfor
```

Gambar 3.3 Pseudocode Pemindahan Data Posisi *Node*

Proses pengumpulan data ini dilakukan sebelum detik ke t, dimana pada detik ke t akan dilakukan proses *clustering*. Proses *clustering* ini dilakukan oleh sebuah *node* yang dinamakan server *node*. Server *node* ini akan melakukan *clustering* dengan menggunakan data yaitu *nodes positions table* yang terdapat pada *node* tersebut. Proses *K-Means Clustering* diawali dengan melakukan generate *cluster* initial *centroid* untuk setiap *cluster* secara random sebanyak k *cluster*, dan selanjutnya akan dilakukan

proses *K-Means Clustering*. Proses *K-Means Clustering* dapat dilihat pada *pseudocode* yang ada pada Gambar 3.4.

```
isClusterChange = true
while isClusterChange=true do
for i=0 to number_of_cluster do
  isClusterChange = false
  for j=0 to number_of_members_on_cluster[i] do
    for k=0 to number_of_cluster do
      distance = euclidean_distance(cluster[i].member[j],cluster[k])
      if (distance < cluster[i].member[j].nearestDistance)
        then
          cluster[i].member[j].setCluster(k)
          isClusterChange = true
      endif
    endfor
  endfor
endfor
update_centroid_all_clusters()
endwhile
```

Gambar 3.4 Pseudocode Proses *K-Means Clustering*

Hasil dari proses *K-Means Clustering* akan menghasilkan beberapa informasi antara lain *cluster* group, *cluster head*, dan *cluster gateway*. Proses pemilihan *cluster head* dan *cluster gateway* dilakukan untuk setiap *cluster*. Pemilihan *cluster head* dilakukan secara internal yakni pemilihan *node* yang terdekat dari *centroid cluster* tersebut sehingga setiap *node* akan dicari jaraknya menuju *centroid cluster* tersebut dengan metode euclidean distance. Proses pemilihan *cluster head* untuk setiap *cluster* dapat dilihat pada *pseudocode* yang ada pada Gambar 3.5. Pemilihan *cluster gateway* diawali dengan mencari titik tengah dari seluruh *node* yang berada pada daerah simulasi dan kemudian akan dicari *node* terdekat terhadap titik tengah tersebut untuk setiap *cluster*. Proses pemilihan *cluster gateway* dapat dilihat pada *pseudocode* pada Gambar 3.6.

```
cluster_centroid = getCentroid_ThisCluster ()
nearestDistance = 1000000.00
iter = 0
for i=0 to number_of_members do
  distance = euclidean_distance(member[i],cluster_centroid)
```

```
  if (distance<nearestDistance)
    then
      nearestDistance = distance
      memberId = i
  endif
endfor
return member[memberId]
```

Gambar 3.5 Pseudocode Proses Pemilihan *Cluster Head*

```
centroid = getCentroid_All Nodes()
nearestDistance = 1000000.00
iter = 0
for i=0 to number_of_members do
  distance = euclidean_distance(member[i],centroid)
  if (distance<nearestDistance)
    then
      nearestDistance = distance
      memberId = i
  endif
endfor
return member[memberId]
```

Gambar 3.6 Pseudocode Proses Pemilihan *Cluster Gateway*

Seluruh Informasi yang didapatkan setelah proses *K-Means Clustering* dilakukan akan dimasukkan ke dalam *nodes clusters table*. Server *node* akan menyebarkan informasi hasil *clustering* ke seluruh *node* yang berada di daerah simulasi. Penerimaan setiap paket RREP dan *Hello Messages* akan diikuti dengan melakukan proses pemindahan informasi hasil *clustering* melalui *nodes clustering table* yang telah disisipkan pada paket tersebut. Tabel 3.3 menjelaskan variabel yang diperlukan didalam *nodes clusters table*.

No	Variabel	Tipe	Keterangan
1	*nodes_cluster*_group	int, array	*cluster node*
2	*nodes_cluster_head*	int, array	*cluster head*
3	*nodes_cluster_gateway*	int, array	*cluster gateway*
4	*nodes_cluster_timestamp*	double	waktu proses *clustering*

Tabel 3.3 *Node*s Clusters Table

*Node*s *clusters table* ini berisi informasi yakni *cluster* group, *cluster head*, dan *cluster gateway*. Proses pemindahan informasi ini memperhatikan *timestamp*, hal ini akan berpengaruh jika dilakukan lebih dari sekali proses *clustering* dalam satu simulasi, sehingga informasi lama akan digantikan dengan informasi yang baru. Proses pemindahan informasi ini dapat dilihat pada *pseudocode* yang terdapat pada Gambar 3.7.

```
if (paket.cluster_table.timestamps > node.cluster_table.timestamps)
 then
  node.cluster_table = paket.cluster_table
  for i=0 to number_of_clusters do
   if (node.cluster_table. head[i] == index || node.cluster_table.gateway[i] == index)
    then
     node.isClusterHead = true
     break
   else
    then
     node.isClusterHead = false
   endif
  endfor
endif
```

Gambar 3.7 Pseudocode Pemindahan Informasi Hasil Clustering

3.3.2 Perancangan *Ant Colony Optimization* (ACO)

Proses penerapan ACO akan diterapkan pada bagian *route discovery* dengan menggunakan 4 parameter yakni RSSM, REM, CM, dan HCM. PC adalah hal yang paling utama pada penerapan ACO karena *route* terbaik didasarkan pada nilai PC. PC akan dihitung dengan menggunakan rumus berikut :

$$PC = \frac{RSSM \; x \; REM}{CM \; x \; HCM}$$

Keterangan

RSSM	=	signal strength dari suatu paket
REM	=	sisa energy yang dimiliki suatu *node*
CM	=	jumlah baris *routing table* pada current *node*

HCM = jumlah *hop count*

RSSM adalah signal strength dari paket yang dikirimkan oleh *node* sebelumnya. RSSM memiliki threshold value yakni -75 dBm. REM adalah energy yang dimiliki oleh suatu *node*. REM memiliki threshold value yakni 10 Joule. CM adalah tingkat kepadatan dari suatu *node* yang dihitung berdasarkan banyaknya baris pada *routing table*. HCM adalah banyak *hop count* suatu paket tiba di *node* tertentu.

Proses diawali dengan *source node* mengirimkan *broadcast* yakni paket *request* dengan menyisipkan variable PC yang diinisiasi sama dengan 0. Paket *request* yang diterima akan diproses untuk menghitung PC sebelum akan dilakukan *forward* ataupun *broadcast* kembali. Proses perhitungan PC dapat dilihat *pseudocode* yang terdapat pada Gambar 3.8.

```
pheromoneCount = ((RSSM+91)*REM)/(HCM*CM)
if (paket->pheromoneCount != NULL)
  then
    lastPheromoneCount = paket->pheromoneCount + pheromoneCount
else
  then
    lastPheromoneCount = pheromoneCount
endif
paket->pheromoneCount = lastPheromoneCount
```

Gambar 3.8 Pseudocode Perhitungan PheromoneCount

Setiap *node* menyimpan variable lastPheromoneCount yakni hasil perhitungan PC terakhir yang dihitung pada *node* tersebut. Proses perhitungan ini akan terus berlanjut hingga paket tiba pada *destination node*. Pada *destination node* akan terjadi perbandingan antara PC yang ada pada *routing table* dengan variable lastPheromoneCount, sehingga PC yang terbesar akan diterima oleh *source node.*

3.4 Perancangan Simulasi pada NS-2

Simulasi pada NS-2 dilakukan dengan menggabungkan *file* skenario dan *script* tcl yang berisikan konfigurasi lingkungan

simulasi. Konfigurasi lingkungan simulasi pada NS-2 dapat dilihat pada Tabel 3.4.

No	Parameter	Spesifikasi
1	Network Simulator Tool	NS2
2	*Routing* Protocol	AODV
3	Simulation Time	200 s
4	*Number* of *Node*s	100, 150, 200, 250, and 300
5	Simulation Area	950 m x 950 m
6	*Number* of Cluster	10, 15, 20
7	*Number* of Clustering	1
8	Antenna Model	Omni Antenna
9	MAC Type	MAC/802_11
10	Network Interfaces Types	Wireless
11	Transmission Range	400 m
12	*Source*/*Destination Node*	Static
13	Initial *Node Energy*	150 Joule

Tabel 3.4 Konfigurasi Lingkungan Simulasi

3.5 Perancangan Metrik Analisis

Perancangan metrik analisisi akan didasarkan pada beberapa parameter yang akan dianalisis pada Tugas Akhir ini untuk dapat membandingkan performa dari AODV asli dengan AODV modifikasi.

3.5.1 Packet Delivery Ratio (PDR)

Packet delivery ratio merupakan perbandingan antara jumlah paket data yang dikirim dengan paket data yang diterima. AODV modifikasi dapat disimpulkan lebih baik dari AODV asli jika nilai PDR dari AODV modifikasi lebih tinggi dari AODV asli. PDR dapat dihitung dengan menggunakan rumus sebagai berikut :

$$PDR = \frac{packet\ received}{packet\ sent}\ x\ 100\%$$

Keterangan :

packet received	=	jumlah paket data yang diterima
packet sent	=	jumlah paket data yang dikirim

3.5.2 Average End-to-End Delay (E2E)

Average End-to-End Delay adalah waktu rata-rata delay antara waktu paket dikirimkan hingga paket tiba pada tujuan. AODV modifikasi dapat disimpulkan lebih baik dari AODV asli jika nilai E2E dari AODV modifikasi lebih rendah dari AODV asli. E2E dapat dihitung dengan menggunakan rumus sebagai berikut :

$$E2E = \frac{\sum_{n=1}^{recvnum} CBR(RecvTime) - CBR(SentTime)}{recvnum}$$

Keterangan :

CBR(RecvTime)	=	waktu *node* tujuan menerima paket
CBR(SentTime)	=	waktu *node* asal mengirim paket
recvnum	=	jumlah paket yang berhasil diterima

3.5.3 *Routing* Over*head* (RO)

Routing Over*head* adalah jumlah *control message* yang ditransmisikan per paket menuju *node* tujuan selama simulasi terjadi. RO ini melibatkan semua jenis *control message* antara lain RREQ, RREP, dan RERR. AODV modifikasi dapat disimpulkan lebih baik dari AODV asli jika nilai RO dari AODV modifikasi lebih rendah dari AODV asli. RO dapat dihitung dengan menggunakan rumus sebagai berikut :

$$RO = \sum_{n=1}^{sentnum} packet\ sent$$

Keterangan :

packet sent	=	jumlah paket yang dikirim
sentnum	=	jumlah paket yang berhasil dikirim

3.5.4 Average *Hop Count*s (HC)

Hop counts adalah jumlah rata-rata hop hount untuk setiap paket menuju *node* tujuan. AODV modifikasi dapat disimpulkan lebih baik dari AODV asli jika HC pada AODV modifikasi lebih kecil dari

AODV asli. HC dapat dihitung dengan menggunakan rumus sebagai berikut :

$$HC = \frac{\sum_{n=1}^{recvnum} hop\ count}{recvnum}$$

Keterangan :

hop count	=	jumlah *hop count* pada suatu paket
recvnum	=	jumlah paket yang berhasil diterima

BAB IV IMPLEMENTASI

Pada bab ini akan diberikan pembahasan mengenai implementasi dari perancangan sistem yang sudah dijelaskan pada bab sebelumnya. Implementasi berupa cara pembuatan skenario mobilitas, modifikasi pada *routing protocol* AODV, dan *script* pengujian metrik analisis.

4.1 Implementasi Skenario *Grid*

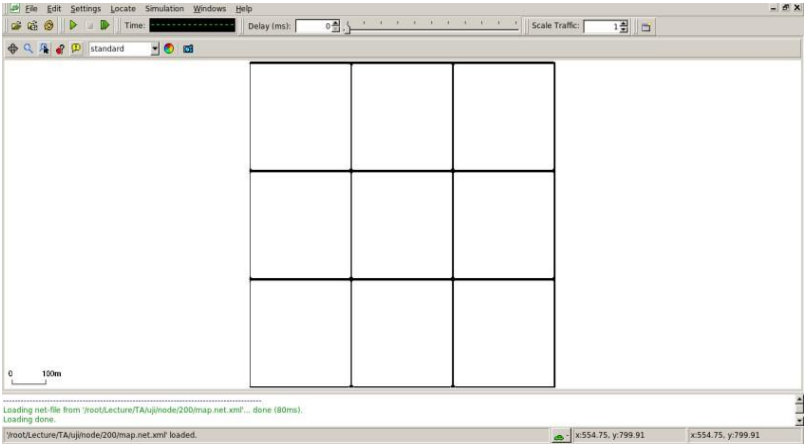
Pada Tugas Akhir ini, penulis menggunakan *tool* netgenerate dari SUMO untuk mengimplementasikan skenario *grid* berupa peta jalan berpetak. Peta skenario *grid* dibuat dengan panjang jalan 300 meter dan luas peta 900 m x 900 m. Dengan melebihkan 50 meter untuk menghindari terjadinya kesalahan pada saat simulasi. Jumlah titik persimpangan antara jalan vertikal dan jalan horisontal sebanyak 4 titik x 4 titik. Dengan jumlah titik persimpangan sebanyak 4 titik tersebut, maka terbentuk 9 buah petak. Kecepatan *node* (kendaraan) secara *default* diatur sebesar 10 m/s.

Perintah netgenerate untuk membuat peta *grid* dengan spesifikasi tersebut dapat dilihat pada Gambar 4.1.

```
netgenerate --grid --grid.number=4 -grid.length=300 --default.speed=10 -tls.guess=1
--output-file=map.net.xml
```

Gambar 4.1 Perintah netgenerate

Setelah itu, akan dihasilkan *file* peta berekstensi .xml. Gambar 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 asal dan titik tujuan setiap *node* secara random menggunakan *tool* randomTrips.py yang terdapat pada SUMO. Perintah penggunaan *tool* randomTrips.py untuk membuat *node* sebanyak n *node* dengan pergerakannya dapat dilihat pada Gambar 4.3. Opsi -e diisi dengan jumlah kendaraan yang diinginkan saat simulasi.

```
python $SUMO_HOME/tools/randomTrips.py -n map.net.xml -e 188 -l --
tripattributes="departLane=\"best\" departSpeed=\"max\"
departPos=\"random_free\"" -o trip.trips.xml
```

Gambar 4.3 Perintah randomTrips.py

Setelah titik asal dan titik akhir didefinisikan, selanjutnya dilakukan pembuatan rute yang akan digunakan oleh kendaraan menggunakan *tool* dua*route*r. Perintah penggunaan *tool* dua*route*r 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 dua*route*r

Ketika menggunakan *tool* dua*route*r, SUMO memastikan bahwa jalur untuk *node-node* yang di*generate* tidak akan melenceng dari jalur peta yang sudah di*generate* menggunakan *tool* randomTrips.py. Selanjutnya untuk menjadikan peta dan pergerakan *node* yang telah di*generate* menjadi sebuah skenario dalam bentuk *file* berekstensi .net.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="route.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 berekstensi .net.xml dan *file* rute pergerakan *node* berekstensi .trips.xml. *File* .sumocfg digunakan untuk mendefinisikan lokasi *file* .net.xml dan .trips.xml serta durasi simulasi. Untuk melihat visualisasi lalu lintas, *file* .sumocfg dapat dibuka dengan menggunakan *tool* sumo-gui. Kemudian lakukan simulasi lalu lintas dengan menggunakan perintah SUMO yang dapat dilihat pada Gambar 4.6.

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

Gambar 4.6 Perintah SUMO untuk Membuat skenario.xml

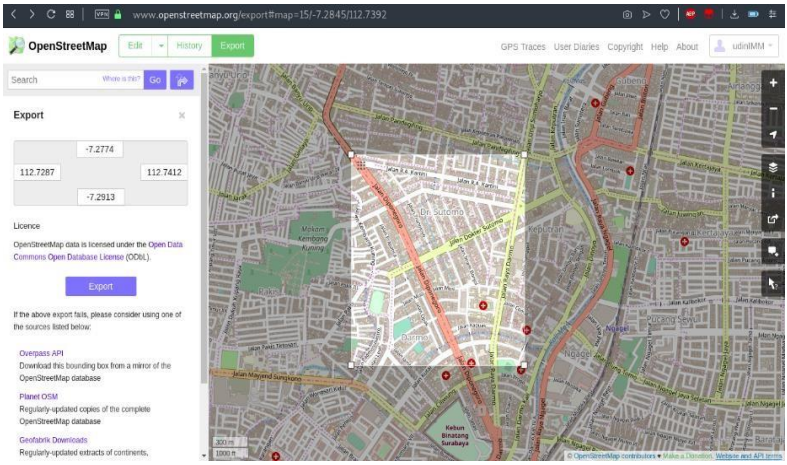
Agar dapat disimulasikan pada NS-2, *file* skenario berekstensi .xml harus dikonversi menjadi *file* berekstensi .tcl dengan menggunakan *tool* traceExporter.py. Perintah untuk menggunakan *tool* traceExporter.py dapat dilihat pada Gambar 4.7.

```
python $SUMO_HOME/tools/traceExporter.py -fcd-input=scenario.xml --
ns2mobilityoutput=scenario.tcl
```

Gambar 4.7 Perintah traceExporter.py

4.2 Implementasi Skenario *Real*

Pada Tugas Akhir ini, penulis mengimplementasikan skenario *real* menggunakan peta hasil pengambilan suatu area di kota Surabaya, yaitu area jalan sekitar Jl. Dr. Soetomo Surabaya. Setelah menentukan area simulasi, ekspor data peta dari situs *OpenStreetMap* seperti yang ditunjukkan pada Gambar 4.8.



Gambar 4.8 Ekspor peta dari *OpenStreetMap*

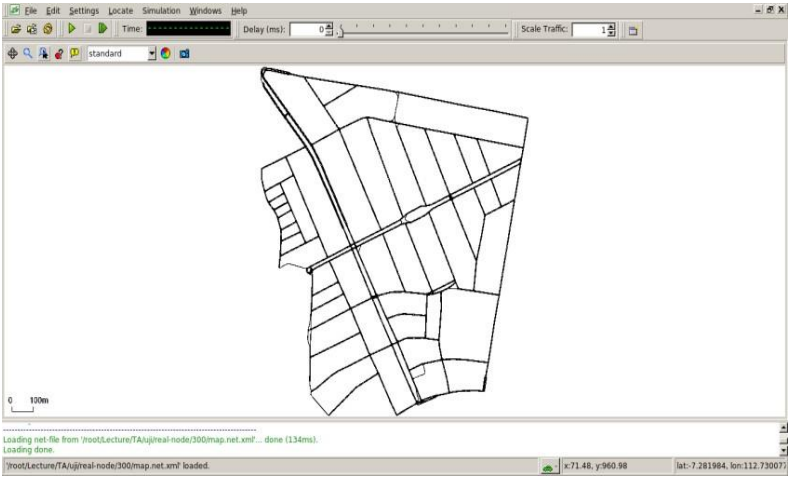
File hasil ekspor dari *OpenStreetMap* tersebut adalah *file* peta dengan ekstensi .osm. Peta disunting terlebih dahulu dengan menggunakan *Java OpenStreetMap Editor* (JOSM) untuk

menghapus jalan yang tidak digunakan. Kemudian *file* .osm hasil suntingan tersebut dikonversi menjadi peta dalam bentuk *file* berekstensi .xml menggunakan *tool* netconvert dari SUMO. Perintah untuk menggunakan netconvert dapat dilihat pada Gambar 4.9.

```
netconvert –osm-files map.osm –outputfile map.net.xml
```

Gambar 4.9 Perintah netconvert

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



Gambar 4.10 Hasil konversi peta *real*

Setelah peta terbentuk, langkah selanjutnya sama dengan ketika membuat skenario *grid*, yaitu membuat *node* asal dan *node* tujuan menggunakan *tool* randomTrips.py. Lalu menggunakan *tool* dua*route*r untuk membuat rute *node* untuk sampai ke tujuan. Kemudian menggunakan *tool* SUMO dengan bantuan *file* skrip berekstensi .sumocfg untuk membuat *file* skenario berekstensi .xml. Agar dapat disimulasikan pada NS-2, *file* skenario

berekstensi ,xml tersebut dikonversi menjadi *file* skenario berekstensi .tcl dengan menggunakan *tool* traceExporter.py. Perintah untuk menggunakan *tool* tersebut sama dengan ketika membuat skenario *grid* di atas.

4.3 Implementasi *K-Means Clustering*

Implementasi *K-Means Clustering* pada Tugas Akhir ini diawali dengan melakukan modifikasi struktur dari beberapa komponen pada AODV antara lain modifikasi *control message* dan modifikasi proses yang menggunakan *control message*.

Proses *K-Means Clustering* akan dilakukan pada *node* yang ditunjuk sebagai server *node* dan dilakukan pada waktu tertentu. Clustering akan menggunakan data yakni *nodes positions table* yang terdapat pada server *node*. Proses akan diawali dengan pengisian data pada *nodes positions table* ke dalam n *cluster* untuk kemudian dilakukan proses penentuan *centroid* secara acak. Proses inisiasi ini dapat dilihat pada Gambar 4.11.

```
void AODV::runCluster(){

.....

  for(int i=0;i<NUMBER_OF_CLUSTER;i++)
  {
    if(i<modClusters Nodes)numberNodes =
NUMBER_OF_NODE/NUMBER_OF_CLUSTER + 1;
    else numberNodes = NUMBER_OF_NODE/NUMBER_OF_CLUSTER ;
    Cluster cluster;
    for(int j=0;j<numberNodes;j++)
    {
      Member member;
      struct point pt;
      pt.x = this->pt.nodes_position_x[memberId];
      pt.y = this->pt.nodes_position_y[memberId];
      member.setMemberPoint(pt);
      member.setMemberIndex(memberId);
      memberId++;
      cluster.assignMember(member);
    }
    cluster.init();

.....
```

```
}
```

Gambar 4.11 Proses Inisiasi Cluster

Kode pada Gambar 4.11 dapat dilihat secara penuh pada lampiran A.8. Proses *K-Means Clustering* akan dilakukan hingga selesai dan akan menghasilkan *nodes clusters table* yang berisi index *node* yang akan bertindak sebagai *cluster head* dan *cluster gateway*. Implementasi pada bagian ini dapat dilihat pada Gambar 4.12.

```
void KMeans::run()
{
   bool change = true;
   while(change)
   {
      change = false;
      for(int i=0;i<this->clusters.size();i++)
      {
         for(int j=0;j<this->clusters[i].members.size();j++)
         {
            int currentClusterIndex=i;
            int nextClusterIndex=i;
            int nodeIndex = clusters[i].members[j].getMemberIndex();
            for(int k=0;k<this->clusters.size();k++)
            {
               double temp =
distanceMemberToCentroid(clusters[i].members[j],clusters[k].getCentroid());
               if(temp<clusters[i].members[j].getNearestDistance())
               {
                  clusters[i].members[j].setNearestDistance(temp);
                  nextClusterIndex = k;
               }
            }
            if(currentClusterIndex != nextClusterIndex)
            {
               Member member = clusters[currentClusterIndex].getMember(nodeIndex);
               clusters[currentClusterIndex].unassignMember(nodeIndex);
               clusters[nextClusterIndex].assignMember(member);
               change = true;
            }
         }
      }
      for(int i=0;i<this->clusters.size();i++)
      {
         this->clusters[i].updateCentroid();
      }
```

```
    }
}
```

Gambar 4.12 Proses *K-Means Clustering*

Kode pada Gambar 4.12 dapat dilihat secara penuh pada lampiran A.9. Hasil dari proses *clustering* akan ditampung pada *nodes clusters table* pada server *node*. Proses pengambilan *cluster head* dan *cluster gateway* kemudian akan dijalankan untuk mendapatkan informasi mengenai *node* yang akan bertindak sebagai *cluster head* dan *cluster gateway*. Implementasi pada bagian ini dapat dilihat pada Gambar 4.13.

```
struct nodes_clusters_table KMeans::getResult()
{

.....

    for(int i=0;i<this->clusters.size();i++)
    {
        if(clusters[i].getSize() != 0)ct.nodes_cluster_head[i] =
clusters[i].getClusterHead().getMemberIndex();
    }

.....

    for(int i=0;i<this->clusters.size();i++)
    {
        if(clusters[i].getSize() != 0)ct.nodes_cluster_gateway[i] =
clusters[i].getClusterGateway(center).getMemberIndex();
    }

.....

}
```

Gambar 4.13 Proses Pengambilan *Cluster Head* dan *Cluster Gateway*

Kode pada Gambar 4.13 dapat dilihat secara penuh pada lampiran A.10. Kode untuk getClusterHead() dan getClusterGateway() dapat dilihat pada lampiran A.11 dan A.12.

4.3.1 Modifikasi Pengiriman *Route Request*(RREQ)

Pengiriman RREQ disertai dengan pengiriman *nodes positions table* yang disisipkan pada paket RREQ. *Node* yang akan mengirimkan RREQ akan terlebih dahulu mengisikan informasi posisi *node* tersebut ke dalam *nodes positions table* yang ada pada *node* tersebut. Implementasi pada modifikasi ini dapat dilihat pada Gambar 4.11.

```
void AODV::send Request (nsaddr_t dst) {

 .....

 ((Mobile Node *) m Node )->getLoc(&posX,&posY);
 pt.nodes _position_x[index] = posX;
 pt.nodes _position_y[index] = posY;
 pt.nodes _timestamp [index] = CURRENT_TIME;
 rq->pt = this->pt;

 Scheduler::instance().schedule(target_, p, 0.);
}
```

Gambar 4.14 Modifikasi Pengiriman RREQ untuk K-Means

Kode yang terdapat pada Gambar 4.14 dapat dilihat secara penuh pada lampiran A.1.

4.3.2 Modifikasi Penerimaan *Route Request*(RREQ)

Paket RREQ yang diterima akan diambil informasi yakni *nodes positions table* yang terdapat pada paket tersebut, kemudian akan dibandingkan dengan *nodes positions table* pada *node* penerima. Perbandingan ini akan didasarkan pada *timestamps* data tersebut didapatkan. *Node*s *positions table* pada paket dan *node* penerima akan saling dibandingkan dan diperbaharui dan *node* penerima akan memasukan data posisi terbarunya pada *nodes positions table* yang ada pada paket yang diterima untuk memastikan bahwa setiap *node* memiliki data yang terbaru nantinya. Implementasi pada modifikasi ini dapat dilihat pada Gambar 4.12.

```
void AODV::recv Request (Packet *p) {
```

```
  struct hdr_ip *ih = HDR_IP(p);
  struct hdr_aodv_request *rq = HDR_AODV_REQUEST(p);

  ((MobileNode*) mNode)->getLoc(&posX,&posY);
  rq->pt.nodes_position_x[index] = this->posX;
  rq->pt.nodes_position_y[index] = this->posY;
  rq->pt.nodes_timestamp[index] = CURRENT_TIME;
  for(int i=0;i<NUMBER_OF_NODE;i++)
  {
    if(this->pt.nodes_timestamp[i] < rq->pt.nodes_timestamp[i])
    {
      this->pt.nodes_position_x[i] = rq->pt.nodes_position_x[i];
      this->pt.nodes_position_y[i] = rq->pt.nodes_position_y[i];
      this->pt.nodes_timestamp[i] = rq->pt.nodes_timestamp[i];
    }
  }

  .....

}
```

Gambar 4.15 Modifikasi Penerimaan RREQ untuk K-Means

Kode yang terdapat pada Gambar 4.15 dapat dilihat secara penuh pada lampiran A.2.

4.3.3 Modifikasi Pengiriman *Route Reply*(RREP)

Pengiriman RREP akan disertai dengan pengiriman *nodes clusters table* yang telah disisipkan pada paket RREP. Implementasi pada modifikasi ini dapat dilihat pada Gambar 4.16.

```
void AODV::sendReply(nsaddr_t ipdst, u_int32_t hop_count, nsaddr_t rpdst, u_int32_t
rpseq, u_int32_t lifetime, double timestamp) {

.....

 rp->rp_dst = rpdst;
 rp->rp_dst_seqno = rpseq;
 rp->rp_src = index;
 rp->rp_lifetime = lifetime;
 rp->rp_timestamp = timestamp;

 rp->ct = this->ct;

......
```

```
}
```

Gambar 4.16 Modifikasi Pengiriman RREP untuk K-Means

Kode yang terdapat pada Gambar 4.16 dapat dilihat secara penuh pada lampiran A.3.

4.3.4 Modifikasi Penerimaan *Route Reply*(RREP)

Paket RREP yang diterima akan diambil informasi mengenai *nodes clusters table* dan akan dibandingkan dengan *nodes clusters table* yang ada pada *node* penerima. Perbandingan ini didasarkan pada *timestamps nodes clusters table*. Tabel ini akan memberikan informasi untuk suatu *node* akan bertindak sebagai *cluster head*, *cluster gateway*, ataupun *cluster* member. Implementasi pada modifikasi ini dapat dilihat pada Gambar 4.17.

```
void AODV::recv Reply (Packet *p) {
  //struct hdr_cmn *ch = HDR_CMN(p);
  struct hdr_ip *ih = HDR_IP(p);
  struct hdr_aodv_reply *rp = HDR_AODV_REPLY(p);
  aodv_rt_entry *rt;
  char suppress_reply = 0;
  double delay = 0.0;

  if(this->ct.nodes_cluster_timestamp < rp->ct.nodes_cluster_timestamp)
  {
    this->ct = rp->ct;
    for(int i=0;i<NUMBER_OF_CLUSTER;i++)
    {
      if(this->ct.nodes_cluster_head[i] == this->index || this-
>ct.nodes_cluster_gateway[i] == this->index)
      {
        this->isClusterHead = true;
        break;
      }
      else
      {
        this->isClusterHead = false;
      }
    }
  }

.......
```

```
}
```

Gambar 4.17 Modifikasi Penerimaan RREP untuk K-Means

Kode yang terdapat pada Gambar 4.17 dapat dilihat secara penuh pada lampiran A.4.

4.3.5 Modifikasi Pengiriman *Hello Messages*

Pengiriman *hello messages* akan diikuti dengan pengiriman informasi *nodes clusters table* dan *nodes positions table*. *Node* pengirim akan mengisikan informasi posisi terbaru tersebut ke dalam *nodes positions table* pada *node* tersebut. Implementasi pada modifikasi ini dapat dilihat pada Gambar 4.18.

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

((Mobile*Node*) m*Node*)->getLoc(&posX,&posY);
this->pt.*nodes*_position_x[index] = posX;
this->pt.*nodes*_position_y[index] = posY;
this->pt.*nodes_timestamp*[index] = CURRENT_TIME;
rh->pt = this->pt;
rh->ct = this->ct;

.....

}
```

Gambar 4.18 Modifikasi Pengiriman *Hello Messages*

Kode yang terdapat pada Gambar 4.18 dapat dilihat secara penuh pada lampiran A.5.

4.3.6 Modifikasi Penerimaan *Hello Messages*

Paket *hello messages* yang diterima akan diambil informasi *nodes clusters table* dan *nodes positions table*. Kedua tabel tersebut akan dibandingkan dengan tabel yang terdapat pada *node* penerima yang didasarkan pada *timestamps table* tersebut. *Node*s *clusters table* akan menentukan suatu *node* bertindak sebagai *cluster head*,

cluster gateway, ataupun *cluster* member. Data pada *nodes positions table* akan diambil oleh *node* penerima untuk memastikan data pada *nodes positions table* yang berada pada *node* penerima adalah data yang terbaru. Implementasi pada modifikasi ini dapat dilihat pada Gambar 4.19.

```
void AODV::recvHello(Packet *p) {
  //struct hdr_ip *ih = HDR_IP(p);
  struct hdr_aodv_reply *rp = HDR_AODV_REPLY(p);
  AODV_Neighbor *nb;

  ((Mobile Node*) m Node)->getLoc(&posX,&posY);
  rp->pt.nodes_position_x[index] = posX;
  rp->pt.nodes_position_y[index] = posY;
  rp->pt.nodes_timestamp[index] = CURRENT_TIME;
  for(int i=0;i<NUMBER_OF_NODE;i++)
  {
    if(pt.nodes_timestamp[i] < rp->pt.nodes_timestamp[i])
    {
      this->pt.nodes_position_x[i] = rp->pt.nodes_position_x[i];
      this->pt.nodes_position_y[i] = rp->pt.nodes_position_y[i];
      this->pt.nodes_timestamp[i] = rp->pt.nodes_timestamp[i];
    }
  }

  if(this->ct.nodes_cluster_timestamp < rp->ct.nodes_cluster_timestamp)
  {
    this->ct = rp->ct;
    for(int i=0;i<NUMBER_OF_CLUSTER;i++)
    {
      if(this->ct.nodes_cluster_head[i] == this->index || this-
>ct.nodes_cluster_gateway[i]  == this->index)
      {
        this->isClusterHead = true;
        break;
      }
      else
      {
        this->isClusterHead = false;
      }
    }
  }

.....

}
```

Gambar 4.19 Modifikasi Penerimaan *Hello Messages*

Kode yang terdapat pada Gambar 4.19 dapat dilihat secara penuh pada lampiran A.6.

4.3.7 Modifikasi Time Scheduling

Proses Clustering akan dilakukan sebanyak n kali dimana n lebih besar sama dengan 1. Proses *clustering* akan dilakukan pada *node* dan waktu tertentu. Implementasi pada modifikasi ini dapat dilihat pada Gambar 4.20.

```
void ClusteringTimer::handle(Event*){
  if(agent->index != SERVER_NODE_INDEX) return;
  if(CURRENT_TIME > 0.0 && (CURRENT_TIME == CLUSTER_TIME_FIRST ||
CURRENT_TIME == CLUSTER_TIME_SECOND))
  {
    agent->runCluster();
    printf("\n\nCLUSTER RUN at %f in %d\n\n",CURRENT_TIME,agent->index);
  }
  Scheduler::instance().schedule(this,&intr,1);
}
```

Gambar 4.20 Modifikasi Time Scheduling

4.4 Implementasi *Ant Colony Optimization* (ACO)

Proses implementasi ACO akan dilakukan dengan memodifikasi struktur dari beberapa *control message*s dan proses yang menggunakan *control message*s tersebut. Pada implementasi ini akan digunakan juga 2 threshold value yakni RSSM threshold value dan REM threshold value yakni sebesar -75 dBm dan 10 Joule.

4.4.1 Modifikasi Pengiriman *Route Request*(RREQ)

Modifikasi yang dilakukan pada pengiriman RREQ adalah pemeriksaan kekuatan sinyal paket tersebut dan keadaan energi dari *node* yang akan mengirimkan paket tersebut. Jika salah satu parameter tidak terpenuhi maka paket akan dibuang. Implementasi pada modifikasi ini dapat dilihat pada Gambar 4.21.

```
void AODV::sendRequest(nsaddr_t dst) {
// Allocate a RREQ packet
Packet *p = Packet::alloc();
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 = rtable.rt_lookup(dst);

  double receiveSignalStrength = p->txinfo_.RxPr;
  double RSSM;
  if(receiveSignalStrength == 0)
  {
    RSSM = -200;
  }
  else
  {
    RSSM = 10*log10(receiveSignalStrength) + 30;
  }
  if(SIGNAL_THRESHOLD >= RSSM)
  {
    Packet::free((Packet *)p);
    return;
  }

  energy = mNode->energy_model()->energy();
  if(ENERGY_THRESHOLD >= energy)
  {
    Packet::free(p);
    return;
  }

.....

}
```

Gambar 4.21 Modifikasi Pengiriman RREQ untuk ACO

Kode yang terdapat pada Gambar 4.21 dapat dilihat secara penuh pada lampiran A.1.

4.4.2 Modifikasi Penerimaan *Route Request*(RREQ)

Modifikasi yang dilakukan pada penerimaan RREQ adalah pemeriksaan kekuatan sinyal paket yang diterima dan keadaan energi dari *node* penerima. Jika salah satu parameter tidak terpenuhi maka paket akan dibuang. Pada bagian ini juga akan

dilakukan perhitungan PC dimana hasilnya akan disimpan pada sebuah variabel bernama lastPheromoneCount di *node* tersebut sebagai nilai PC terakhir dan terbesar yang dihitung pada *node* tersebut. Nilai dari lastPheromoneCount akan disimpan juga pada paket yang akan dikirimkan. Implementasi pada modifikasi ini dapat dilihat pada Gambar 4.22.

```
void AODV::recv Request (Packet *p) {
  struct hdr_ip *ih = HDR_IP(p);
  struct hdr_aodv_request *rq = HDR_AODV_REQUEST(p);

.....

  aodv_rt_entry *rt;
  double receiveSignalStrength = p->txinfo_.RxPr;
  double RSSM;
  if(receiveSignalStrength == 0)
  {
    RSSM = -200;
  }
  else
  {
    RSSM = 10*log10(receiveSignalStrength) + 30;
  }
  if(SIGNAL_THRESHOLD >= RSSM)
  {
    Packet::free((Packet *)p);
    return;
  }
  int queueLength = rqueue.queueLength(index);
  if(queueLength == 0)
  {
    queueLength = 1;
  }
  energy = mNode->energy_model()->energy();
  if(ENERGY_THRESHOLD >= energy)
  {
    Packet::free(p);
    return;
  }
  int hopCount = rq->rq_hop_count;
  double pheromoneCount = ((RSSM+91)*energy)/(queueLength * hopCount);
  if(rq->pheromoneCount != NULL)
  {
    lastPheromoneCount = rq->pheromoneCount + pheromoneCount;
```

```
  }
  else
  {
    lastPheromoneCount = pheromoneCount;
  }
  rq->pheromoneCount = lastPheromoneCount;

.....

     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
  if(lastPheromoneCount != NULL)
  {
    rt0->pheromoneCount = lastPheromoneCount;
  }
}
```

Gambar 4.22 Modifikasi Penerimaan RREQ untuk ACO

Kode yang terdapat pada Gambar 4.22 dapat dilihat secara penuh pada lampiran A.2.

4.4.3 Modifikasi Pengiriman *Route Reply*(RREP)

Modifikasi yang dilakukan pada pengiriman RREP adalah mengisi nilai PC pada paket RREP. Nilai PC yang diisikan adalah nilai PC terbesar antara nilai dari variabel lastPheromoneCount pada *node* tersebut dengan nilai PC menuju *source node* pada *routing table* di *node* tersebut. Implementasi pada modifikasi dapat dilihat pada Gambar 4.23.

```
void AODV::sendReply(nsaddr_t ipdst, u_int32_t hop_count, nsaddr_t rpdst,
            u_int32_t rpseq, u_int32_t lifetime, double timestamp) {

.....

 rp->rp_src = index;
 rp->rp_lifetime = lifetime;
```

```
 rp->rp_timestamp = timestamp;

 if(rp->rp_dst == index)
 {
   rp->pheromoneCount = max(lastPheromoneCount,rt->pheromoneCount);
 }

.....

}
```

Gambar 4.23 Modifikasi Pengiriman RREP untuk ACO

Kode yang terdapat pada Gambar 4.23 dapat dilihat secara penuh pada lampiran A.3.

4.4.4 Modifikasi Penerimaan *Route Reply*(RREP)

Modifikasi yang dilakukan pada penerimaan RREP adalah melakukan *update* pada *routing table node* tersebut yakni memperbaharui PC pada informasi *route* menuju *source node*. Implementasi pada modifikasi dapat dilihat pada Gambar 4.24.

```
void AODV::recvReply(Packet *p) {

.....

  // Update the rt entry
  rt_update(rt, rp->rp_dst_seqno, rp->rp_hop_count,
                        rp->rp_src, rp->pheromoneCount, CURRENT_TIME + rp-
>rp_lifetime);

.....

}
```

Gambar 4.24 Modifikasi Penerimaan RREP untuk ACO

Kode yang terdapat pada Gambar 4.24 dapat dilihat secara penuh pada lampiran A.4.

4.5 Implementasi Simulasi pada NS-2

Implementasi simulasi diawali dengan pendeskripsian lingkungan simulasi pada sebuah *file* berekstensi .tcl. File ini

berisikan konfigurasi umum yang dibutuhkan serta langkah-langkah yang dilakukan selama simulasi. Potongan konfigurasi lingkungan simulasi dapat dilihat pada Gambar 4.23.

```
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)              950                             ;
set opt(y)              950                             ;
set val(ifqlen)         1000                            ;
set val(nn)             200                             ;
set val(seed)           1.0                             ;
set val(adhocRouting)   AODV                            ;
set val(stop)           200                             ;
set val(cp)             "cbr.txt"                       ;
set val(sc)             "scenario.txt"                  ;
```

Gambar 4.25 Konfigurasi Lingkungan Simulasi

Pada konfigurasi dilakukan pemanggilan terhadap *file* traffic yang berisikan konfigurasi *source node*, *destination node*, dan pengiriman paket dengan sesi, serta *file* skenario yang berisikan pergerakan *node* yang telah digenerate menggunakan SUMO. Kode yang terdapat pada Gambar 4.25 dapat dilihat pada lampiran A.13 dan kode konfigurasi traffic dapat dilihat pada lampiran A.14.

4.6 Implementasi Metrik Analisis

Implementasi untuk menguji hasil simulasi yakni dengan membuat sebuah *file shell script* yang akan menjalankan 4 *file* awk *script*. Parameter untuk metrik analisis adalah nama *file* hasil simulasi dengan ekstensi .tr. Implementasi pada metrik analisis dapat dilihat pada Gambar 4.23.

```
awk -f pdr.awk "$1"
awk -f e2e.awk "$1"
awk -f ro.awk "$1"
```

```
awk -f hc.awk "$1"
```

Gambar 4.26 File Metrik Analisis

4.6.1 Implementasi *Packet Delivery Ration(PDR)*

Dalam perhitungan PDR, layer AGT adalah hal yang perlu diperhatikan dalam implementasi PDR. Perhitungan dilakukan dengan menggunakan karakter pada kolom pertama sebagai filter. Rumus perhitungan akan menggunakan rumus yang terdapat pada subbab 3.5.1. Kode implementasi PDR dapat dilihat pada Gambar 4.27.

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

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

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

END {
	print "PDR : " (recvLine/sendLine)*100 " %";
}
```

Gambar 4.27 File pdr.awk

File pdr.awk dapat dieksekusi dengan menajalankan perintah awk -f pdr.awk result.tr pada terminal.

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

Dalam perhitungan E2E, waktu yang tercata pada kolom kedua dengan filter event pada kolom keempat adalah layer AGT dan event pada kolom pertama yang berfungsi membedakan paket dikirim ataupun diterima. Delay akan dihitung dari setiap paket dengan mengurangkan waktu paket diterima dan dikirimkan, perhitungan dilakukan dengan paket yang mempunyai id yang

sama. Rumus perhitungan pada subbab 3.5.2 akan digunakan untuk mendapatkan E2E. Implementasi E2E dapat dilihat pada Gambar 4.28.

```
BEGIN {
   seqno = -1;
   count = 0;
}
{
   if($4 == "AGT" && $1 == "s" && seqno < $6) {
      seqno = $6;
   }
   #end-to-end delay
   if($4 == "AGT" && $1 == "s") {
      start_time[$6] = $2;
   } else if(($7 == "tcp"||$7=="cbr") && ($1 == "r")) {
     end_time[$6] = $2;
   } else if($1 == "D" && ($7 == "tcp" || $7=="cbr")) {
      end_time[$6] = -1;
   }
}
END {
   for(i=0; i<=seqno; i++) {
      if(end_time[i] > 0) {
         delay[i] = end_time[i] - start_time[i];
            count++;
       }
         else
         {
            delay[i] = -1;
         }
   }
   for(i=0; i<=seqno; i++) {
      if(delay[i] > 0) {
         n_to_n_delay = n_to_n_delay + delay[i];
       }
   }
   n_to_n_delay = n_to_n_delay/count;
   print "E2E : " n_to_n_delay * 1000 " ms";
}
```

Gambar 4.28 File e2e.awk

File e2e.awk dapat dieksekusi dengan menjalankan perintah awk -f e2e.awk result.tr pada terminal.

4.6.3 Implementasi *Routing Overhead(RO)*

Dalam perhitungan RO, paket akan diseleksi dengan filter event sent pada kolom pertama dan event layer RTR pada kolom keempat. Jenis paket RREQ, RREP, dan RERR akan digunakan dalam implementasi ini. Rumus perhitungan yang digunakan terdapat pada subbab 3.5.3 untuk mendapatkan RO. Implementasi RO dapat dilihat pada Gambar 4.29.

```
BEGIN{
  rt_pkts = 0;
}
{
  if (($1 == "s" || $1 == "f") && $4 == "RTR" && ($7 =="udp" || $7 == "AODV" || $7
==""message"")) rt_pkts++;
}
END{
  printf("RO : %d\n",rt_pkts);
}
```

Gambar 4.29 File ro.awk

File ro.awk dapat dieksekusi dengan menjalankan perintah awk -f ro.awk result.tr pada terminal.

4.6.4 Implementasi *Average Hop Counts(HC)*

Dalam perhitungan AHC, paket dengan filter event received pada kolom pertama dan filter event AGT pada kolom keempat akan dihitung sebagai paket yang diterima dan juga paket dengan filter event received pada kolom pertama dan filter event RTR pada kolom keempat akan dihitung sebagai *hop count*. Rumus perhitungan yang ada pada subbab 3.5.4 akan digunakan untuk mendapatkan AHC. Implementasi AHC dapat dilihat pada Gambar 4.30.

```
BEGIN{
recvd = 0;
hc = 0;
}

{
if (( $1 == "r") && ( $7 == "cbr" || $7 =="tcp" ) && ( $4=="AGT" )) recvd++;
```

```
if (( $1 == "r") && ( $4 == "RTR") && ( $7 == "cbr")) { hc = hc + 1; }
}

END{
printf("AHC : %.2f\n",hc/recvd);
}
```

Gambar 4.30 File hc.awk

File hc.awk dapat dieksekusi dengan menjalankan perintah awk -f hc.awk result.tr pada terminal.

(Halaman ini sengaja dikosongkan)

BAB V UJI COBA DAN EVALUASI

Pada bab ini akan dilakukan tahap uji coba dan evaluasi sesuai dengan rancangan dan implementasi. Dari hasil uji coba yang didapatkan akan dilakukan evaluasi sehingga dapat menarik kesimpulan.

5.1 Lingkungan Uji Coba

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

Komponen	**Spesifikasi**
CPU	Intel(R) core(TM) i7-6500U CPU @ 2.50GHz
Sistem Operasi	Ubuntu 18.04
Linux Kernel	Linux Kernel 4.4
Memori	12.0 GB
Penyimpanan	50 GB

Tabel 5.1 Spesifikasi Perangkat Lunak

Parameter lingkungan uji coba yang digunakan pada NS-2 dapat dilihat pada Tabel 3.4. Pengujian dilakukan dengan menjalankan skenario yang disimulasikan pada NS-2. Hasil dari simulasi tersebut adalah sebuah *file* dengan ekstensi .tr yang akan dianalisis dengan bantuan skrip awk dan *shell script*. Ada 4 parameter yang akan dianalisis pada bab ini yakni PDR, E2E, RO, dan AHC. Skrip awk pengujian untuk setiap parameter terdapat pada Gambar 4.26, 4.28, 4.29, 4.30. Skrip tersebut akan dijalankan sekaligus dengan bantuan *shell script* yang terdapat pada Gambar 4.25 dengan parameter nama *file* hasil simulasi.

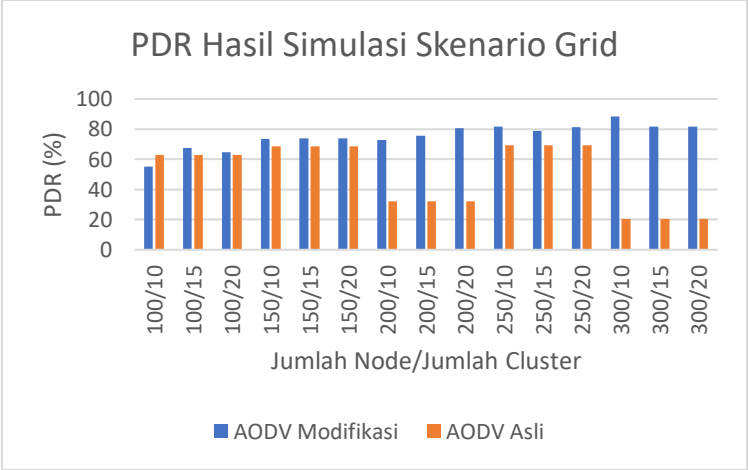
5.2 Skenario Grid

Pengujian pada skenario *grid* digunakan untuk melihat perbandingan PDR, E2E, RO, dan AHC antara *routing protocol* AODV asli dan AODV modifikasi. Pengambilan data pada

skenario *grid* dilakukan sebanyak 5 kali untuk setiap variasi dan data yang ditampilkan pada tabel dan grafik hasil simulasi adalah nilai rata-rata dari 5 kali percobaan untuk setiap variasi.

Jumlah *Node*/ Jumlah Cluster	AODV Modifikasi(%)	AODV Asli(%)	Perbedaan(%)
100/10	55,1732	62,8141	-7,6409
100/15	67,58868	62,8141	+4,77458
100/20	64,51032	62,8141	+1,69622
150/10	73,44996	68,6275	+4,82246
150/15	73,79756	68,6275	+5,17006
150/20	73,82546	68,6275	+5,19796
200/10	72,67744	31,9588	+40,71864
200/15	75,64514	31,9588	+43,68634
200/20	80,75582	31,9588	+48,79702
250/10	81,53368	69,3467	+12,18698
250/15	78,95532	69,3467	+9,60862
250/20	81,16732	69,3467	+11,82062
300/10	88,33096	20,5128	+67,81816
300/15	81,70356	20,5128	+61,19076
300/20	81,75954	20,5128	+61,24674

Tabel 5.2 PDR Hasil Simulasi Skenario Grid

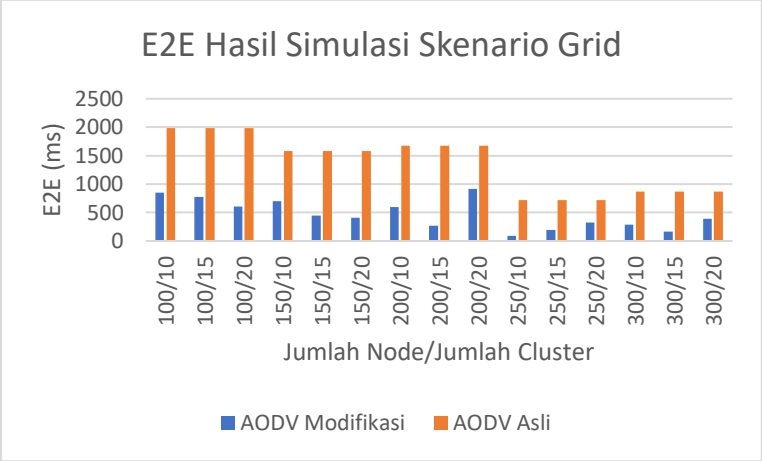


Gambar 5.1 Grafik PDR Hasil Simulasi Skenario Grid

Berdasarkan grafik pada Gambar 5.1 dapat dilihat bahwa AODV modifikasi menghasilkan perubahan PDR yang signifikan terhadap AODV asli. Pada variasi jumlah *node* 100 dan jumlah *cluster* 10, PDR dari AODV modifikasi lebih rendah dibandingkan dengan AODV asli, sedangkan untuk variasi lainnya PDR dari AODV modifikasi lebih tinggi dibandingkan dengan AODV asli. Grafik menunjukan slope positif dimana PDR akan meningkat pada jumlah cluster yang lebih besar. Perbedaan terbesar terjadi pada simulasi denga variasi jumlah *node* 300 dan jumlah *cluster* 10 yakni sebesar 67,81816 %. Pada bagian ini AODV modifikasi menghasilkan PDR yang secara umum lebih baik dibandingkan dengan AODV asli.

Jumlah *Node*/ Jumlah Cluster	AODV Modifikasi(ms)	AODV Asli(ms)	Perbedaan(ms)
100/10	847,7914	1986,61	-1138,8186
100/15	775,83742	1986,61	-1210,77258
100/20	602,8736	1986,61	-1383,7364
150/10	703,4002	1576,39	-872,9898
150/15	447,23224	1576,39	-1129,15776
150/20	407,4974	1576,39	-1168,8926
200/10	595,49194	1672,89	-1077,39806
200/15	269,6807	1672,89	-1403,2093
200/20	918,4836	1672,89	-754,4064
250/10	93,73968	718,837	-625,09732
250/15	193,89002	718,837	-524,94698
250/20	324,91734	718,837	-393,91966
300/10	286,9951	865,675	-578,6799
300/15	165,791	865,675	-699,884
300/20	389,27036	865,675	-476,40464

Tabel 5.3 E2E Hasil Simulasi Skenario Grid



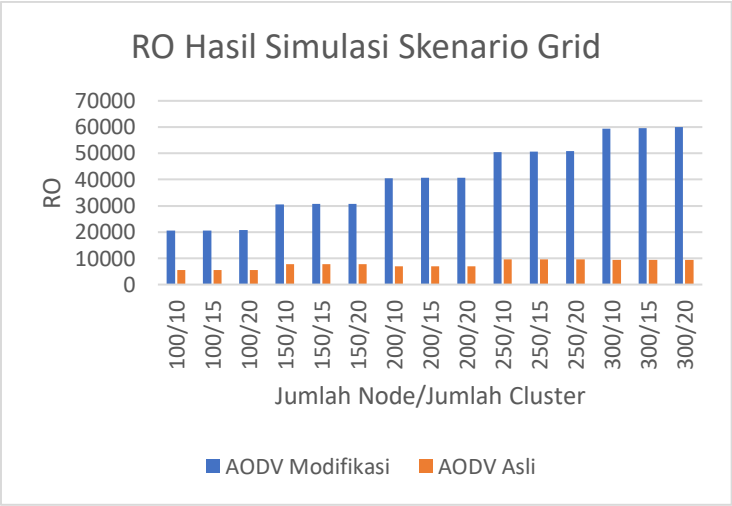
Gambar 5.2 Grafik E2E Hasil Simulasi Skenario Grid

Berdasarkan grafik pada Gambar 5.2 dapat dilihat bahwa AODV modifikasi menghasilkan perubahan E2E yang signifikan terhadap AODV asli. Seluruh variasi yang disimulasikan dengan AODV modifikasi menghasilkan E2E yang lebih kecil dibandingkan dengan AODV asli. Perbedaan E2E terbesar terjadi pada simulasi dengan variasi jumlah *node* 200 dan jumlah *cluster* 15 yakni sebesar 1403,2093 ms. Pada bagian ini AODV modifikasi memilih hasil yang lebih baik dibandingkan dengan AODV asli.

Jumlah *Node*/ Jumlah Cluster	AODV Modifikasi	AODV Asli	Perbedaan
100/10	20597,8	5595	+15002,8
100/15	20674,8	5595	+15079,8
100/20	20769,2	5595	+15174,2
150/10	30620,6	7719	+22901,6
150/15	30665,6	7719	+22946,6
150/20	30727,2	7719	+23008,2
200/10	40558,8	7013	+33545,8
200/15	40621,6	7013	+33608,6
200/20	40686	7013	+33673
250/10	50479,6	9545	+40934,6

250/15	50734,8	9545	+41189,8
250/20	50856,2	9545	+41311,2
300/10	59372,6	9305	+50067,6
300/15	59559,2	9305	+50254,2
300/20	59987	9305	+50682

Tabel 5.4 RO Hasil Simulasi Skenario Grid



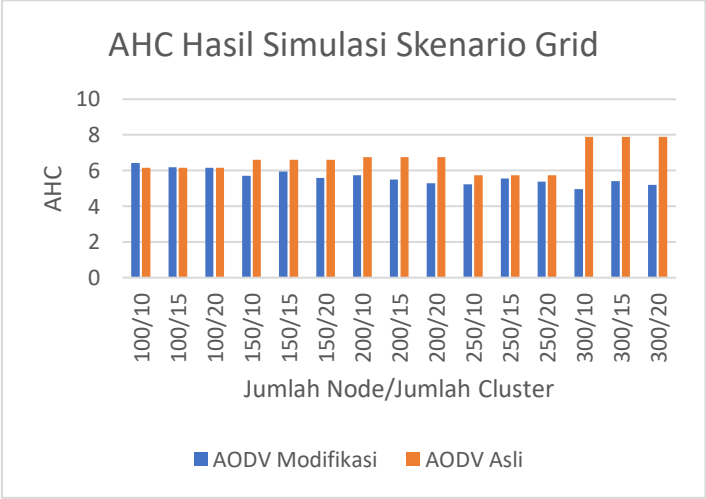
Gambar 5.3 Grafik RO Hasil Simulasi Skenario Grid

Berdasarkan grafik pada Gambar 5.3 dapat dilihat bahwa AODV modifikasi menghasilkan perubahan RO yang lebih besar dari AODV asli. Seluruh variasi yang disimulasikan dengan AODV modifikasi menghasilkan RO yang lebih besar dan memiliki slope positif, dimana RO akan naik secara linear dengan jumlah *node* yang lebih besar. Pada bagian AODV modifikasi memiliki RO hasil simulasi yang lebih buruk dibandingkan dengan AODV asli.

Jumlah *Node*/ Jumlah Cluster	AODV Modifikasi	AODV Asli	Perbedaan
100/10	6,416	6,16	+0,256
100/15	6,198	6,16	+0,038
100/20	6,154	6,16	-0,006

150/10	5,702	6,61	-0,908
150/15	5,948	6,61	-0,662
150/20	5,592	6,61	-1,018
200/10	5,738	6,76	-1,022
200/15	5,502	6,76	-1,258
200/20	5,29	6,76	-1,47
250/10	5,23	5,75	-0,52
250/15	5,558	5,75	-0,192
250/20	5,378	5,75	-0,372
300/10	4,954	7,88	-2,926
300/15	5,404	7,88	-2,476
300/20	5,202	7,88	-2,678

Tabel 5.5 AHC Hasil Simulasi Skenario Grid



Gambar 5.4 Grafik AHC Hasil Simulasi Skenario Grid

Berdasarkan grafik yang terdapat pada Gambar 5.4 dapat dilihat bahwa AHC hasil simulasi AODV modifikasi menghasilkan perubahan yang signifikan. Pada variasi dengan jumlah *node* 100 dan jumlah *cluster* 10 atau 15 AODV asli unggul tipis yakni 0,256 dan 0,038, dimana pada variasi yang lainnya AODV modifikasi unggul. Perbedaan terbesar terdapat pada simulasi dengan variasi jumlah *node* 300 dan jumlah *cluster* 10

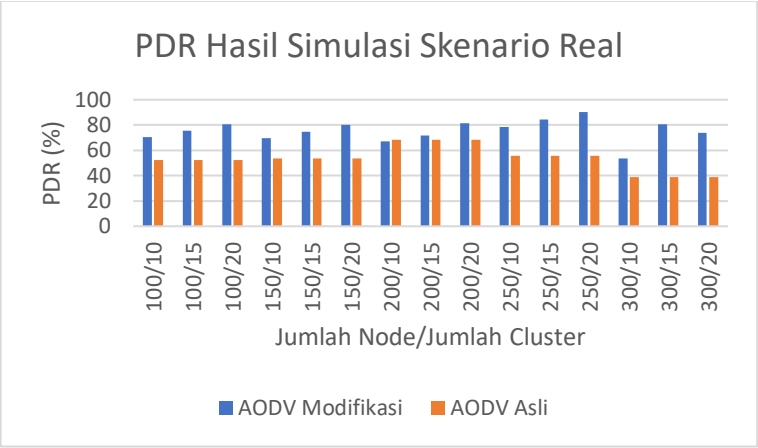
yakni sebesar 2,926. Pada bagian ini AODV modifikasi secara umum memiliki AHC yang lebih baik daripada AODV asli.

5.3 Skenario Real

Pengujian pada skenario *real* digunakan untuk melihat perbandingan PDR, E2E, RO, dan AHC antara *routing protocol* AODV asli dan AODV modifikasi. Pengambilan data pada skenario *real* dilakukan sebanyak 5 kali untuk setiap variasi.

Jumlah *Node*/ Jumlah Cluster	AODV Modifikasi(%)	AODV Asli(%)	Perbedaan(%)
100/10	70,653	52,551	+18,102
100/15	75,62538	52,551	+23,07438
100/20	80,77456	52,551	+28,22356
150/10	69,81712	53,4653	+16,35182
150/15	74,87378	53,4653	+21,40848
150/20	80,05196	53,4653	+26,58666
200/10	66,98668	68,3168	-1,33012
200/15	71,86264	68,3168	+3,54584
200/20	81,57456	68,3168	+13,25776
250/10	78,37696	55,5556	+22,82136
250/15	84,48772	55,5556	+28,93212
250/20	90,12744	55,5556	+34,57184
300/10	53,77018	38,9671	+14,80308
300/15	80,56522	38,9671	+41,59812
300/20	73,78796	38,9671	+34,82086

Tabel 5.6 PDR Hasil Simulasi Skenario Real



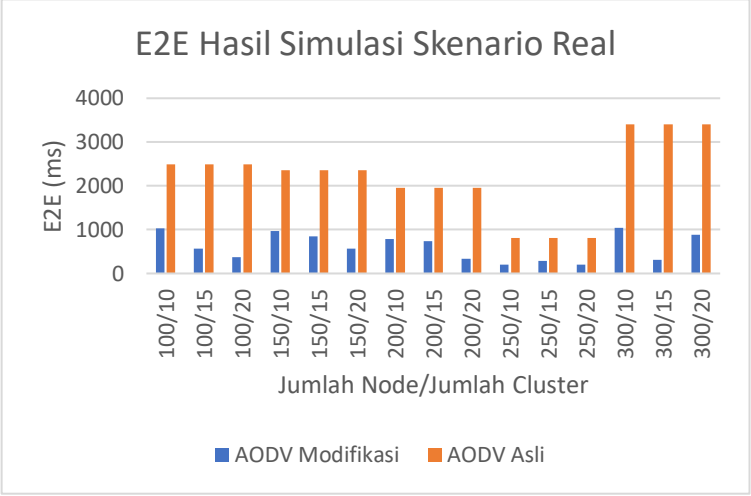
Gambar 5.5 Grafik PDR Hasil Simulasi Skenario Real

Berdasarkan grafik yang terdapat pada Gambar 5.5 dapat dilihat bahwa AODV modifikasi menghasilkan perubahan yang signifikan pada bagian ini. Seluruh simulasi dengan berbagai variasi menunjukan keunggulan AODV modifikasi dibandingkan dengan AODV asli. Grafik menunjukan slope positif dimana PDR akan meningkat dengan jumlah cluster yang lebih besar. Perbedaan terbesar terjadi pada simulasi dengan variasi jumlah *node* 300 dan jumlah *cluster* 15 yakni 41,59812%. Pada bagian ini AODV modifikasi memiliki PDR hasil simulasi yang lebih baik dibandingkan dengan AODV asli.

Jumlah *Node*/ Jumlah Cluster	AODV Modifikasi(ms)	AODV Asli(ms)	Perbedaan(ms)
100/10	1028,8044	2494,57	-1465,7656
100/15	566,479	2494,57	-1928,091
100/20	366,77554	2494,57	-2127,79446
150/10	963,4438	2351,45	-1388,0062
150/15	847,8048	2351,45	-1503,6452
150/20	567,87468	2351,45	-1783,57532
200/10	788,6174	1959,94	-1171,3226
200/15	742,54886	1959,94	-1217,39114
200/20	333,16562	1959,94	-1626,77438

250/10	203,93788	815,993	-612,05512
250/15	289,2226	815,993	-526,7704
250/20	204,78178	815,993	-611,21122
300/10	1037,5684	3403,22	-2365,6516
300/15	309,809	3403,22	-3093,411
300/20	884,17348	3403,22	-2519,04652

Tabel 5.7 E2E Hasil Simulasi Skenario Real



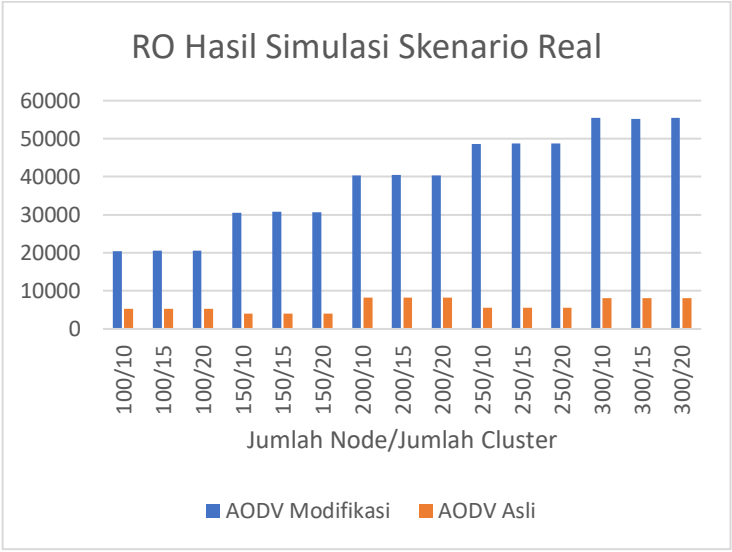
Gambar 5.6 Grafik E2E Hasil Simulasi Skenario Real

Berdasarkan grafik yang terdapat pada Gambar 5.6 dapat dilihat bahwa AODV modifikasi menghasilkan perubahan yang signifikan pada bagian ini. Seluruh simulasi dengan berbagai variasi menunjukan keunggulan AODV modifikasi dibandingkan dengan AODV asli. Perbedaan terbesar terjadi pada simulasi dengan variasi jumlah *node* 300 dan jumlah *cluster* 15 yakni 3093,411 ms. Pada bagian ini AODV modifikasi memiliki E2E hasil simulasi yang lebih baik dibandingkan dengan AODV asli.

Jumlah *Node*/ Jumlah Cluster	AODV Modifikasi	AODV Asli	Perbedaan
100/10	20428	5275	+15153
100/15	20520	5275	+15245

100/20	20641	5275	+15366
150/10	30490,6	4080	+26410,6
150/15	30743,8	4080	+26663,8
150/20	30670,6	4080	+26590,6
200/10	40280,2	8219	+32061,2
200/15	40465,8	8219	+32246,8
200/20	40401,4	8219	+32182,4
250/10	48653	5516	+43137
250/15	48777,8	5516	+43261,8
250/20	48787,2	5516	+43271,2
300/10	55503	8117	+47386
300/15	55141,6	8117	+47024,6
300/20	55438,8	8117	+47321,8

Tabel 5.8 RO Hasil Simulasi Skenario Real

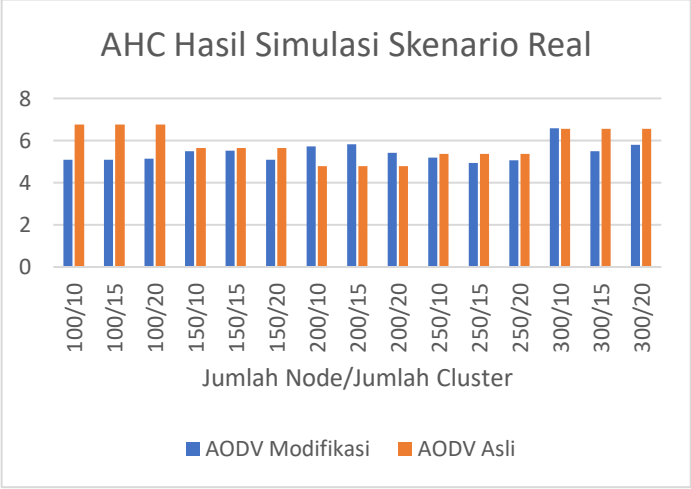


Gambar 5.7 Grafik RO Hasil Simulasi Skenario Real

Berdasarkan grafik yang terdapat pada Gambar 5.7 dapat dilihat bahwa AODV modifikasi menyebabkan RO meningkat jika dibandingkan dengan AODV asli. Seluruh simulasi dengan berbagai variasi menyebabkan RO meningkat fluktuatif. Pada bagian ini AODV modifikasi memiliki RO hasil simulasi yang lebih buruk dibandingkan dengan AODV asli.

Jumlah *Node*/ Jumlah Cluster	AODV Modifikasi	AODV Asli	Perbedaan
100/10	5,096	6,76	-1,664
100/15	5,084	6,76	-1,676
100/20	5,15	6,76	-1,61
150/10	5,506	5,65	-0,144
150/15	5,53	5,65	-0,12
150/20	5,09	5,65	-0,56
200/10	5,724	4,78	+0,944
200/15	5,816	4,78	+1,036
200/20	5,424	4,78	+0,644
250/10	5,18	5,36	-0,18
250/15	4,946	5,36	-0,414
250/20	5,074	5,36	-0,286
300/10	6,582	6,57	+0,012
300/15	5,496	6,57	-1,074
300/20	5,804	6,57	-0,766

Tabel 5.9 AHC Hasil Simulasi Skenario Real



Gambar 5.8 Grafik AHC Hasil Simulasi Skenario Real

Berdasarkan grafik yang terdapat pada Gambar 5.8 dapat dilihat bahwa AHC hasil simulasi AODV modifikasi mengalami

perubahan yang signifikan. Pada jumlah *node* 200 dengan jumlah *cluster* 10, 15, dan 20 AODV asli unggul dibandingkan dengan AODV modifikasi dan juga pada jumlah *node* 300 dengan jumlah *cluster* 10 AODV asli unggul dibandingkan dengan AODV modifikasi, sementara simulasi dengan variasi yang lainnya menunjukan AODV modifikasi unggul dibandingkan dengan AODV asli. Perbedaan terbesar terjadi pada simulasi dengan variasi jumlah *node* 100 dan jumlah *cluster* 15 yakni sebesar 1,676. Pada bagian ini AODV modifikasi memiliki AHC hasil simulasi yang lebih baik dibandingkan dengan AODV asli.

BAB VI KESIMPULAN DAN SARAN

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

6.3 Kesimpulan

Kesimpulan yang diperoleh dari Tugas Akhir ini didasarkan pada hasil uji coba dan evaluasi. Kesimpulan Tugas Akhir ini adalah sebagai berikut :

1. Penerapan *K-Means Clustering* telah berhasil mengurangi jumlah *forwarding node* secara signifikan dan setelah dikombinasikan dengan *Ant Colony Optimization* secara umum menghasilkan hasil yang baik dari sisi Packet Delivery Ration, End-to-End Delay, dan Average *Hop Count*, namun kombinasi ini juga menyebabkan *Routing* Over*head* meningkat secara fluktuatif.
2. Pengaruh implementasi *K-Means Clustering* dan *Ant Colony Optimization* terhadap performa AODV pada skenario *grid* adalah rata-rata kenaikan PDR sebesar 94%, rata-rata penurunan E2E sebesar 66%, rata-rata kenaikan RO sebesar 403%, dan rata-rata penurunan AHC sebesar 14%.
3. Pengaruh implementasi *K-Means Clustering* dan *Ant Colony Optimization* terhadap performa AODV pada skenario *real* adalah rata-rata kenaikan PDR sebesar 44%, rata-rata penurunan E2E sebesar 71%, rata-rata kenaikan RO 539%, dan rata-rata penurunan AHC sebesar 5%.

6.4 Saran

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

1. Melakukan uji coba dengan variasi yang lebih variatif untuk mendapatkan hasil yang lebih akurat.

2. Menerapkan metode lain pada *K-Means Clustering* agar memperoleh jumlah *cluster* yang optimal, seperti metode Genetic Algorithm(GA).
3. Menerapkan sebuah metode untuk mengurangi *Routing* Over*head* pada penerapan *K-Means Clustering* dan *Ant Colony Optimization* pada AODV.

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. Ahmed dan I. Osman, "AODV ROUTING PROTOCOL WORKING PROCESS," *Journal of Convergence Information Technology (JCIT),* vol. 10, no. 2, 2015.

[3] B. A. Kumar, M. V. Subramanyam dan K. S. Prasad, "An Energy Efficient Clustering Using K-Means and AODV Routing Protocol in Ad-hoc Networks," *International Journal of Intelligent Engineering & Systems,* vol. 12, no. 2, 2019.

[4] "OpenStreetMap," [Online]. Available: https://www.openstreetmap.org/.. [Diakses 24 November 2019].

[5] "JOSM," [Online]. Available: josm.openstreetmap.de/.. [Diakses 2019 November 24].

[6] D. Sarkar, S. Choudhury dan A. Majumder, "Enhanced-Ant-AODV for optimal route selection in mobile ad-hoc," *Journal of King Saud University – Computer and Information Sciences,* 2018.

[7] Z. Z. Shirazi dan S. J. Mirabedini, "Dynamic K-Means Algorithm for Optimized Routing in Mobilde Ad Hoc Networks," *International Journal of Computer Science & Engineering Survey (IJCSES),* vol. 7, 2016.

[8] P. K. Maurya, G. Sharma, V. Suhu, A. Roberts dan M. Srivastava, "An Overview of AODV Routing Protocol," *International Journal of Modern Engineering Research (IJMER),* vol. 2, no. 3, pp. 728-732, 2012.

[9] M. Dipobagio, "An Overview on Ad Hoc Networks," Institute of Computer Science (ICS), Freie Universität Berlin, Berlin.

[10] M. Singh, PhD dan S. Kumar, "A Survey: Ad-hoc on Demand Distance Vector (AODV)," *International Journal of Computer Applications (0975 – 8887)*, vol. 161, no. 1, 2017.

[11] A. Ndlovu, "Improved Energy Efficient AODV Routing using K-means Algorithm for Cluster Head Selection," *International Journal of Computer Science and Mobile Computing*, vol. 4, no. 8, pp. 177-187, 2015.

[12] A. M. Abdel-Moniem, M. H. Mohamed dan A.-R. Hedar, "An Ant Colony Optimization Algorithm for the Mobile Ad Hoc Network Routing Problem Based on AODV Protocol," *10th International Conference on Intelligent Systems Design and Applications*, 2010.

LAMPIRAN

A.1 Kode Fungsi send*Request*()

```
void
AODV::send Request (nsaddr_t dst) {
// Allocate a RREQ packet
Packet *p = Packet::alloc();
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 = rtable.rt_lookup(dst);

 //MODIFIED//
 //SIGNAL
 double receiveSignalStrength = p->txinfo_.RxPr;
 double RSSM;
 if(receiveSignalStrength == 0)
 {
  RSSM = -200;
 }
 else
 {
  RSSM = 10*log10(receiveSignalStrength) + 30;
 }
 if(SIGNAL_THRESHOLD >= RSSM)
 {
  Packet::free((Packet *)p);
  return;
 }

 //ENERGY
 energy = mNode->energy_model()->energy();
 if(ENERGY_THRESHOLD >= energy)
 {
  Packet::free(p);
  return;
 }
 // MODIFIED //


assert(rt);

/*
 *  Rate limit sending of Route Requests. We are very conservative
 *  about sending out route requests.
 */
```

```
 if (rt->rt_flags == RTF_UP) {
   assert(rt->rt_hops != INFINITY2);
   Packet::free((Packet *)p);
   return;
 }

 if (rt->rt_req_timeout > CURRENT_TIME) {
   Packet::free((Packet *)p);
   return;
 }

 // rt_req_cnt is the no. of times we did network-wide broadcast
 // RREQ_RETRIES is the maximum number we will allow before
 // going to a long timeout.

 if (rt->rt_req_cnt > RREQ_RETRIES) {
   rt->rt_req_timeout = CURRENT_TIME + MAX_RREQ_TIMEOUT;
   rt->rt_req_cnt = 0;
 Packet *buf_pkt;
   while ((buf_pkt = rqueue.deque(rt->rt_dst))) {
       drop(buf_pkt, DROP_RTR_NO_ROUTE);
   }
   Packet::free((Packet *)p);
   return;
 }

#ifdef DEBUG
   fprintf(stderr, "(%2d) - %2d sending Route Request, dst: %d\n",
                ++route_request, index, rt->rt_dst);
#endif // DEBUG

 // Determine the TTL to be used this time.
 // Dynamic TTL evaluation - SRD

 rt->rt_req_last_ttl = max(rt->rt_req_last_ttl,rt->rt_last_hop_count);

 if (0 == rt->rt_req_last_ttl) {
 // first time query broadcast
   ih->ttl_ = TTL_START;
 }
 else {
 // Expanding ring search.
   if (rt->rt_req_last_ttl < TTL_THRESHOLD)
     ih->ttl_ = rt->rt_req_last_ttl + TTL_INCREMENT;
   else {
   // network-wide broadcast
```

```
    ih->ttl_ = NETWORK_DIAMETER;
    rt->rt_req_cnt += 1;
  }
 }

 // remember the TTL used  for the next time
 rt->rt_req_last_ttl = ih->ttl_;

 // PerHopTime is the roundtrip time per hop for route requests.
 // The factor 2.0 is just to be safe .. SRD 5/22/99
 // Also note that we are making timeouts to be larger if we have
 // done network wide broadcast before.

 rt->rt_req_timeout = 2.0 * (double) ih->ttl_ * PerHopTime(rt);
 if (rt->rt_req_cnt > 0){
  rt->rt_req_timeout *= rt->rt_req_cnt;
  rt->rt_req_timeout += CURRENT_TIME;
 }

 // Don't let the timeout to be too large, however .. SRD 6/8/99
 if (rt->rt_req_timeout > CURRENT_TIME + MAX_RREQ_TIMEOUT){
  rt->rt_req_timeout = CURRENT_TIME + MAX_RREQ_TIMEOUT;
  rt->rt_expire = 0;
 }

#ifdef DEBUG
 fprintf(stderr, "(%2d) - %2d sending Route Request, dst: %d, tout %f ms\n",
                      ++route_request,
                            index, rt->rt_dst,
                            rt->rt_req_timeout - CURRENT_TIME);
#endif       // DEBUG


 // Fill out the RREQ packet
 // ch->uid() = 0;
 ch->ptype() = PT_AODV;
 ch->size() = IP_HDR_LEN + rq->size();
 ch->iface() = -2;
 ch->error() = 0;
 ch->addr_type() = NS_AF_NONE;
 ch->prev_hop_ = index;          // AODV hack

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

```
 // Fill up some more fields.
 rq->rq_type = AODVTYPE_RREQ;
 rq->rq_hop_count = 1;
 rq->rq_bcast_id = bid++;
 rq->rq_dst = dst;
 rq->rq_dst_seqno = (rt ? rt->rt_seqno : 0);
 rq->rq_src = index;
 seqno += 2;
 assert ((seqno%2) == 0);
 rq->rq_src_seqno = seqno;
 rq->rq_timestamp = CURRENT_TIME;

  // MODIFIED //
  ((MobileNode*) mNode)->getLoc(&posX,&posY);
  pt.nodes_position_x[index] = posX;
  pt.nodes_position_y[index] = posY;
  pt.nodes_timestamp[index] = CURRENT_TIME;
  rq->pt = this->pt;
  // MODIFIED //

 Scheduler::instance().schedule(target_, p, 0.);
}
```

A.2 Kode Fungsi recv*Request*()

```
void
AODV::recvRequest(Packet *p) {
struct hdr_ip *ih = HDR_IP(p);
struct hdr_aodv_request *rq = HDR_AODV_REQUEST(p);

// MODIFIED //
  ((MobileNode*) mNode)->getLoc(&posX,&posY);
  rq->pt.nodes_position_x[index] = this->posX;
  rq->pt.nodes_position_y[index] = this->posY;
  rq->pt.nodes_timestamp[index] = CURRENT_TIME;
  for(int i=0;i<NUMBER_OF_NODE;i++)
  {
    if(this->pt.nodes_timestamp[i] < rq->pt.nodes_timestamp[i])
    {
      this->pt.nodes_position_x[i] = rq->pt.nodes_position_x[i];
      this->pt.nodes_position_y[i] = rq->pt.nodes_position_y[i];
      this->pt.nodes_timestamp[i] = rq->pt.nodes_timestamp[i];
    }
    else if(this->pt.nodes_timestamp[i] > rq->pt.nodes_timestamp[i])
    {
      rq->pt.nodes_position_x[i] = this->pt.nodes_position_x[i];
      rq->pt.nodes_position_y[i] = this->pt.nodes_position_y[i];
```

```
      rq->pt.nodes_timestamp[i] = this->pt.nodes_timestamp[i];
    }
  }
// MODIFIED //

aodv_rt_entry *rt;

// MODIFIED //
  //SIGNAL
  double receiveSignalStrength = p->txinfo_.RxPr;
  double RSSM;
  if(receiveSignalStrength == 0)
  {
    RSSM = -200;
  }
  else
  {
    RSSM = 10*log10(receiveSignalStrength) + 30;
  }
  if(SIGNAL_THRESHOLD >= RSSM)
  {
    Packet::free((Packet *)p);
    return;
  }

  //CONGESTION
  int queueLength = rqueue.queueLength(index);
  if(queueLength == 0)
  {
    queueLength = 1;
  }

  //ENERGY
  energy = mNode->energy_model()->energy();
  if(ENERGY_THRESHOLD >= energy)
  {
    Packet::free(p);
    return;
  }

  //HOPCOUNT
  int hopCount = rq->rq_hop_count;

  //PHEROMONE COUNT
  double pheromoneCount = ((RSSM+91)*energy)/(queueLength * hopCount);
  if(rq->pheromoneCount != NULL)
  {
```

```
    lastPheromoneCount = rq->pheromoneCount + pheromoneCount;
   }
   else
   {
    lastPheromoneCount = pheromoneCount;
   }
   rq->pheromoneCount = lastPheromoneCount;

// MODIFIED //


   /*
    * Drop if:
    *      - I'm the source
    *      - I recently heard this request.
    */

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

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

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

    Packet::free(p);
    return;
  }

  /*
   * 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(), rq-
>pheromoneCount,
                    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,
    // 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

 // MODIFIED //
 if(lastPheromoneCount != NULL)
 {
```

```
    rt0->pheromoneCount = lastPheromoneCount;
  }
  // MODIFIED //

 /*
  * 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 ..

 if(rq->rq_dst == index) {
printf("\nSAMPAI DI DESTINATION -> %d Time : %f",index,CURRENT_TIME);
#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(rq->rq_src,          // IP Destination
           1,                  // Hop Count
           index,               // Dest IP Address
           seqno,               // Dest Sequence Num
           MY_ROUTE_TIMEOUT,    // Lifetime
           rq->rq_timestamp);   // timestamp
   Packet::free(p);
 }

 // 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?
   if (rq->rq_timestamp == NULL) rq->rq_timestamp = CURRENT_TIME;
   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);

    // MODIFIED //
    if(!isClusterHead)
    {
      Packet::free(p);
      return;
    }
    // MODIFIED //
```

```
   forward((aodv_rt_entry*) 0, p, DELAY);
 }

}
```

A.3 Kode Fungsi send*Reply*()

```
void
AODV::send Reply(nsaddr_t ipdst, u_int32_t hop_count, nsaddr_t rpdst,
                u_int32_t rpseq, u_int32_t lifetime, double timestamp) {
Packet *p = Packet::alloc();
struct hdr_cmn *ch = HDR_CMN(p);
struct hdr_ip *ih = HDR_IP(p);
struct hdr_aodv_reply *rp = HDR_AODV_REPLY(p);
aodv_rt_entry *rt = rtable.rt_lookup(ipdst);
  printf("\n SEND REPLY BY %d to %d at %f",rpdst,ipdst,CURRENT_TIME);
#ifdef DEBUG
fprintf(stderr, "sending Reply from %d at %.2f\n", index,
Scheduler::instance().clock());
#endif // DEBUG
 assert(rt);

 rp->rp_type = AODVTYPE_RREP;
 //rp->rp_flags = 0x00;
 rp->rp_hop_count = hop_count;
 rp->rp_dst = rpdst;
 rp->rp_dst_seqno = rpseq;
 rp->rp_src = index;
 rp->rp_lifetime = lifetime;
 rp->rp_timestamp = timestamp;

 // MODIFIED //
 rp->ct = this->ct;
 if(rp->rp_dst == index)
 {
   rp->pheromoneCount = max(lastPheromoneCount,rt->pheromoneCount);
 }
 // MODIFIED //

 // ch->uid() = 0;
 ch->ptype() = PT_AODV;
 ch->size() = IP_HDR_LEN + rp->size();
 ch->iface() = -2;
 ch->error() = 0;
 ch->addr_type() = NS_AF_INET;
 ch->next_hop_ = rt->rt_nexthop;
 ch->prev_hop_ = index;          // AODV hack
```

```
 ch->direction() = hdr_cmn::DOWN;

 ih->saddr() = index;
 ih->daddr() = ipdst;
 ih->sport() = RT_PORT;
 ih->dport() = RT_PORT;
 ih->ttl_ = NETWORK_DIAMETER;

 Scheduler::instance().schedule(target_, p, 0.);

}
```

A.4 Kode Fungsi recv*Reply*()

```
void
AODV::recvReply(Packet *p) {
//struct hdr_cmn *ch = HDR_CMN(p);
struct hdr_ip *ih = HDR_IP(p);
struct hdr_aodv_reply *rp = HDR_AODV_REPLY(p);
aodv_rt_entry *rt;
char suppress_reply = 0;
double delay = 0.0;
printf("\nSAMPAI KEMBALI (PERJALANAN) KE SOURCE -> %d Time :
%f",index,CURRENT_TIME);
// MODIFIED //
  if(this->ct.nodes_cluster_timestamp < rp->ct.nodes_cluster_timestamp)
  {
    this->ct = rp->ct;
    for(int i=0;i<NUMBER_OF_CLUSTER;i++)
    {
      if(this->ct.nodes_cluster_head[i] == this->index || this-
>ct.nodes_cluster_gateway[i] == this->index)
      {
        this->isClusterHead = true;
        break;
      }
      else
      {
        this->isClusterHead = false;
      }
    }
  }
// MODIFIED //

#ifdef DEBUG
 fprintf(stderr, "%d - %s: received a REPLY\n", index, __FUNCTION__);
#endif // DEBUG
```

```
/*
 *  Got a reply. So reset the "soft state" maintained for
 *  route requests in the request table. We don't really have
 *  have a separate request table. It is just a part of the
 *  routing table itself.
 */
// Note that rp_dst is the dest of the data packets, not the
// the dest of the reply, which is the src of the data packets.

rt = rtable.rt_lookup(rp->rp_dst);

/*
 *  If I don't have a rt entry to this host... adding
 */
if(rt == 0) {
  rt = rtable.rt_add(rp->rp_dst);
}

/*
 * Add a forward route table entry... here I am following
 * Perkins-Royer AODV paper almost literally - SRD 5/99
 */

if ( (rt->rt_seqno < rp->rp_dst_seqno) ||   // newer route
     ((rt->rt_seqno == rp->rp_dst_seqno) &&
      (rt->rt_hops > rp->rp_hop_count)) ) { // shorter or better route

  // Update the rt entry
  rt_update(rt, rp->rp_dst_seqno, rp->rp_hop_count,
                           rp->rp_src, rp->pheromoneCount, CURRENT_TIME + rp-
>rp_lifetime);

  // reset the soft state
  rt->rt_req_cnt = 0;
  rt->rt_req_timeout = 0.0;
  rt->rt_req_last_ttl = rp->rp_hop_count;

if (ih->daddr() == index) { // If I am the original source
  // Update the route discovery latency statistics
  // rp->rp_timestamp is the time of request origination
  printf("\nSAMPAI KEMBALI DI SOURCE -> %d Time :
%f",index,CURRENT_TIME);
    rt->rt_disc_latency[(unsigned char)rt->hist_indx] = (CURRENT_TIME - rp-
>rp_timestamp)
                                / (double) rp->rp_hop_count;
```

```
    // increment indx for next time
    rt->hist_indx = (rt->hist_indx + 1) % MAX_HISTORY;
   }

   /*
    * Send all packets queued in the sendbuffer destined for
    * this destination.
    * XXX - observe the "second" use of p.
    */
   Packet *buf_pkt;
   while((buf_pkt = rqueue.deque(rt->rt_dst))) {
    if(rt->rt_hops != INFINITY2) {
         assert (rt->rt_flags == RTF_UP);
    // Delay them a little to help ARP. Otherwise ARP
    // may drop packets. -SRD 5/23/99
      forward(rt, buf_pkt, delay);
      delay += ARP_DELAY;
    }
   }
  }
  else {
   suppress_reply = 1;
  }

  /*
   * If reply is for me, discard it.
   */

 if(ih->daddr() == index || suppress_reply) {
    Packet::free(p);
  }
  /*
   * Otherwise, forward the Route Reply.
   */
  else {
  // Find the rt entry
 aodv_rt_entry *rt0 = rtable.rt_lookup(ih->daddr());
    // If the rt is up, forward
    if(rt0 && (rt0->rt_hops != INFINITY2)) {
         assert (rt0->rt_flags == RTF_UP);
      rp->rp_hop_count += 1;
      rp->rp_src = index;
      forward(rt0, p, NO_DELAY);
      // Insert the nexthop towards the RREQ source to
      // the precursor list of the RREQ destination
      rt->pc_insert(rt0->rt_nexthop); // nexthop to RREQ source
```

```
   }
   else {
   // I don't know how to forward .. drop the reply.
#ifdef DEBUG
     fprintf(stderr, "%s: dropping Route Reply\n", __FUNCTION__);
#endif // DEBUG
     drop(p, DROP_RTR_NO_ROUTE);
   }
 }
}
```

A.5 Kode Fungsi sendHello()

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

// MODIFIED //
((MobileNode*) mNode)->getLoc(&posX,&posY);
this->pt.nodes_position_x[index] = posX;
this->pt.nodes_position_y[index] = posY;
this->pt.nodes_timestamp[index] = CURRENT_TIME;
rh->pt = this->pt;
rh->ct = this->ct;
// MODIFIED //

#ifdef DEBUG
fprintf(stderr, "Sending Hello from %d at %.2f\n", index,
Scheduler::instance().clock());
#endif // DEBUG

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

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

```
 ch->prev_hop_ = index;          // AODV hack

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

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

A.6 Kode Fungsi recvHello()

```
void
AODV::recvHello(Packet *p) {
//struct hdr_ip *ih = HDR_IP(p);
struct hdr_aodv_reply *rp = HDR_AODV_REPLY(p);
AODV_Neighbor *nb;

  // MODIFIED //
  ((MobileNode*) mNode)->getLoc(&posX,&posY);
  rp->pt.nodes_position_x[index] = posX;
  rp->pt.nodes_position_y[index] = posY;
  rp->pt.nodes_timestamp[index] = CURRENT_TIME;
  for(int i=0;i<NUMBER_OF_NODE;i++)
  {
    if(pt.nodes_timestamp[i] < rp->pt.nodes_timestamp[i])
    {
      this->pt.nodes_position_x[i] = rp->pt.nodes_position_x[i];
      this->pt.nodes_position_y[i] = rp->pt.nodes_position_y[i];
      this->pt.nodes_timestamp[i] = rp->pt.nodes_timestamp[i];
    }
  }
  // MODIFIED //
  // MODIFIED //
  if(this->ct.nodes_cluster_timestamp < rp->ct.nodes_cluster_timestamp)
  {
    this->ct = rp->ct;
    for(int i=0;i<NUMBER_OF_CLUSTER;i++)
    {
      if(this->ct.nodes_cluster_head[i] == this->index || this-
>ct.nodes_cluster_gateway[i] == this->index)
      {
        this->isClusterHead = true;
        break;
      }
      else
```

```
    {
      this->isClusterHead = false;
    }
   }
  }
  // MODIFIED //

 nb = nb_lookup(rp->rp_dst);
 if(nb == 0) {
   nb_insert(rp->rp_dst);
 }
 else {
   nb->nb_expire = CURRENT_TIME +
                (1.5 * ALLOWED_HELLO_LOSS * HELLO_INTERVAL);
 }

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

A.7 Kode Fungsi rt_*update*()

```
void
AODV::rt_update(aodv_rt_entry *rt, u_int32_t seqnum, u_int16_t metric,
                        nsaddr_t nexthop, double phCount, double expire_time) {

    // MODIFIED //
    rt->pheromoneCount = phCount;
    // MODIFIED //

     rt->rt_seqno = seqnum;
     rt->rt_hops = metric;
     rt->rt_flags = RTF_UP;
     rt->rt_nexthop = nexthop;
     rt->rt_expire = expire_time;
}
```

A.8 Kode Fungsi runCluster()

```
void AODV::runCluster(){
  KMeans km;
  srand(time(0));
  int memberId=0;
  int modClustersNodes = NUMBER_OF_NODE % NUMBER_OF_CLUSTER;
  int numberNodes;
  for(int i=0;i<NUMBER_OF_CLUSTER;i++)
  {
```

```
    if(i<modClusters Nodes)numberNodes =
NUMBER_OF_NODE/NUMBER_OF_CLUSTER + 1;
    else numberNodes = NUMBER_OF_NODE/NUMBER_OF_CLUSTER ;
    Cluster cluster;
    for(int j=0;j<numberNodes;j++)
    {
      Member member;
      struct point pt;
      pt.x = this->pt.nodes_position_x[memberId];
      pt.y = this->pt.nodes_position_y[memberId];
      member.setMemberPoint(pt);
      member.setMemberIndex(memberId);
      memberId++;
      cluster.assignMember(member);
    }
    cluster.init();
    km.assignCluster(cluster);
  }
  km.run();

  if(this->ct.nodes_cluster_timestamp < km.getResult().nodes_cluster_timestamp)
    this->ct = km.getResult();
```

A.9 Kode Fungsi run()

```
void KMeans::run()
{
    bool change = true;

    while(change)
    {
        change = false;
        //check every member in cluster to each centroid of clusters
        for(int i=0;i<this->clusters.size();i++)
        {
            for(int j=0;j<this->clusters[i].members.size();j++)
            {
                int currentClusterIndex=i;
                int nextClusterIndex=i;
                int nodeIndex = clusters[i].members[j].getMemberIndex();
                for(int k=0;k<this->clusters.size();k++)
                {
                    double temp =
distanceMemberToCentroid(clusters[i].members[j],clusters[k].getCentroid());
                    //shorter distance is found
                    if(temp<clusters[i].members[j].getNearestDistance())
                    {
```

```
                    clusters[i].members[j].setNearestDistance(temp);
                    nextClusterIndex = k;
                }
            }
            //changing of member cluster
            if(currentClusterIndex != nextClusterIndex)
            {
                Member member =
clusters[currentClusterIndex].getMember(nodeIndex);
                clusters[currentClusterIndex].unassignMember(nodeIndex);
                clusters[nextClusterIndex].assignMember(member);
                change = true;
            }
        }
    }
    //update centroid
    for(int i=0;i<this->clusters.size();i++)
    {
        this->clusters[i].updateCentroid();
    }
  }
}
```

A.10 Kode Fungsi getResult()

```
struct nodes_clusters_table KMeans::getResult()
{
  struct nodes_clusters_table ct;
  memset(ct.nodes_cluster_head,-1,sizeof(ct.nodes_cluster_head));
  memset(ct.nodes_cluster_gateway,-1,sizeof(ct.nodes_cluster_gateway));
  for(int i=0;i<this->clusters.size();i++)
  {
    if(clusters[i].getSize() != 0)ct.nodes_cluster_head[i] =
clusters[i].getClusterHead().getMemberIndex();
  }

  struct point center;
  double cX,cY;
  for(int i=0;i<this->clusters.size();i++)
  {
    for(int j=0;j<this->clusters[i].getSize();j++)
    {
      cX+=clusters[i].members[j].getMemberPointX();
      cY+=clusters[i].members[j].getMemberPointY();
    }
  }
```

```
    center.x = cX/NUMBER_OF_NODE;
    center.y = cY/NUMBER_OF_NODE;
    for(int i=0;i<this->clusters.size();i++)
    {
        if(clusters[i].getSize() != 0)ct.nodes_cluster_gateway[i] =
clusters[i].getClusterGateway(center).getMemberIndex();
    }

    ct.nodes_cluster_timestamp = CURRENT_TIME;
    return ct;
}
```

A.11 Kode Fungsi getClusterHead()

```
Member Cluster::getClusterHead()
{
    struct point center = this->getCentroid();
    double nearestDistance = 1000000.00;
    int iter=0;
    for(int i=0;i<this->members.size();i++)
    {
        double temp = distanceMemberToCentroid(members[i],center);
        if(temp<nearestDistance)
        {
            nearestDistance=temp;
            iter=i;
        }
    }
    return members[iter];
}
```

A.12 Kode Fungsi getClusterGateway()

```
Member Cluster::getClusterGateway(struct point clustersCenter)
{
    double nearestDistance = 1000000.00;
    int iter=0;
    for(int i=0;i<this->members.size();i++)
    {
        double temp = distanceMemberToCentroid(members[i],clustersCenter);
        if(temp<nearestDistance)
        {
            nearestDistance=temp;
            iter=i;
        }
    }
```

```
    return members[iter];
}
```

A.13 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)                      905                                     ;
set opt(y)                      905                                     ;
set val(ifqlen)                 1000                                    ;
set val(nn)                     200                                     ;
set val(seed)                   1.0                                     ;
set val(adhocRouting)           AODV                                    ;
set val(stop)                   200                                     ;
set val(cp)                      "cbr.txt"                              ;
set val(sc)                     "scenario.txt"                          ;

set val(energy_mod)             EnergyModel
set val(energy_init)            150
set val(tx_power)               8.0
set val(rx_power)               6.0
set val(idle_power)             0.0
set val(sleep_power)            0.0

set ns_                 [new Simulator]

set topo    [new Topography]

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

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

set topo                [new Topography]
$topo load_flatgrid $opt(x) $opt(y)

set god_ [create-god $val(nn)]

$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) \
                -energyModel $val(energy_mod) \
                -initialEnergy $val(energy_init) \
                -txPower $val(tx_power) \
                -rxPower $val(rx_power) \
                -idlePower $val(idle_power) \
                -sleepPower $val(sleep_power) \
                -topoInstance $topo \
                -agentTrace ON \
                -routerTrace ON \
                -macTrace ON \
                -movementTrace ON

Phy/WirelessPhy      set          RXThresh_ 5.57189e-11 ;
Phy/WirelessPhy      set          CSThresh_ 5.57189e-11 ;

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

puts "Loading connection pattern..."
source $val(cp)

puts "Loading scenario file..."
source $val(sc)

for {set i 0} {$i < $val(nn)} {incr i} {
   $ns_ initial_node_pos $node_($i) 20
}

for {set i 0} {$i < $val(nn) } {incr i} {
   $ns_ at $val(stop).0 "$node_($i) reset";
}

$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.14 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_ 1.0
$cbr_(0) set random_ 1
$cbr_(0) set maxpkts_ 10000
$cbr_(0) attach-agent $udp_(0)
$ns_ connect $udp_(0) $null_(0)
$ns_ at 2.5568388786897245 "$cbr_(0) start"
$ns_ at 100.00 "$cbr_(0) stop"

set udp_(1) [new Agent/UDP]
$ns_ attach-agent $node_(198) $udp_(1)
set null_(1) [new Agent/Null]
$ns_ attach-agent $node_(199) $null_(1)
set cbr_(1) [new Application/Traffic/CBR]
$cbr_(1) set packetSize_ 512
$cbr_(1) set interval_ 1.0
$cbr_(1) set random_ 1
$cbr_(1) set maxpkts_ 10000
$cbr_(1) attach-agent $udp_(1)
$ns_ connect $udp_(1) $null_(1)
$ns_ at 101.5568388786897245 "$cbr_(1) start"
$ns_ at 200.00 "$cbr_(1) stop"
```

BIODATA PENULIS

ALCREDO SIMANJUNTAK, lahir di Balikpapan, 8 Juni 1998. Penulis adalah anak kedua dari tiga bersaudara. Penulis menempuh pendidikan sekolah dasar di SD Methodist Pematangsiantar, lalu melanjutkan pendidikan sekolah menengah pertama di SMP Bintang Timur Pematangsiantar, kemudian menempuh pendidikan sekolah menengah atas di SMAN 2 – Asrama Yayasan Soposurung Balige dan akhirnya menempuh pendidikan sarjana di Departemen Informatika, Fakultas Teknologi Informasi dan Komunikasi, Institut Teknologi Sepuluh Nopember, Surabaya.

Dalam menempuh gelar sarjana, penulis mengambil bidang minat Arsitektur Jaringan dan Komputer(AJK). Sebagai mahasiswa penulis berperan aktif dalam beberapa organisasi kampus seperti Schematics dan Himpunan Mahasiswa Teknik Komputer-Informatika(HMTC). Pada tahun 2018 penulis pernah melakukan magang di PT SCHLUMBERGER GEOPHYSICS NUSANTARA yakni perusahaan multinasional yang bergerak pada bidang oil service dan pada tahun 2019 penulis juga pernah magang di E-Life Solutions Sdn. Bhd. Johor Bahru, Malaysia. Penulis dapat dihubungi melalui nomor handphone 087808061998 atau email ke simajuntakalcredo@gmail.com.