TUGAS AKHIR – IF184802

IMPLEMENTASI K-MEANS CLUSTERING DENGAN METODE LINK COEFFICIENT PADA AODV UNTUK MENINGKATKAN STABILITAS RUTE DALAM MANETS

AKMAL DARARI RAFIF BASKORO
NRP 05111640000148

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

Dosen Pembimbing II
Ary Mazharuddin Shidiqi., S.Kom., M.Comp.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 DENGAN METODE LINK COEFFICIENT PADA AODV UNTUK MENINGKATKAN STABILITAS RUTE DALAM MANETS

AKMAL DARARI RAFIF BASKORO
NRP 05111640000148

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

Dosen Pembimbing II
Ary Mazharuddin Shidiqi., S.Kom., M.Comp.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 WITH COEFFICIENT LINK METHOD ON AODV FOR IMPROVING ROUTE STABILITY ON MANETS

AKMAL DARARI RAFIF BASKORO
NRP 05111640000148

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

Second Advisor
Ary Mazharuddin Shidiqi., S.Kom., M.Comp.Sc., Ph.D.

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

(Halaman ini sengaja dikosongkan)

LEMBAR PENGESAHAN

IMPLEMENTASI K-MEANS CLUSTERING DENGAN METODE LINK COEFFICIENT PADA AODV UNTUK MENINGKATKAN STABILITAS RUTE DALAM MANETS

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:
AKMAL DARARI RAFIF BASKORO
NRP: 05111640000148

Disetujui oleh Pembimbing Tugas Akhir

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

2. Ary Mazharuddin Shidiqi., S.Kom., M.Comp.Sc., Ph.D. ..... (Pembimbing 2)
   (NIP. 198106202005011003)

SURABAYA
JUNI, 2020

(Halaman ini sengaja dikosongkan)

IMPLEMENTASI K-MEANS CLUSTERING DENGAN METODE LINK COEFFICIENT PADA AODV UNTUK MENINGKATKAN STABILITAS RUTE DALAM MANETS

Nama Mahasiswa : Akmal Darari Rafif Baskoro
NRP : 05111640000148
Departemen : Teknik Informatika
Dosen Pembimbing 1 : Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.
Dosen Pembimbing 2 : Ary Mazharuddin Shidiqi., S.Kom., M.Comp.Sc., Ph.D.

Abstrak

Adhoc On-Demand Distance Vector (AODV) adalah salah satu proses *routing* yang dapat diimplementasikan pada *mobile adhoc network*(MANET). Sejatinya, AODV mempunyai dua fase, pertama adalah *Route Discovery* dan yang kedua adalah *Route Maintenance. Route Discovery* adalah fase dimana AODV melakukan proses pencarian suatu rute menuju tujuan oleh *source node.* Sedangkan *Route Maintenance* adalah sebuah proses yang bertujuan untuk mengecek keadaan *route* yang sudah ada.

Pada AODV, rute yang dipilih adalah *route* dengan jumlah lompatan 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* seperti posisi, energi, kepadatan, dan kekuatan sinyal. Penggunaan faktor-faktor tambahan tersebut akan meningkatkan akurasi didalam pemilihan rute yang terbaik untuk AODV.

Pada Tugas Akhir ini diusulkan suatu algoritma *routing* sebagai modifikasi dari AODV yaitu dinamakan AODV *with K-Means and Link Coefficient*. 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), dan *Residual Energy Metric*(REM) yang akan menjadi pertimbangan didalam pemilihan *route*.

Kata kunci: MANET, AODV, K-Means, LCF, Coefficient Link

IMPLEMENTATION OF K-MEANS CLUSTERING WITH COEFFICIENT LINK METHOD ON AODV FOR IMPROVING ROUTE STABILITY ON MANETS

Student's Name	**: Akmal Darari Rafif Baskoro**
Student's ID	**: 05111640000148**
Department	**: Informatics Engineering**
First Advisor	**: Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.**
Second Advisor	**: Ary Mazharuddin Shidiqi., S.Kom., M.Comp.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 is a phase where AODV doing some route searching process to destination with source node, and Route Maintenance is a process for checking route conditions in existing routes.

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 as energy, congestion, and signal strength. The using of that additional factors will enhance accuracy in choosing route for AODV.

Consequently, In this thesis, routing algorithm as a modification of AODV called AODV with K-Means and Link Coefficient 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), and Residual Energy Metric(REM) which will be the considerations in route election.

Keyword: MANET, AODV, K-Means, LCF Coefficient Link

KATA PENGANTAR

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

Puji syukur penulis panjatkan ke hadirat Allah SWT, atas rahmat, barokah, dan ridho-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul ***"Implementasi K-Means Clustering dengan metode Link Coefficient pada AODV untuk Meningkatkan Stabilitas Rute Dalam MANETS".***

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 Yanuariadi Kusuma Baskoro, Ibu Indah Cahyani) 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 Ary Mazharuddin Shidiqi., S.Kom., M.Comp.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. Teman saya yang bernama Irham Muhammad Fadhil dan Alcredo Simanjuntak yang telah memberikan saya dukungan baik berupa bantuan berupa doa dan moral dan materi-materi yang saya butuhkan dalam pengerjaan Tugas Akhir ini
6. Teman-teman dari BrainWash Syndicate yang telah sudah memberikan saya dukungan baik berupa doa dan moral dalam pengerjaan Tugas Akhir ini.

7. Teman-teman dari Tim Sukses Kampanye Rumah Manfaat dan juga teman saya yang bernama Daud Wahyu Imani yang telah sudah memberikan saya dukungan baik berupa doa dan moral dalam pengerjaan Tugas Akhir ini.
8. 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, Juni 2020

Akmal Darari Rafif Baskoro

(Halaman ini sengaja dikosongkan)

DAFTAR ISI

Abstrak v
Abstract vii
KATA PENGANTAR ix
DAFTAR ISI xii
DAFTAR GAMBAR xv
DAFTAR TABEL xvii
BAB I PENDAHULUAN 1
1.1 Latar Belakang 1
1.2 Rumusan Masalah 2
1.3 Batasan Permasalahan 2
1.4 Tujuan 3
1.5 Manfaat 3
1.6 Metodlogi 3
1.6.1 Penyusunan Proposal Tugas Akhir 3
1.6.2 Studi Literatur 4
1.6.3 Analisis dan Desain Sistem 4
1.6.4 Implementasi Sistem 4
1.6.5 Pengujian dan Evaluasi 4
1.6.6 Penyusunan Buku 5
1.7 Sistematika Penulisan Laporan 5
BAB II TINJAUAN PUSTAKA 8
2.1 *Mobile Ad-hoc Networks* (MANETs) 8
2.2 *Ad-hoc On-demand Distance Vector* (AODV) 10
2.2.1 Pesan Kontrol 11
2.2.2 *Route Discovery* 12
2.2.3 *Route Maintenance* 14
2.3 *K-Means Clustering* 15
2.4 *Coefficient Link* 16
2.5 *Network Simulator-2* (NS2) 18
2.5.1 Instalasi 19

2.5.2 *Trace File* 21
2.6 OpenStreetMap(OSM) 22
2.7 Java OpenStreetMap(JOSM) 23
2.8 Simulation of Urban Mobilitiy(SUMO) 23
2.9 AWK 24
BAB III PERANCANGAN 26
3.1 Deskripsi Umum 26
3.2 Perancangan Skenario Mobilitas 29
3.2.1 Perancangan Skenario Grid 30
3.2.2 Perancangan Skenario Real 31
3.3 Perancangan Modifikasi *Routing* Protocol AODV 32
3.3.1 Perancangan *K-Means Clustering* 32
3.3.2 Perancangan *Coefficient Link* 36
3.4 Perancangan Simulasi pada NS2 38
3.5 Perancangan Metrik Analisis 38
3.5.1 Packet Delivery Ratio (PDR) 38
3.5.2 Average End-to-End Delay (E2E) 39
3.5.3 *Routing* Over*head* (RO) 39
BAB IV IMPLEMENTASI 42
4.1 Implementasi Skenario *Grid* 42
4.2 Implementasi Skenario *Real* 45
4.3 Implementasi *K-Means Clustering* 47
4.3.1 Modifikasi Pengiriman *Route Request*(RREQ) 50
4.3.2 Modifikasi Penerimaan *Route Request*(RREQ) 50
4.3.3 Modifikasi Pengiriman *Route Reply*(RREP) 51
4.3.4 Modifikasi Penerimaan *Route Reply*(RREP) 52
4.3.5 Modifikasi Pengiriman *Hello Messages* 53
4.3.6 Modifikasi Penerimaan *Hello Messages* 53
4.3.7 Modifikasi Time Scheduling 55
4.4 Implementasi *Coefficient Link* (LCF) 55
4.4.1 Modifikasi Pengiriman *Route Request*(RREQ) 55
4.4.2 Modifikasi Penerimaan *Route Request*(RREQ) 56
4.4.3 Modifikasi Pengiriman *Route Reply*(RREP) 58
4.4.4 Modifikasi Penerimaan *Route Reply*(RREP) 59
4.5 Implementasi Simulasi pada NS2 59
4.5.1 Implementasi Simulasi pada *K-Means Clustering* 60

4.6 Implementasi Metrik Analisis 61
4.6.1 Implementasi *Packet Delivery Ration(PDR)* 61
4.6.2 Implementasi *Average End-to-End Delay(E2E)* 62
4.6.3 Implementasi *Routing Overhead(RO)* 63
BAB V UJI COBA DAN EVALUASI 66
5.1 Lingkungan Uji Coba 66
5.2 Cara Uji Coba 66
5.3 Skenario Grid 67
5.4 Skenario Real 72
5.5 Perbandingan Hasil AODV Modifikasi Pada Skenario *Grid* dan *Real* 76
BAB VI KESIMPULAN DAN SARAN 79
6.1 Kesimpulan 79
6.2 Saran 79
DAFTAR PUSTAKA 81
LAMPIRAN 84
A.1 Kode Fungsi send*Request*() 84
A.2 Kode Fungsi recv*Request*() 87
A.3 Kode Fungsi send*Reply*() 94
A.4 Kode Fungsi recv*Reply*() 95
A.5 Kode Fungsi sendHello() 98
A.6 Kode Fungsi recvHello() 99
A.7 Kode Fungsi rt_*update*() 100
A.8 Kode Fungsi runCluster() 100
A.9 Kode Fungsi run() 101
A.10 Kode Fungsi getResult() 102
A.11 Kode Fungsi getClusterHead() 103
A.12 Kode Fungsi getClusterGateway() 103
A.13 Kode Skenario NS2 104
A.14 Kode Konfigurasi Traffic 106
A.15 Kode Dalam File AODV.h 106
A.16 Kode Dalam File AODV_packet.h 112
A.17 Kode Dalam File AODV_rtable.h 114
A.18 Hasil Simulasi AODV Real yang sudah di modifikasi 117
A.19 Hasil Simulasi AODV Grid yang sudah di modifikasi 118
BIODATA PENULIS 120

DAFTAR GAMBAR

Gambar 2.1 Jaringan MANETs 10
Gambar 2.2 Contoh AODV 11
Gambar 2.3 Flowchart Pemrosesan Pesan Kontrol 13
Gambar 2.4 Proses *Route Maintenance* 14
Gambar 2.5 Flowchart *K-Means Clustering* 16
Gambar 2.6 Flowchart *Coefficient Link* 18
Gambar 2.7 Arsitektur dasar dari *Network Simulator* 19
Gambar 2.8 Perintah untuk memasang paket untuk persyaratan pemasangan NS2 19
Gambar 2.9 Cara mengekstrak paket NS2 yang telah diunduh 19
Gambar 2.10 Baris Kode yang Diubah Pada File MakeFile.in 20
Gambar 2.11 Perintah Instalasi NS2 20
Gambar 2.12 Skrip instalasi GCC versi 5 20
Gambar 2.13 Pembuatan lintasan cepat dan menjalankan NS2 21
Gambar 3.1 Diagram perencanaan simulasi AODV modifikasi 27
Gambar 3.2 Alur Perancangan Skenario 30
Gambar 3.3 *Pseudocode* Pemindahan Data Posisi *Node* 33
Gambar 3.4 *Pseudocode* Proses *K-Means Clustering* 34
Gambar 3.5 *Pseudocode* Proses Pemilihan *Cluster Head* 35
Gambar 3.6 *Pseudocode* Proses Pemilihan *Cluster Gateway* 35
Gambar 3.7 *Pseudocode* Pemindahan Informasi Hasil *Clustering* 36
Gambar 3.8 *Pseudocode* Pemindahan Informasi Hasil *Coefficient Link* 37
Gambar 4.1 Perintah *netgenerate* 42
Gambar 4.2 Hasil *Generate* Peta *Grid* 43
Gambar 4.3 Perintah randomTrips.py 43
Gambar 4.4 Perintah duarouter 43
Gambar 4.5 File skrip .sumocfg 44
Gambar 4.6 Perintah SUMO untuk Membuat skenario.xml 44
Gambar 4.7 Perintah traceExporter.py 45
Gambar 4.8 Ekspor peta dari *OpenStreetMap* 45

Gambar 4.9 Perintah netconvert 46
Gambar 4.10 Hasil konversi peta *real* 46
Gambar 4.11 Proses Inisiasi *Cluster* 47
Gambar 4.12 Proses *K-Means Clustering* 49
Gambar 4.13 Proses Pengambilan *Cluster Head* dan *Cluster Gateway* 49
Gambar 4.14 Modifikasi Pengiriman RREQ untuk *K-Means* 50
Gambar 4.15 Modifikasi Penerimaan RREQ untuk *K-Means* 51
Gambar 4.16 Modifikasi Pengiriman RREP untuk *K-Means* 52
Gambar 4.17 Modifikasi Penerimaan RREP untuk *K-Means* 53
Gambar 4.18 Modifikasi Pengiriman *Hello Messages* 53
Gambar 4.19 Modifikasi Penerimaan *Hello Messages* 55
Gambar 4.20 Modifikasi *Time Scheduling* 55
Gambar 4.21 Modifikasi Pengiriman RREQ untuk LCF 56
Gambar 4.22 Modifikasi Penerimaan RREQ untuk LCF 58
Gambar 4.23 Modifikasi Pengiriman RREP untuk LCF 59
Gambar 4.24 Modifikasi Penerimaan RREP untuk LCF 59
Gambar 4.25 Konfigurasi Lingkungan Simulasi 60
Gambar 4.26 Konfigurasi File AODV.h 61
Gambar 4.27 File Metrik Analisis 61
Gambar 4.28 File pdr.awk 62
Gambar 4.29 File e2e.awk 63
Gambar 4.30 File ro.awk 64
Gambar 5.1 File skrip.sh 67
Gambar 5.2 Tambahan skrip di dalam file *MakeFile* 67
Gambar 5.3 Skrip *Compile* 67
Gambar 5.4 Grafik PDR Hasil Simulasi Skenario *Grid* 68
Gambar 5.5 Grafik E2E Hasil Simulasi Skenario *Grid* 70
Gambar 5.6 Grafik RO Hasil Simulasi Skenario *Grid* 71
Gambar 5.7 Grafik PDR Hasil Simulasi Skenario *Real* 73
Gambar 5.8 Grafik E2E Hasil Simulasi Skenario *Real* 74
Gambar 5.9 Grafik RO Hasil Simulasi Skenario *Real* 75

DAFTAR TABEL

Tabel 2.1 Format RREQ 11
Tabel 2.2 Format RREP 12
Tabel 2.3 Detail Informasi *Trace* File AODV 22
Tabel 2.4 Alat pada *Simulation of Urban Mobility* (SUMO)..... 24
Tabel 3.1 Daftar Istilah 28
Tabel 3.2 Tabel Posisi *Node* 32
Tabel 3.3 *Nodes Clusters Table* 35
Tabel 3.4 Konfigurasi Lingkungan Simulasi 38
Tabel 5.1 Spesifikasi Perangkat yang Digunakan..... 66
Tabel 5.2 PDR Hasil Simulasi Skenario *Grid*..... 68
Tabel 5.3 E2E Hasil Simulasi Skenario *Grid* 70
Tabel 5.4 RO Hasil Simulasi Skenario *Grid*..... 71
Tabel 5.5 PDR Hasil Simulasi Skenario *Real*..... 72
Tabel 5.6 E2E Hasil Simulasi Skenario *Real*..... 74
Tabel 5.7 RO Hasil Simulasi Skenario Real 75
Tabel 5.8 Perbandingan hasil PDR pada AODV *Grid* dan AODV *Real*..... 76
Tabel 5.9 Perbandingan hasil E2E pada AODV *grid* dan AODV *real*..... 76
Tabel 5.10 Perbandingan hasil RO antara AODV Grid dan AODV Real 77

BAB I PENDAHULUAN

1.1 Latar Belakang

Teknologi komunikasi nirkabel pada saat ini telah berkembang secara pesat, hal ini dapat dilihat dari keinginan pengguna yang dapat digunakan untuk berkomunikasi dimanapun dikarenakan kelebihan dari jaringan nirkabel sendiri yang dapat digunakan dimanapun dan kapanpun tanpa harus menetap di suatu tempat seperti pada jaringan komunikasi berkabel.

Terdapat beberapa jenis dari jaringan nirkabel berdasarkan cara untuk berkoneksi yang dapat dikelompokan atas 2 bagian yaitu jaringan dengan *Fixed Infrastructure* dan *Wireless Adhoc Network. Fixed Infrastructure* menggunakan *access point* sebagai suatu pembeda dengan *wireless adhoc network*. *Access point* dalam *Fixed Infrastructure* berfungsi sebagai *router* atau jembatan yang menghubungkan antar *node* di dalam jaringan tersebut dan sekaligus menyimpan semua informasi yang ada dialam jaringan tersebut, contohnya dapat dilihat pada GSM (*Global System for Mobile Communication*) dan UMTS (*Universal Mobile Telecommunication System*). Hal ini berbanding terbalik dengan *wireless adhoc network* yang tidak mempunyai infrastruktur yang tetap dan *access point* sama sekali. Topologi pada *wireless adhoc network* berubah secara dinamis karena *node* yang masuk dan keluar dari jaringan tersebut. Disebabkan tidak adanya *access point* di dalam *wireless adhoc network* maka *node* yang berada didalamnya berperan juga sebagai *router* ataupun jembatan antar *node* lainnya dalam pengiriman pesan kontrol.

Dalam pengiriman pesan kontrol pada *wireless adhoc network* diperlukan adanya proses *routing* dari *node* pusat ke *node* destinasi yang handal agar pesan kontrol tersebut dapat sampai secara utuh. Dalam hal ini terdapat beberapa jenis *routing* yang dapat digunakan yang diklasifikasikan atas 3 bagian yaitu *Proactive Routing, Reactive Routing,* dan *Hybrid Routing. Reactive Routing* adalah sebuah *routing* yang dilakukan ketika suatu node akan melakukan pengiriman pesan kontrol ke *node* destinasi, maka proses pencarian rute akan dilakukan. *Reactive*

Routing yang berdasarkan vektor jarak juga terbukti dapat menurunkan secara signifikan biaya atau usaha yang diberikan dalam melakukan *routing*. Dalam beberapa observasi yang membandingkan antara *reactive routing* dan *proactive routing* menunjukan bahwa *reactive routing* unggul dari sisi rasio pengiriman paket, biaya routing, efisiensi energi, dan stabilitas.

Sering kali, dikarenakan pergerakan yang dinamis dalam *MANETs*, juga akan membuat rentan dalam serangan sekaligus juga pemilihan jalur juga hal yang vital dalam jaringan ini. Karena itu, dibutuhkan stabilitas dalam pemilihan rute dalam *MANETs*.

Pada tugas akhir ini akan dibahas sebuah metode yang dapat menstabilkan pemilihan rute yang dipilih pada saat berkomunikasi, yaitu dengan penerapan metode *clustering*. *Routing* ini bergantung pada tingkat energi dan perpindahan *node* pada suatu jaringan. Perkiraan biaya energi dihitung berdasarkan energi yang digunakan pada suatu *node* dalam penerapan *K-Means Clustering* yang diikuti oleh AODV dengan faktor *Coefficient Link* dengan perhitungan kecepatan antar *node* supaya dapat meningkatkan stabilitas pada saat berkomunikasi dengan menggunakan MANETs.

1.2 Rumusan Masalah

Tugas Akhir ini mengangkat beberapa rumusan masalah sebagai berikut:

1. Bagaimana melakukan implementasi *K-Means Clustering* terhadap seluruh *node* yang berada pada suatu jaringan?
2. Bagaimana melakukan optimasi pemilihan *route* terbaik berdasarkan *signal strength*, *energy*, dan *congestion*, protokol *Adhoc On-Demand Distance Vector* dengan menggunakan metode *Coefficient Link* ?

1.3 Batasan Permasalahan

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

1. Jaringan yang digunakan adalah jaringan *Mobile Ad hoc Networks* (MANET).
2. Simulasi pengujian jaringan menggunakan *Network Simulator 2* (NS2) dengan bahasa pemrograman C++.
3. Pembuatan skenario uji coba menggunakan *Simulation of Urban Mobility* (SUMO).
4. Melakukan evaluasi performa di setiap rute dengan menggunakan *performance metrics*, antara lain PDR *(Packet Delivery Ratio), Energy Consumption, Delay, Packet Loss.*

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 rute terbaik berdasarkan beberapa faktor tambahan di lingkungan MANETs.

1.5 Manfaat

Manfaat yang diperoleh dari pengerjaan Tugas Akhir ini adalah mengetahui faktor lain dalam memilih *route* terbaik di lingkungan MANETs.

1.6 Metodlogi

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 MANET, AODV, *K-Means Clustering, Network Simulator* 2 (NS2), *Coefficient Link, 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 *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 MANET. Setelah itu, dilakukan simulasi dengan bantuan SUMO.

1.6.4 Implementasi Sistem

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

1.6.5 Pengujian dan Evaluasi

Pada tahap ini dilakukan pengujian menggunakan SUMO, supaya dapat membuat simulasi dengan keadaan yang diinginkan. Hasil dari simulasi yang sudah ditambah oleh 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*, peralatan, serta definisi yang digunakan dalam pembuatan Tugas Akhir.

2.1 *Mobile Ad-hoc Networks* (MANETs)

Ad Hoc Network adalah jaringan nirkabel yang terdiri dari kumpulan mobile node (mobile station) yang bersifat dinamik dan spontan, dapat diaplikasikan di mana pun tanpa menggunakan jaringan infrastruktur (seluler ataupun PSTN) yang telah ada. Contoh mobile node adalah notebook, PDA dan ponsel. Jaringan ad hoc disebut juga dengan spontaneous network atau disebut MANET [1].

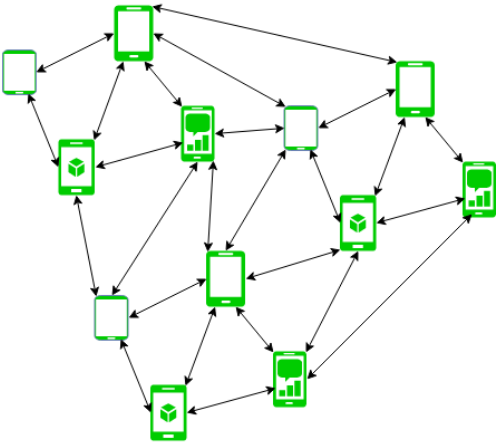
MANET ini berhubungan dengan IP bergerak dan DHCP. Ketika IP bergerak dan DHCP memegang peranan dalam koneksi dengan perangkat bergerak ke infrastruktur yang tetap, maka MANET berisi router bergerak. MANET juga berhubungan dengan pengembangan protokol dan komponen supaya dapat membentuk jaringan ad-hoc diantara perangkat bergerak [2].

MANET memiliki beberapa karakteristik yaitu:

- Topologi yang Dinamis : *Node* pada MANET dapat berpindah-pindah secara bebas dan topologinya (yang biasanya *multihop*) dapat berganti secara acak dan cepat dalam waktu yang tidak dapat diprediksi, dan dapat berisikan jalur yang terarah maupun tidak terarah [3].
- Bandwidth yang Dibatasi, dengan kapasitas jalur yang bervariasi : Jalur (*link*) nirkabel dapat mempunyai kapasitas yang rendah secara signifikan dan terus menerus dibandingkan dengan yang berkabel. Dalam *throughput* yang terealisasi dalam komunikasi nirkabel

setelah menghitung dari akses yang banyak, *fading, noise,* dan kondisi interfensi, biasanya kurang dari rata-rata transmisi maksimal. Salah satu efek dari kapasitas jalur yang rendah hingga menengah adalah sering terjadinya kemacetan pada saat berkomunikasi [3].

- Pengoperasian Energi yang Dibatasi : Dikarenakan beberapa atau semua *node* dalam MANET yang berhubungan dengan baterai atau energi yang cepat habis, konservasi energi adalah hal yang terpenting dalam optimalisasi dalam kriteria desain sistem [3].
- Keterbatasan Keamanan : Karena jaringan nirkabel biasanya lebih tidak tahan terhadap ancaman keamanan dibandingkan jaringan berkabel, kemungkinan gangguan keamanan seperti *eavesdropping, spoofing,* dan *denial-of-service attacks* harus sangat dipertimbangkan secara hati-hati. Keamanan jalur yang sudah ada biasanya diaplikasikan didalam jaringan nirkabel untuk mengurangi ancaman terhadap keamanan [3].
- Sedikit Interfensi oleh Manusia: MANETs tidak perlu banyak interfensi oleh manusia dikarenakan MANETs dapat terkonfigurasi secara *autonomous* [4].



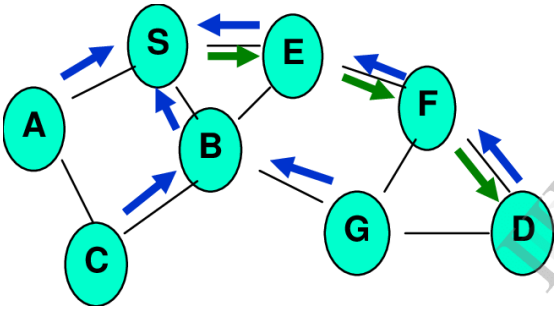
Gambar 2.1 Jaringan MANETs

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

Ad-hoc On-demand Distance Vector (AODV) adalah salah satu dari jenis jalur protokol reaktif untuk jaringan *ad hoc* dengan pendekatan secara langsung dimana rute jaringan dibuat hanya pada saat dibutuhkan yaitu pada saat suatu node sumber akan mengirimkan sebuah paket ke suatu node tujuan [5].

AODV merupakan *routing protocol* yang dapat digunakan dalam MANETs. AODV menawarkan adaptasi yang cepat dalam kondisi jalur yang dinamis, *processing* yang rendah dan *overhead* memori, ultilisasi jaringan yang rendah, dan menjabarkan rute *unicast* ke destinasi didalam jaringan *ad-hoc* [6].

Selai itu, AODV juga termasuk dalam kategori *reactive routing protocol*. Seperti *reactive routing protocol* lainnya, AODV hanya menyimpan informasi *routing* seputar *path* dan *host* yang aktif. Didalam AODV, informasi *routing* disimpan di semua *node*. Setiap *node* menyimpan tabel *routing next-hop*, dimana menyimpan informasi tujuan ke *hop* berikutnya dimana *node* tersebut memiliki *route* tertentu. Didalam AODV, ketika *node* asal ingin mengirim *packet* ke tujuan namun tidak ada *route* yang tersedia, *node* tersebut akan memulai proses *route discovery* [7].



Gambar 2.2 Contoh AODV

2.2.1 Pesan Kontrol

AODV mempunyai 4 jenis pesan kontrol antara lain *Route Request*(RREQ), *Route Reply*(RREP), *Route* Error(RERR), dan *Hello Message* [7].

a *Route Request* (RREQ)

Ketika rute yang digunakan untuk mencapai tujuan tidak bisa digunakan maka paket RREQ akan membanjiri jaringan dengan format seperti yang ditunjukan pada Tabel 2.1 [7].

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

Tabel 2.1 Format RREQ

b *Route Reply*(RREP)

Jika sebuah *node* tujuan atau *node* yang bukan merupakan jalur yang benar, maka node tersebut akan mengirimkan RREP

kembali ke *node* pengirim dengan format seperti yang ditunjukan pada Tabel 2.2 [7].

Source Address
Destination Address
Destination Sequence No
Hop Count
Life Time

Tabel 2.2 Format RREP

c *Route* Error (RERR)

Setiap *node* mengawasi semua *node* tetangga dan penyebaran pesan ketika *node* tujuan tidak lagi ditemukan, jika data yang seharusnya dikirimkan ke sebuah *node* akan tetapi *node* tersebut tidak memiliki rute aktif dan rute tidak diperbaiki, jika *node* menerima RERR dari *node* tetangga untuk satu atau lebih dari rute yang aktif, untuk memberikan informasi ke *node* yang lainnya dari kedua sisi tentang ketidaktersediaan jaringan [7].

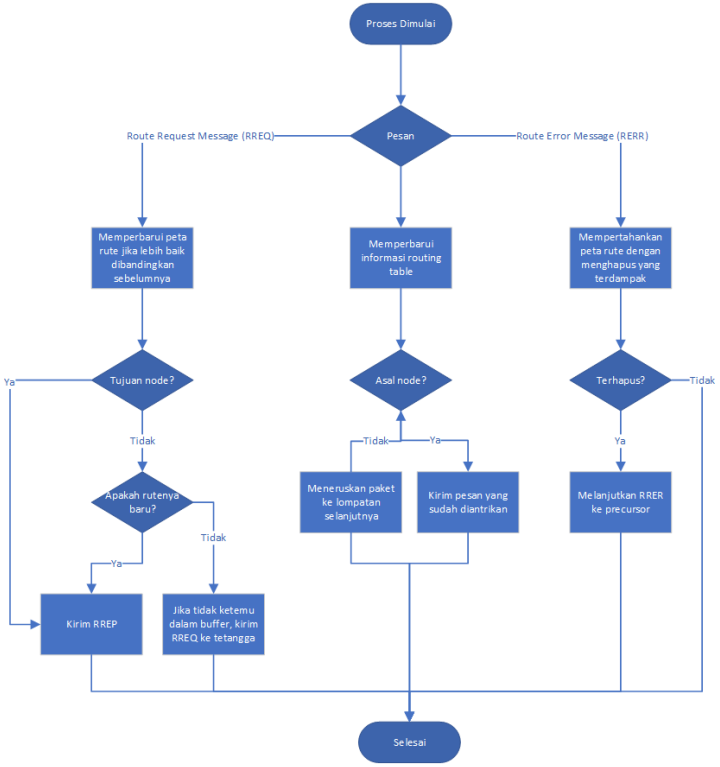
d *Hello Messages*

Setiap *node* dapat mengetahui *node* tetangganya dengan menggunakan *local broadcast*, biasa disebut *HELLO message*. *Node* tetangga adalah *node* yang bisa terhubung secara langsung. AODV menggunakan *HELLO message* untuk melakukan pembaharuan *table routing* karena AODV adalah *reactive* sehingga diperlukan pembaharuan rute secara berkala [7].

2.2.2 *Route Discovery*

Route discovery adalah proses pencarian suatu *route* menuju *destination* oleh *source node*. Didalam proses *route discovery*, *node* asal membroadcast *route request* (RREQ) *packets* dimana disertakan nomor *sequence* tujuan. Ketika *node* tujuan atau *node* yang memiliki *route* ke tujuan menerima *packet* RREQ, *node* tersebut akan memeriksa nomor *sequence* tujuan yang sampai pada *node*nya ketika *packet* tiba dan nomor *sequence* sama didalam RREQ. Untuk memastikan bahwa *packet* tersebut masih

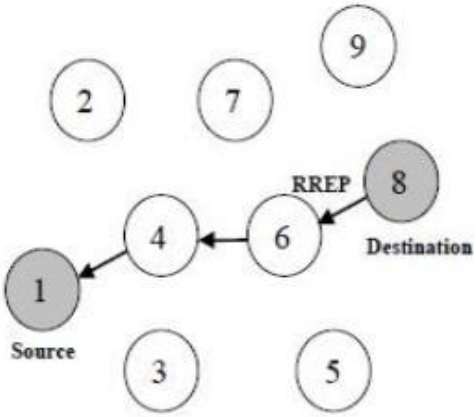
bersifat baru, *node* tujuan membalas paket RREQ dengan *route reply* (RREP) *packet*. RREP dibuat dan dikirim kembail ke *node* asal hanya jika nomor *sequence* tujuan sama dengan atau lebih besar dari yang dispesifikasikan di RREQ. AODV hanya menggunakan jalur yang bersifat simetris dan RREP mengikuti *path* sebaliknya dari *path* yang dihasilkan oleh RREQ. Ketika menerima RREP, setiap *node* diantara asal dan tujuan mengupdate *routing* table *nexthop* dengan RREP ke tujuannya. *Node* asal kemudian memilih *route* dengan jumlah *hop* paling sedikit untuk mengirimkan *packet* tujuannya [7].



Gambar 2.3 Flowchart Pemrosesan Pesan Kontrol

2.2.3 *Route Maintenance*

Route maintenance adalah sebuah proses yang bertujuan untuk mengecek keadaan *route* yang sudah ada. Perpindahan *node* dari jaringan *ad-hoc* berimbas pada rute yang berisikan *node* seperti pada *path* yang aktif. Jika *node* pusat berpindah pada saat proses transmisi data, dapat memulai ulang pencarian rute untuk membuat rute baru menuju destinasi. Ketika tujuan atau beberapa *node intermediate* berpindah, maka pesan RERR terkirim kepada *node* pusat yang terimbas. RERR dimulai dengan *node upstream* pada saat *break*, dan menuliskan setiap destinasi yang tidak dapat dicapai karena jalur yang hilang. *Node* yang rusak menyebarkan RERR kepada tetangga. Ketika tetangga menerima RERR, mereka menuliskan rute ke destinasi yang salah dengan menyunting destinasi dari sama menuju tak terbatas. Ketika *node* pusat menerima RERR, mereka dapat memulai ulang proses pencarian rute ketika rute tersebut masih dibutuhkan [8].



Gambar 2.4 Proses *Route Maintenance*

Ada dua jenis dari *route maintenance.* Salah satunya adalah pemulihan proses pencarian rute pusat yang dimaksud pada Gambar 2.4 yang rute pusat menyebarkan RREQ ke RREQ tetangga yang mempunyai alamat IP pusat dan tujuan dan ID

broadcast dan bilangan sekuensial. Ketika tujuan mengirimkan RREP, akan diterima oleh *node* pusat didalam *intermediate node.* Cara kedua adalah pemulihan rute lokal dengan menggunakan *intermediate node*. Didalam metode ini, jalur yang rusak dipulihkan oleh *intermediate node. Intermediate node* menyebarkan RREQ ke tetangga dan meminta rute saat di tujuan. Tujuan memberikan balasan dengan RREP dan rute tersebut terbentuk kembali mulai dari asal ke tujuan [8].

2.3 *K-Means Clustering*

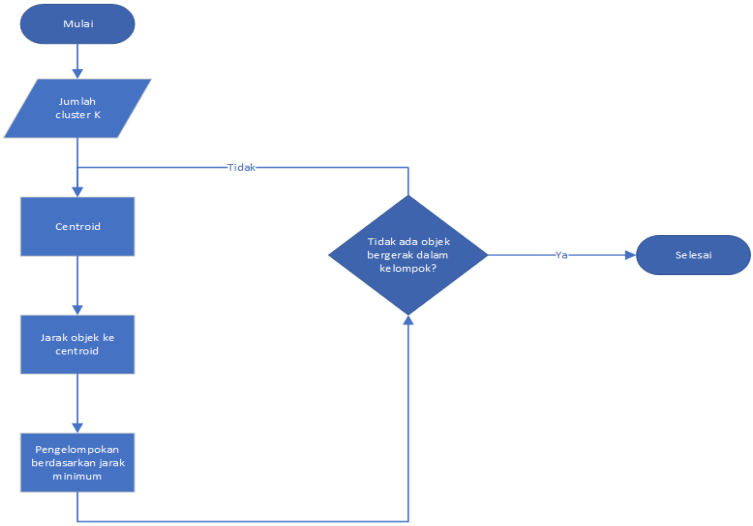
K-Means Clustering merupakan salah satu teknik pengurutan yang cepat dibandingkan pengelompokan secara hieraki dengan banyak variabel dengan memproduksi *cluster* yang rapat dibandingkan dengan *clustering* secara hieraki [9]. Ada empat urutan utama dalam proses algoritma *K-Means Clustering*:

- Langkah 1 : *K Cluster* dibentuk dari SN dengan meng*compile* bilangan K dari *centeroid* dalam tempat yang berbeda [9].
- Langkah 2 : Menghitung *Euclidean Distance* dari setiap SN ke *centroid* yang dikomputasikan untuk membuat *cluster* pertama. Disarankan untuk setiap *node* harus yang terdekat dari *centroid*. Untuk rumusnya terdapat dibawah ini.

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

  Dimana x dan y adalah koordinat yang merepresentasikan x_1,x_2 dan y_1,y_2 [9].
- Langkah 3 : Di dalam suatu situasi dalam setiap node, akan diperiksa dari posisi sebelumnya dan setiap luas *cluster* akan diciptakan kedalam jaringan [9].
- Langkah 4 : Jika lokasi dari *centroid* berubah, dan bisa mengulangi kembali dari langkah kedua untuk mendapatkan hasil *cluster* yang efisien atau memproses *clustering* hingga akhir. Terakhir, *centeroid* yang ditugaskan dari *K-Means Clustering* menjadi *Cluster Head* dari kelompok di dalam jaringan [9].

Gambar 2.5 akan menjelaskan alur bekerjanya proses *K-Means Clustering*.



Gambar 2.5 Flowchart *K-Means Clustering*

2.4 *Coefficient Link*

Coefficient link adalah faktor utama dari kecepatan diantara *node* yang mempengaruhi masa berlaku dari hubungan *node*. Ketika kecepatan dari dua *node* relatif tinggi, maka jarak diantara *node* akan bertambah secara cepat, dan menyebabkan interupsi dari hubungan [10].

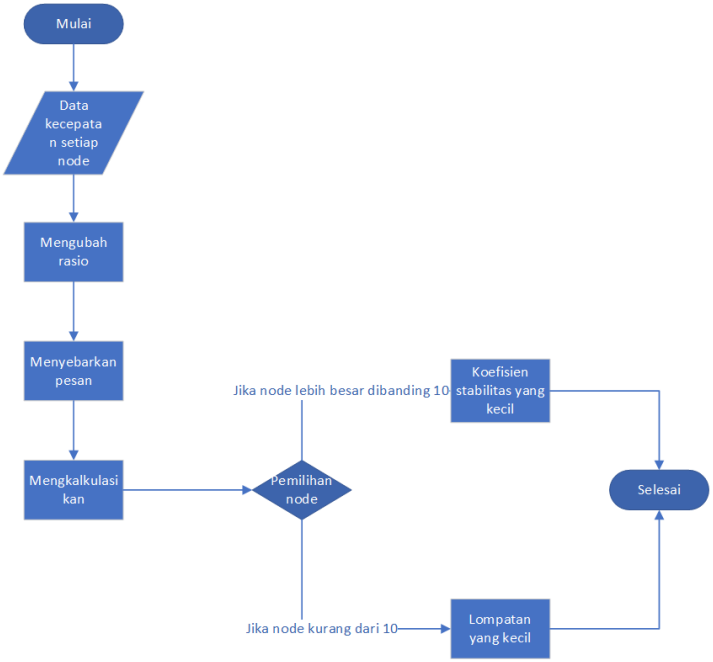
Jumlah dari koefisien stabilitas dari setiap kedua *node* yang sama dapat didefinisikan menjadi faktor stabilitas hubungan, yang berbasis dengan kecepatan *node* secara horizontal dan vertikal lalu dijumlahkan sehingga terbentuklah koefisien stabilitas diantara *node* [10].

Basis dari *coefficient link* ini adalah protokol *routing* RREQ dan RREP. Dalam fase permintaan (*request*), *source code* meminta kecepatan dari diri sendiri dan *node* tetangga mengubah rasio sehingga dapat disimpan secara terpisah. Lalu, rute tersebut

menyebarkan pesan permintaan kepada *node* tetangga, ketika *node* tetangga menerima rute, *node* akan mengkalkulasikan koefisien stabilitas diantara lompatan *node* terakhir dan dirinya. Lalu, memperbarui jumlah koefisien stabilitas di dalam pesan permintaan *routing*. Selain itu juga menerima pesan permintaan *routing* yang telah diperbarui kedalam kecepatan yang dipunyai. Setelah itu, meneruskan RREQ hingga *node* tujuan menerimanya.

Node tujuan akan memilih pesan permintaaan *route* melalui rasio pergantian *node* tetangga atau jumlah jalur koefisien stabilitas untuk membalas sebelum membuat jalur. Ketika *node* lebih besar dibanding sepuluh, maka *node* tujuan akan memilih rute yang mempunya jumlah koefisien stabilitas yang kecil. Selain itu juga dapat dipilih melalui jalur yang mempunyai lompatan yang kecil akan terpilih.

Di bawah ini adalah *flowchart* dari cara kerja *coefficient link*

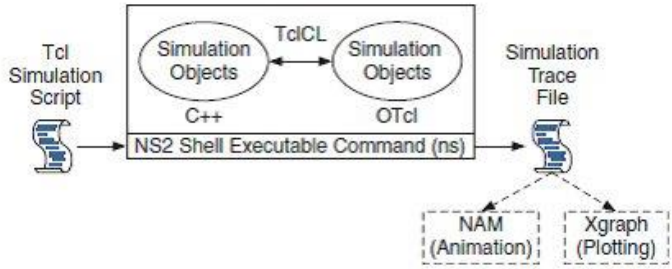


Gambar 2.6 Flowchart *Coefficient Link*

2.5 *Network Simulator-2* (NS2)

NS2 adalah singkatan dari *Network Simulator Version 2*. Program simulasi ini bersifat platform terbuka *(open-source)* yang didesain secara spesifik untuk penelitian dalam jaringan komunikasi komputer.

NS2 berisi dua bahasa pemrograman kunci: C++ dan Object-oriented Tool Command Language (OTcl). Ketika bahasa C++ mendefinisikan mekanisme internalnya *(backend)* dalam objek simulasi, OTcl membuat simulasi dengan menyusun dan mengkonfigurasi object sekaligus juga menjadwalkan penugasan yang diskrit. Bahasa C++ dan OTcl saling berhubungan satu sama lain dengan TclCL. Proses simulasi pada NS-2 akan memberikan output berupa *file* NAM dan trace *file* [11].



Basic architecture of NS.

Gambar 2.7 Arsitektur dasar dari *Network Simulator*

2.5.1 Instalasi

NS2 membutuhkan beberapa paket yang harus ada sebelum proses intalasi dilakukan. Dalam proses install paket yang dibutuhkan dapat dilakukan dengan perintah yang ditunjukkan pada Gambar 2.8.

```
sudo apt install build-essential autoconf automake libxmu-dev wget gdb bzip2 flex
nano nam software-properties-common python python-pip python-matplotlib
```

Gambar 2.8 Perintah untuk memasang paket untuk persyaratan pemasangan NS2

Selanjutnya, ekstrak paket NS2 dengan cara seperti pada gambar 2.9

```
cd ~/Downloads
mkdir ../ns2
mv ns-allinone-2.35_gcc5.tar.gz ../ns2/ns-allinone-2.35_gcc5.tar.gz
cd ../ns2
tar -xvzf ns-allinone-2.35_gcc5.tar.gz
```

Gambar 2.9 Cara mengekstrak paket NS2 yang telah diunduh

Kemudian, lakukan perubahan pada baris kode pada *file*s MakeFile.in di folder ns-2.35 menjadi seperti pada Gambar 2.10 untuk mengaktifkan fitur *C Debugging*.

```
# Rubah
CC     = @CC@
CPP    = @CXX@
# Menjadi
CC     = gcc-5
CPP    = g++-5

# Rubah
CCOPT  = @V_CCOPT@
# Menjadi
CCOPT  = @V_CCOPT@ -g

# Rubah
DEFINE = -DTCP_DELAY_BIND_ALL -DNO_TK @V_DEFINE@
@V_DEFINES@ @DEFS@ -DNS_DIFFUSION -DSMAC_NO_SYNC -
DCPP_NAMESPACE=@CPP_NAMESPACE@ -
DUSE_SINGLE_ADDRESS_SPACE -Drng_test
# Menjadi
DEFINE = -DTCP_DELAY_BIND_ALL -DNO_TK @V_DEFINE@
@V_DEFINES@ @DEFS@ -DNS_DIFFUSION -DSMAC_NO_SYNC -
DCPP_NAMESPACE=@CPP_NAMESPACE@ -
DUSE_SINGLE_ADDRESS_SPACE -Drng_test -DNDEBUG -DDEBUG
```

Gambar 2.10 Baris Kode yang Diubah Pada File MakeFile.in

Proses instalasi dilakukan dengan menjalankan perintah seperti pada Gambar 2.11.

```
export CC=gcc-5 CXX=g++-5
sudo bash install
```

Gambar 2.11 Perintah Instalasi NS2

Namun, perintah *export* hanya dijalankan jika mempunyai aplikasi gcc dengan lebih dari dua versi. Jika hanya mempunyai aplikasi gcc dengan satu versi saja, dapat langsung menjalankan perintah sudo bash install. Jika masih mengeluarkan kesalahan karena tidak ada gcc-5, dapat memasangnya dengan cara mengetikan skrip seperti pada Gambar 2.12.

```
sudo apt install g++-5
sudo apt install gcc-5
```

Gambar 2.12 Skrip instalasi GCC versi 5

Jika pemasangan telah selesai, buat lintasan cepat program NS2 supaya mudah untuk dijalankan lalu jalankan programnya seperti pada Gambar 2.13.

```
cd ~/ns2/ns-allinone-2.35/ns-2.35
sudo cp ns /usr/local/bin/ns2
ns2
```

Gambar 2.13 Pembuatan lintasan cepat dan menjalankan NS2

2.5.2 *Trace File*

Trace file merupakan kode *ASCII* yang yang berisikan deskripsi *event* yang berisikan waktu simulasi, tujuan dan asal, yang mengidentifikasikan jalur yang terjadi [12]. 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	Layer	AGT : application RTR : *routing* LL : *link* layer IFQ : *packet* queue MAC : MAC PHY : physical
4	Flag	--- : Tidak ada
5	*Sequence Number*	Nomor paket
6	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*
7	Ukuran	Ukuran paket pada layer saat itu
8	Detail MAC	[a b c d] a : perkiraan waktu paket b : alamat penerima c : alamat penerima d : IP *head*er
9	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 peta dunia yang bebas untuk digunakan di bawah lisensi terbuka. Semua orang dapat menggunakan untuk tujuan apapun dikarenakan lisensi dari OSM sendiri yang bersifat terbuka di bawah *Open Data Commons Open Database License* (ODbL) oleh *OpenStreet Map Foundation* [13].

OpenStreetMap dibangun oleh komunitas pembuat peta yang berkontribusi dan juga kontributor menggunakan citra udara, perangkat GPS, dan peta lapangan berteknologi rendah untuk memverifikasi bahwa OSM akurat dan termutakhir [13].

OpenStreetMap bersifat digital, yang membuatnya sangat berguna untuk kita dan dapat dibagikan, yang berarti bahwa semua orang dapat mengambil manfaat dari pekerjaan orang lain [14].

2.7 Java OpenStreetMap(JOSM)

Java OpenStreetMap merupakan aplikasi tambahan untuk menyunting JOSM tanpa harus terus menerus tersambung pada internet. Aplikasi ini sangat membantu apabila memiliki masalah dalam koneksi internet Anda. Dengan JOSM, anda hanya membutuhkan koneksi internet dalam hal mendownload dan mengunggah file OpenStreetMap [14].

JOSM dapat juga menyunting hasil peta OSM yang telah diekspor seperti mengubah objek, menambah dan menghilangkan benda, dan juga menambah jalan [15].

2.8 Simulation of Urban Mobilitiy(SUMO)

Simulation of Urban Mobility(SUMO) merupakan paket simulasi lalu lintas yang bersifat *open-source*. SUMO dapat digunakan untuk simulasi intermodal termasuk pedestrian dan peralatan yang besar dalam pembuatan skenario [16].

SUMO juga memungkinkan untuk implementasi manajemen lalu lintas secara akurat dan juga banyak improvisasi dan penambahan simulasi yang terdapat dalam SUMO [17].

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 jaringan jalan secara abstrak yang dapat digunakan untuk aplikasi SUMO yang lainnya
netconvert	Mengimpor jaringan jarak digital dari beberapa pusat dan membuat jaringan jalan yang dapat digunakan oleh peralatan yang lain dalam suatu paket
randomTrips.py	Membuat rute acak untuk diberikan kepada jaringan dengan memilih asal dan tujuan secara seragam dalam acak atau memodifikasi distribusinya
dua*route*r	Mengimpor beberapa definisi permintaan yang berbeda, menghitung rute dalam kendaraan yang dapat dipakai oleh SUMO dengan menggunakan komputasi *path* terpendek
traceExporter.py	Dapat melakukan konversi dan penyaringan *fcd output* SUMO menuju *"trace file"* yang berbeda

Tabel 2.4 Alat pada *Simulation of Urban Mobility* (SUMO)

2.9 AWK

AWK adalah bahasa *script* yang digunakan untu memanipulasi data dan membuat laporan. Perintah AWK dalam bahasa pemrograman tidak perlu di*compile*, dan pengguna dapat menggunakan variabel, fungsi numerik, fungsi *string*, dan operator logika [18].

Dalam penggunaanya, AWK adalah alat yang dapat menuliskan program yang kecil tapi efektif didalam pernyataan yang ada didalam tekstur teks yang dapat dicari dengan setiap baris didalam dokumen dan dapat beraksi ketika ada kesamaan

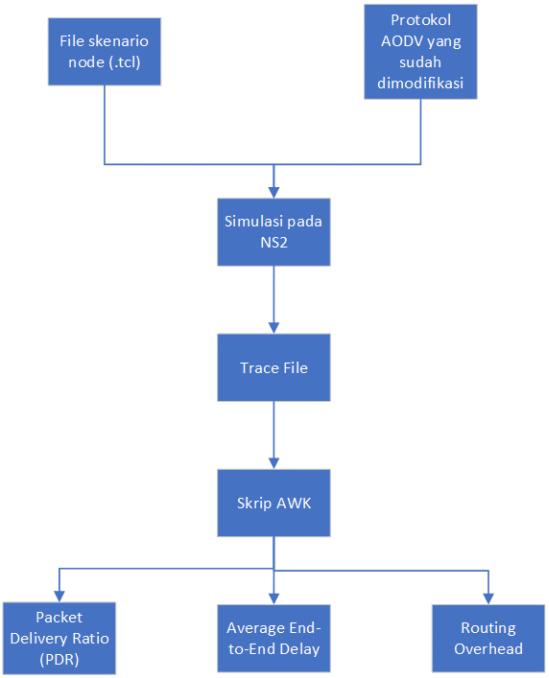
didalam suatu baris. AWK sering digunakan untuk melakukan pencarian dan proses tekstur. AWK mencari satu atau lebih *file* untuk melihat jika terdapat baris yang sama seperti tekstur yang spesifik dan melakukan aksi yang berhubungan [18].

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 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	V	*Velocity* atau Kecepatan. Kecepatan dari *node* yang mempengaruhi masa aktif dari hubungan *node*
9	LCF	*Link Stability Coefficient.* Adalah faktor dimana kecepatan dari kedua node yang mempengaruhi dari stabilitas hubungan *node.* Rumus dari *Link Stability Coefficient* ini adalah $lcf = (Vx - Vy)^2$. Dimana Vi adalah kecepatan di *node* horizontal dan Vy adalah kecepatan di *node* vertikal
10	HC	*Hop Count.* Adalah adalah jumlah rata-rata *hop count* atau bisa disebut juga dengan jumlah lompatan untuk setiap paket menuju *node* tujuan
11	RQ	*Route Request.* Adalah rute *node* AODV yang diminta

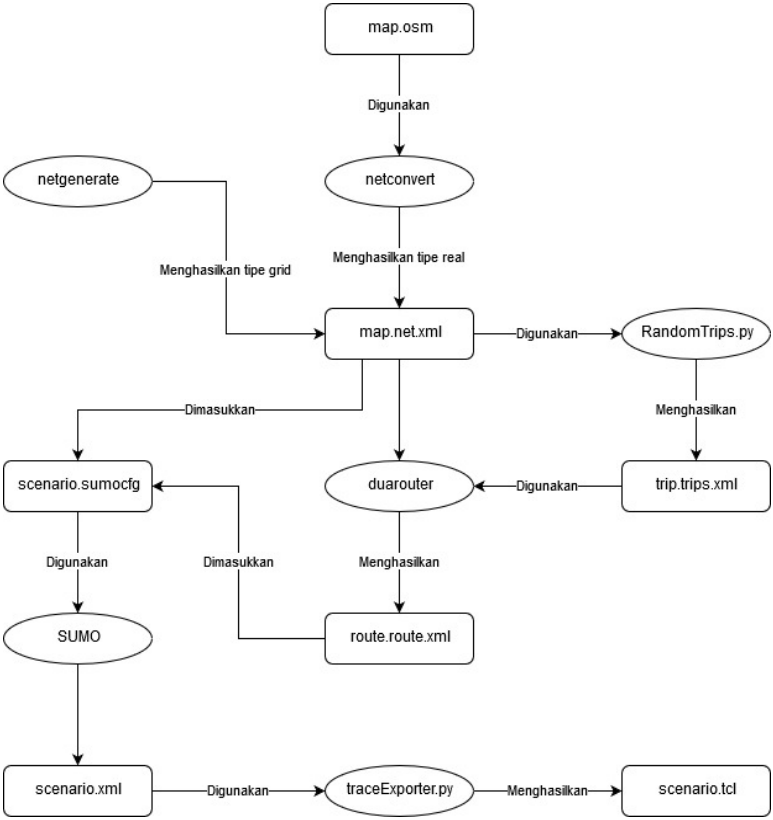
Tabel 3.1 Daftar Istilah

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 *Coefficient Link* pada proses *route discovery*. Dalam proses ini akan dilakukan perhitungan LCF untuk setiap paket yang tiba pada suatu *node*. LCF akan dihitung menggunakan sebuah rumus dengan beberapa parameter. Pada saat paket sampai pada *destination node*, nilai LCF akan dikirim kembali menuju *source node* dengan disisipkan pada paket reply. Proses perbandingan nilai LCF yang ada pada *destination node* dengan nilai LCF paket yang tiba bertujuan untuk mencari *route* yang lebih baik dengan nilai LCF 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 1000 m x 1000 m. Dengan 4 titik persimpangan, maka akan didapatkan 9 petak dan panjang

tiap peta adalah 1000 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 *Coefficient Link*. 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 Tabel Posisi *Node*

*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 *Nodes 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 *Coefficient Link*

Coefficient Link ini adalah faktor dimana kecepatan diantara *node* sangat mempengaruhi masa berlaku dari hubungan *node. Coefficient Link* ini dapat dirumuskan sebagai berikut:

$$lcf = (Vx - Vy) \times (Vx - Vy)$$
$$lcf_sum = (Vx - Vy)^2 + (Vx - Vy)^2 + \ldots$$

Pada saat fase RREQ, urutan cara kerjanya adalah sebagai berikut:

- Pertama, node pusat akan menambahkan V pada node ke-i (dimulai dari node pusat), *hop_count*, dan *lcf_sum* = 0.
- Kedua RREQ akan disebar melalui *node* tetangga. Ketika paket telah diterima, *node* tetangga tersebut akan menghitung

jumlah *lcf_sum*, jumlah *hop_count*, dan memperbarui nilai V dengan kecepatan node tersebut.

- Ketiga, paket RREQ ini akan terus disalurkan dengan cara yang sama seperti diatas hingga ke *node* tujuan
- Keempat, *node* tujuan akan memilih jalur destinasi berdasarkan:
    - Jika, total *hop_count* > 10, maka akan diambil path dengan nilai *lcf_sum* terkecil
    - Jika total *hop_count* < 10, makan akan diambil path dengan total *hop_count* terkecil

Untuk penerapan *pseudocode* nya, dapat dilihat pada gambar dibawah ini

```
linkcoef = pow(rq->rq_speed_prev - n_speed, 2);
total_linkcoef = linkcoef + rq->rq_prev_linkcoef;
```

Gambar 3.8 *Pseudocode* Pemindahan Informasi Hasil *Coefficient Link*

Terlihat di *pseudocode* diatas dimana RQ dikurangi oleh kecepatan lalu dikuadratkan. Lalu untuk jumlah LCF nya tinggal dijumlahkan dari hasil perhitungan *linkcoef* dan juga perhitungan *linkcoef* sebelumnya.

Sehingga, hasil yang diharapkan adalah *overhead* dari proses pencarian rute akan berkurang karena protokol yang telah diimprovisasi dengan menggunakan *coefficient link*. Selain itu juga keterlambatan *end to end* dapat dikurangi karena dimungkinkan dari hasil dari pencarian rute dengan cara adaptasi diri sendiri sehingga protokol tersebut mempunyai keterlambatan *end to end* yang lebih baik dibanding AODV yang biasa dikarenakan pencarian rute dengan melihat faktor kecepatan dari setiap node.

3.4 Perancangan Simulasi pada NS2

Simulasi pada NS2 dilakukan dengan menggabungkan *file* skenario dan *script* tcl yang berisikan konfigurasi lingkungan simulasi. Konfigurasi lingkungan simulasi pada NS2 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	50, 100, 150, 200, 250, and 300
5	Simulation Area	1000 m x 1000 m
6	*Number* of Cluster	10, 15, 20
7	*Number* of Clustering	1
6	Antenna Model	Omni Antenna
7	MAC Type	MAC/802_11
8	Network Interfaces Types	Wireless
9	Transmission Range	400 m
10	*Source*/*Destination Node*	Static
11	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

(Halaman ini sengaja dikosongkan)

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*

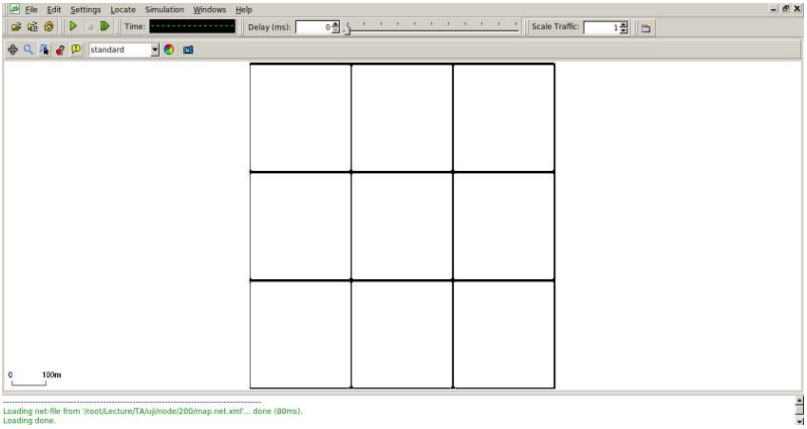
Pada Tugas Akhir ini, penulis menggunakan *tool* netgenerate dari SUMO untuk mengimplementasikan skenario *grid* berupa peta jalan berpetak. Peta skenario *grid* dibuat dengan panjang jalan 300 meter dan luas peta 1000 m x 1000 m. Dengan melebihkan 5 meter untuk menghindari terjadinya kesalahan pada 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*router*. Perintah penggunaan *tool* dua*router* dapat dilihat pada Gambar 4.4.

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

Gambar 4.4 Perintah duarouter

Ketika menggunakan *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

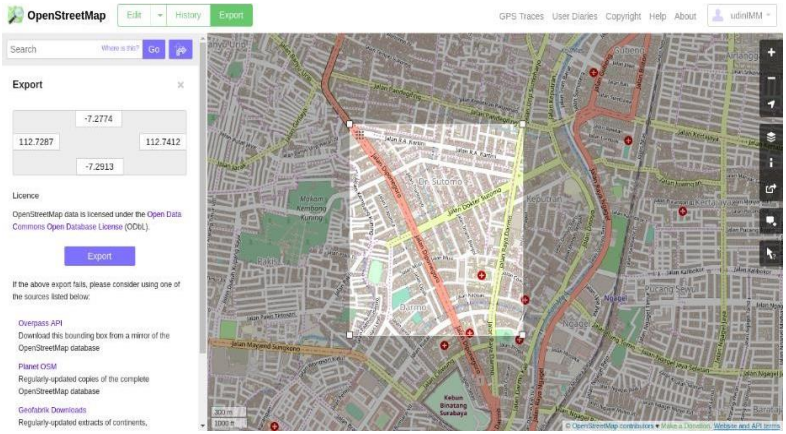
Agar dapat disimulasikan pada NS-2, *file* skenario berekstensi .xml harus dikonversi menjadi *file* berekstensi .tcl dengan menggunakan *tool* traceExporter.py. Perintah untuk menggunakan *tool* traceExporter.py dapat dilihat pada Gambar 4.7.

```
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*

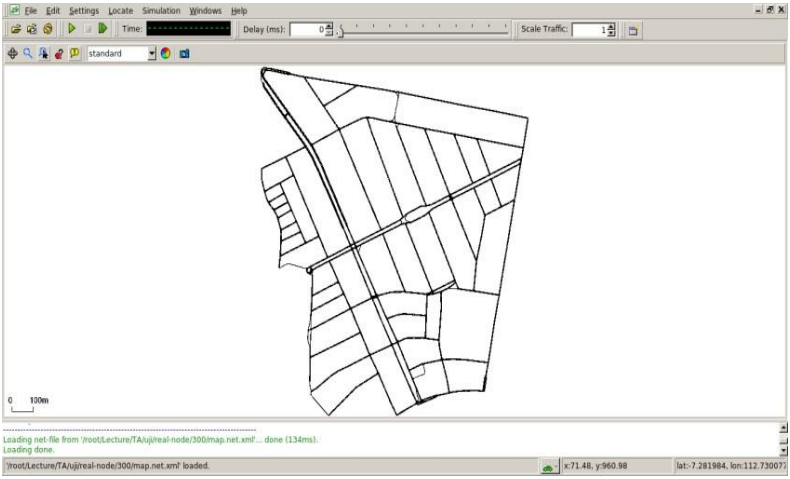
File hasil ekspor dari *OpenStreetMap* tersebut adalah *file* peta dengan ekstensi .osm. Peta disunting terlebih dahulu dengan menggunakan *Java OpenStreetMap Editor* (JOSM) untuk menghapus jalan yang tidak digunakan. Kemudian *file* .osm hasil suntingan tersebut dikonversi menjadi peta dalam bentuk *file*

berekstensi .xml menggunakan *tool* netconvert dari SUMO. Perintah untuk menggunakan netconvert dapat dilihat pada Gambar 4.9.

```
netconvert --osm-files map.osm -o 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 4.14.

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

 .....

 ((MobileNode*) t_node)->getLoc(&posX,&posY,0);
 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.);
```

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);

  ((Mobile Node*) t_node)->getLoc(&posX,&posY,0);
  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::send Reply(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);

((MobileNode*) t_Node)->getLoc(&posX,&posY,0);
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;

  ((MobileNode*) t_node)->getLoc(&posX,&posY,0);
  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 *Coefficient Link* (LCF)

Proses implementasi LCF akan dilakukan dengan memodifikasi struktur dari beberapa *control message*s dan proses yang menggunakan *control message*s tersebut.

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) {
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);

……..
 if (index == 8){

   t_node=(MobileNode*)(Node::get_node_by_address(index));
   ((MobileNode*) t_node)->getLoc(&xpos,&ypos,&zpos,0);
   t_node=(MobileNode*)(Node::get_node_by_address(index));
   n_speed=((MobileNode*)t_node)->speed();

   RAODV raodv_data;
   raodv_data.current_speed = n_speed;
   raodv_data.current_linkcoef = 0;
   raodv_data.jml_linkcoef = 0;
   nodedata[index] = raodv_data;

 }
……..

}
```

Gambar 4.21 Modifikasi Pengiriman RREQ untuk LCF

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 LCF dimana hasilnya akan disimpan pada sebuah variabel bernama jml_linkcoef di *node* tersebut sebagai hasil penjumlahan nilai LCF terakhir dan yang dihitung pada *node* tersebut. Nilai dari jml_linkcoef akan disimpan juga pada paket

yang akan dikirimkan. Implementasi pada modifikasi ini dapat dilihat pada Gambar 4.22.

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

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

  if (flag_counter == 1) {
        Packet::free(p);

  }

((MobileNode*) t_node)->getLoc(&posX,&posY,0);
  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];
   }
  }
.....
  RAODV raodv_data;
  raodv_data.current_speed = n_speed;
  raodv_data.current_linkcoef = pow(n_speed - nodedata[rq->rq_src-
1].current_speed, 2);
  raodv_data.jml_linkcoef = raodv_data.current_linkcoef + nodedata[rq->rq_src-
1].jml_linkcoef;
  nodedata[index] = raodv_data;

……

   counter = counter - 1;
```

```
    #ifdef DEBUG
    fp = fopen("debug.txt", "a");
    fprintf(fp, "\n%.6f  node %d - %s, BEFORE IF DEST counter : %d |
dest_linkcoef: %f | total_linkcoef: %f ",  CURRENT_TIME, index,
__FUNCTION__, counter, dest_linkcoef, nodedata[index].jml_linkcoef);
    fclose(fp);
    #endif // DEBUG

    if(dest_linkcoef > nodedata[index].jml_linkcoef){
      dest_linkcoef = nodedata[index].jml_linkcoef;
      dest_index_terkecil = rq->rq_src;
    }


    #ifdef DEBUG
    fp = fopen("debug.txt", "a");
    fprintf(fp, "\n%.6f  node %d - %s, counter : %d | dest_linkcoef: %f |
total_linkcoef: %f ",  CURRENT_TIME, index, __FUNCTION__, counter,
dest_linkcoef, nodedata[index].jml_linkcoef);
    fclose(fp);
    #endif // DEBUG

…….
```

Gambar 4.22 Modifikasi Penerimaan RREQ untuk LCF

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 LCF pada paket RREP. Nilai LCF yang diisikan adalah nilai LCF terbesar antara nilai dari variabel jml_linkcoef pada *node* tersebut dengan nilai LCF menuju *source node* pada *routing table* di *node* tersebut. Implementasi pada modifikasi dapat dilihat pada Gambar 4.23.

```
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*) {

.....

rp->rp_type = AODVTYPE_RREP;
 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;

if(rp->rp_dst == index)
{
  rp->linkcoef = max(jml_linkcoef,rt->linkcoef);
}
.....
}
```

Gambar 4.23 Modifikasi Pengiriman RREP untuk LCF

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 LCF pada informasi *route* menuju *source node*. Implementasi pada modifikasi dapat dilihat pada Gambar 4.24.

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

.....

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

}
```

Gambar 4.24 Modifikasi Penerimaan RREP untuk LCF

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

4.5 Implementasi Simulasi pada NS2

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)                  1000
        ;
set opt(y)                  1000
        ;
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.

4.5.1 Implementasi Simulasi pada *K-Means Clustering*

Dikarenakan dalam file berekstensi .tcl tidak terdapat konfigurasi untuk mengatur *clustering* pada file berekstensi .tcl, maka dilakukanlah pendefinisian dan juga penyuntingan supaya

dapat dilakukan simulasi dengan menggunakan jumlah *clustering* yang terdapat pada Gambar 4.26.

```
.....
#define NUMBER_OF_CLUSTER      15
#define NUMBER_OF_NODE         200
.....
```

Gambar 4.26 Konfigurasi File AODV.h

File tersebut akan mendefinisikan jumlah *cluster* dan *node* yang akan disimulasikan dengan menggunakan metode *K-Means Clustering*.

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 4.27.

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

Gambar 4.27 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 4.28.

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

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

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

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

Gambar 4.28 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 4.29.

```
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 4.29 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 4.30.

```
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 4.30 File ro.awk

File ro.awk dapat dieksekusi dengan menjalankan perintah awk -f ro.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) i7-7700 HQ CPU @ 2.80GHz
Sistem Operasi	Ubuntu 18.04
Linux Kernel	4.4
Memori	16 GB
Penyimpanan	50 GB

Tabel 5.1 Spesifikasi Perangkat yang Digunakan

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, dan RO. Skrip awk pengujian untuk setiap parameter terdapat pada Gambar 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 Cara Uji Coba

Untuk cara pengujian, supaya lebih mudah dibuatkan skrip seperti dibawah ini didalam folder skenario agar tidak perlu mengetik perintah secara berulang dan simpan sebagai skrip.sh.

```
bash skrip.sh
bash skenario.sh
bash cbr.sh
bash run.sh
```

Gambar 5.1 File skrip.sh

Setelah itu, tambahkan skrip dibawah ini di file *MakeFile* supaya dapat menjalankan AODV yang menggunakan *K-Means*

```
#Tambahkan ini
aodv/aodv_kmeans.o aodv/aodv_kmeans.o \

#Setelah baris yang tertulis seperti ini
aodv/aodv_logs.o aodv/aodv.o \
aodv/aodv_rtable.o aodv/aodv_rqueue.o \
```

Gambar 5.2 Tambahan skrip di dalam file *MakeFile*

Setelah itu, *compile* semua file dengan cara mengetikan seperti yang ada dibawah ini

```
Sudo make clean; sudo make; sudo make install
```

Gambar 5.3 Skrip *Compile*

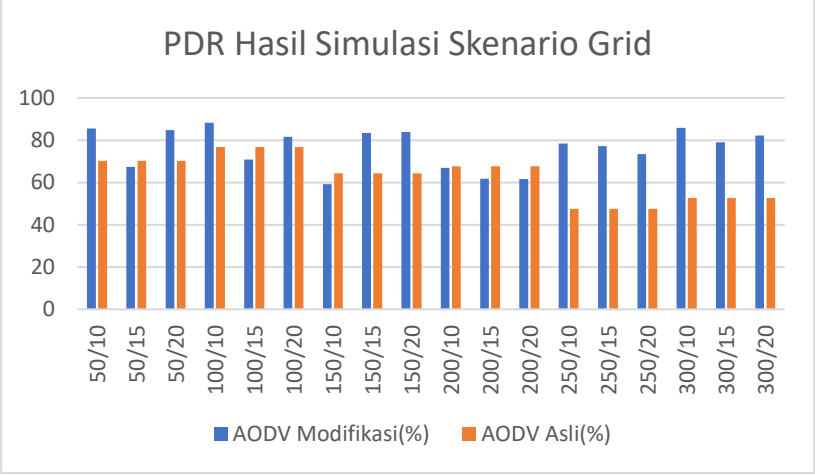
Setelah itu, buka jendela *Terminal* baru dan arahkan menuju folder berisi skenario dan ketik bash jalan.sh (Catatan: Tidak perlu pakai awalan sudo maupun dimasukkan dalam mode sudo su. Dikarenakan jika pakai perintah sudo dapat mengakibatkan tidak jalannya program.)

5.3 Skenario Grid

Pengujian pada skenario *grid* digunakan untuk melihat perbandingan PDR, E2E, dan RO antara *routing protocol* AODV asli dan AODV modifikasi. Pengambilan data pada skenario *grid* dilakukan sebanyak 3 kali untuk setiap variasi dan data yang ditampilkan pada tabel dan grafik hasil simulasi adalah nilai rata-rata dari 3 kali percobaan untuk setiap variasi (Tersedia dalam lampiran untuk hasil yang tidak dirata-rata).

Jumlah *Node*/ Jumlah Cluster	AODV Modifikasi(%)	AODV Asli(%)	Perbedaan(%)
50/10	85,55	70,22	15,33
50/15	67,38	70,22	-2,84
50/20	84,84	70,22	14,62
100/10	88,33	76,78	11,55
100/15	70,91	76,78	-5,87
100/20	81,56	76,78	4,78
150/10	59,11	64,38	-5,27
150/15	83,45	64,38	19,07
150/20	83,90	64,38	19,52
200/10	66,91	67,64	-0,73
200/15	61,76	67,64	-5,88
200/20	61,58	67,64	-6,06
250/10	78,43	47,59	30,84
250/15	77,26	47,59	29,67
250/20	73,44	47,59	25,85
300/10	85,78	52,66	33,12
300/15	79,01	52,66	26,35
300/20	82,26	52,66	29,6

Tabel 5.2 PDR Hasil Simulasi Skenario *Grid*



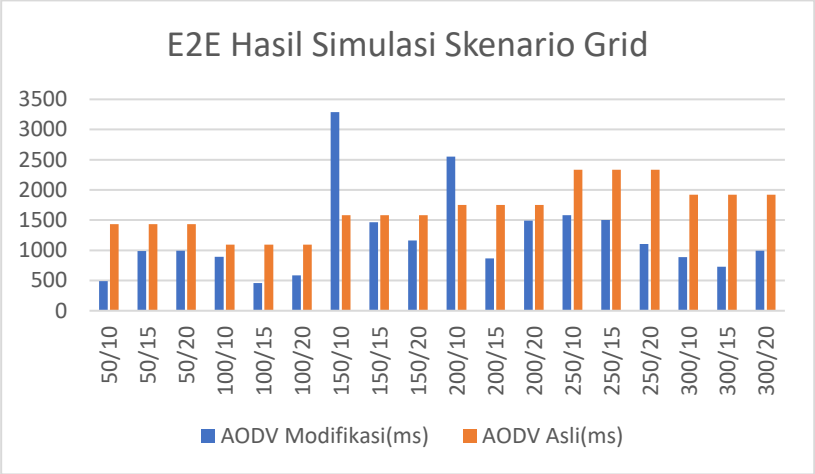
Gambar 5.4 Grafik PDR Hasil Simulasi Skenario *Grid*

Berdasarkan grafik pada Gambar 5.4 dapat dilihat bahwa AODV modifikasi menghasilkan perubahan PDR yang lumayan signifikan terhadap AODV asli. Pada variasi jumlah node 150, 200 dalam jumlah *cluster* 10, dan jumlah *node* 50, 100, 200 dalam jumlah *cluster* 15, dan juga jumlah *node* 200 dalam jumlah node 200 dalam jumlah *cluster* 20, PDR dari AODV modifikasi lebih rendah dibandingkan dengan AODV asli, terutama pada jumlah *node* 200 dan jumlah *cluster* 10, PDR dari AODV asli dan modifikasi bisa dibilang hampir sama (selisihnya tipis sekali, AODV modifikasi hanya beda 0,73 lebih rendah dibanding AODV asli). Grafik menunjukan kenaikan yang positif dimana PDR akan naik secara signifikan jika jumlah *node* sudah diatas 200. Untuk jumlah *node* 50, 100, 150, dan 200, grafik menunjukan kenaikan yang positif walaupun ada beberapa yang jumlah AODV asli nya lebih bagus dibanding AODV modifikasi. Perbedaan terbesar terjadi pada simulasi dengan variasi jumlah *node* 300 dan jumlah cluster 10 yang menghasilkan angka sebesar 30,84%. Pada bagian ini, AODV modifikasi menghasilkan PDR yang sebagian besar lebih baik dibandingkan dengan AODV asli.

Jumlah *Node*/ Jumlah Cluster	AODV Modifikasi(ms)	AODV Asli(ms)	Perbedaan(ms)
50/10	488,434	1432,248	-943,814
50/15	985,939	1432,248	-446,309
50/20	995,713	1432,248	-436,535
100/10	892,204	1094,233	-202,029
100/15	456,176	1094,233	-638,057
100/20	585,336	1094,233	-508,897
150/10	3286,267	1580,118	1706,149
150/15	1465,053	1580,118	-115,065
150/20	1164,102	1580,118	-416,016
200/10	2553,437	1749,8	803,637
200/15	869,127	1749,8	-880,673
200/20	1493,540	1749,8	-256,26
250/10	1580,818	2334,237	-753,419
250/15	1505,113	2334,237	-829,124
250/20	1105,390	2334,237	-1228,85
300/10	887,255	1923,498	-1036,24
300/15	727,272	1923,498	-1196,23

300/20	996,326	1923,498	-927,172

Tabel 5.3 E2E Hasil Simulasi Skenario *Grid*

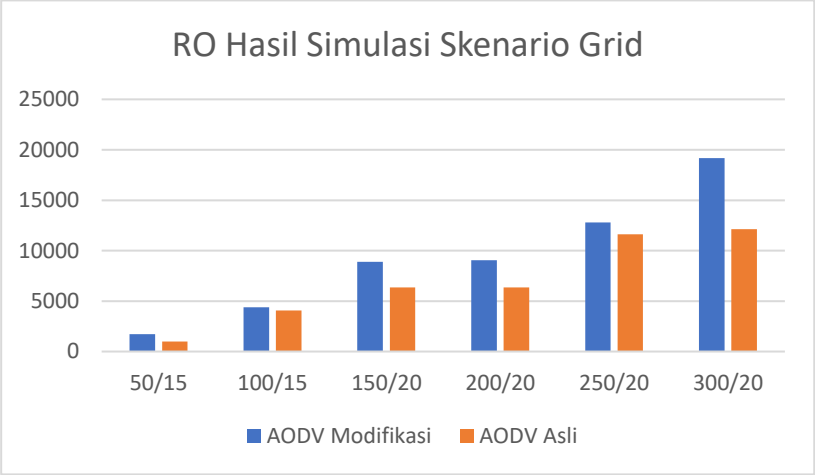


Gambar 5.5 Grafik E2E Hasil Simulasi Skenario *Grid*

Berdasarkan grafik pada Gambar 5.5 dapat dilihat bahwa AODV modifikasi menghasilkan perubahan E2E yang lumayan signifikan terhadap AODV asli. Pada variasi jumlah *node* 150 dan 200 dalam jumlah *cluster* 10, masih mengkasilkan E2E yang lebih besar dibandingkan dengan AODV asli, dan juga ditemukan perbedaan yang sangat tipis sekali (beda 115,065 ms) dalam variasi jumlah *node* 150 dalam jumlah *cluster* 15. Perbedaan E2E terbesar terjadi pada simulasi dengan variasi jumlah *node* 150 dalam jumlah *cluster* 10 yang perbedaannya hingga 1706,149 ms. Namun, untuk perbandingan hasil terbaik pada AODV Modifikasi (*alias* hasil AODV Modifikasi lebih kecil dibandingkan AODV asli), terdapat pada *node* 250 dalam jumlah *cluster* 20 yang perbedaannya mencapai 1228,85 ms. Pada bagian ini, AODV modifikasi secara garis besar memiliki hasil yang lebih baik dibandingkan AODV asli

Jumlah *Node*/ Jumlah Cluster	AODV Modifikasi	AODV Asli	Perbedaan
50/15	1722	1003	719
100/15	4393	4055	338
150/20	8881	6353	2528
200/20	9038	6344	2694
250/20	12808	11624	1184
300/20	19173	12119	7054

Tabel 5.4 RO Hasil Simulasi Skenario *Grid*



Gambar 5.6 Grafik RO Hasil Simulasi Skenario *Grid*

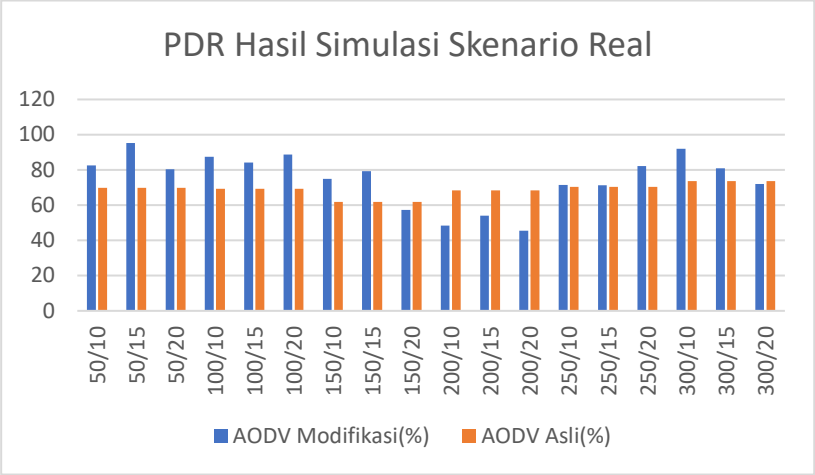
Berdasarkan grafik pada Gambar 5.6 dapat dilihat bahwa AODV modifikasi menghasilkan RO yang lebih ditinggi dibandingkan AODV asli. Dikarenakan perhitungan menggunakan *K-Means Clustering* yang menyebabkan grafik RO yang tinggi. Pada bagian ini, AODV modifikasi menghasilkan RO yang lebih buruk dibandingkan AODV asli dikarenakan perhitungan *K-Means Clustering*.

5.4 Skenario Real

Pengujian pada skenario *real* digunakan untuk melihat perbandingan PDR, E2E, dan RO antara *routing protocol* AODV asli dan AODV modifikasi. Pengambilan data pada skenario *real* dilakukan sebanyak 3 kali untuk setiap variasi dan data yang ditampilkan pada tabel dan grafik hasil simulasi adalah nilai rata-rata dari 3 kali percobaan untuk setiap variasi (Tersedia dalam lampiran untuk hasil yang tidak dirata-rata).

Jumlah *Node*/ Jumlah Cluster	AODV Modifikasi(%)	AODV Asli(%)	Perbedaan(%)
50/10	82,63	69,78	12,85
50/15	95,30	69,78	25,52
50/20	80,37	69,78	10,59
100/10	87,42	69,37	18,05
100/15	84,14	69,37	14,77
100/20	88,79	69,37	19,42
150/10	74,84	61,76	13,08
150/15	79,37	61,76	17,61
150/20	57,28	61,76	-4,48
200/10	48,30	68,29	-19,99
200/15	54,08	68,29	-14,21
200/20	45,45	68,29	-22,84
250/10	71,49	70,34	1,15
250/15	71,23	70,34	0,89
250/20	82,16	70,34	11,82
300/10	91,99	73,71	18,28
300/15	80,96	73,71	7,25
300/20	72,03	73,71	12,85

Tabel 5.5 PDR Hasil Simulasi Skenario *Real*



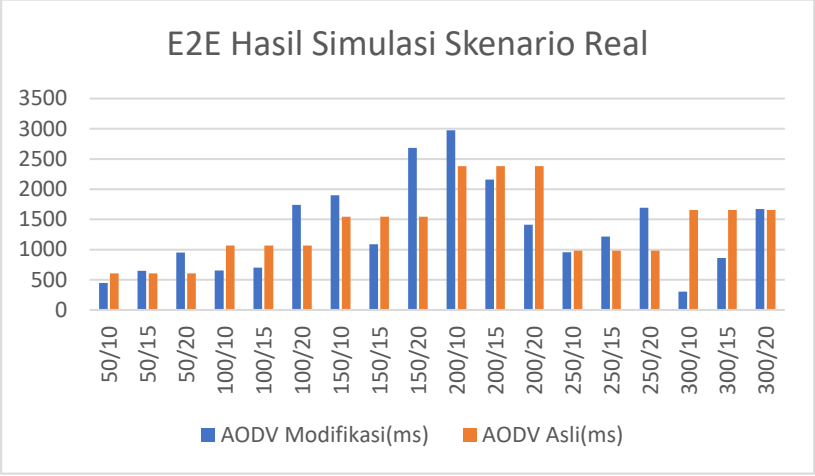
Gambar 5.7 Grafik PDR Hasil Simulasi Skenario *Real*

Berdasarkan grafik pada Gambar 5.7 dapat dilihat bahwa AODV modifikasi menghasilkan perubahan PDR yang lumayan signifikan terhadap AODV asli. Pada variasi jumlah node 200 dalam seluruh *cluster* 10, dan jumlah *node* 150 dalam jumlah *cluster* 20, dan juga jumlah *node* 300 dalam jumlah *cluster* 20, PDR dari AODV modifikasi lebih rendah dibandingkan dengan AODV asli. Untuk jumlah *node* 50, 100, 150, 250, dan 300, grafik menunjukan kenaikan yang positif walaupun ada beberapa yang jumlah AODV asli nya lebih bagus dibanding AODV modifikasi dan juga hasilnya sama saja antara AODV asli dan AODV modifikasi. Perbedaan terbesar terjadi pada simulasi dengan variasi jumlah *node* 50 dan jumlah *cluster* 15 yang menghasilkan angka sebesar 25,52%. Pada bagian ini, AODV modifikasi menghasilkan PDR yang sebagian besar lebih baik dibandingkan dengan AODV asli.

Jumlah *Node*/ Jumlah Cluster	AODV Modifikasi(ms)	AODV Asli(ms)	Perbedaan(ms)
50/10	448,205	607,199	-158,994

50/15	645,365	607,199	38,166
50/20	950,174	607,199	342,975
100/10	654,592	1067,061	-412,469
100/15	699,196	1067,061	-367,865
100/20	1737,608	1067,061	670,547
150/10	1896,910	1545,533	351,377
150/15	1087,814	1545,533	-457,719
150/20	2683,429	1545,533	1137,896
200/10	2976,866	2382,553	594,313
200/15	2158,270	2382,553	-224,283
200/20	1409,009	2382,553	-973,544
250/10	954,987	981,678	-26,691
250/15	1215,256	981,678	233,578
250/20	1690,867	981,678	709,189
300/10	301,539	1656,177	-1354,638
300/15	857,239	1656,177	-798,938
300/20	1670,030	1656,177	-158,994

Tabel 5.6 E2E Hasil Simulasi Skenario *Real*



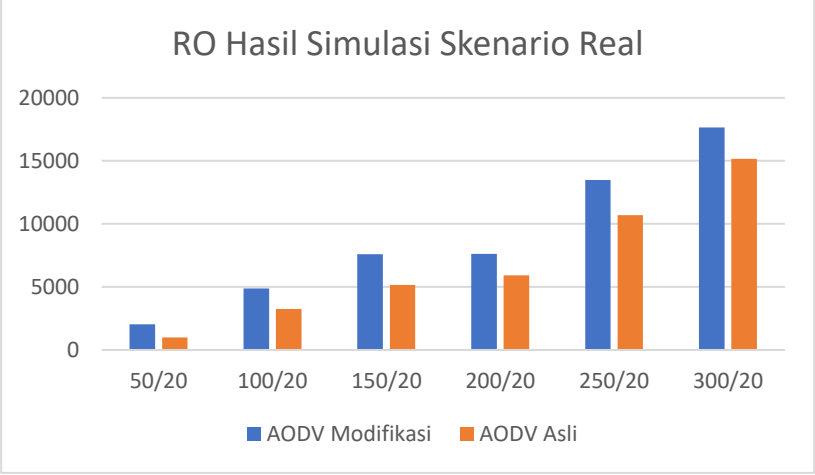
Gambar 5.8 Grafik E2E Hasil Simulasi Skenario *Real*

Berdasarkan grafik pada Gambar 5.8 dapat dilihat bahwa AODV modifikasi menghasilkan perubahan E2E yang fluktutatif dibandingkan dengan AODV asli. Semua *node* dan *cluster* yang terdapat dalam grafik pada Gambar 5.8 terdapat hasil E2E yang baik dan E2E yang buruk. Perbedaan terbesar pada jumlah *node*

300 dalam jumlah *cluster* 10 yang menghasilkan angka 1354,638. Pada bagian ini, AODV modifikasi terkadang hasilnya lebih buruk dibandingkan AODV asli, maupun AODV modifikasi hasilnya lebih baik dibandingkan AODV asli, untuk probabilitasnya sekitar 50:50.

Jumlah *Node*/ Jumlah Cluster	AODV Modifikasi	AODV Asli	Perbedaan
50/20	2045	992	1053
100/20	4877	3259	1618
150/20	7596	5159	2437
200/20	7616	5911	1705
250/20	13478	10702	2776
300/20	17647	15157	2490

Tabel 5.7 RO Hasil Simulasi Skenario Real



Gambar 5.9 Grafik RO Hasil Simulasi Skenario *Real*

Berdasarkan grafik pada Gambar 5.9 dapat dilihat bahwa AODV modifikasi menghasilkan RO yang lebih ditinggi dibandingkan AODV asli. Dikarenakan perhitungan menggunakan *K-Means Clustering* yang menyebabkan grafik RO

yang tinggi. Pada bagian ini, AODV modifikasi menghasilkan RO yang lebih buruk dibandingkan AODV asli dikarenakan perhitungan *K-Means Clustering*.

5.5 Perbandingan Hasil AODV Modifikasi Pada Skenario *Grid* dan *Real*

Dibawah ini adalah perbandingan hasil dari pengujian AODV modifikasi. Pertama adalah hasil pengujian PDR dari AODV yang telah dimodifikasi

Jumlah *Node*	AODV Grid (%)	AODV Real (%)	Perbedaan(%)
50	67,38	95,30	27,92
100	70,91	84,14	13,23
150	83,90	57,28	-26,62
200	61,58	45,45	-16,13
250	73,44	82,16	8,72
300	82,26	72,03	-10,23

Tabel 5.8 Perbandingan hasil PDR pada AODV *Grid* dan AODV *Real*

Berdasarkan tabel pada Tabel 5.8 dapat dilihat bahwa hasil pengujian PDR terlihat fluktutatif, terkadang hasil PDRnya lebih bagus pada AODV *grid* dan juga terkadang lebih bagus pada AODV *real.*

Dibawah ini adalah perbandingan hasil pengujian E2E dari AODV yang telah dimodifikasi

Jumlah *Node*	AODV Grid (ms)	AODV Real (ms)	Perbedaan (ms)
50	985,939	645,365	27,92
100	456,176	699,196	13,23
150	1164,102	2683,429	-26,62
200	1493,540	1409,009	-16,13
250	1102,390	1690,867	8,72
300	996,326	1670,030	-10,23

Tabel 5.9 Perbandingan hasil E2E pada AODV *grid* dan AODV *real*

Berdasarkan tabel pada Tabel 5.9 dapat dilihat bahwa hasil pengujian E2E secara garis besar lebih baik pada saat pengujian di AODV *grid.* Dikarenakan pengaruh peta skenario itu sendiri (AODV *grid* hanya peta berbentuk tabel ukuran 9x9 sedangkan AODV *real* adalah peta berbentuk jalan raya sebenarnya).

Dibawah ini adalah perbandingan hasil pengujian RO dari AODV yang telah dimodifikasi

Jumlah *Node*	AODV Grid	AODV Real	Perbedaan
50	1722	2045	323
100	4393	4877	484
150	8881	7596	-1285
200	9038	7616	-1422
250	12808	13478	670
300	19173	17647	1526

Tabel 5.10 Perbandingan hasil RO antara AODV Grid dan AODV Real

Berdasarkan tabel pada Tabel 5.10 dapat dilihat bahwa hasil pengujian RO pada AODV *grid* dan AODV *real* hasilnya fluktutatif antara AODV *grid* dan AODV *real.* Terlihat dimana untuk *node* ke 50,100,250,300 lebih banyak di AODV *real,* sedangkan untuk *node* ke 150 dan 200 lebih banyak di AODV *grid.*

(Halaman ini sengaja dikosongkan)

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.

6.1 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 *Coefficient Link* secara umum menghasilkan hasil yang baik dari sisi *Packet Delivery Ratio*, dan *End-to-End Delay*. Sehingga, jika nilai kenaikan PDR dan E2E dirata-rata, kombinasi ini menghasilkan hasil sebesar 44,6% lebih baik pada skenario *grid* dan 31,2% lebih baik pada skenario *real*. Namun kombinasi ini juga menyebabkan *Routing Overhead* meningkat dikarenakan penggunaan *K-Means Clustering* dan rata-rata kenaikan *Routing Overhead* nya adalah 221,6%.
2. Pengaruh implementasi *K-Means Clustering* dan *Coefficient Link* terhadap performa AODV pada skenario *grid* adalah rata-rata kenaikan PDR sebesar 21,69%, rata-rata penurunan E2E sebesar 67,6%, dan rata-rata kenaikan RO sebesar 241,95%
3. Pengaruh implementasi *K-Means Clustering* dan *Coefficient Link* terhadap performa AODV pada skenario *real* adalah rata-rata kenaikan PDR sebesar 13,15%, rata-rata penurunan E2E sebesar 49,3%, rata-rata kenaikan RO sebesar 201,316%

6.2 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 menstabilkan hasil E2E pada simulasi skenario *Real* supaya hasilnya tidak fluktutatif dan grafiknya cenderung membaik seperti pada simulasi skenario *Grid*.

DAFTAR PUSTAKA

[1] Yudianto dan M. J. Noor, "Ad Hoc Network: IlmuKomputer.com," 31 Januari 2013. [Online]. Available: https://ilmukomputer.org/2013/01/31/ad-hoc-network/. [Diakses 06 Januari 2020].

[2] Schiller dan J. H., "Mobile ad-hoc networks," dalam *Mobile Communications*, Harlow, Pearson Education Limited, 2003, p. 330.

[3] S. Corson dan J. Macker, "RFC 2501 - Mobile Ad hoc Networking (MANET):," Januari 1999. [Online]. Available: https://tools.ietf.org/html/rfc2501. [Diakses 30 Juni 2020].

[4] A. Khattri, "Introduction of Mobile Ad hoc Network (MANET) - GeeksforGeeks," [Online]. Available: https://www.geeksforgeeks.org/introduction-of-mobile-ad-hoc-network-manet/. [Diakses 30 Juni 2020].

[5] A. Faried, "EVALUASI PERFOMANSI PROTOCOL ROUTING AD-HOC ON-DEMAND DISTANCE VECTOR (AODV) PADA JARINGAN HYBRID AD HOC | Faried, Achmad | Undergraduate Theses - ITS Institutional Repository," 27 April 2010. [Online]. Available: http://digilib.its.ac.id/ITS-Undergraduate-3100010038678/9496. [Diakses 21 Agustus 2019].

[6] C. Perkins, E. Belding-Royer dan S. Das, "RFC 3561 - Ad hoc On-Demand Distance Vector (AODV) Routing," Juli 2003. [Online]. Available: https://tools.ietf.org/html/rfc3561#section-1. [Diakses 30 Juni 2020].

[7] A. A. Rahadiyanto dan A. Fatulloh, "Simulasi Pengiriman Data Pada Topologi MANET Dengan

Protokol AODV Menggunakan Aplikasi NS2," Batam, 2017.

[8] A. S. Bhalodia, N. B. Gohil dan C. Bhalodia, "Route Maintenance in AODV Routing Protocol: A Review," *International Journal of Advance Engineering and Research Development,* vol. 2, no. 1, p. 5, 2015.

[9] 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.

[10] T. Liu, Z. Xia, S. Shi dan X. Gu, "A Modified AODV Protocol Based on Nodes Velocity," *Machine Learnings and Intelligent Communications,* pp. 545-554, 2017.

[11] TutorialsWeb, "Network Simulator 2 (NS2) : Features & Basic Architecture Of NS2," TutorialsWeb, [Online]. Available: https://www.tutorialsweb.com/ns2/NS2-1.htm. [Diakses 18 Desember 2019].

[12] A. U. Salleh, Z. Ishak, N. M. Din dan M. Z. Jamaludin, "Trace Analyzer for NS-2," *4th Student Confrence on Research and Development (SCOReD 2006),* pp. 29-32, 2006.

[13] "OpenStreetMap," [Online]. Available: https://www.openstreetmap.org/.. [Diakses 30 Mei 2020].

[14] Humanitarian OpenStreetMap Team, Panduan Dasar OpenStreetMap dengan iD Editor, 2013.

[15] HOT OSM, "LearnOSM," 12 Juli 2015. [Online]. Available: https://learnosm.org/id/josm/start-josm/. [Diakses 30 Juni 2020].

[16] German Aerospace Center, "Documentation - SUMO Documentation," 20 Mei 2020. [Online]. Available:

https://sumo.dlr.de/docs/SUMO_User_Documentation.html. [Diakses 01 Juni 2020].

[17] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. H. L. Lücken, J. Rummel, P. Wagner dan E. Wießner, "Microscopic Traffic Simulation using SUMO," *21st International Conference on Intelligent Transportation Systems,* vol. 21, pp. 2575-2582, 2018.

[18] S. Singh, "AWK command in Unix/Linux with examples - GeeksforGeeks," GeeksforGeeks, [Online]. Available: https://www.geeksforgeeks.org/awk-command-unixlinux-examples/. [Diakses 30 Juni 2020].

LAMPIRAN

A.1 Kode Fungsi send*Request*()

```
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);

 //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 = t_node->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;
 }

  // modifikasi
  if (index == 8){

    t_node=(MobileNode*)(Node::get_node_by_address(index));
    ((MobileNode*) t_node)->getLoc(&xpos,&ypos,&zpos);
    t_node=(MobileNode*)(Node::get_node_by_address(index));
    n_speed=((MobileNode*)t_node)->speed();

    RAODV raodv_data;
    raodv_data.current_speed = n_speed;
    raodv_data.current_linkcoef = 0;
    raodv_data.jml_linkcoef = 0;
    nodedata[index] = raodv_data;

  }

#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*) t_node)->getLoc(&posX,&posY,0);
 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);
//struct node_data nodedata[400];

if (flag_counter == 1) {
   Packet::free(p);
```

```
  }
// MODIFIED //
  ((MobileNode*) t_node)->getLoc(&posX,&posY, 0);
  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 = t_node->energy_model()->energy();
   if(ENERGY_THRESHOLD >= energy)
   {
    Packet::free(p);
    return;
   }

   //HOPCOUNT
   int hopCount = rq->rq_hop_count;

// MODIFIED //
   // linkcoef = pow(rq->rq_speed_prev - n_speed, 2);
   // total_linkcoef = linkcoef + rq->rq_prev_linkcoef;
   RAODV raodv_data;
   raodv_data.current_speed = n_speed;
   raodv_data.current_linkcoef = pow(n_speed - nodedata[rq->rq_src-1].current_speed,
2);
   raodv_data.jml_linkcoef = raodv_data.current_linkcoef + nodedata[rq->rq_src-
1].jml_linkcoef;
   nodedata[index] = raodv_data;


   #ifdef DEBUG
    FILE *fp;
    fp = fopen("debug.txt", "a");
    fprintf(fp, "\n%.6f  %s: in node %d, packet from node %d to node destination %d,
total linkcoef : %2f, linkcoef current : %f |  current_speed: %d | prev speed: %d",
CURRENT_TIME, __FUNCTION__, index, rq->rq_src, rq->rq_dst,
nodedata[index].jml_linkcoef, nodedata[index].current_linkcoef,
nodedata[index].current_speed, nodedata[rq->rq_src-1].current_speed);
    fclose(fp);
   #endif // DEBUG

   /*
    * 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;
  }

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

 aodv_rt_entry *rt0; // rt0 is the reverse route

   rt0 = rtable.rt_lookup(rq->rq_src);
   if(rt0 == 0)
   {
     rt0 = rtable.rt_add(rq->rq_src);
   }

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

   if ( (rq->rq_src_seqno > rt0->rt_seqno ) ||
      ((rq->rq_src_seqno == rt0->rt_seqno) &&
   (rq->rq_hop_count < rt0->rt_hops)) )

   {

rt_update(rt0, rq->rq_src_seqno, rq->rq_hop_count, ih->saddr(), rq->linkcoef,
            max(rt0->rt_expire, (CURRENT_TIME + REV_ROUTE_LIFE)) );
     if (rt0->rt_req_timeout > 0.0)
     {
       rt0->rt_req_cnt = 0;
       rt0->rt_req_timeout = 0.0;
       rt0->rt_req_last_ttl = rq->rq_hop_count;
       rt0->rt_expire = CURRENT_TIME + ACTIVE_ROUTE_TIMEOUT;
     }

     assert (rt0->rt_flags == RTF_UP);
     Packet *buffered_pkt;
     while ((buffered_pkt = rqueue.deque(rt0->rt_dst)))
     {
```

```
      if (rt0 && (rt0->rt_flags == RTF_UP))
      {
        assert(rt0->rt_hops != INFINITY2);
        forward(rt0, buffered_pkt, NO_DELAY);
      }
     }
   }
   // End for putting reverse route in rt table

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

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

if(rq->rq_dst == index) {
    //modifikasi
    counter = counter - 1;

    #ifdef DEBUG
    fp = fopen("debug.txt", "a");
    fprintf(fp, "\n%.6f  node %d - %s, BEFORE IF DEST counter : %d | dest_linkcoef:
%f | total_linkcoef: %f ",  CURRENT_TIME, index, __FUNCTION__, counter,
dest_linkcoef, nodedata[index].jml_linkcoef);
    fclose(fp);
    #endif // DEBUG

    if(dest_linkcoef > nodedata[index].jml_linkcoef)
    {
      dest_linkcoef = nodedata[index].jml_linkcoef;
      dest_index_terkecil = rq->rq_src;
    }


    #ifdef DEBUG
    fp = fopen("debug.txt", "a");
    fprintf(fp, "\n%.6f  node %d - %s, counter : %d | dest_linkcoef: %f | total_linkcoef:
%f ",  CURRENT_TIME, index, __FUNCTION__, counter, dest_linkcoef,
nodedata[index].jml_linkcoef);
    fclose(fp);
    #endif // DEBUG

    if(counter <= 0 && flag_counter == 0){

      flag_counter = 1;
      if(rq->rq_hop_count < 8){
```

```
      #ifdef DEBUG
      // FILE *fp;
      fp = fopen("debug.txt", "a");
      fprintf(fp, "\n%.6f  node %d - %s: destination sending reply via hopcount to :
%d ",  CURRENT_TIME, index, __FUNCTION__, rq->rq_src);
      fclose(fp);
      #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);
    }
    else {

    #ifdef DEBUG
      // FILE *fp;
      fp = fopen("debug.txt", "a");
      fprintf(fp, "\n%.6f  node %d - %s: destination sending reply via link coef via
node : %d",  CURRENT_TIME, index, __FUNCTION__, dest_index_terkecil);
      fclose(fp);
    #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);
   }
   }
  }
```

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

 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::sendReply(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);
double jml_linkcoef;
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_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;

 if(rp->rp_dst == index)
 {
   rp->rp_linkcoef = max(jml_linkcoef,rt->rt_linkcoef);
 }


 // 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->rp_linkcoef, 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*) t_node)->getLoc(&posX,&posY,0);
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*) t_node)->getLoc(&posX,&posY,0);
  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) {

    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;
  Cluster cluster;

  for(int i=0;i<NUMBER_OF_CLUSTER;i++)
  {
   if(i<modClustersNodes) {
     numberNodes = NUMBER_OF_NODE/NUMBER_OF_CLUSTER + 1;
   }
   else {
     numberNodes = NUMBER_OF_NODE/NUMBER_OF_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();
                }

  for(int i=0;i<NUMBER_OF_CLUSTER;i++)
  {
    printf("\nCLUSTER HEAD OF CLUSTER %d : INDEX %d",i,this-
>ct.nodes_cluster_head[i]);
  }
  for(int i=0;i<NUMBER_OF_CLUSTER;i++)
  {
    printf("\nCLUSTER GATEWAY OF CLUSTER %d : INDEX %d",i,this-
>ct.nodes_cluster_gateway[i]);
  }
}
```

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 NS2

```
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(adhoc*Routing*)"
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"
```

A.15 Kode Dalam File AODV.h

```
#ifndef __aodv_h__
#define __aodv_h__



#include <cmu-trace.h>
#include <priqueue.h>
```

```
#include <aodv/aodv_rtable.h>
#include <aodv/aodv_rqueue.h>
#include <classifier/classifier-port.h>
#include <mobilenode.h>
#include <aodv/aodv_kmeans.h>

#define AODV_LOCAL_REPAIR
#define AODV_LINK_LAYER_DETECTION
#define AODV_USE_LL_METRIC

class AODV;

#define MY_ROUTE_TIMEOUT       10
#define ACTIVE_ROUTE_TIMEOUT    10
#define REV_ROUTE_LIFE         6
#define BCAST_ID_SAVE          6
#define RREQ_RETRIES           3
#define MAX_RREQ_TIMEOUT               10.0
#define TTL_START     5
#define TTL_THRESHOLD 7
#define TTL_INCREMENT 2
#define NODE_TRAVERSAL_TIME     0.03
#define LOCAL_REPAIR_WAIT_TIME  0.15
#define NETWORK_DIAMETER       30
#define RREP_WAIT_TIME         1.0
#define ID_NOT_FOUND    0x00
#define ID_FOUND       0x01
#define DELAY 1.0
#define NO_DELAY -1.0
#define ARP_DELAY 0.01
#define HELLO_INTERVAL         1
#define ALLOWED_HELLO_LOSS     3
#define BAD_LINK_LIFETIME      3
#define MaxHelloInterval       (1.25 * HELLO_INTERVAL)
#define MinHelloInterval       (0.75 * HELLO_INTERVAL)

class RAODV {

public:
  int current_speed;
  double current_linkcoef;
  double jml_linkcoef;

};
RAODV nodedata[500];

class ClusteringTimer : public Handler {
```

```
public:
        ClusteringTimer(AODV* a) : agent(a) {}
        void handle(Event*);
private:
        AODV *agent;
        Event intr;
};

class BroadcastTimer : public Handler {
public:
        BroadcastTimer(AODV* a) : agent(a) {}
        void   handle(Event*);
private:
        AODV    *agent;
                Event       intr;
};

class HelloTimer : public Handler {
public:
        HelloTimer(AODV* a) : agent(a) {}
        void   handle(Event*);
private:
        AODV    *agent;
                Event       intr;
};

class NeighborTimer : public Handler {
public:
        NeighborTimer(AODV* a) : agent(a) {}
        void   handle(Event*);
private:
        AODV    *agent;
                Event       intr;
};

class RouteCacheTimer : public Handler {
public:
        RouteCacheTimer(AODV* a) : agent(a) {}
        void   handle(Event*);
private:
        AODV    *agent;
                Event       intr;
};

class LocalRepairTimer : public Handler {
public:
        LocalRepairTimer(AODV* a) : agent(a) {}
```

```
        void   handle(Event*);
private:
        AODV    *agent;
                Event        intr;
};

class BroadcastID {
        friend class AODV;
 public:
        BroadcastID(nsaddr_t i, u_int32_t b) { src = i; id = b;  }
 protected:
        LIST_ENTRY(BroadcastID) link;
        nsaddr_t        src;
        u_int32_t       id;
        double          expire;         // now + BCAST_ID_SAVE s
};

LIST_HEAD(aodv_bcache, BroadcastID);

#define NUMBER_OF_CLUSTER       15
#define NUMBER_OF_NODE          200
#define SIGNAL_THRESHOLD        -75.00
#define ENERGY_THRESHOLD        10.00

struct nodes_positions_table {
        double          nodes_position_x[NUMBER_OF_NODE];
        double          nodes_position_y[NUMBER_OF_NODE];
        double          nodes_timestamp[NUMBER_OF_NODE];
};

struct nodes_clusters_table {
        int             nodes_cluster_head[NUMBER_OF_CLUSTER];
        int             nodes_cluster_gateway[NUMBER_OF_CLUSTER];
        double          nodes_cluster_timestamp;
};

class AODV: public Agent {

        friend class aodv_rt_entry;
        friend class BroadcastTimer;
        friend class HelloTimer;
        friend class NeighborTimer;
        friend class RouteCacheTimer;
        friend class LocalRepairTimer;
        friend class ClusteringTimer;

 public:
```

```
    AODV(nsaddr_t id);

    void                recv(Packet *p, Handler *);

protected:
    int         command(int, const char *const *);
    int         initialized() { return 1 && target_; }

    MobileNode *t_node;
        MobileNode *mNode;

    void        rt_resolve(Packet *p);
    void        rt_update(aodv_rt_entry *rt, u_int32_t seqnum,
                                        u_int16_t metric, nsaddr_t nexthop,
                                        double expire_time);
    void        rt_down(aodv_rt_entry *rt);
    void        local_rt_repair(aodv_rt_entry *rt, Packet *p);
public:
    void        rt_ll_failed(Packet *p);
    void        handle_link_failure(nsaddr_t id);
protected:
    void        rt_purge(void);

    void        enque(aodv_rt_entry *rt, Packet *p);
    Packet*     deque(aodv_rt_entry *rt);

    void        nb_insert(nsaddr_t id);
    AODV_Neighbor*      nb_lookup(nsaddr_t id);
    void        nb_delete(nsaddr_t id);
    void        nb_purge(void);

    void        id_insert(nsaddr_t id, u_int32_t bid);
    bool        id_lookup(nsaddr_t id, u_int32_t bid);
    void        id_purge(void);

    void        forward(aodv_rt_entry *rt, Packet *p, double delay);
    void        sendHello(void);
    void        sendRequest(nsaddr_t dst);
    void        sendReply(nsaddr_t ipdst, u_int32_t hop_count,
                        nsaddr_t rpdst, u_int32_t rpseq,
                        u_int32_t lifetime, double timestamp);
    void        sendError(Packet *p, bool jitter = true);

    void        recvAODV(Packet *p);
    void        recvHello(Packet *p);
    void        recvRequest(Packet *p);
    void        recvReply(Packet *p);
```

```
        void        recvError(Packet *p);

                double                  PerHopTime(aodv_rt_entry *rt);

        nsaddr_t    index;              // IP Address of this node
        u_int32_t   seqno;              // Sequence Number
        int         bid;                // Broadcast ID

        aodv_rtable     rthead;         // routing table
        aodv_ncache     nbhead;         // Neighbor Cache
        aodv_bcache     bihead;         // Broadcast ID Cache

        BroadcastTimer  btimer;
        HelloTimer      htimer;
        NeighborTimer   ntimer;
        RouteCacheTimer rtimer;
        LocalRepairTimer lrtimer;
        ClusteringTimer ctimer;

        aodv_rtable     rtable;
        aodv_rqueue     rqueue;
        Trace       *logtarget;
        PriQueue    *ifqueue;

        void        log_link_del(nsaddr_t dst);
        void        log_link_broke(Packet *p);
        void        log_link_kept(nsaddr_t dst);

                PortClassifier *dmux_;

        struct nodes_positions_table pt;
        struct nodes_clusters_table ct;
        bool isClusterHead;
        double energy;
public:
        void runCluster();
        double      posX;
        double      posY;
        double      posZ;
                double          xpos;
                double          ypos;
                double          zpos;
        double      iEnergy;
        MobileNode  *iNode;
        int n_speed;
        double linkcoef;
        double total_linkcoef;
```

```
        int rq_speed_prev;
        int temp_speed;
        int temp_speed_i;
        int jml_packet;
        int timernya;
        int counter;
        double dest_linkcoef;
        int dest_index_terkecil;
        int flag_counter;
};

#endif /* __aodv_h__ */
```

A.16 Kode Dalam File AODV_packet.h

```
#ifndef __aodv_packet_h__
#define __aodv_packet_h__

#include "aodv.h"

#define AODV_MAX_ERRORS 100
#define AODVTYPE_HELLO    0x01
#define AODVTYPE_RREQ     0x02
#define AODVTYPE_RREP     0x04
#define AODVTYPE_RERR     0x08
#define AODVTYPE_RREP_ACK           0x10
#define HDR_AODV(p)                 ((struct hdr_aodv*)hdr_aodv::access(p))
#define HDR_AODV_REQUEST(p)         ((struct
hdr_aodv_request*)hdr_aodv::access(p))
#define HDR_AODV_REPLY(p)           ((struct
hdr_aodv_reply*)hdr_aodv::access(p))
#define HDR_AODV_ERROR(p)           ((struct
hdr_aodv_error*)hdr_aodv::access(p))
#define HDR_AODV_RREP_ACK(p)        ((struct
hdr_aodv_rrep_ack*)hdr_aodv::access(p))

struct hdr_aodv {
      u_int8_t       ah_type;

            static int offset_; // required by PacketHeaderManager
            inline static int& offset() { return offset_; }
            inline static hdr_aodv* access(const Packet* p) {
                  return (hdr_aodv*) p->access(offset_);
            }
};

struct hdr_aodv_request {
```

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

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

        double          rq_timestamp;   // when REQUEST sent;
                                                        // used to compute route
discovery latency

        // MODIFIED //
        struct nodes_positions_table pt;
        int     rq_speed_prev;          // add speed / velocity (Vi)
                double rq_prev_linkcoef;
                double rq_linkcoef;
                double linkcoef;

  inline int size() {
  int sz = 0;
                sz = 8*sizeof(u_int32_t);
                assert (sz >= 0);
                return sz;
  }
};

struct hdr_aodv_reply {
        u_int8_t        rp_type;        // Packet Type
        u_int8_t        reserved[2];
        u_int8_t        rp_hop_count;           // Hop Count
        nsaddr_t        rp_dst;                 // Destination IP Address
        u_int32_t       rp_dst_seqno;           // Destination Sequence Number
        nsaddr_t        rp_src;                 // Source IP Address
        double                  rp_lifetime;            // Lifetime

        double          rp_timestamp;           // when corresponding REQ sent;

        struct nodes_positions_table pt;
        struct nodes_clusters_table ct;
        double rp_linkcoef;                 // add link coefficient, for check later
                                int             rp_speed_i;             // add speed / velocity (Vi)
                                int             rp_speed_j;             // add speed ke j alias speed
di node skrg
        // MODIFIED //
```

```
  inline int size() {
  int sz = 0;
                sz = 6*sizeof(u_int32_t);
                assert (sz >= 0);
                return sz;
  }

};

struct hdr_aodv_error {
        u_int8_t        re_type;                // Type
        u_int8_t        reserved[2];            // Reserved
        u_int8_t        DestCount;              // DestCount
        // List of Unreachable destination IP addresses and sequence numbers
        nsaddr_t        unreachable_dst[AODV_MAX_ERRORS];
        u_int32_t       unreachable_dst_seqno[AODV_MAX_ERRORS];

  inline int size() {
  int sz = 0;
                sz = (DestCount*2 + 1)*sizeof(u_int32_t);
                assert(sz);
        return sz;
  }

};

struct hdr_aodv_rrep_ack {
                u_int8_t    rpack_type;
                u_int8_t    reserved;
};

union hdr_all_aodv {
  hdr_aodv          ah;
  hdr_aodv_request  rreq;
  hdr_aodv_reply    rrep;
  hdr_aodv_error    rerr;
  hdr_aodv_rrep_ack rrep_ack;
};

#endif /* __aodv_packet_h__ */
```

A.17 Kode Dalam File AODV_rtable.h

```
#ifndef __aodv_rtable_h__
#define __aodv_rtable_h__
```

```
#include <assert.h>
#include <sys/types.h>
#include <config.h>
#include <lib/bsd-list.h>
#include <scheduler.h>

#define CURRENT_TIME    Scheduler::instance().clock()
#define INFINITY2       0xff

class AODV_Neighbor {
        friend class AODV;
        friend class aodv_rt_entry;
 public:
        AODV_Neighbor(u_int32_t a) { nb_addr = a; }

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

LIST_HEAD(aodv_ncache, AODV_Neighbor);

class AODV_Precursor {
        friend class AODV;
        friend class aodv_rt_entry;
 public:
        AODV_Precursor(u_int32_t a) { pc_addr = a; }

 protected:
        LIST_ENTRY(AODV_Precursor) pc_link;
        nsaddr_t        pc_addr;        // precursor address
};

LIST_HEAD(aodv_precursors, AODV_Precursor);

class aodv_rt_entry {
        friend class aodv_rtable;
        friend class AODV;
                friend class LocalRepairTimer;
 public:
        aodv_rt_entry();
        ~aodv_rt_entry();

        void            nb_insert(nsaddr_t id);
        AODV_Neighbor*  nb_lookup(nsaddr_t id);
```

```
        void            pc_insert(nsaddr_t id);
        AODV_Precursor* pc_lookup(nsaddr_t id);
        void            pc_delete(nsaddr_t id);
        void            pc_delete(void);
        bool            pc_empty(void);

        double          rt_req_timeout;      // when I can send another req
        u_int8_t        rt_req_cnt;          // number of route requests

        // Modifikasi
        double     rq_linkcoef;    // add link coefficient, for check later
        double     rq_speed_i;     // add speed / velocity (Vi)
        double     rq_speed_j;     // add speed ke j alias speed di node skrg

    protected:
        LIST_ENTRY(aodv_rt_entry) rt_link;

        nsaddr_t        rt_dst;
        u_int32_t       rt_seqno;
        u_int16_t       rt_hops;            // hop count
          int           rt_last_hop_count;  // last valid hop count
        nsaddr_t        rt_nexthop;         // next hop IP address
        aodv_precursors rt_pclist;
        double          rt_expire;          // when entry expires
        u_int8_t        rt_flags;
        double          rt_linkcoef;        // link coefficient value

    #define RTF_DOWN 0
    #define RTF_UP 1
    #define RTF_IN_REPAIR 2
    #define MAX_HISTORY           3
                double              rt_disc_latency[MAX_HISTORY];
                char                hist_indx;
        int             rt_req_last_ttl;       // last ttl value used
        aodv_ncache     rt_nblist;
    };

    class aodv_rtable {
     public:
                aodv_rtable() { LIST_INIT(&rthead); }

        aodv_rt_entry*      head() { return rthead.lh_first; }

        aodv_rt_entry*      rt_add(nsaddr_t id);
        void            rt_delete(nsaddr_t id);
        aodv_rt_entry*      rt_lookup(nsaddr_t id);
```

```
 private:
      LIST_HEAD(aodv_rthead, aodv_rt_entry) rthead;
};

#endif /* _aodv__rtable_h__ */
```

A.18 Hasil Simulasi AODV Real yang sudah di modifikasi

Catatan: Hasil RO dibulatkan keatas

AODV Real Cluster 10									
Jumlah node	PDR 1	PDR 2	PDR 3	Rata2	Jumlah node	E2E 1 (ms)	E2E 2 (ms)	E2E 3 (ms)	Rata2
50 node	0,8316	0,9276	0,7196	0,8263	50 node	319,325	483,468	541,823	448,205
100 node	0,9529	0,8081	0,8617	0,8742	100 node	95,0509	929,718	939,008	654,592
150 node	0,8196	0,7026	0,7231	0,7484	150 node	1581,15	2747,5	1362,08	1896,910
200 node	0,1951	0,2953	0,9585	0,4830	200 node	2441,47	6102,3	386,828	2976,866
250 node	0,6979	0,5590	0,8878	0,7149	250 node	734,674	1394,44	735,847	954,987
300 node	0,8878	0,9643	0,9077	0,9199	300 node	519,286	243,048	142,284	301,539

AODV Real Cluster 15									
Jumlah node	PDR 1	PDR 2	PDR 3	Rata2	Jumlah node	E2E 1 (ms)	E2E 2 (ms)	E2E 3 (ms)	Rata2
50 node	0,9577	0,9433	0,9579	0,9530	50 node	864,522	526,452	545,121	645,365
100 node	0,9026	0,8162	0,8053	0,8414	100 node	887,021	1088,39	122,178	699,196
150 node	0,6193	0,8848	0,8769	0,7937	150 node	1848,14	681,637	733,665	1087,814
200 node	0,5330	0,3951	0,6943	0,5408	200 node	2287,51	2560,6	1626,7	2158,270
250 node	0,7990	0,6237	0,7143	0,7123	250 node	1584,6	1307,35	753,819	1215,256
300 node	0,9100	0,7879	0,7310	0,8096	300 node	182,428	1269,88	1119,41	857,239

AODV Real Cluster 20									
Jumlah node	PDR 1	PDR 2	PDR 3	Rata2	Jumlah node	E2E 1 (ms)	E2E 2 (ms)	E2E 3 (ms)	Rata2
50 node	0,7173	0,9188	0,7750	0,8037	50 node	942,672	1332,66	575,189	950,174
100 node	0,7868	0,9538	0,9231	0,8879	100 node	1508,58	3086,46	617,785	1737,608
150 node	0,7680	0,3918	0,5585	0,5728	150 node	431,676	3337,41	4281,2	2683,429
200 node	0,2121	0,3904	0,7610	0,4545	200 node	3024,5	847,508	355,02	1409,009
250 node	0,7577	0,8806	0,8265	0,8216	250 node	3617,51	537,946	917,145	1690,867
300 node	0,7588	0,7989	0,6031	0,7203	300 node	956,66	2047,57	2005,86	1670,030

RO Revisi Cluster 20				
Jumlah node	RO 1	RO 2	RO 3	Rata2
50 node	2014	2015	2107	2045
100 node	4912	4906	4813	4877
150 node	6203	9335	7250	7596
200 node	7669	7346	7834	7616
250 node	12981	12789	14663	13478
300 node	15704	20967	16270	17647

A.19 Hasil Simulasi AODV Grid yang sudah di modifikasi

Catatan: Hasil RO dibulatkan keatas

AODV Grid Cluster 10									
Jumlah node	PDR 1	PDR 2	PDR 3	Rata2	Jumlah node	E2E 1 (ms)	E2E 2 (ms)	E2E 3 (ms)	Rata2
50 node	0,7249	0,9394	0,9021	0,8555	50 node	642,044	491,771	331,487	488,434
100 node	0,9219	0,9350	0,7931	0,8833	100 node	549,226	741,495	1385,89	892,204
150 node	0,4789	0,6915	0,6029	0,5911	150 node	4269,20	2417,68	3171,92	3286,267
200 node	0,9100	0,2216	0,8756	0,6691	200 node	351,182	6843,37	465,76	2553,437
250 node	0,9215	0,6839	0,7474	0,7843	250 node	779,725	2501,95	1460,78	1580,818
300 node	0,8196	0,9062	0,8477	0,8578	300 node	1205,73	358,525	1097,51	887,255

AODV Grid Cluster 15									
Jumlah node	PDR 1	PDR 2	PDR 3	Rata2	Jumlah node	E2E 1 (ms)	E2E 2 (ms)	E2E 3 (ms)	Rata2
50 node	0,9574	0,4684	0,5956	0,6738	50 node	510,446	1425,01	1022,36	985,939
100 node	0,9397	0,5632	0,6243	0,7091	100 node	162,368	587,624	618,536	456,176
150 node	0,6436	0,9588	0,9010	0,8345	150 node	2960,3	336,509	1098,35	1465,053
200 node	0,6617	0,8995	0,2915	0,6176	200 node	1025,9	683,254	898,227	869,127
250 node	0,8131	0,8571	0,6477	0,7726	250 node	1491,92	1255,82	1767,6	1505,113
300 node	0,9231	0,6667	0,7806	0,7901	300 node	182,239	1267,58	731,998	727,272

AODV Grid Cluster 20									
Jumlah node	PDR 1	PDR 2	PDR 3	Rata2	Jumlah node	E2E 1 (ms)	E2E 2 (ms)	E2E 3 (ms)	Rata2
50 node	0,9634	0,6134	0,9683	0,8484	50 node	586,072	2100	301,066	995,713
100 node	0,8788	0,8281	0,7398	0,8156	100 node	487,049	685,149	583,811	585,336
150 node	0,8031	0,7761	0,9378	0,8390	150 node	1179,31	1952,54	360,456	1164,102
200 node	0,8223	0,6332	0,3918	0,6158	200 node	746,221	2335,45	1398,95	1493,540
250 node	0,6425	0,8158	0,7448	0,7344	250 node	1444,77	822,671	1039,73	1102,390
300 node	0,8317	0,8394	0,7968	0,8226	300 node	522,172	635,566	1831,24	996,326

Jumlah node	RO 1	RO 2	RO 3	Rata2
50 node/15	1773	1737	1655	1722
100 node/15	4379	4360	4440	4393
150 node/20	8210	7516	10917	8881
200 node/20	8981	8295	9839	9038
250 node/20	12374	11845	14205	12808
300 node/20	20688	20447	16384	19173

(Halaman ini sengaja dikosongkan)

BIODATA PENULIS

AKMAL DARARI RAFIF BASKORO, lahir di Banyuwangi, 17 Januari 1997. Penulis adalah anak tunggal. Penulis menempuh pendidikan sekolah dasar di SD Islam Al-Azhar 11 Surabaya, lalu melanjutkan pendidikan sekolah menengah pertama di SMP Negeri 1 Surabaya, kemudian menempuh pendidikan sekolah menengah atas di SMA Negeri 2 Surabaya dan akhirnya menempuh pendidikan sarjana di Departemen Informatika, Fakultas Teknologi Elektro dan Informatika Cerdas, 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, Dewan Perwakilan Mahasiswa Fakultas Teknologi Informasi dan Komunikasi, dan Badan Eksekutif Mahasiswa Fakultas Teknologi Informasi dan Komunikasi Pada tahun 2018 penulis pernah melakukan magang di BADAN PENGKAJIAN DAN PENERAPAN TEKNOLOGI (BPPT), dimana BPPT adalah adalah Lembaga Pemerintah Non Departemen yang berada di bawah koordinasi Kementerian Riset, Teknologi dan Pendidikan Tinggi yang mempunyai tugas melaksanakan tugas pemerintahan di bidang pengkajian dan penerapan teknologi. Penulis dapat dihubungi melalui nomor handphone 085258405676 atau e-mail di akmal.drb@gmail.com