



TUGAS AKHIR - IF184802

IMPLEMENTASI K-MEANS CLUSTERING DAN MODIFIKASI ADHOC ON-DEMAND DISTANCE VECTOR ROUTING PROTOCOL PADA MOBILE ADHOC NETWORK UNTUK MENINGKATKAN PERFORMA PENGIRIMAN ANTAR NODE

FAHRIZAL NAUFAL AHMAD
NRP 05111640000135

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

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

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

(Halaman ini sengaja dikosongkan)



TUGAS AKHIR - IF184802

IMPLEMENTASI K-MEANS CLUSTERING DAN MODIFIKASI ADHOC ON-DEMAND DISTANCE VECTOR ROUTING PROTOCOL PADA MOBILE ADHOC NETWORK UNTUK MENINGKATKAN PERFORMA PENGIRIMAN ANTAR NODE

FAHRIZAL NAUFAL AHMAD
NRP 05111640000135

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

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

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

(Halaman ini sengaja dikosongkan)



UNDERGRADUATE THESIS - IF184802

IMPLEMENTATION OF K-MEANS CLUSTERING AND BACKUP ROUTING ALGORITHM IN ADHOC ON-DEMAND DISTANCE VECTOR TO ENHANCE MOBILE NODE PERFORMANCE ON WIRELESS ADHOC NETWORK

**FAHRIZAL NAUFAL AHMAD
NRP 05111640000135**

First Advisor

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

Second Advisor

Ir.F.X. Arunanto M.Sc.

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 DAN MODIFIKASI ADHOC ON-DEMAND DISTANCE VECTOR ROUTING PROTOCOL PADA MOBILE ADHOC NETWORK UNTUK MENINGKATKAN PERFORMA PENGIRIMAN ANTAR NODE

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:

FAHRIZAL NAUFAL AHMAD
NRP: 05111640000135

Disetujui oleh Pembimbing Tugas Akhir:

- | | |
|--|-------------------------|
| 1. Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.
(NIP. 198410162008121002) |
(Pembimbing 1) |
| 2. Ir.F.X. Arunanto M.Sc.
(NIP. 195701011983031004) |
(Pembimbing 2) |

SURABAYA
JUNI, 2020

(Halaman ini sengaja dikosongkan)

IMPLEMENTASI K-MEANS CLUSTERING DAN MODIFIKASI ADHOC ON-DEMAND DISTANCE VECTOR ROUTING PROTOCOL PADA MOBILE ADHOC NETWORK UNTUK MENINGKATKAN PERFORMA PENGIRIMAN ANTAR NODE

Nama Mahasiswa : Fahrizal Naufal Ahmad
NRP : 05111640000135
Departemen : Teknik Informatika
Dosen Pembimbing 1 : Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.
Dosen Pembimbing 2 : Ir.F.X. Arunanto M.Sc.

Abstrak

Adhoc On-Demand Distance Vector(AODV) merupakan salah satu *routing protocol* yang dapat diaplikasikan ke pada *Mobile Adhoc Network* (MANET). AODV sendiri memiliki dua fase, yaitu *route discovery* dan *route maintenance*. *Route Discovery* sendiri memiliki dua fase yaitu *Route Request* (RREQ) dan *Route Reply* (RREP)

Dalam AODV, *route* yang dipilih adalah *route* dengan jumlah hop terkecil tanpa memperhatikan faktor-faktor esensial lainnya yang dapat memengaruhi dalam pemilihan suatu *route*. Ada beberapa faktor lain yang bisa mempengaruhi pemilihan rute.

Pada Tugas Akhir ini, Penulis mengusulkan sebuah algoritma bernama *Backup Routing* untuk diimplementasikan di dalam AODV. Untuk lebih meningkatkan performa, Penulis juga mengusulkan sebuah metode *clustering*, yaitu *K-Means Clustering*. Dua metode tersebut akan menghitung seluruh rute yang tersedia lalu melakukan *clustering* sebelum mengirimkan sebuah paket.

Pada akhir pengujian, dua metode tersebut yang telah diimplementasikan akan disimulasikan. Setelah disimulasikan, terdapat data yang menunjukkan bahwa adanya peningkatan performa dari AODV yang telah dimodifikasi.

Kata kunci: MANET, AODV, Backup Routing, K-Means, Clustering

IMPLEMENTATION OF K-MEANS CLUSTERING AND BACKUP ROUTING ALGORITHM IN ADHOC ON-DEMAND DISTANCE VECTOR TO ENHANCE MOBILE NODE PERFORMANCE ON WIRELESS ADHOC NETWORK

Student Name : Fahrizal Naufal Ahmad
Student ID : 05111640000135
Department : Teknik Informatika
First Advisor : Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.
Second Advisor : Ir.F.X. Arunanto M.Sc.

Abstract

Adhoc On-Demand Distance Vector (AODV) is a routing protocol that can be applied to the Mobile Adhoc Network (MANET). AODV itself has two phases, namely route discovery and route maintenance. Route Discovery itself has two phases namely Route Request (RREQ) and Route Reply (RREP)

In AODV, the selected route is the route with the smallest number of hops without looking to other essential factors that can influence the selection of a route. There are several factors that can influence route selection.

In this Undergraduate Thesis, the author proposes an algorithm called Backup Routing to be implemented in AODV. To improve performance, the author also proposes a clustering method, namely K-Means Clustering. The two methods will calculate all available routes and then cluster before sending a packet.

At the end of the test, the two methods that have been implemented will be simulated. After being simulated, there are data that show an increase in the performance of the modified AODV.

Keyword: MANET, AODV, Backup Routing, K-Means, Clustering

KATA PENGANTAR

Puji syukur kepada Allah SWT atas segala karunia dan rahmat-Nya penulis dapat menyelesaikan Tugas Akhir yang berjudul ***“IMPLEMENTASI K-MEANS CLUSTERING DAN ALGORITMA BACKUP ROUTING PADA AD-HOC ON-DEMAND DISTANCE VECTOR UNTUK MENINGKATKAN PERFORMA MOBILE NODE PADA MOBILE AD-HOC NETWORK”***.

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

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

1. Allah SWT atas semua rahmat yang diberikan sehingga penulis dapat menyelesaikan Tugas Akhir ini.
2. Keluarga penulis (Bapak Muhammad Thobii, Ibu Maria Magdalena Sri Haryani, Maulida Rahma Britania, dan Risyad Athaya Muhammad) 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 Ir.F.X. Arunanto M.Sc. selaku dosen pembimbing, atas arahan dan bantuannya dalam pengerjaan Tugas Akhir ini.
4. Herninta Lagoon Fatika, selaku kerabat terdekat saya yang selalu menemani saya serta memberikan dukungan dan semangat yang tulus dalam suka dan duka mengerjakan Tugas Akhir ini.
5. Alcredo Simanjuntak selaku kerabat dekat yang mempunyai peran krusial dalam memberikan arahan dalam pengerjaan Tugas Akhir ini.

6. Irham Muhammad Fadhil, selaku kerabat saya yang selalu membantu dan memberikan arahan saat mengerjakan Tugas Akhir
7. Angkatan 2016 Informatika ITS (C20) yang telah memberikan dukungan saat 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

Fahrizal Naufal Ahmad

(Halaman ini sengaja dikosongkan)

DAFTAR ISI

Abstrak	vii
<i>Abstract</i>	<i>ix</i>
DAFTAR ISI	xiv
DAFTAR GAMBAR	xviii
DAFTAR TABEL	xx
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 Metodologi.....	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 <i>Mobile Ad-hoc Networks</i> (MANETs).....	8
2.2 <i>Ad-hoc On-demand Distance Vector</i> (AODV).....	10
2.2.1 <i>Control Messages</i>	10
2.2.2 <i>Route Discovery</i>	12
2.2.3 <i>Route Maintenance</i>	14
2.3 <i>K-Means Clustering</i>	15
2.4 <i>Backup Routing</i>	16
2.4.1 <i>Refined Route Request</i> (RRREQ)	17
2.4.2 <i>Refined Route Reply</i> (RRREP)	18
2.4.3 <i>Refined Route Error</i> (RRERR).....	19

2.5 <i>Network Simulator - 2</i>	20
2.6 OpenStreetMap (OSM).....	20
2.7 Java OpenStreetMap (JOSM)	21
2.8 Simulation of Urban Mobility (SUMO)	21
2.9 AWK.....	22
BAB III PERANCANGAN	25
3.1 Deskripsi Umum.....	25
3.2 Perancangan Skenario Mobilitas	27
3.3 Perancangan Modifikasi Protokol AODV	27
3.3.1 Perancangan K-Means Clustering	29
3.3.2 Perancangan Backup Routing	33
3.4 Perancangan Simulasi pada NS-2.....	34
3.5 Perancangan Metrik Analisis	35
3.5.1 Packet Delivery Ratio (PDR).....	35
3.5.2 Average End-to-End Delay (E2E)	36
3.5.3 Routing Overhead (RO).....	36
3.5.4 Average Hop Count (HC)	36
BAB IV IMPLEMENTASI.....	39
4.1 Implementasi Skenario <i>Grid</i>	39
4.2 Implementasi Skenario <i>Real</i>	42
4.3 Implementasi <i>K-Means Clustering</i> pada AODV	43
4.3.1 Modifikasi Pengiriman RREQ.....	46
4.3.2 Modifikasi Penerimaan RREQ	47
4.3.3 Modifikasi Pengiriman RREP	48
4.3.4 Modifikasi Penerimaan RREP	48
4.3.5 Modifikasi Pengiriman Hello Messages	49
4.4 Implementasi algoritma <i>Backup Routing</i> pada AODV	50
4.4.1 Modifikasi Pengiriman RREQ.....	51
4.4.2 Modifikasi Penerimaan RREQ	52
4.5 Implementasi Simulasi pada NS-2.....	53
4.6 Implementasi Metrik Analisis	53
BAB V UJI COBA DAN EVALUASI.....	56
5.1 Lingkungan Uji Coba	56
5.2 Skenario <i>Grid</i>	56
5.3 Skenario <i>Real</i>	62

BAB VI KESIMPULAN DAN SARAN	69
6.1 Kesimpulan	69
6.2 Saran	70
DAFTAR PUSTAKA	72
LAMPIRAN	73
A.1 Kode runCluster()	73
A.2 Kode run().....	73
A.3 Kode getResult().....	74
A.4 Kode getClusterHead()	75
A.5 Kode getClusterGateway()	76
A.6 Kode sendRequest().....	76
A.7 Kode recvRequest()	80
A.8 Kode sendReply()	85
A.9 Kode recvReply().....	87
A.10 Kode sendHello().....	90
A.11 Kode konfigurasi skenario NS-2.....	91
A.12 Kode konfigurasi awk.....	94
BIODATA PENULIS	97

(Halaman ini sengaja dikosongkan)

DAFTAR GAMBAR

Gambar 2.1 Ilustrasi Jaringan MANETs.....	9
Gambar 2.2 Flowchart <i>Proses</i> Control Message.....	14
Gambar 2.3 <i>Ilustrasi terjadinya RERR</i>	15
Gambar 2.4 <i>Flowchart dari K-Means Clustering</i>	16
Gambar 2.5 Ilustrasi proses berjalannya RRREQ.....	17
Gambar 2.6 Ilustrasi proses berjalannya RRREP.....	18
Gambar 2.7 Ilustrasi proses berjalannya RRERR.....	19
Gambar 2.8 Struktur <i>File AWK</i>	23
Gambar 3.1 Diagram Simulasi AODV.....	25
Gambar 3.2 Alur Protokol AODV Modifikasi.....	28
Gambar 3.3 <i>Pseudocode Proses Pemindahan Data Posisi Node</i>	30
Gambar 3.4 <i>Pseudocode Proses K-Means Clustering</i>	30
Gambar 3.5 <i>Pseudocode Pemilihan Cluster Head</i>	31
Gambar 3.6 <i>Pseudocode Pemilihan Cluster Gateway</i>	31
Gambar 3.7 <i>Pseudocode Pergantian Informasi</i>	33
Gambar 3.8 <i>Pseudocode pemilihan forwarding node</i>	34
Gambar 4.1 Perintah saat melakukan netgenerate.....	39
Gambar 4.2 Peta Grid Hasil netgenerate.....	40
Gambar 4.3 Perintah dari Tools randomTrips.....	40
Gambar 4.4 Perintah dari duarouter.....	40
Gambar 4.5 Isi dari File Skrip .sumocfg.....	41
Gambar 4.6 Perintah SUMO untuk melakukan konversi.....	41
Gambar 4.7 Perintah dari traceExporter.py.....	41
Gambar 4.8 Ekspor area dari OpenStreetMap.....	42
Gambar 4.9 Perintah dari netconvert.....	42
Gambar 4.10 Hasil Konversi dari Peta Real.....	43
Gambar 4.11 Proses Inisiasi K-Means Clustering.....	43
Gambar 4.12 Proses <i>K-Means Clustering</i>	44
Gambar 4.13 Proses Pengambilan <i>Cluster Head</i> dan <i>Cluster Gateway</i> . 45	
Gambar 4.14 Modifikasi Pengiriman RREQ untuk K-Means.....	46
Gambar 4.15 Modifikasi Penerimaan RREQ untuk K-Means.....	47

Gambar 4.16 Modifikasi Pengiriman RREP untuk K-Means.....	48
Gambar 4.17 Modifikasi Penerimaan RREP untuk K-Means.....	49
Gambar 4.18 Modifikasi Pengiriman <i>Hello Messages</i>	50
Gambar 4.19 Proses Inisiasi <i>Backup Routing</i>	51
Gambar 4.20 Modifikasi Pengiriman RREQ pada <i>Backup Routing</i>	51
Gambar 4.21 Modifikasi Penerimaan RREQ pada <i>Backup Routing</i>	52
Gambar 4.22 Konfigurasi Lingkungan Simulasi.....	53
Gambar 4.23 File Metrik Analisis.....	54
Gambar 5.1 Grafik PDR Hasil Simulasi Skenario Grid.....	57
Gambar 5.2 Grafik E2E Hasil Simulasi Skenario Grid.....	59
Gambar 5.3 Grafik RO Hasil Simulasi Skenario Grid.....	60
Gambar 5.4 Grafik AHC Hasil Simulasi Skenario Grid.....	61
Gambar 5.5 Grafik PDR Hasil Simulasi Skenario Real.....	63
Gambar 5.6 Grafik E2E Hasil Simulasi Skenario Real.....	64
Gambar 5.7 Grafik RO Hasil Simulasi Skenario Real.....	65
Gambar 5.8 Grafik AHC Hasil Simulasi Skenario Real.....	66

DAFTAR TABEL

Tabel 2.1 Format dari RREQ.....	11
Tabel 2.2 Format dari RREP.....	11
Tabel 2.3 <i>List Tools</i> yang tersedia pada SUMO.....	21
Tabel 3.1 Daftar Istilah.....	26
Tabel 3.2 <i>Nodes Position Table</i>	29
Tabel 3.3 <i>Nodes Clusters Table</i>	32
Tabel 3.4 Spesifikasi Skenario NS2.....	35
Tabel 5.1 Spesifikasi Perangkat Simulasi.....	56
Tabel 5.2 PDR dari Hasil Simulasi Skenario Grid.....	57
Tabel 5.3 E2E dari Hasil Simulasi Skenario Grid.....	58
Tabel 5.4 RO dari Hasil Simulasi Skenario Grid.....	59
Tabel 5.5 HC dari Hasil Simulasi Skenario Grid.....	61
Tabel 5.6 PDR dari Hasil Simulasi Skenario Real.....	62
Tabel 5.7 E2E dari Hasil Simulasi Skenario Real.....	63
Tabel 5.8 RO dari Hasil Simulasi Skenario Real.....	65
Tabel 5.9 HC dari Hasil Simulasi Skenario Real.....	66

BAB I PENDAHULUAN

1.1 Latar Belakang

Dalam kehidupan sehari-hari, banyak sekali aktivitas yang dilakukan, mulai dari pekerjaan, hiburan, olahraga, bersosialisasi, dan aktivitas-aktivitas lain yang sangatlah banyak. Bersosialisasi pun menjadi salah satu aktivitas yang sangatlah sering dilakukan. Maka dari itu muncul teknologi-teknologi yang berkembang dari aktivitas tersebut. Seperti telepon, ponsel pintar, handie talkie, dan teknologi-teknologi lain yang sudah berkembang di era informatika. Salah satu teknologi yang cukup sering ditemui adalah Handie Talkie, sebuah perangkat nirkabel yang terhubung dengan perangkat lainnya tanpa menggunakan satu perangkat sebagai pusat menyalurkan data. Teknologi yang digunakan pada perangkat tersebut dinamakan Ad-hoc On demand Distance Vector (AODV).

AODV adalah distance vector routing protocol yang termasuk dalam klasifikasi reaktif routing protocol, yang hanya me-request sebuah rute saat dibutuhkan. AODV yang standar ini dikembangkan oleh C. E. Perkins, E.M. Belding-Royer dan S. Das pada RFC 3561. Teknologi AODV pun pada akhirnya dikembangkan dan akhirnya muncul teknologi baru yang bisa digunakan pada perangkat bergerak, yaitu Mobile Ad-hoc Network (MANET).

Di dalam MANET, antara node yang berbeda dapat terhubung melalui transmisi wireless secara langsung, akan tetapi jika salah satu node diluar jangkauan transmisi maka membutuhkan node lain untuk meneruskan pesan. Oleh karena itu muncul skenario multi hop, dimana ada beberapa host yang berfungsi sebagai relay untuk meneruskan paket dari host sumber menuju kepada host target.

Sama seperti AODV, MANET juga menggunakan pesan route request (RREQ), route reply (RREP), and route error (RERR). Akan tetapi, karena sifat MANET yang sangat dinamis, maka routing tidak dapat dilakukan secara efektif. Beberapa penelitian juga melakukan evaluasi dari MANET. Dihasilkan bahwa jika MANET

mengirim paket data melalui rute yang telah rusak, maka sistem tersebut tidak dapat memberikan rute cadangan langsung. Maka dari itu, perlu sekali dilakukan optimasi performa dari MANET.

Beberapa penelitian juga sudah memodifikasi AODV menjadi lebih efektif dengan melakukan menemukan rute-rute yang ada terlebih dahulu, baru kemudian dikirimkan paket data oleh sumber. Teknologi tersebut dinamakan AODV-ABR atau AODV-Advanced Backup Routing.

Berdasarkan permasalahan yang telah disebutkan, pada tugas akhir ini penulis akan melakukan modifikasi terhadap MANET dengan melakukan implementasi K-Means Clustering pada AODV Routing Protocol untuk meningkatkan *Packet Delivery Ratio*, dan menurunkan *routing overhead*, *average hop count* dan *end-to-end delay*.

1.2 Rumusan Masalah

Dari permasalahan di atas, maka dapat dirumuskan beberapa masalah:

1. Bagaimana melakukan implementasi K-Means Clustering dan Backup Routing pada AODV Routing Protocol pada MANET?
2. Bagaimana peranan K-Means Clustering dalam AODV Based Backup Routing dalam mengurangi Packet Loss Ratio pada MANET?
3. Bagaimana peranan algoritma K-means dalam AODV Based Backup Routing memengaruhi performa MANET secara keseluruhan diukur dari Packet Delivery Ratio, End-to-end Delay, dan Routing Overhead

1.3 Batasan Permasalahan

Permasalahan yang ada pada Tugas Akhir ini memiliki pembatasan sebagai berikut:

1. Jaringan yang digunakan adalah jaringan Mobile Adhoc Network (MANET).

2. Routing protocol yang diujicoba pada MANET adalah AODV.
3. Simulasi pengujian jaringan menggunakan Network Simulator 2 (NS-2).
4. Pembuatan skenario uji coba menggunakan Simulation of Urban Mobility (SUMO) dengan melihat indikator Packet Delivery Ratio, Packet loss ratio, End-to-end Delay, dan Routing Overhead sebagai penilaian.

1.4 Tujuan

Tujuan pembuatan Tugas Akhir ini adalah sebagai berikut:

1. Melakukan implementasi K-means dalam AODV routing protocol pada MANET.
2. Menganalisis peranan Algoritma K-means dalam AODV Based Backup Routing dalam mengurangi Packet Loss Ratio pada MANET.
3. Menganalisis performa AODV Based Backup Routing yang telah ditambah algoritma K-means dengan mengukur matriks Packet Delivery Ratio (PDR), End-to-end Delay, dan Routing Overhead.

1.5 Manfaat

Tugas Akhir ini dapat dijadikan referensi ilmiah berupa hasil analisis performa AODV Routing Protocol pada MANET yang sudah dimodifikasi dengan metode AODV Based Backup Routing.

1.6 Metodologi

Pembuatan Tugas Akhir ini dilakukan menggunakan beberapa Metode, yaitu:

1.6.1 Penyusunan Proposal Tugas Akhir

Tahap awal untuk memulai pengerjaan tugas akhir adalah penyusunan proposal tugas akhir. Proposal ini akan menyajikan ringkasan dari metode dan teori dasar yang digunakan pada paper ini.

1.6.2 Studi Literatur

Tahap Studi Literatur dimulai setelah proposal diterima oleh Dosen Pembimbing, pada tahap ini penulis akan mempelajari seluruh teori-teori yang akan dijadikan landasan pada pengerjaan Tugas Akhir dari buku dan jurnal-jurnal yang diterbitkan oleh badan jurnal internasional yang berhubungan dengan AODV, K-Means Clustering, dan Backup Routing.

1.6.3 Analisis dan Desain Sistem

Pada tahap ini dilakukan perancangan alur kerja dari modifikasi AODV berupa flowchart, dan setelah dilakukan perancangan akan dilakukan implementasi dari AODV Based Backup Routing pada AODV Routing Protocol dalam MANET menggunakan Network Simulator 2.

1.6.4 Implementasi Sistem

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

1.6.5 Pengujian dan Evaluasi

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

1.6.6 Penyusunan Buku

Pada tahap ini dilakukan penyusunan laporan yang menjelaskan dasar teori dan metode yang digunakan dalam tugas akhir ini serta hasil dari implementasi aplikasi perangkat lunak yang telah dibuat.

1.7 Sistematika Penulisan Laporan

Sistematika penulisan laporan Tugas Akhir adalah sebagai berikut:

1. Bab I. Pendahuluan
Pada bagian ini berisi penjelasan mengenai latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, metodologi, dan sistematika penulisan dari pembuatan Tugas Akhir.
2. Bab II. Tinjauan Pustaka
Pada bagian ini berisi kajian teori atau penjelasan dari metode, algoritma, library, dan tools yang digunakan dalam penyusunan Tugas Akhir ini.
3. Bab III. Perancangan
Pada bagian ini berisi pembahasan mengenai perancangan beberapa skenario antara lain modifikasi, pengujian, dan perhitungan metrik analisis.
4. Bab IV. Implementasi
Pada bagian ini menjelaskan implementasi yang berbentuk kode program dari proses modifikasi, proses pengujian. Dan perhitungan metrik analisis.
5. Bab V. Uji Coba dan Evaluasi
Pada bagian ini berisi hasil uji coba dan evaluasi dari implementasi yang telah dilakukan untuk menyelesaikan masalah yang dibahas pada Tugas Akhir.
6. Bab VI. Kesimpulan dan Saran
Pada bagian ini berisi kesimpulan dari hasil uji coba yang dilakukan, masalah yang dialami pada proses pengerjaan Tugas Akhir, dan saran untuk pengembangan dari solusi yang ada.
7. Daftar Pustaka

Pada bagian ini berisi daftar pustaka yang dijadikan literatur dalam pengerjaan Tugas Akhir.

8. Lampiran

Pada bagian ini terdapat tabel-tabel yang berisi data hasil uji coba dan beberapa kode program.

(Halaman ini sengaja dikosongkan)

BAB II TINJAUAN PUSTAKA

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

2.1 *Mobile Ad-hoc Networks (MANETs)*

Mobile Ad hoc Network(MANET) merupakan sebuah jaringan yang terbentuk dari beberapa *node* yang bergerak bebas dan dinamis. MANET memungkinkan terjadinya komunikasi jaringan tanpa bergantung pada ketersediaan infrastruktur jaringan yang tetap [1]. Dalam MANET, setiap *node* yang ada dapat bertindak sebagai host maupun router. Setiap *node* juga dapat saling berkomunikasi antara satu sama lain walaupun tidak ada *access point*. Perangkat pada MANET harus dapat mendeteksi lokasi atau keberadaan dari setiap perangkat yang ada. MANET dapat dimodifikasi sesuai dengan kebutuhan di lapangan, karena hampir tidak membutuhkan support dari infrastruktur yang telah ada. Antar node akan terhubung sebagai jaringan ad hoc sebagai autonomous system of mobile hosts (MH) yang juga bertindak sebagai router dan terhubung menggunakan jaringan wireless. Hal ini kontras dengan jaringan seluler single hop yang membutuhkan BTS (Base Tranceiver Station) sebagai akses poin. Dalam jaringan seluler, komunikasi antar mobile node tergantung terhadap wired backbone dan dukungan sinyal dari BTS. Di dalam MANET tidak dibutuhkan infrastruktur dan topologi jaringan dapat berubah secara dinamis mengikuti perubahan node karena node dapat bergerak secara bebas. MANET mempunyai beberapa karakteristik, yaitu:

1. Topologi yang Dinamis:

Node pada MANET dapat bergerak secara bebas dan mampu bergerak ke segala arah layaknya sebuah perangkat

mobile. Hal ini menyebabkan topologi berubah secara acak dan tidak mempunyai pola.

2. Keterbatasan Energi:

Karena bersifat *mobile*, maka setiap perangkat yang ada tidak tercolok langsung ke sambungan listrik. Melainkan menggunakan baterai yang tentu mempunyai batas energi dan umur baterai.

3. Keterbatasan *bandwith*:

Link pada *wireless* jelas memiliki kapasitas yang lebih sedikit daripada perangkat berkabel. Selain itu seringkali terjadi kemacetan ketika menggunakan perangkat *wireless*.

4. Keterbatasan Keamanan:

Jaringan nirkabel seringkali menjadi sasaran empuk bagi penyerang keamanan. Contoh seperti *spoofing*, *denial of services*, *eavesdropping*, harus menjadi pertimbangan utama saat membuat suatu jaringan.



Gambar 2.1 Ilustrasi Jaringan MANETs

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

AODV merupakan perpaduan antara DSR dan DSDV. AODV mengambil karakteristik DSR yaitu melakukan route discovery bila dibutuhkan. Perbedaan AODV dan DSR adalah AODV menggunakan routing table tradisional yaitu satu entri per tujuan [2]. Seperti DSDV, AODV menjamin tidak akan terjadi loops. AODV menggunakan komunikasi broadcast untuk route discovery dan komunikasi unicast untuk route reply. Sedangkan pada DSR digunakan routing yang telah ditentukan oleh node asal, bukan rute yang ditentukan oleh node perantara yang ada di antara node asal dan node tujuan. Node perantara memiliki route cache yang akan menyimpan informasi untuk keperluan routing. AODV sendiri dapat dikatakan sebuah *routing protocol* yang sangat direkomendasikan untuk sebuah MANET. Karena AODV dapat mengatasi keterbatasan *bandwidth* yang dimiliki oleh MANET dengan sangat baik. Salah satu keunggulan dari AODV adalah *routing protocol* tersebut tidak menyimpan informasi tentang *route* yang sudah tidak aktif dan penggunaan *sequence number* yang disisipkan pada setiap paket oleh setiap *node* yang mengirimkan paket tersebut yang bertujuan untuk menggantikan cached *routes* sekaligus memastikan bahwa *routes* yang ada pada *routing table* adalah *routes* yang terbaik berdasarkan *sequence number* dan *timestamps*.

AODV sendiri mempunyai 2 fase utama, yaitu *Route Discovery* dan *Route Maintenance*. Pada setiap fase, proses ini melibatkan beberapa jenis paket yang disebut *control messages*.

2.2.1 *Control Messages*

Control Messages adalah paket untuk mengatur proses *route discovery* dan *route maintenance*. *Control Messages* sendiri mempunyai 4 jenis, yaitu *Route Request*, *Route Reply*, *Route Error*, *Hello Messages*.

2.2.1.1 Route Request (*RREQ*)

RREQ adalah jenis paket ketika *source node* dihubungkan ke *destination node*. *Source node* sendiri akan menyebarkan *broadcast* untuk RREQ terhadap *node* di sekitarnya. RREQ sendiri memiliki beberapa *fields* seperti yang tertera pada Tabel 2.1.

<i>Source Address</i>
<i>Request ID</i>
<i>Source Sequences Number</i>
<i>Destination Address</i>
<i>Destination Sequences Number</i>
<i>Hop Count</i>

Tabel 2.1 Format dari RREQ

Setiap kali *source node* mengirimkan RREQ baru, maka *request id* pasti akan bertambah, sehingga dapat dikatakan bahwa RREQ yang dikirimkan adalah unik sesuai dengan *request id* dan *source address*. *Node* yang menerima RREQ dengan identitas parameter yang sama akan membuang RREQ tersebut, sedangkan yang diterima, *hop count* dan *sequence number* RREQ tersebut akan dibandingkan dengan informasi yang ada di dalam *node* tersebut.

2.2.1.2 Route Reply (*RREP*)

RREP adalah jenis paket yang digunakan ketika *node* tersebut adalah *node* tujuan atau menyimpan sebuah rute menuju *node* tujuan. Setelah itu *node* tersebut akan melakukan *unicast* RREP ke pada *node* sumber atau *source node*. *Fields* pada RREP ini tercantum pada Tabel 2.2.

<i>Source Address</i>
<i>Destination Address</i>
<i>Destination Sequence No</i>
<i>Hop Count</i>

Life Time

Tabel 2.2 Format dari RREP

Unicast dapat dilakukan karena setiap node pada suatu rute memiliki data *route* untuk kembali ke *source node*. Proses ini dinamakan *Reverse Path Setup* (RPS). Sedangkan dalam pengiriman RREP secara *unicast* setiap *node* yang berada pada *route* tersebut menyimpan informasi darimana *node* tersebut mendapatkan paket RREP atau yang disebut juga dengan proses *Forward Path Setup*(FPS).

2.2.1.3 Route Error (*RERR*)

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

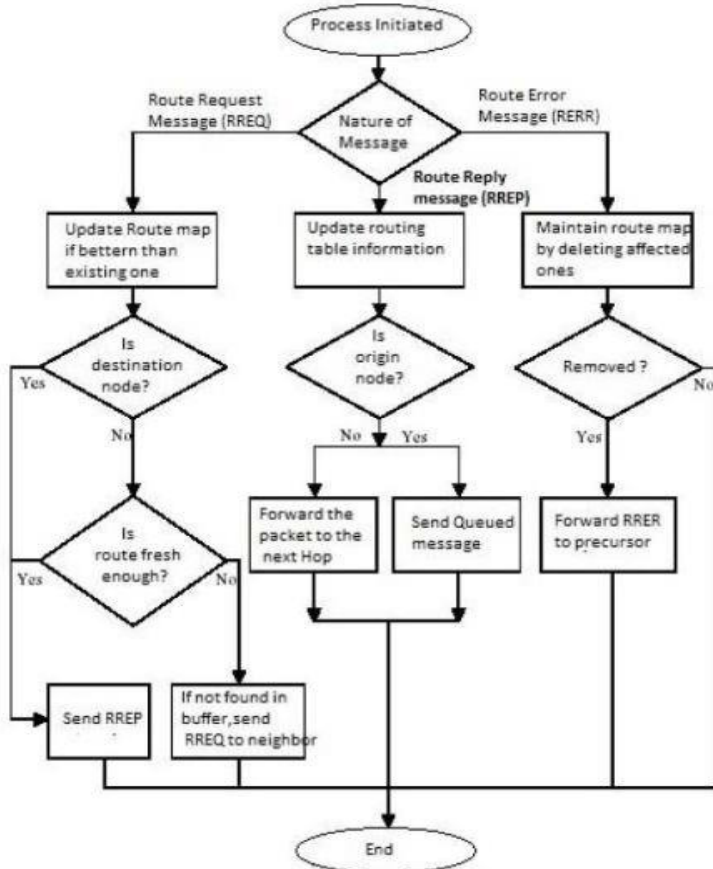
2.2.1.4 Hello Messages

Setiap *node* dapat mengetahui tetangganya dengan mengirimkan local broadcast atau juga yang disebut dengan Hello Messages. Node tetangga adalah *node* yang diasumsikan dapat berkomunikasi secara langsung dengan suatu *node* tanpa melalui suatu intermediate *node*.

2.2.2 Route Discovery

Route discovery adalah proses pencarian suatu *route* menuju *destination* oleh *source node*. Dalam prosesnya *node* akan menggunakan RREQ *message* dan RREP *message* untuk mencari *route* yang diinginkan oleh *source node*. Ketika suatu *node* mengirimkan RREQ, *node* penerima mungkin saja mendapatkan RREQ yang sama kembali dari *node* tetangganya, sehingga untuk

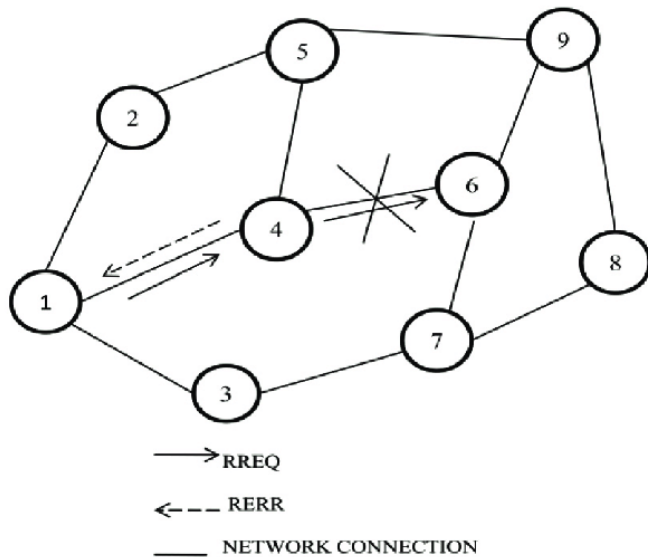
mencegah *infinite cycles* setiap *node* memiliki *buffer*, yang berisi daftar RREQ yang telah dilakukan *broadcast*. Sebelum RREQ dikirim, *buffer* akan selalu diperiksa untuk memastikan RREQ yang sama tidak akan dikirimkan lagi. Setiap intermediate *node* akan melakukan penambahan *sequence number* untuk memastikan bahwa *route* akan selalu diperbaharui dengan informasi terbaru. Pada setiap RREQ *message* akan dimasukkan *source node sequence number* dan *destination node sequence number*. Gambar 2.1 akan menjelaskan bagaimana suatu *node* melakukan proses terhadap suatu paket *message* yang diterimanya.



Gambar 2.2 Flowchart Proses Control Message

2.2.3 Route Maintenance

Route Maintenance adalah sebuah proses untuk mengecek segala rute yang ada. Jika ada koneksi yang putus di antara dua node, maka RERR dikirimkan ke beberapa tujuan.



Gambar 2.3 Ilustrasi terjadinya RERR

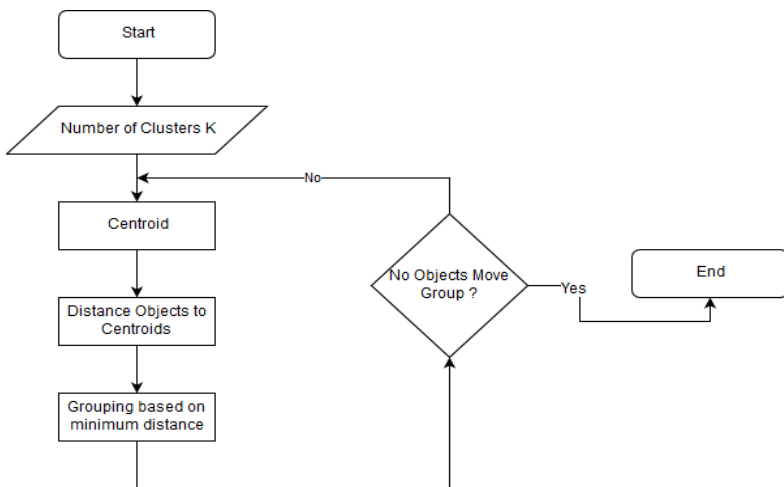
Pada Gambar 2.3, terlihat ada 9 node yang beroperasi dengan rutennya masing-masing, lalu ketika node 1 mengirimkan kepada node 4 pengiriman berhasil, namun ketika node 4 mengirimkan ke pada node 6, pengiriman gagal, karena pengiriman gagal akhirnya node 4 mengirimkan RERR kembali ke pada node 1. Lalu ketika *source node* mendapatkan RERR, *source node* akan memulai untuk melakukan proses *route discovery* kembali.

2.3 K-Means Clustering

K-Means Clustering merupakan teknik pengelompokkan terhadap objek n kedalam kluster k berdasarkan jarak terdekat ke centroid cluster atau pusat dari suatu kluster. Langkah-langkah untuk melakukan K-Means Clustering adalah sebagai berikut

1. Tentukan jumlah cluster
2. Alokasikan data ke dalam cluster secara random

3. Hitung centroid/rata-rata dari data yang ada di masing-masing cluster
4. Alokasikan masing-masing data ke centroid/rata-rata terdekat
5. Kembali ke Step 3, apabila masih ada data yang berpindah cluster atau apabila perubahan nilai centroid, ada yang di atas nilai threshold yang ditentukan atau apabila perubahan nilai pada objective function yang digunakan di atas nilai threshold yang ditentukan.



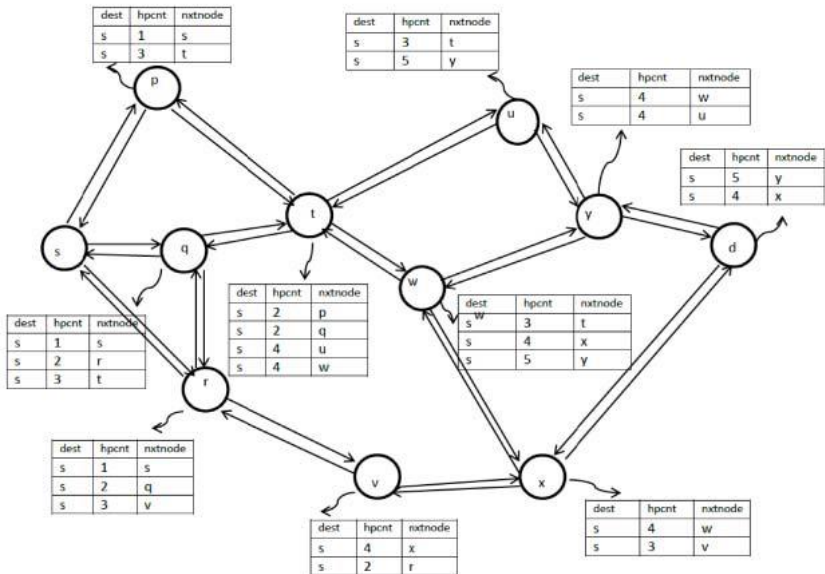
Gambar 2.4 Flowchart dari *K-Means Clustering*

2.4 Backup Routing

Backup Routing merupakan sebuah algoritma yang dikemukakan oleh Struthy dkk. [3] Algoritma tersebut bertujuan untuk memperbaiki atau melakukan *Refinement* terhadap beberapa *Control Message* yang ada pada AODV, seperti *Refined Route Request* (RRREQ), *Refined Route Reply* (RRREP), *Refined Route Error* (RRERR).

2.4.1 Refined Route Request (RRREQ)

RRREQ merupakan sebuah modifikasi dan inovasi dari RREQ standar yang ada pada AODV [3]. Ilustrasi dari perubahan RREQ ada pada Gambar 2.5 di bawah ini.



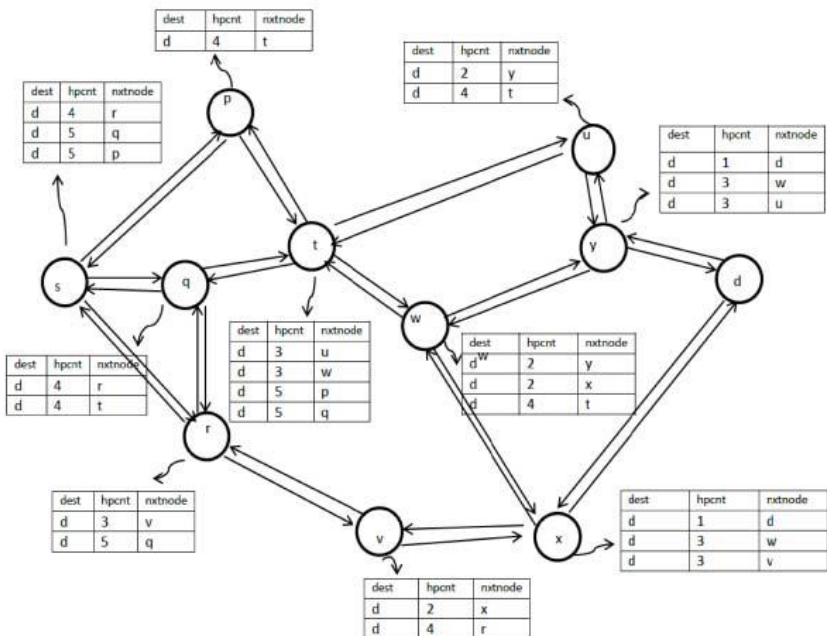
Gambar 2.5 Ilustrasi proses berjalannya RRREQ

Di RREQ, setiap node memiliki rute masing-masing, terutama memiliki rute arah balik ke pada *Source Node*. Akan tetapi, rute arah balik yang disimpan oleh node pada RREQ biasa hanya ada satu rute. Namun di RRREQ, seperti pada gambar, semua node tidak hanya menyimpan satu rute arah balik, melainkan semua rute arah balik yang mungkin dilakukan akan disimpan oleh node. Misalnya, ketika node p mengirim ke node r, node tersebut akan menghitung seluruh rute terlebih dahulu, lalu ketika saat mengirim ternyata node

tujuan berbeda dengan yang seharusnya, maka paket tersebut akan dikirimkan melalui node lain sesuai rute yang ada. Hal ini diklaim lebih efektif oleh penulis paper [3].

2.4.2 Refined Route Reply (RRREP)

Sama halnya dengan RRREQ, RRREP menyimpan seluruh rute yang tersedia dari *Destination Node* menuju *source node*. Berbeda dengan RREP biasa yang hanya menyimpan satu rute ketika *destination node* akan melakukan reply kepada *source node*. Pada Gambar 2.6 akan menggambarkan ilustrasi bagaimana proses RRREP berjalan.

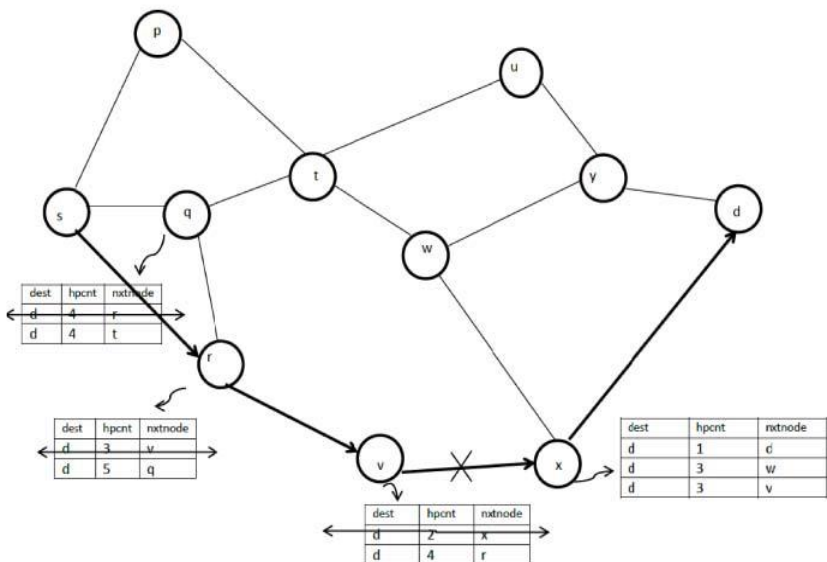


Gambar 2.6 Ilustrasi proses berjalannya RRREP

Gambar tersebut menjelaskan hamper sama seperti RREP, namun dapat dilihat bahwa pada RRREP setiap node menyimpan seluruh rute yang tersedia sebelum ia mengirimkan kembali paket data. Sebagai contoh, misal node p ingin melakukan *reply* terhadap node d, makai a bisa melewati node s maupun node r karena kedua rute tersebut sudah disimpan dan dapat dilewati ketika dibutuhkan.

2.4.3 Refined Route Error (RRERR)

Dalam algoritma ini, seluruh rute pengiriman *forwarding* dan *reversing* disimpan sebelum pengiriman dijalankan. Namun, jika pengiriman gagal, sama halnya pada AODV biasa node akan mencari RERR atau *Route Error*. Namun pada metode *Backup Routing*, RERR dimodifikasi lebih lanjut dan dinamakan RRERR. Ilustrasi proses terjadinya RRERR terdapat pada Gambar 2.7.



Gambar 2.7 Ilustrasi proses berjalannya RRERR

Dapat dilihat pada gambar, ketika node v mengirimkan kepada node x, terjadi error. Karena terjadi *error*, maka *source node* sebelum mengirimkan paket ke pada node d akan menghitung seluruh rute yang ada dengan asumsi node tujuan adalah d, maka dari itu *source node* akan menggunakan rute selain rute yang sama karena rute yang sama telah terjadi *error*. Namun, ketika rute yang digunakan tidak terjadi error maka rute yang sama tetap dilewati sesuai dengan nilai a. Berdasarkan gambar, rute tersebut bisa menjadi bermacam-macam, namun dilihat dari tabel pada gambar, ada beberapa factor yang diperhitungkan, yaitu sebuah *value* bernilai a. Nilai a merupakan hasil perhitungan dari suatu rute, dengan menghitung jumlah energi dibagi dengan *hop count*, setelah itu nilai a terbesar dari setiap rute akan dipilih menjadi rute yang dipakai, dengan catatan selain rute yang terjadi error.

2.5 Network Simulator - 2

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

2.6 OpenStreetMap (OSM)

OSM merupakan sebuah web untuk membuat sebuah peta dan dapat diakses secara gratis. OSM dibangun oleh sukarelawan dengan melakukan survei menggunakan GPS dan mendigitalisasi citra satelit

serta juga mengumpulkan data geografis yang tersedia di publik. Open Data Commons Commons Open Database License 1.0 adalah sarana dimana kontributor OSM dapat memiliki, memodifikasi, dan membagikan data geografis secara luas [4].

Di dalam OSM sendiri ada beberapa atribut yang digunakan dalam membuat peta, seperti jalan raya, stasiun, bandara, kantor, rumah sakit, dan jenis lain yang tersebar di seluruh dunia. Data yang tersedia pada OSM pun dapat digunakan secara bebas dan dapat digunakan untuk kepentingan berbeda.

2.7 Java OpenStreetMap (JOSM)

Java OpenStreetMap(JOSM) adalah aplikasi yang dapat digunakan untuk melakukan penyuntingan terhadap data yang didapatkan dari OpenStreetMap [6]. Penyuntingan dapat berupa merapikan data yang ada pada OSM, *filtering*, dan beberapa filter lainnya.

2.8 Simulation of Urban Mobility (SUMO)

SUMO merupakan aplikasi atau program *open source* yang digunakan untuk melakukan simulasi lalu lintas. SUMO sendiri saat ini sudah semakin berkembang dan semakin banyak fitur yang dapat digunakan dalam pemodelan dan membaca format-format yang berbeda. SUMO sendiri dapat mendefinisikan beberapa atribut tertentu, seperti panjang kendaraan, kecepatan maksimum, percepatan maupun perlambatannya, dan fitur lainnya.

SUMO memberikan beberapa *tools* untuk membuat simulasi lalu-lintas pada tahap-tahap yang berbeda. Seluruh penjelasan mengenai tools tersebut terdapat pada Tabel 2.3 dibawah.

Tool	Keterangan
netgenerate	Membuat peta seperti <i>grid</i> , spider, dan bahkan random network. Sebelum melakukan proses ini, pengguna dapat

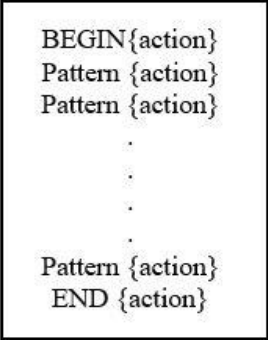
	menentukan kecepatan maksimum dan membuat traffic light pada peta. Hasil dari proses ini akan menghasilkan <i>file</i> dengan ekstension .net.xml.
netconvert	Program CLI yang berfungsi untuk melakukan konversi dari peta seperti OpenStreetMap menjadi format native SUMO.
randomTrips.py	Membuat rute acak yang akan dilalui oleh kendaraan dalam simulasi
duarouter	Melakukan perhitungan rute berdasarkan definisi yang diberikan dan memperbaiki kerusakan rute.
sumo	Program yang melakukan simulasi lalu lintas berdasarkan data yang didapatkan dari netgenereate atau netconvert dari randomTrips.py. Hasil simulasi merupakan <i>file</i> hasil <i>export</i> untuk dikonversi menjadi format lain.
sumo-gui	GUI untuk melihat simulasi yang dilakukan oleh SUMO dengan grafis
traceExporter.py	Melakukan konversi output dari sumo menjadi format yang dapat digunakan pada simulator lain.

Tabel 2.3 *List Tools* yang tersedia pada SUMO

2.9 AWK

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

Dalam penggunaannya AWK pattern dapat diketikkan dalam terminal bersamaan dengan text sebagai paramater, selain itu AWK *script* dapat dibuatkan kedalam suatu *file* dengan beberapa pattern tertentu. Berikut ini adalah struktur dari *file* AWK ditunjukkan pada Gambar 2.8



```
BEGIN {action}
Pattern {action}
Pattern {action}
.
.
.
Pattern {action}
END {action}
```

Gambar 2.8 Struktur kodingan pada file AWK

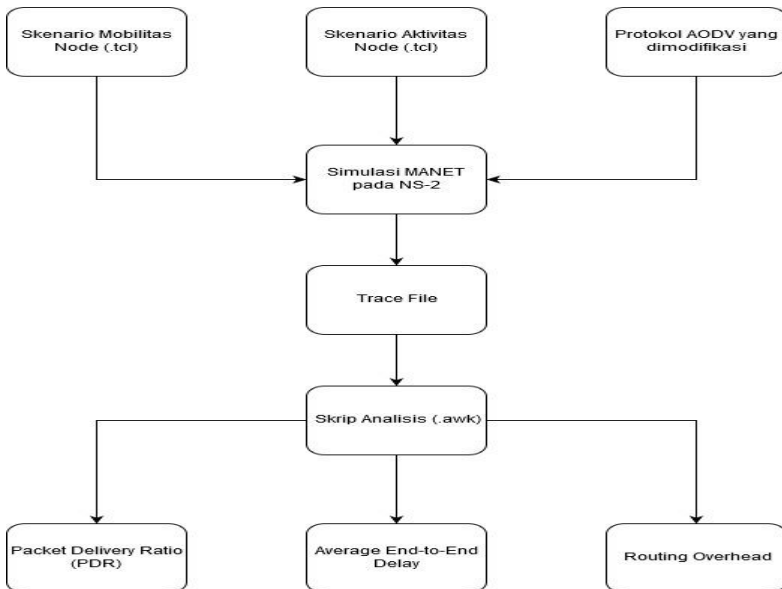
(Halaman ini sengaja dikosongkan)

BAB III PERANCANGAN

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

3.1 Deskripsi Umum

Pada Tugas Akhir ini akan diimplementasikan sebuah *routing protocol* yaitu AODV, dengan melakukan beberapa modifikasi pada *route discovery* menggunakan metode *K-means Clustering* dan *Backup Routing* lalu akan disimulasikan menggunakan NS-2. Ilustrasi proses simulasi AODV modifikasi ini dapat dilihat pada Gambar 3.1.



Gambar 3.1 Diagram Simulasi AODV

Secara umum alur simulasi antara AODV asli dan AODV modifikasi tidak ada perbedaan. Dalam Tugas Akhir ini akan ada beberapa istilah yang akan sering digunakan, 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	<i>Routing Overhead</i> . Jumlah <i>control message</i> yang terkirim.
5	RREQ	<i>Route Request</i> . Paket <i>request</i> pada AODV yang dikirim untuk mencari <i>route</i>
6	RREP	<i>Route Reply</i> . Paket <i>reply</i> pada AODV yang dikirim ke <i>node</i> sumber melalui <i>route</i> yang sudah dibuat.
7	RERR	<i>Route Error</i> . Paket <i>error</i> pada AODV yang dikirim untuk memberitahukan <i>node</i> pada suatu <i>route</i> bahwa <i>route</i> tersebut tidak bisa digunakan lagi.
8	BR	<i>Backup Routing</i> , merupakan sebuah algoritma untuk aodv, yang akan menghitung seluruh jalur yang tersedia terlebih dahulu sebelum paket dikirimkan.

Tabel 3.1 Daftar Istilah

Modifikasi yang dilakukan terhadap *routing protocol* AODV sendiri akan dibagi menjadi 2 bagian, yang pertama melakukan *clustering* pada seluruh node menggunakan metode *K-Means Clustering*, proses *clustering* ini akan dilakukan di awal detik t , sehingga tentu data yang diperlukan untuk menggunakan *clustering* harus diperoleh sebelum detik t . Hasil dari pengelompokkan ini akan menghasilkan n cluster dan ada *cluster head* dan *cluster gateway* pada setiap clusternya. Lalu modifikasi yang kedua akan menerapkan algoritma *Backup Routing*, dalam algoritma ini maka total energi dari setiap rute yang mungkin pada tiap node akan dihitung lalu akan dibagi jumlah *hop count* pada tiap rute, setelah itu perhitungan tersebut akan menghasilkan nilai a . Nilai a yang mempunyai nilai paling besar akan dipilih sebagai rute untuk mengirimkan paket dari *source node* ke pada *destination node*.

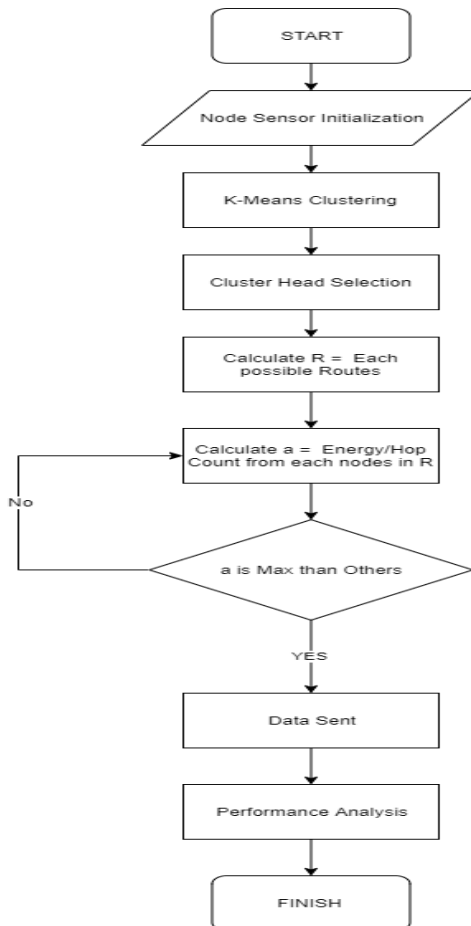
3.2 Perancangan Skenario Mobilitas

Tugas Akhir ini menggunakan 2 jenis skenario, yaitu skenario *grid* dan skenario *real*. Skenario *grid* akan netconvert terlebih dahulu. Tahap berikutnya memiliki tahapan yang sama seperti merancang skenario *grid*, yaitu membuat pergerakan *node* menggunakan randomTrips dan duarouter. Kemudian dilakukan penggabungan *file* peta *real* yang sudah dikonversi ke dalam *file* dengan ekstensi .net.xml dan *file* pergerakan *node* yang sudah dibuat sebelumnya. Hasil dari penggabungan tersebut merupakan *file* skenario berekstensi .xml. File yang dihasilkan tersebut dikonversi ke dalam bentuk *file* dengan ekstensi .tcl agar dapat diterapkan pada NS-2.

3.3 Perancangan Modifikasi Protokol AODV

AODV sebagai *routing protocol* yang digunakan untuk MANET dalam Tugas Akhir ini nantinya akan dimodifikasi, menggunakan metode *K-Means Clustering* terlebih dahulu untuk melakukan pengelompokkan dari beberapa node berdasarkan jarak antar node

terdekat. Lalu akan ditambahkan metode *Backup Routing* untuk menghitung seluruh rute yang tersedia untuk mencapai node tujuan, lalu dihitung energinya dan akan dibagi dengan hop count lalu menghasilkan nilai a . Nilai a terbesar inilah yang akan dijadikan rute untuk mengirimkan paket dari *source node*. Alur protokol AODV yang telah dimodifikasi terdapat pada Gambar 3.2.



Gambar 3.2 Alur Protokol AODV Modifikasi

3.3.1 Perancangan K-Means Clustering

Proses penerapan *K-Means Clustering* dimulai dari mengumpulkan beberapa data yang diperlukan terlebih dahulu. Data yang diperlukan untuk melakukan *clustering* adalah posisi dua dimensi dari seluruh *node* yang berada di daerah simulasi. Data tersebut akan ditampung ke dalam *nodes positions table* yang ada pada *node*. Variabel yang diperlukan untuk *node position table* terdapat pada tabel 3.2.

No	Variabel	Tipe	Keterangan
1	<i>nodes_position_x</i>	double, array	posisi absis <i>node</i>
2	<i>nodes_position_y</i>	double, array	posisi ordinat <i>node</i>
3	<i>nodes_timestamp</i>	double, array	waktu data diperoleh

Tabel 3.2 *Nodes Position Table*

Nodes Position Table sendiri merupakan tabel yang berisi data 2 dimensi posisi dari setiap *node* yang ada, dan berisi *timestamp* untuk memastikan bahwa data yang akan diambil adalah data yang paling baru sesuai *timestamp*. Oleh karena itu, paket tersebut harus memindahkan data dari *node* kepada paket tersebut dengan memperhatikan *timestamp*. Proses pemindahan data tersebut ada pada *pseudocode* 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
    endif
  endfor

```

Gambar 3.3 *Pseudocode* Proses Pemindahan Data Posisi Node

Proses pemindahan data ini akan dilakukan sebelum detik t , yang nantinya akan diteruskan ke proses *clustering*. Proses *Clustering* ini akan dilakukan oleh node yang berperan sebagai *Server Node*. Node tersebut nantinya akan melakukan *clustering* sesuai data dari *Nodes Position Table*. Setelah itu proses ini akan memilih *centroid cluster* awal secara acak sebanyak k *cluster*, lalu setelah itu dapat dilakukan proses *K-Means Clustering*. *Pseudocode* untuk melakukan *K-Means Clustering* terdapat pada Gambar 3.4 di bawah ini.

```

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
        endif
      endfor
    endfor
  endfor
  update_centroid_all_clusters()
endwhile

```

Gambar 3.4 Pseudocode K-Means Clustering

Setelah melakukan *K-Means Clustering* maka akan ada beberapa informasi penting, seperti *cluster head*, *cluster group*, *cluster gateway*. Kemudian, *cluster head* dilakukan secara internal dengan pemilihan *node* yang terdekat dari *centroid cluster* tersebut sehingga setiap *node* akan dicari jaraknya menuju *centroid cluster* tersebut dengan metode euclidean distance. *Pseudocode* untuk pemilihan *cluster head* ini terdapat pada Gambar 3.5. Setelah memilih *cluster head* maka selanjutnya akan dipilih *cluster gateway* dengan mencari titik tengah terlebih dahulu yang ada diantara seluruh *node* yang ada pada peta simulasi, lalu akan dicari setiap *node* pada tiap cluster yang paling dekat jaraknya dengan *cluster gateway*. *Pseudocode* pemilihan *Cluster Gateway* ini terdapat 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 Pemilihan Cluster Head

```

centroid = getCentroid_AllNodes()
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 Pemilihan Cluster Gateway

Setelah seluruh proses *Clustering* berjalan, maka akan ada beberapa informasi yang dimasukkan kedalam *nodes cluster tables*. Setelah itu, *server node* akan menyampaikan seluruh informasi hasil dari *clustering* sebelumnya ke pada seluruh node yang ada pada peta simulasi. Penerimaan setiap paket RRREP dan *Hello Messages* diikuti dengan melakukan proses pemindahan informasi hasil *clustering* melalui *nodes clustering table* yang telah disimpan pada paket tersebut. Penjelasan tentang *nodes cluster table* ada pada Tabel 3.3.

No	Variabel	Tipe	Keterangan
1	<i>nodes_cluster_group</i>	int, array	<i>cluster node</i>
2	<i>nodes_cluster_head</i>	int, array	<i>cluster head</i>
3	<i>nodes_cluster_gateway</i>	int, array	<i>cluster gateway</i>
4	<i>nodes_cluster_timestamp</i>	double	waktu proses <i>clustering</i>

Tabel 3.3 *Nodes Cluster Table*

Dapat dilihat pada tabel, bahwa ada beberapa informasi penting beserta tipenya. Seperti *Cluster Node*, *Cluster Head*, *Cluster Gateway*, dan waktu dari proses *clustering*. Proses ini seperti sebelumnya juga memperhatikan *timestamp*, karena hal tersebut akan sangat memengaruhi jika *clustering* dilakukan lebih dari 1 kali, yang nantinya tentu data pada tabel tersebut diganti oleh informasi yang terbaru. *Pseudocode* untuk proses pergantian informasi ini dapat dilihat di 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
endif

```

Gambar 3.7 Pseudocode Pergantian Informasi

3.3.2 Perancangan Backup Routing

Backup Routing akan dijalankan setelah melakukan *clustering*, yakni melakukan pemilihan forwarding node dengan yaitu node yang berhak untuk melakukan rebroadcast paket RREQ. Penentuan forwarding node dilakukan dengan cara membandingkan antara minimum energi dan rata-rata energi yang dimiliki oleh rute RREQ. Apabila nilai minimum energi pada rute lebih kecil nilai minimum energi pada node yang dilalui oleh RREQ tersebut, bandingkan rata-rata energi rute tersebut dengan rata-rata energi pada node yang dilalui oleh rute tersebut. Apabila rata-rata energi pada rute lebih kecil rata-rata energi pada node yang dilalui, maka paket RREQ akan diteruskan. Jika rute tersebut tidak memenuhi kriteria tersebut, maka paket RREQ akan di-drop. *Pseudocode* untuk pemilihan forwarding node ada pada Gambar 3.8.

```

rq->rq_min_energy = energy
rq->rq_energy = rq->rq_energy + energy
avg_energy = rq->rq_energy / (hop count + 1)
if (rq->rq_min_energy < node's minimum energy) then
  update node's minimum energy
  if (avg_energy < node avg energy) then
    drop RREQ
  else
    forward RREQ
    update node avg energy
  end if
else
  drop RREQ
end if

```

Gambar 3.8 Pseudocode pemilihan forwarding node

Dapat dilihat pada *Pseudocode* tersebut bahwa energi setiap rute akan dihitung terlebih dahulu, lalu dari setiap node pada rute akan dipilih energi yang terkecil, setelah itu jumlah energi pada suatu rute akan dibagi dengan *hop count* untuk mendapatkan nilai *a* sesuai pada rumus berikut:

$$a = \frac{\text{total route energy}}{\text{hop count}}$$

Setelah proses berikut, maka nilai *a* yang paling besar dibandingkan rute lain akan menjadi rute yang digunakan. Algoritma ini akan membantu jika terjadi *route error* di suatu rute. Jika *error* ditemukan maka node mempunyai rute cadangan lain untuk dilalui.

3.4 Perancangan Simulasi pada NS-2

Simulasi dari program yang telah dibuat akan menggunakan NS-2, dengan menggabungkan skrip *tcl* dan skenario yang telah dibuat sebelumnya. *File* gabungan tersebut memiliki konfigurasi lingkungan simulasi yang sesuai pada Tabel 3.4.

No	Parameter	Spesifikasi
1	Network Simulator Tool	NS2
2	<i>Routing Protocol</i>	AODV
3	Simulation Time	200 s
4	<i>Number of Nodes</i>	50, 100, 150, 200
5	Simulation Area	700 m x 700 m
6	<i>Number of Cluster</i>	5,10, 15, 20
7	<i>Number of Clustering</i>	1
8	Antenna Model	Omni Antenna
9	MAC Type	MAC/802_11
10	Network Interfaces Types	Wireless
11	Transmission Range	400 m
11	<i>Source/Destination Node</i>	Static
12	Initial <i>Node Energy</i>	100 Joule

Tabel 3.4 Spesifikasi Skenario NS2

3.5 Perancangan Metrik Analisis

Dalam Tugas Akhir ini akan ada beberapa Metrik Analisis yang digunakan sebagai data utama untuk membandingkan hasil AODV modifikasi dan hasil AODV asli.

3.5.1 Packet Delivery Ratio (PDR)

PDR merupakan rasio dari jumlah terkirimnya paket dibandingkan dengan jumlah paket yang dikirim. Jika hasil PDR tinggi, maka dapat dikatakan bahwa AODV tersebut mempunyai performa yang baik. Perhitungan PDR ada pada rumus sebagai berikut:

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

Keterangan:

PDR: Packet Delivery Ratio

Packet Received: Jumlah paket yang terkirim atau diterima oleh node tujuan

Packet Sent: Jumlah paket yang dikirim oleh *Source Node*

3.5.2 Average End-to-End Delay (E2E)

E2E merupakan rata-rata delay dari waktu pengiriman paket dari *source node* hingga paket tersebut mencapai *destination node*. Jika E2E memiliki waktu yang sedikit, maka dapat dikatakan AODV tersebut memiliki performa yang baik. Rumus dari E2E adalah sebagai berikut:

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

Keterangan :

CBR(RecvTime) = waktu *destination node* menerima paket

CBR(SentTime) = waktu *source node* mengirimkan paket

recvnum = jumlah paket yang berhasil diterima

3.5.3 Routing Overhead (RO)

Routing Overhead adalah jumlah *control message* yang dilakukan per paket menuju *node* tujuan selama simulasi. RO melibatkan semua *control message* meliputi RREP, RREQ, RRER. Jika nilai RO lebih rendah, maka dapat disimpulkan bahwa performa AODV tersebut lebih baik. Rumus dari RO adalah sebagai berikut:

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

Keterangan :

packet sent = jumlah paket yang dikirim

sentnum = jumlah paket yang terkirim

3.5.4 Average Hop Count (HC)

HC merupakan hasil rata-rata dari *hop count* di setiap paket menuju *destination node*. Jika nilai HC lebih kecil, maka dapat

dikatakan performa AODV tersebut lebih baik. Rumus untuk HC adalah sebagai berikut:

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

Keterangan :

<i>hop count</i>	=	jumlah <i>hop count</i> pada suatu paket
<i>recvnum</i>	=	jumlah paket yang berhasil diterima

(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.

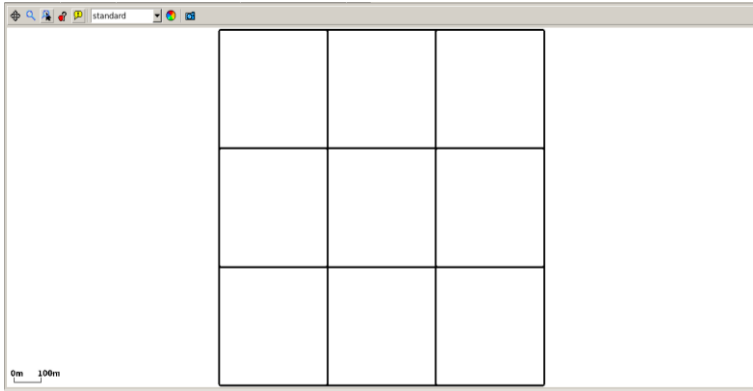
4.1 Implementasi Skenario *Grid*

Pada skenario *grid* yang akan digunakan untuk simulasi, SUMO mempunyai tools untuk melakukan generasi peta bernama *netgenerate*. Dalam Tugas Akhir ini, penulis membuat peta dengan titik horizontal x vertical yaitu 4 x 4 titik. Titik-titik tersebut nantinya akan membuat sebuah 9 petak segiempat. Sehingga untuk membuat sebuah peta seluas 700 m x 700 m, maka dibutuhkan seluas 200 m x 200 m per petak. Perintah *netgenerate* untuk membuat peta tersebut ada pada Gambar 4.1.

```
netgenerate --grid --grid.number=4 --grid.length=200 --default.speed=20 --tls.guess=1 --  
output-file=map.net.xml
```

Gambar 4.1 Perintah saat melakukan *netgenerate*

Setelah perintah *netgenerate* berhasil dilakukan, maka perintah tersebut akan menghasilkan *file* dengan ekstensi *.xml*. Peta yang telah dibuat ada pada Gambar 4.2.



Gambar 4.2 Peta Grid Hasil Netgenerate

Setelah berhasil melakukan pembuatan peta, maka dibuatlah *node* serta pergerakan *node* dengan menentukan titik awal dan akhir dari setiap *node* yang telah dibuat. Dalam pembuatan *node* tersebut maka digunakanlah *tools* bernama randomTrips untuk membuat *node* sebanyak *n* sesuai pada Gambar 4.3.

```
python $SUMO_HOME/tools/randomTrips.py -n map.net.xml -e 48 -l --trip-attributes="departLane=\"best\" departSpeed=\"max\" departPos=\"random_free\"" -o trip.trips.xml
```

Gambar 4.3 Perintah dari Tools randomTrips

Setelah melakukan randomTrips, maka dibuatlah rute yang digunakan kendaraan untuk mencapai tujuannya dengan menggunakan *tools* bernama duarouter sesuai pada Gambar 4.4.

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

Gambar 4.4 Perintah dari duarouter

Saat menggunakan *tools* duarouter, SUMO akan memastikan bahwa kendaraan-kendaraan yang ada tidak melenceng dari jalur peta yang telah dibuat sebelumnya. Maka untuk menjadikan peta dan pergerakan *node* yang telah dibuat menjadi sebuah skenario dalam

bentuk *file* berekstensi .xml, dibutuhkan sebuah *file* skrip dengan ekstensi .sumocfg untuk menggabungkan file peta dengan rute pergerakan dari *node*. *File* .sumocfg yang akan dibuat sesuai pada Gambar 4.5.

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

Gambar 4.5 Isi dari File Skrip .sumocfg

Setelah *file* .sumocfg dibuat, maka file tersebut akan disimpan di folder yang sama dengan file peta dan file rute pergerakan *node* yang telah dibuat sebelumnya. *File* .sumocfg ini dapat dibuka dengan menggunakan sumo-gui. Setelah itu *file* .sumocfg tersebut dapat dikonversi menjadi file .xml menggunakan *tools* dari SUMO. Perintah untuk melakukan konversi tersebut terdapat pada Gambar 4.6.

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

Gambar 4.6 Perintah SUMO untuk melakukan konversi

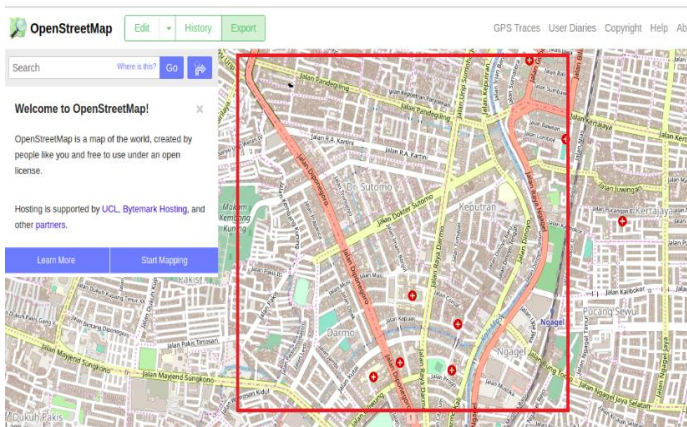
Setelah *file* .xml tersebut telah dihasilkan, maka *file* .xml tersebut akan dikonversi menjadi *file* .tcl agar dapat dibaca dan disimulasikan menggunakan NS-2. Dalam melakukan konversi maka dibutuhkan *tools* bernama traceExporter. Perintah untuk menggunakan traceExporter terdapat pada Gambar 4.7.

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

Gambar 4.7 Perintah dari traceExporter

4.2 Implementasi Skenario *Real*

Dalam pembuatan skenario *real*, maka dibutuhkan sebuah area dari dunia nyata yang akan dikonversi menjadi area simulasi. Pada Tugas Akhir ini penulis mengambil area di daerah Jl. Dr Soetomo kota Surabaya. Area yang diambil untuk Tugas Akhir ini terdapat pada Gambar 4.8.



Gambar 4.8 Ekspor Area dari OpenStreetMap

Kemudian, *file* hasil ekspor tersebut mempunyai ekstensi *.osm*. Setelah itu *file* tersebut akan dikonversi menjadi *file .xml* sama halnya dengan peta grid, namun untuk aktivitas ini menggunakan *tools* yang berbeda yaitu perintah *netconvert* sesuai dengan Gambar 4.9.

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

Gambar 4.9 Perintah dari *netconvert*

File .xml yang dihasilkan dapat dilihat pada Gambar 4.10 dengan menggunakan *tools* *sumo-gui*.



Gambar 4.10 Hasil konversi dari Peta Real

Setelah peta dihasilkan, maka langkah selanjutnya sama seperti dengan pembuatan peta grid, yaitu melakukan konversi ke *file* .sumocfg terlebih dahulu menggunakan randomTrips dan mengkonversi *file* .sumocfg ke .tcl dengan menggunakan traceExporter.

4.3 Implementasi *K-Means Clustering* pada AODV

Implementasi *K-Means Clustering* pada AODV ini dengan melakukan modifikasi terutama di bagian *control message* dan segala 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*. Pertama-tama pengisian data dari *nodes position table* ke dalam *n cluster* terlebih dahulu, setelah itu dilakukan proses penentuan *centroid* secara *random*. Potongan kode proses inisialisasi tersebut terdapat pada Gambar 4.11.

```
void AODV::runCluster(){
.....

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 ;
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 K-means Clustering

Seluruh kode yang terdapat pada Gambar 4.11 akan ditampilkan pada lampiran A.1. Proses tersebut akan dilakukan hingga selesai lalu akan menghasilkan *node clusters table* yang berisi index *node* yang nantinya akan bertindak sebagai *cluster head* dan *cluster getaway* sesuai dengan 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

Seluruh kode yang terdapat pada Gambar 4.12 akan ditampilkan pada lampiran A.2. Hasil dari proses *clustering* akan ditampilkan pada *nodes clusters table* pada server *node*. Proses pemilihan *cluster head* dan *cluster gateway* akan dilakukan selanjutnya untuk mengetahui *node* manakah yang akan menjadi *cluster head* dan *cluster gateway*. Proses tersebut terdapat 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 Pemilihan Cluster Head dan Cluster Gateway

Seluruh kode yang terdapat pada Gambar 4.13 akan ditampilkan pada lampiran A.3, dan untuk kode `getClusterHead()` dan `getClusterGateway()` ada pada lampiran A.4 dan A.5.

4.3.1 Modifikasi Pengiriman RREQ

Pengiriman RREQ nantinya akan disertai *nodes positions table* yang sebelumnya sudah disipsikan ke pada RREQ. Setiap *Node* akan mengirimkan RREQ sebelumnya akan mengisi informasi mengenai posisi *node* tersebut ke dalam *nodes positions table* yang ada di *node* tersebut. Potongan kode dari implementasi tersebut terdapat pada Gambar 4.14.

```

void AODV::sendRequest(nsaddr_t dst) {
.....

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

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

```

Gambar 4.14 Modifikasi Pengiriman RREQ untuk K-Means

Seluruh kode yang terdapat pada Gambar 4.14 terdapat pada lampiran A.6.

4.3.2 Modifikasi Penerimaan RREQ

Paket RREQ yang telah diterima oleh sebuah *node* akan diambil informasi mengenai *nodes positions table* yang ada pada paket tersebut, setelah itu akan dibandingkan dengan *nodes positions table* yang ada pada penerima. Perbandingan tersebut akan melihat *timestamps* atau waktu saat data tersebut didapatkan. Setelah itu *nodes positions table* yang baru akan memperbarui data *timestamps* yang ada pada paket yang telah diterima. Proses ini bertujuan agar setiap *node* yang menerima paket memiliki data yang terbaru. Potongan kode dari proses tersebut terdapat pada Gambar 4.15.

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

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

Gambar 4.15 Modifikasi Penerimaan RREQ untuk K-Means

Seluruh kode yang terdapat pada Gambar 4.15 terdapat pada lampiran A.7.

4.3.3 Modifikasi Pengiriman RREP

Pengiriman RREP nantinya akan disisipkan *nodes clusters table*. Potongan kode dari proses tersebut terdapat pada Gambar 4.16

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

.....

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

rp->ct = this->ct;

.....

}
```

Gambar 4.16 Modifikasi Pengiriman RREP untuk K-Means

Seluruh kode yang terdapat pada Gambar 4.16 terdapat pada lampiran A.8.

4.3.4 Modifikasi Penerimaan RREP

Paket RREP yang telah diterima akan diambil informasi mengenai *nodes clusters table* setelah itu dibandingkan *node clusters table* yang ada pada penerima. Perbandingan ini akan dilakukan dengan mendasarkan *timestamps* yang ada pada *nodes clusters table*. Tabel ini akan memberikan informasi untuk suatu *node* akan bertindak sebagai *cluster head*, *cluster gateway*, ataupun member yang ada dari tiap *cluster*. Potongan kode dari proses tersebut terdapat pada Gambar 4.17.


```

void AODV::recvReply(Packet *p) {
    //struct hdr_cmh *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

Seluruh kode yang terdapat pada Gambar 4.17 terdapat pada lampiran A.9.

4.3.5 Modifikasi Pengiriman Hello Messages

Pengiriman *hello messages* akan disisipkan *nodes clusters table* dan *nodes positions table*. *Node* pengirim akan memberikan informasi posisi baru ke dalam *nodes positions table* yang ada pada *node* tersebut. Potongan kode dari proses tersebut terdapat 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*) mNode)->getLoc(&posX,&posY);
    this->pt.nodes_position_x[index] = posX;
    this->pt.nodes_position_y[index] = posY;
    this->pt.nodes_timestamp[index] = CURRENT_TIME;
    rh->pt = this->pt;
    rh->ct = this->ct;

    .....
}

```

Gambar 4.18 Modifikasi Hello Messages untuk K-Means

Seluruh kode yang terdapat pada Gambar 4.18 terdapat pada lampiran A.10.

4.4 Implementasi algoritma *Backup Routing* pada AODV

Implementasi dari algoritma *Backup Routing* akan dilakukan dengan modifikasi dari *control message* di bagian RREQ, RREP, RERR yang nantinya hasil modifikasi per proses tersebut akan dinamakan RRREQ, RRREP, RRERR. Namun dari ketiga perubahan tersebut memiliki perubahan yang sama, yaitu perhitungan seluruh rute sebelum mengirimkan RREQ, maka dari itu perubahan yang dilakukan adalah dari RREQ saja.

Proses dari *Backup Routing* nantinya akan menghitung seluruh proses perhitungan rute dari setiap node, yang nantinya setiap node akan menghitung seluruh jalan yang tersedia untuk mencapai node tujuan.

Proses inisiasi dari *Backup Routing* diawali dengan menambah sebuah variabel bertipe array untuk menampung energi dari setiap node dan akan menganggap seluruh *node* yang ada bukanlah sebuah *cluster head* sebelum menjalankan *K-Means Clustering* sesuai pada Gambar 4.19.

```

double avg_energy_node[1000];
double min_energy_node[1000];

MobileNode *iNode;
xpos = 0;
ypos = 0;
iEnergy = 0.0000;

mNode = (MobileNode *) (Node::get_node_by_address(index));
isClusterHead = false;

```

Gambar 4.19 Proses Inisiasi *Backup Routing*

4.4.1 Modifikasi Pengiriman RREQ

Pada saat node sumber mengirim RREQ ke pada *node* tujuan dari *node* sumber, maka akan secara otomatis menambahkan energi yang dimiliki oleh *node* sumber kepada jumlah energi pada rute lalu melakukan *reset* terhadap nilai rata-rata energi maksimum yang dimiliki oleh semua node yang ada. Seluruh kode dari proses tersebut terdapat pada Gambar 4.20.

```

iNode= (MobileNode *) (Node::get_node_by_address (index) );
xpos= iNode->X();
ypos= iNode->Y();
iEnergy= iNode->energy_model()->energy();

for (int i=0;i<1000;i++) {
    avg_energy_node[i] = 0;
}

rq->rq_energy = iEnergy;

FILE *fp;
fp = fopen("debug.txt", "a");
fprintf(fp, "\n node %d mengirim route request: energy: %f", index, rq->rq_energy);

```

```
fclose(fp);
```

Gambar 4.20 Modifikasi Pengiriman RREQ pada *Backup Routing*

4.4.2 Modifikasi Penerimaan RREQ

Setelah *node* menerima paket, setiap *node* akan menghitung seluruh rute yang tersedia lalu dari setiap rute akan dihitung jumlah energinya, dari setiap rute dipilih *node* dengan energi yang terkecil setelah itu jumlah energi pada suatu rute akan dibagi dengan *hop count* untuk mendapatkan nilai *a*, setelah mendapatkan nilai *a*, maka dipilihlah nilai *a* yang paling besar sesuai pada Gambar 4.21.

```
double avg_energy, min_energy;
rq->rq_energy = rq->rq_energy + iEnergy;
rq->rq_min_energy = iEnergy;
avg_energy = rq->rq_energy / (rq->rq_hop_count + 1);
if (rq->rq_min_energy < min_energy_node[index]) {
    //Packet::free(p);
    min_energy_node[index] = rq->rq_min_energy; //update minimum energi pada node
    if (avg_energy < avg_energy_node[index]) {
        FILE *fp;
        fp = fopen("debug.txt", "a");
        fprintf(fp, "\n node %d didrop: avg energi: %f", index, avg_energy);
        fclose(fp);
        Packet::free(p); //drop paket jika avg energi yang diterima < max avg energi pada
node
        return;
    }
    else {
        FILE *fp;
        fp = fopen("debug.txt", "a");
        fprintf(fp, "\n node %d diteruskan: avg energi: %f", index, avg_energy);
        fclose(fp);
        avg_energy_node[index] = avg_energy; //update avg energi
    }
}
else {
    Packet::free(p);
    return;
}
```

Gambar 4.21 Modifikasi Penerimaan RREQ pada *Backup Routing*

4.5 Implementasi Simulasi pada NS-2

Implementasi untuk melakukan simulasi diawali dengan melakukan pendeskripsian dari *file* .tcl. File ini berisikan konfigurasi umum dan langkah-langkah yang dijalankan saat simulasi. Potongan kode konfigurasi dari proses tersebut terdapat pada Gambar 4.22.

set val(chan)	Channel/WirelessChannel	;
set val(prop)	Propagation/TwoRayGround	;
set val(netif)	Phy/WirelessPhy	;
set val(mac)	Mac/802_11	;
set val(ifq)	Queue/DropTail/PriQueue	;
set val(ll)	LL	;
set val(ant)	Antenna/OmniAntenna	;
set opt(x)	700	;
set opt(y)	700	;
set val(ifqlen)	1000	;
set val(nn)	200	;
set val(seed)	1.0	;
set val(adhocRouting)	AODV	;
set val(stop)	200	;
set val(cp)	"cbr.txt"	;
set val(sc)	"scenario.txt"	;

Gambar 4.22 Konfigurasi Lingkungan Simulasi

Pada konfigurasi di Gambar 4.21, dilakukan pemanggilan *file traffic* yang berisikan konfigurasi *source node*, *destination node*, dan pengiriman paket dengan sesi, serta *file* skenario yang berisikan pergerakan *node* yang telah dibuat menggunakan SUMO sebelumnya. Seluruh kode yang terdapat pada Gambar 4.22 terdapat pada lampiran A.11.

4.6 Implementasi Metrik Analisis

Implementasi untuk menguji hasil simulasi yang telah dilakukan sebelumnya akan mengukur dari beberapa metrik, yaitu PDR, E2E, HC, dan RO. Seluruh perhitungan metrik akan disatukan ke dalam *file* yang mempunyai ekstensi .awk. Seluruh kode untuk

file .awk yang akan melakukan pengujian terdapat pada lampiran A.12. Namun untuk melakukan eksekusi terhadap *file* awk tersebut terdapat pada Gambar 4.23.

```
awk -f result.awk $1
```

Gambar 4.23 File Metrik Analisis

(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 dari Tugas Akhir ini dilakukan di perangkat dengan spesifikasi seperti pada Tabel 5.1.

Komponen	Spesifikasi
CPU	Intel(R) core(TM) i7-8750H CPU @ 2.20GHz
Sistem Operasi	Linux Mint 19.3 "Tricia" - Cinnamon (64-bit)
Linux Kernel	Linux Kernel 4.4
Memori	16.4 GB
Penyimpanan	50 GB

Tabel 5.1 Spesifikasi Perangkat Simulasi

Parameter lingkungan uji coba yang digunakan pada NS-2 dapat dilihat pada Tabel 3.4. Pengujian dilakukan dengan melakukan simulasi menggunakan NS-2. Hasil simulasi tersebut berupa *file* dengan ekstensi *.tr* yang akan dianalisis dan diuji menggunakan skrip *awk* dan *shell script*. Ada 4 parameter yang diuji, yaitu PDR, E2E, RO, HC. Seluruh kode skrip *awk* terdapat pada lampiran A.12.

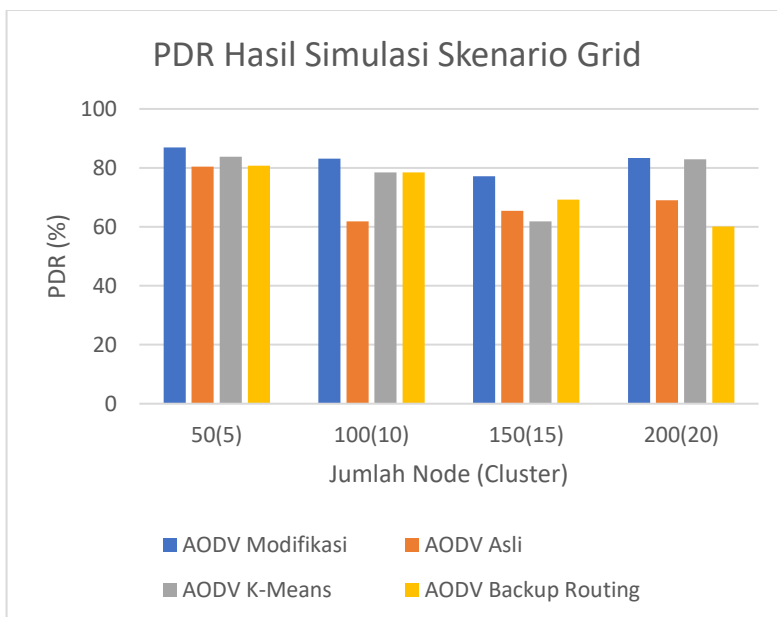
5.2 Skenario Grid

Pengujian pada skenario *grid* digunakan untuk melihat perbandingan PDR, E2E, RO, dan AHC antara AODV asli dan AODV modifikasi. Data pada skenario ini diambil sebanyak 10 kali

dalam setiap variasi kemudian hasil tersebut akan dilakukan perhitungan rata-rata untuk masing-masing variasi. *Cluster* dari setiap variasi diambil berdasarkan 10% dari jumlah *node* yang diuji. Berikut ini adalah hasil pengujian serta analisis dari keempat metrik yang diuji dalam Tugas Akhir ini.

Jumlah Node (Cluster)	AODV Modifikasi (%)	AODV Asli (%)	Perbedaan (%)	AODV K-Means (%)	AODV BR (%)
50(5)	86,973	80,368	6,605	83.78	80.73
100(10)	83,139	61,841	21,298	78.45	78.46
150(15)	77,145	65,476	11,669	61.80	69.26
200(20)	83,295	69,041	14,254	82.95	60.10

Tabel 5.2 PDR dari Hasil Simulasi Skenario *Grid*

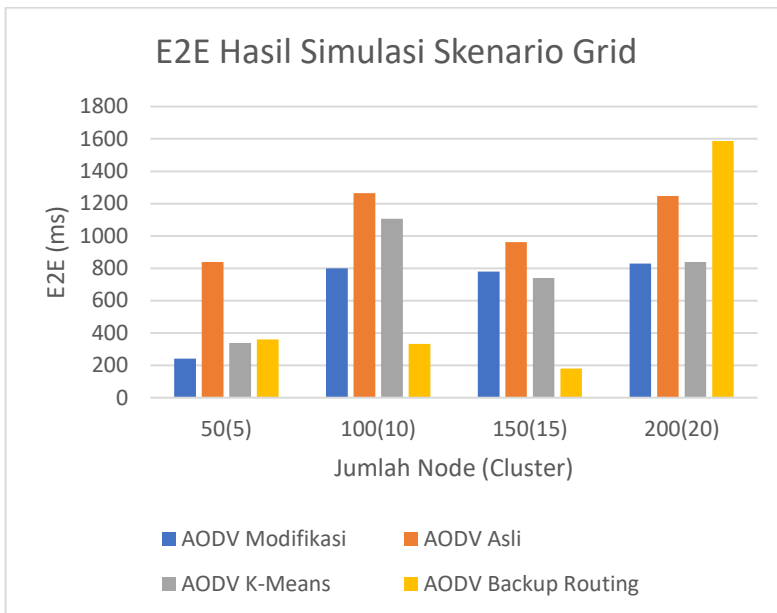


Gambar 5.1 Grafik PDR dari Hasil Simulasi Skenario *Grid*

Berdasarkan grafik pada Gambar 5.1, dapat dilihat bahwa dalam metrik PDR dalam setiap variasi AODV Modifikasi lebih unggul daripada AODV Asli. Perbedaan terjauh terjadi pada variasi jumlah 100 *node* dan 10 *cluster* dengan perbedaan mencapai 21,298%. Dalam grafik tersebut juga menunjukkan bahwa performa PDR dari AODV modifikasi tidak berbeda jauh dari setiap variasi dan dapat dikatakan cukup konsisten, kecuali pada variasi jumlah *node* 150 dan jumlah *cluster* 15, PDR dari AODV Modifikasi menunjukkan angka terendah dibandingkan dengan variasi lainnya, yaitu 77,145%, meskipun angka tersebut masih lebih tinggi daripada angka yang diperoleh AODV asli. Pada bagian ini, AODV Modifikasi menunjukkan angka yang lebih baik daripada AODV Asli dalam setiap variasi dalam skenario *grid*.

Jumlah Node (Cluster)	AODV Modifikasi (ms)	AODV Asli (ms)	Perbedaan (ms)	AODV K-Means (ms)	AODV BR (ms)
50(5)	242,34141	838,2864	-595,9449	337.851	360.758
100(10)	799,6421	1265,1203	-465,4782	1106.93	331.529
150(15)	780,5188	962,08983	-181,5710	740.687	179.998
200(20)	829,46716	1245,8534	-416,3862	839.888	1586.74

Tabel 5.3 E2E dari Hasil Simulasi Skenario *Grid*



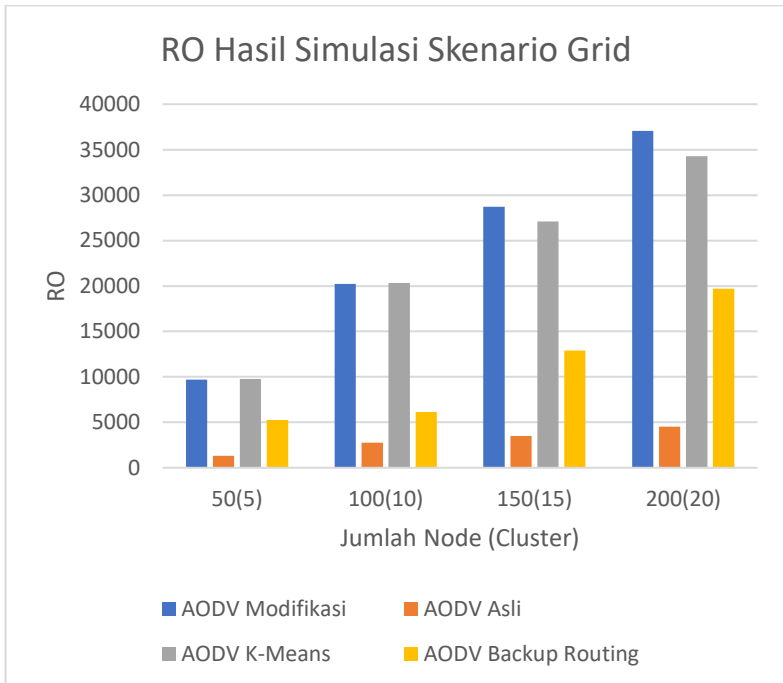
Gambar 5.2 Grafik E2E dari Hasil Simulasi Skenario *Grid*

Berdasarkan grafik pada Gambar 5.2, dapat dilihat bahwa dalam metrik E2E dalam setiap variasi AODV Modifikasi lebih unggul daripada AODV Asli. Sama seperti halnya PDR, dalam setiap variasi pun E2E dari AODV Modifikasi cukup konsisten, terkecuali pada variasi jumlah *node* 50 dan *cluster* 5, justru AODV Modifikasi mempunyai hasil yang amat baik, dengan waktu 242,34141 ms, AODV Modifikasi pada variasi tersebut memiliki keunggulan yang jauh dalam E2E dibandingkan dengan variasi lain. Pada bagian ini, AODV Modifikasi menunjukkan angka yang lebih baik daripada AODV Asli dalam setiap variasi dalam skenario *grid*.

Jumlah Node (Cluster)	AODV Modifikasi	AODV Asli	Perbedaan	AODV K-Means	AODV BR
50(5)	9701,1	1323,6	8377,5	9762	5257

100(10)	20234,5	2752,1	17482,4	20335	6152
150(15)	28704,2	3492,4	25211,8	27113	12904
200(20)	37069,7	4537,3	32532,4	34283	19710

Tabel 5.4 RO dari Hasil Simulasi Skenario *Grid*



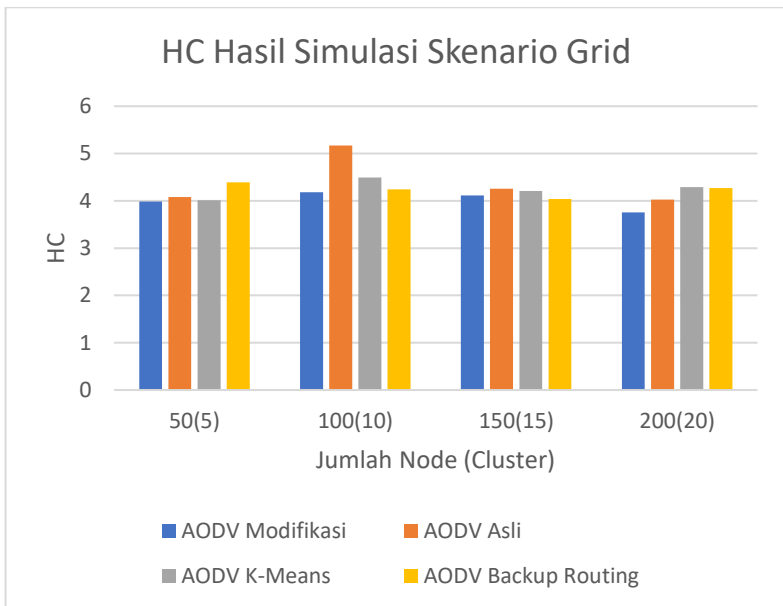
Gambar 5.3 Grafik RO dari Hasil Simulasi Skenario *Grid*

Berdasarkan grafik pada Gambar 5.3, dapat dilihat bahwa dalam metrik RO dalam setiap variasi AODV Modifikasi lebih buruk daripada AODV Asli. Hal itu dikarenakan RO AODV Modifikasi jauh lebih tinggi daripada AODV Asli. Terlihat juga dalam grafik tersebut, jumlah RO AODV Modifikasi dan AODV Asli meningkat seiring dengan bertambahnya jumlah *node* dan *cluster*. Namun perbedaan RO AODV Modifikasi dan AODV Asli paling banyak pada variasi jumlah *node* 200 dan jumlah *cluster* 20,

dengan selisih sebanyak 32532,4. Dari hasil pengujian teratas, maka dapat disimpulkan bahwa dalam metrik RO, AODV Modifikasi lebih buruk daripada AODV Asli dalam skenario *grid*.

Jumlah Node (Cluster)	AODV Modifikasi	AODV Asli	Perbedaan	AODV K-Means	AODV BR
50(5)	3,985	4,078	-0,093	4.01	4.39
100(10)	4,178	5,167	-0,989	4.49	4.24
150(15)	4,112	4,253	-0,141	4.21	4.04
200(20)	3,757	4,026	-0,269	4.29	4.27

Tabel 5.5 HC dari Hasil Simulasi Skenario *Grid*



Gambar 5.4 Grafik HC dari Hasil Simulasi Skenario *Grid*

Berdasarkan grafik pada Gambar 5.4, dapat dilihat bahwa dalam metrik HC dalam setiap variasi AODV Modifikasi lebih

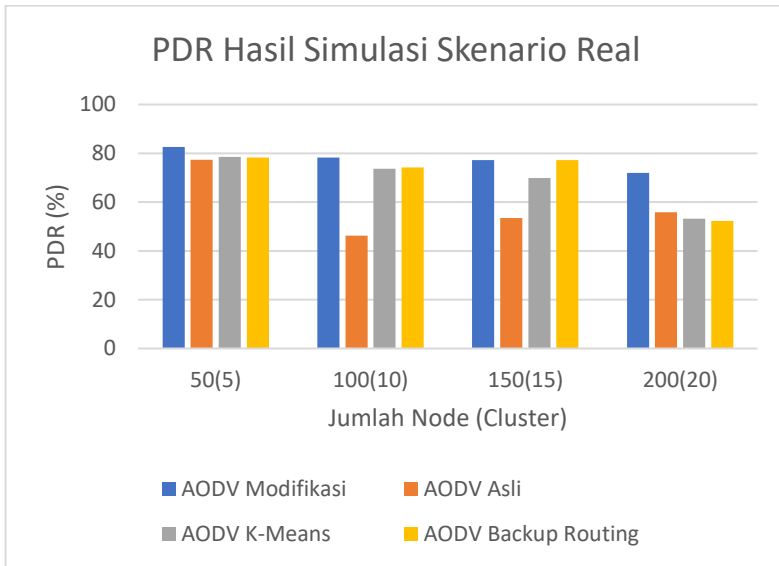
unggul daripada AODV Asli. Perbedaan yang terjauh terjadi pada variasi jumlah *node* 100 dan jumlah *cluster* 10, yaitu selisih 0,989. Namun untuk variasi lainnya, perbedaan HC antara AODV Modifikasi dan AODV Asli Relatif tidak berbeda jauh, namun jumlah HC yang dimiliki oleh AODV Modifikasi lebih kecil. Dalam hal HC, dapat disimpulkan bahwa AODV Modifikasi lebih unggul daripada AODV Asli dalam skenario *grid*.

5.3 Skenario Real

Pengujian pada skenario *real* digunakan untuk melihat perbandingan PDR, E2E, RO, dan AHC antara AODV asli dan AODV modifikasi. Data pada skenario ini diambil sebanyak 10 kali dalam setiap variasi kemudian hasil tersebut akan dilakukan perhitungan rata-rata untuk masing-masing variasi. *Cluster* dari setiap variasi diambil berdasarkan 10% dari jumlah *node* yang diuji. Berikut ini adalah hasil pengujian serta analisis dari keempat metrik yang diuji dalam Tugas Akhir ini.

Jumlah Node (Cluster)	AODV Modifikasi (%)	AODV Asli (%)	Perbedaan (%)	AODV K-Means (%)	AODV BR (%)
50(5)	82,608	77,379	5,229	78.50	78.22
100(10)	78,333	46,266	32,067	73.70	74.14
150(15)	77,269	53,447	23,822	69.90	77.23
200(20)	71,927	55,834	16,093	53.20	52.24

Tabel 5.6 PDR dari Hasil Simulasi Skenario *Real*



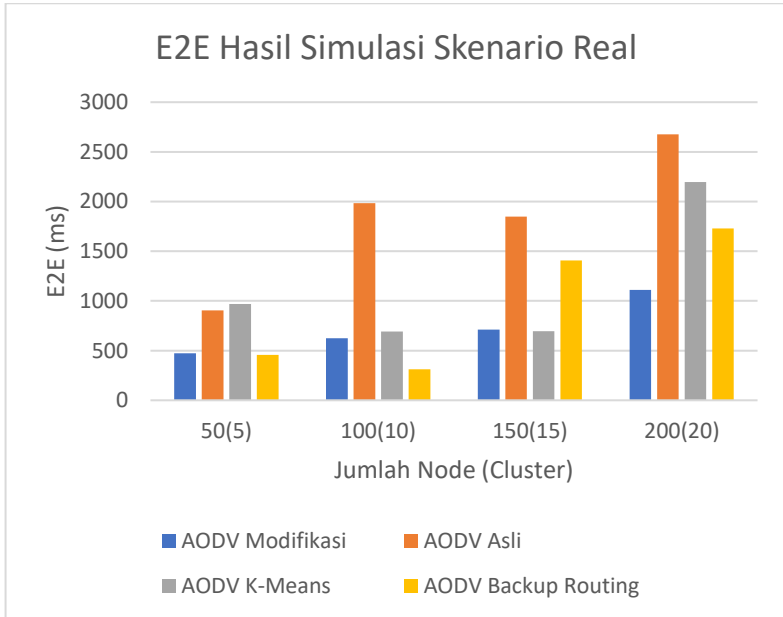
Gambar 5.5 Grafik PDR dari Hasil Simulasi Skenario *Real*

Berdasarkan grafik pada Gambar 5.5, dapat dilihat bahwa dalam metrik PDR dalam setiap variasi AODV Modifikasi lebih unggul daripada AODV Asli dari simulasi tersebut. Namun pada skenario *real*, semakin banyak jumlah *node* pada simulasi, maka rasio PDR dari AODV Modifikasi semakin menurun walaupun tidak terlalu banyak jumlah penurunannya dan tetap mengungguli hasil dari AODV Asli pada skenario *real*. Perbedaan terbanyak terdapat pada variasi jumlah *node* 100 dan jumlah *cluster* 10, yaitu dengan jumlah perbedaan mencapai 32,067%. Dalam metrik PDR, dapat disimpulkan bahwa AODV Modifikasi lebih baik daripada AODV Asli dalam skenario *real*.

Jumlah Node (Cluster)	AODV Modifikasi (ms)	AODV Asli (ms)	Perbedaan (ms)	AODV K-Means (ms)	AODV BR (ms)
50(5)	471,53471	904,2857	-432,75099	969.656	458.161

100(10)	625,37918	1982,5671	-1357,1879	691.944	311.212
150(15)	711,1943	1848,9637	-1137,769	696.609	1407.16
200(20)	1111,78121	2676,64494	-1564,8637	2197.95	1728.3

Tabel 5.7 E2E dari Hasil Simulasi Skenario *Real*



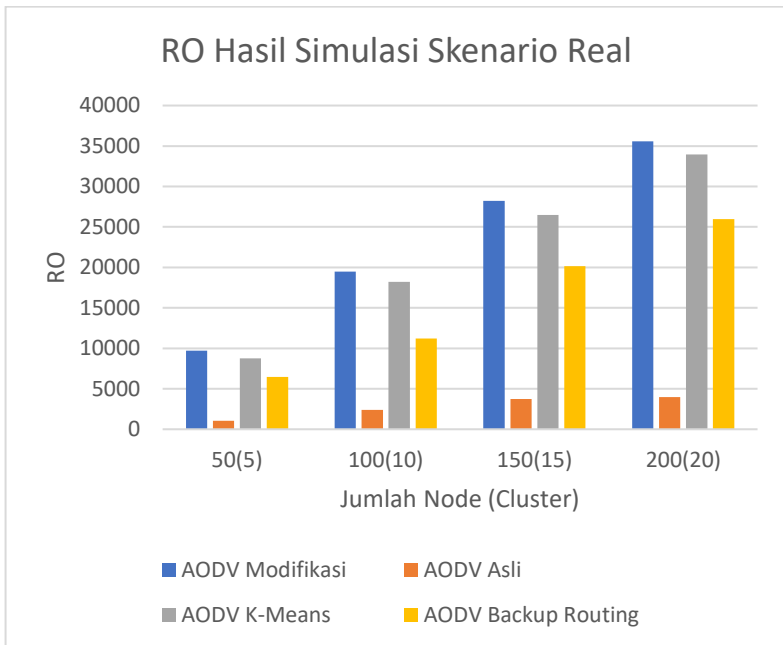
Gambar 5.6 Grafik E2E dari Hasil Simulasi Skenario *Real*

Berdasarkan grafik pada Gambar 5.6, dapat dilihat bahwa dalam metrik E2E dalam setiap variasi AODV Modifikasi lebih unggul daripada AODV Asli dari simulasi tersebut. Dalam grafik tersebut, terlihat bahwa angka dari E2E pada AODV Modifikasi bertambah seiring dengan bertambahnya jumlah *node* pada saat simulasi, namun angka dari E2E pada AODV Modifikasi masih menungguli angka dari AODV Asli. Perbedaan terbanyak terlihat pada variasi jumlah *node* 200 dan jumlah *cluster* 20, dengan perbedaan sebanyak 1564,86373 ms dengan AODV Modifikasi

lebih unggul. Dari hasil pengujian diatas, maka dalam metrik E2E AODV Modifikasi lebih baik daripada AODV Asli pada skenario *real*.

Jumlah Node (Cluster)	AODV Modifikasi	AODV Asli	Perbedaan	AODV K-Means	AODV BR
50(5)	9718,8	1058,4	8660,4	9760	6468
100(10)	19474,2	2375,4	17098,8	18199	11211
150(15)	28210,8	3723,6	24487,2	26482	20167
200(20)	35575,1	3988,862	31586,238	33959	25955

Tabel 5.8 RO dari Hasil Simulasi Skenario *Real*



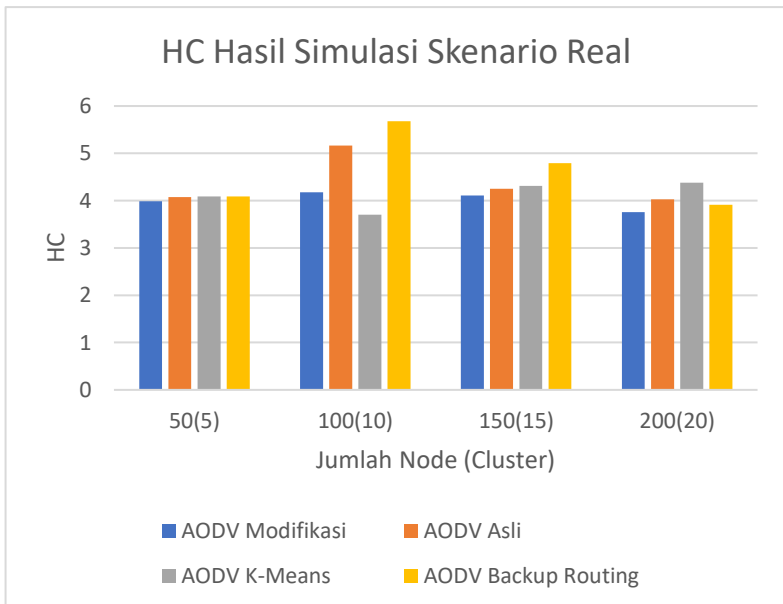
Gambar 5.7 Grafik RO dari Hasil Simulasi Skenario *Real*

Berdasarkan grafik pada Gambar 5.7, dapat dilihat bahwa dalam metrik RO dalam setiap variasi AODV Modifikasi lebih

buruk daripada AODV Asli dari simulasi tersebut. Terlihat juga perbedaan RO pada AODV Modifikasi dan AODV Asli terlampau cukup jauh, bahkan perbedaan mencapai ribuan, semakin banyak jumlah *node*, maka semakin jauh juga perbedaan RO antara AODV Modifikasi dan AODV Asli, dengan perbedaan terjauh pada jumlah *node* 200 dan jumlah *cluster* 20, dengan perbedaan mencapai 31586,238. Dalam metrik RO, dapat disimpulkan bahwa AODV Modifikasi lebih buruk daripada AODV Asli pada skenario *real*.

Jumlah Node (Cluster)	AODV Modifikasi	AODV Asli	Perbedaan	AODV K-Means	AODV BR
50(5)	3,727	4,335	-0,608	4.09	4.09
100(10)	3,971	5,301	-1,33	3.7	5.68
150(15)	4,002	5,606	-1,604	4.31	4.79
200(20)	3,893	5,252	-1,359	4.38	3.91

Tabel 5.9 HC dari Hasil Simulasi Skenario *Real*



Gambar 5.8 Grafik HC dari Hasil Simulasi Skenario *Real*

Berdasarkan grafik pada Gambar 5.8, dapat dilihat bahwa dalam metrik RO dalam setiap variasi AODV Modifikasi lebih baik daripada AODV Asli dari simulasi tersebut. Dengan perbedaan yang meningkat sebanding dengan jumlah *node*, namun terlihat pada variasi jumlah *node* 100 dan *cluster* 10 terdapat selisih yang paling besar dibandingkan dengan variasi lainnya, yaitu berjumlah 1,604. Jumlah HC pada AODV Modifikasi relatif konsisten, terlihat dari perbedaan yang tidak signifikan. Dalam metrik HC dapat disimpulkan bahwa AODV Modifikasi memiliki performa yang lebih baik daripada AODV Asli.

(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 untuk pengembangan selanjutnya 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 adalah sebagai berikut:

1. Penerapan *K-Means Clustering* telah berhasil mengurangi jumlah forwarding node secara signifikan dan setelah dikombinasikan dengan algoritma *Backup Routing* secara umum menghasilkan hasil yang lebih baik daripada AODV Asli berdasarkan *Packet Delivery Ratio*, *End-to-End Delay*, dan *Average Hop Count*, namun kombinasi ini juga menyebabkan *Routing Overhead* meningkat secara fluktuatif, hal ini dikarenakan pekerjaan yang dilakukan oleh setiap *node* yang jauh lebih banyak dibandingkan dengan AODV Asli.
2. Penerapan *K-Means Clustering* dan algoritma *Backup Routing* dalam skenario *grid* telah memengaruhi performa AODV dengan rata-rata kenaikan pada PDR sebesar 20%, rata-rata penurunan pada E2E sebesar 40%, rata-rata penurunan HC sebesar 8%, dan rata-rata kenaikan RO sebesar 677%.
3. Penerapan *K-Means Clustering* dan algoritma *Backup Routing* dalam skenario *real* telah memengaruhi performa AODV dengan rata-rata kenaikan pada PDR sebesar 37%, rata-rata penurunan pada E2E sebesar 59%, rata-rata

penurunan HC sebesar 23%, dan rata-rata kenaikan RO sebesar 747%.

6.2 Saran

Saran yang diberikan dari hasil pengujian dan analisis pada Tugas Akhir ini adalah sebagai berikut:

1. Melakukan uji coba dengan variasi yang lebih banyak sehingga mendapatkan data yang lebih akurat.
2. Menerapkan sebuah metode untuk mengurangi angka *Routing Overhead* dari AODV yang diterapkan *K-Means Clustering* dan algoritma *Backup Routing*.
3. Menerapkan metode selain *K-Means Clustering* dan algoritma *Backup Routing* yang dapat memaksimalkan kinerja dari AODV. Seperti *Particle Swarm Optimization* (PSO).

(Halaman ini sengaja dikosongkan)

DAFTAR PUSTAKA

- [1] M. L. Raja dan C. D. S. S. Baboo, "An Overview of MANET: Applications, Attacks and Challenges," *International Journal of Computer Science and Mobile Computing*, vol. 3, no. 1, pp. 408-417, 2014.
- [2] M. Singh dan S. Kumar, "A Survey: Ad-hoc on Demand Distance Vector (AODV) Protocol," *International Journal of Computer Applications*, vol. 161, no. 1, pp. 38-44, 2017.
- [3] G. G. S. Sruthy, "AODV based backup routing for optimized performance in mobile ad-hoc networks," *2017 International Conference on Computing Methodologies and Communication (ICCMC)*, pp. 684-688, 2017.
- [4] "OpenStreetMap," [Online]. Available: [https://www.openstreetmap.org/..](https://www.openstreetmap.org/) [Diakses 22 May 2020].
- [5] 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.
- [6] "JOSM," [Online]. Available: [josm.openstreetmap.de/..](https://josm.openstreetmap.de/) [Diakses 22 5 2020].
- [7] K. Shaymala, S. K. Lokhande, R. B. P dan S. Kumar, "Efficient backup routing scheme in AODV with unidirectional links," *2011 Annual IEEE India Conference*, 2011.

LAMPIRAN

A.1 Kode runCluster()

```

void AODV::runCluster(){
    KMeans km;
    srand(time(0));
    int memberId=0;
    int modClustersNodes = NUMBER_OF_NODE % NUMBER_OF_CLUSTER;
    int numberNodes;
    for(int i=0;i<NUMBER_OF_CLUSTER;i++)
    {
        if(i<modClustersNodes)numberNodes =
NUMBER_OF_NODE/NUMBER_OF_CLUSTER + 1;
        else numberNodes = NUMBER_OF_NODE/NUMBER_OF_CLUSTER ;
        Cluster cluster;
        for(int j=0;j<numberNodes;j++)
        {
            Member member;
            struct point pt;
            pt.x = this->pt.nodes_position_x[memberId];
            pt.y = this->pt.nodes_position_y[memberId];
            member.setMemberPoint(pt);
            member.setMemberIndex(memberId);
            memberId++;
            cluster.assignMember(member);
        }
        cluster.init();
        km.assignCluster(cluster);
    }
    km.run();

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

```

A.2 Kode 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.3 Kode 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.4 Kode 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.5 Kode 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.6 Kode sendRequest()

```

void
AODV::sendRequest(nsaddr_t dst) {
  // Allocate a RREQ packet
  Packet *p = Packet::alloc();
  struct hdr_cmh *ch = HDR_CMH(p);
  struct hdr_ip *ih = HDR_IP(p);
  struct hdr_aodv_request *rq = HDR_AODV_REQUEST(p);
  aodv_rt_entry *rt = rtable.rt_lookup(dst);

  //MODIFIED//
  //SIGNAL
  double receiveSignalStrength = p->txinfo_.RxPr;
  double RSSM;
  if(receiveSignalStrength == 0)
  {
    RSSM = -200;
  }
  else

```

```

{
    RSSM = 10*log10(receiveSignalStrength) + 30;
}
if(SIGNAL_THRESHOLD >= RSSM)
{
    Packet::free((Packet *)p);
    return;
}

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

assert(rt);

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

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

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

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

if (rt->rt_req_cnt > RREQ_RETRIES) {
    rt->rt_req_timeout = CURRENT_TIME + MAX_RREQ_TIMEOUT;
    rt->rt_req_cnt = 0;
    Packet *buf_pkt;
    while ((buf_pkt = rqueue.deque(rt->rt_dst))) {

```

```

    drop(buf_pkt, DROP_RTR_NO_ROUTE);
}
Packet::free((Packet *)p);
return;
}

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

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

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

if (0 == rt->rt_req_last_ttl) {
    // first time query broadcast
    ih->ttl_ = TTL_START;
}
else {
    // Expanding ring search.
    if (rt->rt_req_last_ttl < TTL_THRESHOLD)
        ih->ttl_ = rt->rt_req_last_ttl + TTL_INCREMENT;
    else {
        // network-wide broadcast
        ih->ttl_ = NETWORK_DIAMETER;
        rt->rt_req_cnt += 1;
    }
}

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

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

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

// Don't let the timeout to be too large, however ... SRD 6/8/99

```

```

if (rt->rt_req_timeout > CURRENT_TIME + MAX_RREQ_TIMEOUT){
    rt->rt_req_timeout = CURRENT_TIME + MAX_RREQ_TIMEOUT;
    rt->rt_expire = 0;
}

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

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

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

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

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

```

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

A.7 Kode recvRequest()

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

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

    aodv_rt_entry *rt;

    // MODIFIED //
    //SIGNAL
    double receiveSignalStrength = p->txinfo_.RxPr;
    double RSSM;
    if(receiveSignalStrength == 0)
    {
        RSSM = -200;
    }
    else
```



```

{
    RSSM = 10*log10(receiveSignalStrength) + 30;
}
if(SIGNAL_THRESHOLD >= RSSM)
{
    Packet::free((Packet *)p);
    return;
}

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

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

//HOPCOUNT
int hopCount = rq->rq_hop_count;

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

// MODIFIED //

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

```

```

    */

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

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

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

        Packet::free(p);
        return;
    }

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

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

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

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

    if ( (rq->rq_src_seqno > rt0->rt_seqno) ||
        ((rq->rq_src_seqno == rt0->rt_seqno) &&
         (rq->rq_hop_count < rt0->rt_hops)) ) {
        // If we have a fresher seq no. or lesser #hops for the

```

```

// same seq no., update the rt entry. Else don't bother.
rt_update(rt0, rq->rq_src_seqno, rq->rq_hop_count, ih->saddr(), rq->pheromoneCount,
          max(rt0->rt_expire, (CURRENT_TIME + REV_ROUTE_LIFE)));
if (rt0->rt_req_timeout > 0.0) {
// Reset the soft state and
// Set expiry time to CURRENT_TIME + ACTIVE_ROUTE_TIMEOUT
// This is because route is used in the forward direction,
// but only sources get benefited by this change
rt0->rt_req_cnt = 0;
rt0->rt_req_timeout = 0.0;
rt0->rt_req_last_ttl = rq->rq_hop_count;
rt0->rt_expire = CURRENT_TIME + ACTIVE_ROUTE_TIMEOUT;
}

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

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

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

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

// First check if I am the destination ..

if(rq->rq_dst == index) {

```

```

printf("\nSAMPAI DI DESTINATION -> %d Time : %f",index,CURRENT_TIME);
#ifdef DEBUG
    fprintf(stderr, "%d - %s: destination sending reply\n",
        index, __FUNCTION__);
#endif // DEBUG

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

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

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

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

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

#ifdef RREQ_GRAT_RREP

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

```

```

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

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

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

    Packet::free(p);
}
/*
 * Can't reply. So forward the Route Request
 */
else {
    ih->saddr() = index;
    ih->daddr() = IP_BROADCAST;
    rq->rq_hop_count += 1;
    // Maximum sequence number seen en route
    if (rt) rq->rq_dst_seqno = max(rt->rt_seqno, rq->rq_dst_seqno);

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

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

```

A.8 Kode sendReply()

```

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_cmh *ch = HDR_CMH(p);

```

```

struct hdr_ip *ih = HDR_IP(p);
struct hdr_aodv_reply *rp = HDR_AODV_REPLY(p);
aodv_rt_entry *rt = rtable.rt_lookup(ipdst);
    printf("\n SEND REPLY BY %d to %d at %f",rpdst,ipdst,CURRENT_TIME);
#ifdef DEBUG
fprintf(stderr, "sending Reply from %d at %.2f\n", index, Scheduler::instance().clock());
#endif // DEBUG
assert(rt);

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

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

// ch->uid() = 0;
ch->ptype() = PT_AODV;
ch->size() = IP_HDR_LEN + rp->size();
ch->iface() = -2;
ch->error() = 0;
ch->addr_type() = NS_AF_INET;
ch->next_hop_ = rt->rt_nexthop;
ch->prev_hop_ = index; // AODV hack
ch->direction() = hdr_cmnn::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.9 Kode recvReply()

```

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("\nSAMP AI KEMBALI (PERJALANAN) KE SOURCE -> %d Time :
%f",index,CURRENT_TIME);
// MODIFIED //
if(this->ct.nodes_cluster_timestamp < rp->ct.nodes_cluster_timestamp)
{
    this->ct = rp->ct;
    for(int i=0;i<NUMBER_OF_CLUSTER;i++)
    {
        if(this->ct.nodes_cluster_head[i] == this->index || this->ct.nodes_cluster_gateway[i]
== this->index)
        {
            this->isClusterHead = true;
            break;
        }
        else
        {
            this->isClusterHead = false;
        }
    }
}
// MODIFIED //

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

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

```

```

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

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

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

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

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

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

    if (ih->daddr() == index) { // If I am the original source
        // Update the route discovery latency statistics
        // rp->rp_timestamp is the time of request origination
        printf("\nSAMPAI KEMBALI DI SOURCE -> %d Time :
%f",index,CURRENT_TIME);
        rt->rt_disc_latency[(unsigned char)rt->hist_indx] = (CURRENT_TIME - rp-
>rp_timestamp)
            / (double) rp->rp_hop_count;
        // increment indx for next time
        rt->hist_indx = (rt->hist_indx + 1) % MAX_HISTORY;
    }

/*
 * Send all packets queued in the sendbuffer destined for
 * this destination.
 * XXX - observe the "second" use of p.
 */

```



```

Packet *buf_pkt;
while((buf_pkt = rqueue.deque(rt->rt_dst))) {
    if(rt->rt_hops != INFINITY2) {
        assert(rt->rt_flags == RTF_UP);
        // Delay them a little to help ARP. Otherwise ARP
        // may drop packets. -SRD 5/23/99
        forward(rt, buf_pkt, delay);
        delay += ARP_DELAY;
    }
}
else {
    suppress_reply = 1;
}

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

if(ih->daddr() == index || suppress_reply) {
    Packet::free(p);
}
/*
 * Otherwise, forward the Route Reply.
 */
else {
    // Find the rt entry
    aodv_rt_entry *rt0 = rtable.rt_lookup(ih->daddr());
    // If the rt is up, forward
    if(rt0 && (rt0->rt_hops != INFINITY2)) {
        assert(rt0->rt_flags == RTF_UP);
        rp->rp_hop_count += 1;
        rp->rp_src = index;
        forward(rt0, p, NO_DELAY);
        // Insert the nexthop towards the RREQ source to
        // the precursor list of the RREQ destination
        rt->pc_insert(rt0->rt_nexthop); // nexthop to RREQ source
    }
    else {
        // I don't know how to forward .. drop the reply.
#ifdef DEBUG
        fprintf(stderr, "%s: dropping Route Reply\n", __FUNCTION__);
#endif
        drop(p, DROP_RTR_NO_ROUTE);
    }
}

```

```
}
}
```

A.10 Kode sendHello()

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

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

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

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

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

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

```

ih->ttl_ = 1;

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

```

A.11 Kode konfigurasi skenario NS-2

```

#
=====
# Define options
#
=====
=====

set val(chan)          Channel/WirelessChannel    ;# channel type
set val(prop)          Propagation/TwoRayGround   ;# radio-propagation model
set val(netif)         Phy/WirelessPhy           ;# network interface type
set val(mac)           Mac/802_11                ;# MAC type
set val(ifq)           Queue/DropTail/PriQueue    ;# interface queue type
set val(ll)            LL                        ;# link layer type
set val(ant)           Antenna/OmniAntenna        ;# antenna model
set opt(x)             700                      ;# X dimension of the topography
set opt(y)             700                      ;# Y dimension of the topography
set val(ifqlen)        1000                     ;# max packet in ifq
set val(nn)            50                       ;# how many nodes are simulated
set val(seed)          1.0                      ;
set val(adhocRouting)  AODV                     ;# routing protocol
set val(stop)          200                      ;# simulation time
set val(cp)            "cbr.txt"                 ;#<-- traffic file
set val(sc)            "scenario.txt"             ;#<-- mobility file

set val(energy_mod)    EnergyModel               ;# energy model
set val(energy_init)   100                       ;# init val for energy
set val(tx_power)      1.65                      ;# energy consume for transmitting packet
set val(rx_power)      1.4                      ;# energy consume for receiving packet
set val(idle_power)    0.5                      ;# energy consume for idle
set val(sleep_power)   0.3                      ;# energy consume for sleep mode

#
=====
# Main Program

```

```

#
=====
# Initialize Global Variables
# create simulator instance

set ns_ [new Simulator]

# setup topography object

set topo [new Topography]

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

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

# set up topology object
set topo [new Topography]
$topo load_flatgrid $opt(x) $opt(y)

# Create God
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

###

# 802.11p default parameters
Phy/WirelessPhy set RXThresh_ 3.65262e-10 ; #250m
Phy/WirelessPhy set CSThresh_ 3.65262e-10 ; #250m

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

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

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

# Define node initial position in nam

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

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

    $ns_ initial_node_pos $node_($i) 20
}

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

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

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

```

```
puts "Starting Simulation..."
$ns_run
```

A.12 Kode konfigurasi awk

```
BEGIN {
    sendLine = 0;
    recvLine = 0;
    fowardLine = 0;
    TC = 0;
    rt_pkts = 0;
    rt_send = 0;
    rt_forward = 0;
    recvd = 0;
    hc = 0;
}

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

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

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

$0 ~ /^s.* \[TC / {
    TC ++ ;
}

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

{
    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 == "cbr") && ($1 == "r")) {

        end_time[$6] = $2;

    } else if($1 == "D" && $7 == "cbr") {

        end_time[$6] = -1;

    } else if (($1 == "s" || $1 == "f") && ($4 == "RTR") && ($7 == "AODV")) {

        rt_pkts++;
    }
    if (($1 == "s") && ($4 == "RTR") && ($7 == "AODV") && ($25 ==
"(REQUEST)")) {

        rt_send++;
    }
    if (($1 == "s") && ($4 == "RTR") && ($7 == "AODV") && ($25 ==
"(REQUEST)") && ($3 != "_58_")) {

        rt_forward++;
    }

}

END {

    for(i=0; i<=seqno; i++) {

        if(end_time[i] > 0) {

            delay[i] = end_time[i] - start_time[i];

            count++;

        }

        else

```

```

    {

        delay[i] = -1;

    }

}

for(i=0; i<=seqno; i++) {

    if(delay[i] > 0) {

        n_to_n_delay = n_to_n_delay + delay[i];

    }

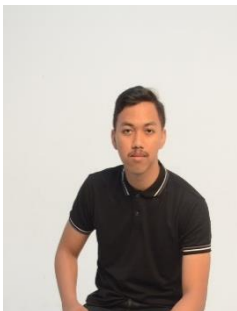
}

n_to_n_delay = n_to_n_delay/count;

printf "Packet sendLine \t= %d \n", sendLine;
printf "Packet rcvLine \t= %d \n", rcvLine;
printf "Packet PDR Ratio \t= %.4f \n", (rcvLine/sendLine);
printf "Packet loss \t= %d \n", (sendLine-rcvLine);
printf "Packet forwardLine\t= %d \n", fowardLine;
printf "End-to-End Delay \t= " n_to_n_delay * 1000 " ms \n";
printf "Topology Control \t= %d \n", TC;
printf "Routing Packets \t= %d \n", rt_pkts;
printf "Route Request Send \t= %d \n", rt_send;
printf "Route Request Forwarded \t= %d \n", rt_forward;
printf sendLine"\t"rcvLine"\t%.4f\t"(sendLine-rcvLine)"\t" n_to_n_delay * 1000
ms\t" rt_pkts"\t"rt_send"\t" rt_forward"\n", (rcvLine/sendLine);
printf("AHC : %.2f\n",hc/rcvd);
}

```


BIODATA PENULIS



FAHRIZAL NAUFAL AHMAD, lahir di Kediri, 8 Agustus 1998. Penulis merupakan anak sulung dari tiga bersaudara. Penulis menempuh Pendidikan sekolah dasar di SDI Al-Azhar 20 Cibubur, kemudian melanjutkan sekolah di SMP Labschool Jakarta, setelah itu menempuh Pendidikan sekolah menengah atas di SMA Labschool Jakarta, setelah lulus dari Pendidikan SMA, Penulis melanjutkan

Pendidikan Sarjana di Departemen Informatika, Fakultas Teknologi Informasi dan Komunikasi, Institut Teknologi Sepuluh Nopember, Surabaya.

Dalam menempuh pendidikan sarjana, Penulis mengambil bidang minat Arsitektur Jaringan dan Komputer (AJK). Penulis juga aktif dalam bersosialisasi dan berorganisasi selama menempuh pendidikan di kampus perjuangan. Penulis mengikuti beberapa organisasi, seperti HMTC ITS, dan menjadi Kepala Departemen Hubungan Luar BEM FTIK ITS. Penulis juga sempat berkegiatan di UKM CLICK ITS selama awal perkuliahan. Selain itu, penulis juga sempat menjadi penanggung jawab dari REEVA SCHEMATICS 2018, salah satu sub-event dari SCHEMATICS 2018. Penulis juga sempat melakukan *internship* di PT. SIGMA CIPTA CARAKA atau yang biasa kita kenal sebagai TELKOMSIGMA. Jika ada pertanyaan atau saran, silakan menghubungi penulis di +6281320465964 atau fahrizal.8898@hotmail.com.