TUGAS AKHIR - KI141502

IMPLEMENTASI METODE *CLUSTERING* BERDASARKAN *NODE DEGREE* PADA *AD-HOC ON DEMAND DISTANCE VECTOR* (AODV) DI LINGKUNGAN VANETS

NAUFAL PRANASETYO FODENSI
NRP 05111540000057

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

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

Departemen Informatika
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya 2019

TUGAS AKHIR – KI141502

IMPLEMENTASI METODE *CLUSTERING* BERDASARKAN *NODE DEGREE* PADA *AD-HOC ON DEMAND DISTANCE VECTOR* (AODV) DI LINGKUNGAN VANETS

NAUFAL PRANASETYO FODENSI
NRP 05111540000057

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

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

Departemen Informatika
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya 2019

(Halaman ini sengaja dikosongkan)

UNDERGRADUATE THESES – KI141502

IMPLEMENTATION OF CLUSTERING METHOD BASED ON NODE DEGREE LEVEL IN AD-HOC ON DEMAND DISTANCE VECTOR (AODV) IN VANETS

NAUFAL PRANASETYO FODENSI
NRP 05111540000057

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

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

Department of Informatics
Faculty of Information Technology and Communication
Sepuluh Nopember Institute of Technology
Surabaya 2019

(Halaman ini sengaja dikosongkan)

LEMBAR PENGESAHAN

IMPLEMENTASI METODE *CLUSTERING* BERDASARKAN *NODE DEGREE* PADA *AD-HOC ON DEMAND DISTANCE VECTOR* (AODV) DI LINGKUNGAN *VANETS*

TUGAS AKHIR

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

Oleh:
NAUFAL PRANASETYO FODENSI
NRP: 05111540000057

Disetujui oleh Pembimbing Tugas Akhir:

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

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

SURABAYA
JUNI, 2019

(Halaman ini sengaja dikosongkan)

IMPLEMENTASI METODE *CLUSTERING* BERDASARKAN *NODE DEGREE* PADA *AD-HOC ON DEMAND DISTANCE VECTOR* (AODV) DI LINGKUNGAN VANETS

Nama Mahasiswa	**: NAUFAL PRANASETYO FODENSI**
NRP	**: 05111540000057**
Departemen	**: Informatika FTIK-ITS**
Dosen Pembimbing 1	**: Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.**
Dosen Pembimbing 2	**: Prof. Ir. Supeno Djanali, M.Sc., Ph.D.**

Abstrak

Teknologi internet saat ini dapat digunakan sebagai suatu pemecahan suatu masalah. Contoh perkembangannya adalah jaringan nirkabel *ad-hoc.* Seperti contohnya proses penentuan rute perjalanan berhubungan dengan rute pengiriman data informasi dalam jaringan internet. Untuk melakukan pemcecahan permasalahan hal tersebut dapat memanfaatkan teknologi jaringan *Ad-Hoc* yang mana mendasari pembuatan Vehicular Ad-Hoc Networks (VANETs). *Vehicular Ad hoc Networks* (VANETs) merupakan pengembangan dari *Mobile Ad hoc Network* (MANET) dimana *node* memiliki karakteristik dengan mobilitas yang sangat tinggi dan terbatas pada pola pergerakannya. Ada banyak *routing protocol* yang dapat diimplementasikan pada VANETs, salah satunya adalah *Ad hoc On demand Distance Vector* (AODV).

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

digunakan untuk mengetahui informasi adanya kesalahan pada rute. Pada fase ini terdapat proses pengiriman *Route Error* (RERR).

Pada kinerja AODV, pemilihan rute yang stabil saat proses pencarian rute diperlukan untuk memperpanjang waktu penggunaan *node*. Seperti contohnya ketika tiap *node* melakukan *broadcast* paket RREQ, hal tersebut membutuhkan *delay* yang lama dan *resource* yang banyak saat pengiriman. Hal tersebut berpengaruh pada tingkat kestabilan rute dan rentan terputus yang mengakibatkan paket yang dikirim berkurang.

Pada Tugas Akhir ini mengimplementasikan suatu algoritma clustering yang bernama *AODV based on Node Degree Clustering* untuk mengontrol mekanisme *flooding* saat membangun suatu rute baru di protokol AODV. Di dalam algoritma ini, dilakukan pemilihan *Cluster Head* yang didapatkan berdasarkan tingkat derajat *node* tetangganya. Lalu tiap *node* akan di klasifikasikan sesuai *Cluster*nya. Maka saat melakukan *Rebroadcast* RREQ, tidak perlu mengirim ke semua node, melainkan hanya melalui *node Cluster Head* dan node *gateway* untuk menghubungkan antar *Cluster*. Dari hasil uji coba, AODV yang dimodifikasi pada skenario *grid* berhasil meningkatkan nilai rata-rata *Packet Delivery Ratio* (PDR) hingga 13,41%, menurunkan *Routing Overhead* hingga 17,85% dan penurunan nilai rata-rata *Forwarded Route Request (RREQ F)* hingga 16,36%. Sedangkan pada skenario *real* berhasil meningkatkan nilai rata-rata *Packet Delivery Ratio* (PDR) hingga 13,03%, menurunkan *Routing Overhead* hingga 11,52% dan penurunan nilai rata-rata *Forwarded Route Request (RREQ F)* hingga 10,39%.

Kata kunci: VANETs, AODV, Node Degree Clustering, Node Tetangga

IMPLEMENTATION OF CLUSTERING METHOD BASED ON NODE DEGREE LEVEL IN AD-HOC ON DEMAND DISTANCE VECTOR (AODV) IN VANETS

Student's Name	**:**	**NAUFAL PRANASETYO FODENSI**
Student's ID	**:**	**05111540000057**
Department	**:**	**Informatics – FTIK ITS**
First Advisor	**:**	**Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.**
Second Advisor	**:**	**Prof. Ir. Supeno Djanali, M.Sc., Ph.D.**

Abstract

In this era, Internet technology can be used as problem solving. An example of its development is ad-hoc wireless networks. For example, the process of determining the route of travel is related to sending data information on the internet network. To do this problem solver can use Ad-Hoc network technology which underlies the creation of Ad-Hoc Vehicle Networks (VANETs). Ad hoc Network Vehicle (VANET) is the development of an Ad hoc Mobile Network (MANET) where nodes have characteristics with very high mobility and are limited to their movement patterns. There are many routing protocols that can be implemented on VANET, one of which is Ad hoc On demand Distance Vector (AODV).VANETs are an improvement of MANET which have high mobility node characteristic and limited movement pattern. There are many routing protocols that can be implemented on VANETs and one of them is AODV.

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

maintenance that consist of Route Error (RERR) is used for finding out an error information in route.

In normal AODV performance, stable route selection is needed in route search process to extend the lifetime of a node. For example, when each node broadcasts a RREQ packet, it requires a long delay and a lot of resources when sending. This affects the level of stability of the route and is susceptible to interruption which results in the package being sent reduced.

In this Final Project implement a clustering algorithm called AODV based on Node Degree Clustering to control the flooding mechanism when building a new route in the AODV protocol. In this algorithm, Cluster Head selection is obtained based on the degree of neighboring node level. Then each node will be classified according to the cluster. So when doing Rebroadcast RREQ, there is no need to send to all nodes, but only through Cluster Head nodes and gateway nodes to connect between Clusters. From the test results, modified AODV in the grid scenario has increased the average value of the Packet Delivery Ratio (PDR) by 13,41%, decreased Routing Overhead value by 17,85%, and the value of Forwarded Route Request (RREQ F) has decreased by 16,36%. While in the real scenario the average value of the Packet Delivery Ratio (PDR) has increased by 13,03% , decreased Routing Overhead value by 11,52%, and the value of Forwarded Route Request (RREQ F) has decreased by 10,39%.

Keyword: VANETs, AODV, Node Degree Clustering, Neighbor Node

KATA PENGANTAR

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

Puji syukur kepada Allah Yang Maha Esa atas segala karunia dan rahmat-Nya penulis dapat menyelesaikan Tugas Akhir yang berjudul **"Implementasi Metode *Clustering* berdasarkan *Node Degree* pada *Ad-hoc On Demand Distance Vector* (AODV) di Lingkungan VANETs"**.

Harapan penulis semoga apa yang tertulis di dalam buku Tugas Akhir ini dapat bermanfaat bagi pengembangan ilmu pengetahuan saat ini dan ke depannya, serta dapat memberikan kontribusi yang nyata.

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

1. Allah SWT atas segala rahmat yang diberikan sehingga penulis dapat menyelesaikan Tugas Akhir ini.
2. Keluarga penulis terutama Bapak dan Ibu selaku orang tua penulis dan adik-adik penulis atas segala dukungan berupa motivasi, doa, moral, dan material sehingga penulis tetap semangat dan dapat menyelesaikan Tugas Akhir ini.
3. Bapak Dr. Eng. Radityo Anggoro, S.Kom., M.Sc., dan Bapak Prof. Ir. Supeno Djanali, M.Sc., Ph.D. selaku dosen pembimbing penulis atas nasihat, arahan dan bantuannya sehingga penulis dapat menyeleksaikan Tugas Akhir ini.
4. Teman-teman dari Penghuni Sekretariat Himpunan x Warkop NF (Ronald Sumbayak, Tegar, Djohan, Ichsan, Aqil, Aidil, Maile, Pentol, Ariya, Illham, Yasin, Faiq, Adam, Himawho, Hilmi, Fatur, Cak To, Adit DC dan Pak Wit) yang selalu memberikan semangat, hiburan, dan menjadi tempat bertukar pikiram dan pendapat serta menemani penulis tidur sehari-hari di ruang himpunan selama penulis berkuliah di Departemen Informatika ITS.

5. Teman-teman dari keluarga besar Laboratorium AJK (Ilham Penyok, Fuad, Didin, Awan, Satriya, Daus, Nahda, dan Hana), Laboratorium MI (Huda, Narendra, Ivan, Adib, Yudhis, Rezky, Unggul, Yola, Bela, Salma, Ajeng, Nafi, Dio, Subhan, Yuga, Zahri, Byan, Adi, GD, dan Azzam), serta teman-teman Informatika angkatan 2015 yang telah menemani, memotivasi, memberikan doa, memberikan hiburan di kala penulis sedang jenuh saat pengerjaan Tugas Akhir ini, serta menyediakan fasilitas selama penulis mengerjakan Tugas Akhir ini.
6. Teman – teman GND ITS (Aisyah, Wafi, Mamad, Thareq, Kiki, Dikky, dan Daniel) yang selalu dapat diajak untuk bertukar pikiran dan pendapat, serta menjadi penghibur penulis sejak SMA hingga saat ini.
7. Teman teman HMTC Inspirasi dan HMTC Kreasi yang sudah memberikan kesibukan lebih untuk penulis semasa kuliah di Informatika ITS.

Penulis telah berusaha sebaik-baiknya dalam menyusun Tugas Akhir ini, namun 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, Mei 2019

NAUFAL PRANASETYO FODENSI

DAFTAR ISI

(Halaman ini sengaja dikosongkan)

DAFTAR GAMBAR

DAFTAR TABEL

(Halaman ini sengaja dikosongkan)

BAB I
PENDAHULUAN

1.1 Latar Belakang

Saat ini, perkembangan teknologi informasi dan komunikasi sudah mengalami kemajuan yang didukung dengan perkembangan dunia internet yang sangat pesat. Teknologi internet saat ini juga digunakan sebagai suatu pemecahan suatu masalah. Seperti contohnya pada permasalahan di jalan raya dengan melakukan pencarian rute tercepat agar sampai tujuan. Proses penentuan rute perjalanan berhubungan dengan rute pengiriman data dalam jaringan internet. Untuk melakukan pemcecahan permasalahan hal tersebut dapat memanfaatkan teknologi jaringan *Ad-Hoc* yang mana mendasari pembuatan *Vehicular Ad-Hoc Networks* (VANETs). *Vehicle Ad hoc Networks* (VANETs) merupakan perkembangan dari *Mobile Ad hoc Network* (MANET) dimana setiap node memiliki mobilitas yang tinggi dan dikhususkan untuk jaringan yang dinamis. [1].

Routing protocol dalam VANET dibedakan menjadi dua model, yaitu *proactive* dan *reactive routing*. *Proactive routing* adalah protokol yang bekerja dengan cara melakukan *update* tabel *routing* setiap saat pada waktu tertentu tanpa memperhatikan beban jaringan, *bandwidth* dan ukuran jaringan. Sedangkan *reactive routing* adalah merupakan mekanisme *routing* yang membentuk tabel *routing* hanya jika ada permintaan pengiriman data atau paket [2].

Salah satu contoh *reactive routing protocol* adalah *routing protocol* AODV. Contoh penggunaan protokol AODV adalah diimplementasikannya protokol tersebut kedalam jaringan sensor nirkabel melalui simulasi VANETs. *Adhoc on-Demand Distance Vector Protocol* merupakan salah satu *routing protocol* reaktif yang dikenal luas, sehingga sudah banyak penelitian untuk yang memodifikasi untuk meningkatkan kinerjanya. Pada kinerja

AODV, pemilihan rute yang stabil dan cepat saat proses pencarian rute diperlukan untuk memperpanjang waktu penggunaan *node*. Seperti contohnya ketika semua node melakukan *rebroadcast* paket RREQ untuk mengetahui rute menuju *node* tujuan, proses *rebroadcast* tersebut membutuhkan waktu lama dan *resource node* yang banyak. Hal tersebut berpengaruh pada lamanya pencarian rute yang stabil sehingga paket yang dikirim mengalami *delay* dan jumlah paket yang seharusnya terkirim berkurang [2].

Perlu adanya mekanisme pemilihan *node* mana saja yang bertugas untuk melakukan *rebroadcast,* maka dari itu penulis mengusulkan suatu algoritma *Node Degree Clustering*. Di dalam algoritma ini, dilakukan pemilihan *Cluster Head* yang didapatkan berdasarkan tingkat derajat *node* tetangga atau *node degree* tertinggi. Yang berarti *Cluster Head* berperan sebagai pemimpin atau perwakilan di lingkungan transmisinya. Maka saat melakukan *rebroadcast* RREQ, hanya *node* dengan *node degree* tertinggi yang berhak melakukan *rebroadcast* RREQ dan *node gateway* yang digunakan untuk mendukung komunikasi antar *cluster,* sehingga *resource node* yang dibutuhkan sedikit serta mempercepat waktu ketika melakukan pencarian rute [3].

Pada Tugas Akhir ini, penulis mengimplementasikan metode *clustering* pada AODV yaitu *AODV based on Node Degree Clustering* untuk mengontrol mekanisme *flooding* saat membangun suatu rute dan mencari rute stabil di protokol AODV. Hasil akhir yang diharapkan adalah mengetahui perbandingan kinerja antara AODV dan AODV yang telah dimodifikasi diukur berdasarkan performansi *Packet Delivery Ratio, End-to-end Delay, Routing Overhead,* dan *Forwarded Route Request.*

1.2 Rumusan Masalah

Tugas Akhir ini mengangkat beberapa rumusan masalah sebagai berikut:

1. Bagaimana melakukan pemilihan *Cluster Head* berdasarkan *node degree* di lingkungan VANETs?

2. Bagaimana peranan *Cluster Head* dalam mereduksi jumlah pengiriman *Rebroadcast Route Request*?
3. Bagaimana peranan *Cluster Head* terhadap performa protocol AODV secara keseluruhan diukur berdasarkan *Packet Delivery Ratio, End-to-end Delay, Routing Overhead,* dan *Forwarded Route Request.*

1.3 Batasan Permasalahan

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

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

1.4 Tujuan

Tujuan dari Tugas Akhir ini adalah sebagai berikut:

1. Melakukan pemilihan *Cluster Head* berdasarkan *node degree* di lingkungan VANETs.
2. Menganalisa peranan *Cluster Head* dalam mereduksi jumlah pengiriman *Rebroadcast Route Request (RE-RREQ) message*.
3. Menganalisa performa AODV yang telah dimodifikasi berdasarkan matriks *Packet Delivery Ratio, End-to-end Delay, Routing Overhead, Forwarded Route Request.*.

1.5 Manfaat

Manfaat yang diperoleh dari pengerjaan Tugas Akhir ini adalah dapat memberikan informasi tentang dampak dari modifikasi dengan melalukan pemilihan *Cluster Head* berdasarkan

node degree serta peran *gateway* terhadap kinerja *routing protocol* AODV di lingkungan VANETs.

1.6 Metodologi

Pembuatan Tugas Akhir ini dilakukan dengan menggunakan metodologi sebagai berikut:

1.6.1 Penyusunan Proposal Tugas Akhir

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

1.6.2 Studi Literatur

Pada tahap ini, dipelajari sejumlah referensi yang diperlukan dalam melakukan implementasi yaitu mengenai VANETs, AODV, *Network Simulator* NS2, OpenStreetMap, Java OpenStreetMap (JOSM), SUMO, dan AWK.

1.6.3 Analisis dan Desain Sistem

Pada tahap ini dilakukan analisis dari hasil percobaan modifikasi AODV yang dibuat. Data yang dianalis berasal dari perhitungan *Packet Delivery Ratio, Routing Overhead, Forwarded Route Request*, dan *End-to-End Delay* paket dari *node* ke *node* lainnya. Hal ini bertujuan untuk merumuskan solusi yang tepat untuk konfigurasi AODV yang dimodifikasi dalam lingkungan topologi MANET. Setelah selesai diaplikasikan pada MANET,

dilakukan simulasi yang dilakukan pada VANETs dengan bantuan SUMO.

1.6.4 Implementasi Sistem

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

1.6.5 Pengujian dan Evaluasi

Pada tahap ini dilakukan pengujian menggunakan SUMO, sebuah *traffic generator* untuk membuat simulasi keadaan topologi yang diujikan. Hasil dari SUMO tersebut akan dijalankan pada NS-2 yang akan menghasilkan *trace file*. *Packet Delivery Ratio, End-to-end Delay, Routing Overhead,* dan *Forwarded Route Request.*akan dihitung dari *trace file* tersebut untuk menguji performa AODV yang telah dimodifikasi.

1.6.6 Penyusunan Buku

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

1.7 Sistematika Penulisan Laporan

Sistematika penulisan laporan Tugas Akhir adalah sebagai berikut:

1. Bab I. Pendahuluan

Bab ini berisikan penjelasan mengenai latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, metodologi, dan sistematika penulisan dari pembuatan Tugas Akhir.

2. Bab II. Tinjauan Pustaka
   Bab ini berisi kajian teori atau penjelasan dari metode, algoritma, *library*, dan *tools* yang digunakan dalam penyusunan Tugas Akhir ini. Bab ini berisi tentang penjelasan singkat mengenai VANETs, AODV, NS2, OpenStreetMap, Java OpenStreetMap (JOSM), SUMO, dan AWK.
3. Bab III. Perancangan
   Bab ini berisi pembahasan mengenai perancangan skenario mobilitas *grid* dan *real*, perancangan simulasi pada NS2, perancangan modifikasi AODV, serta perancangan metrik analisis (*Packet Delivery Ratio, End-to-end Delay, Routing Overhead,* dan *Forwarded Route Request.*).
4. Bab IV. Implementasi
   Bab ini menjelaskan implementasi yang berbentuk kode sumber dari proses modifikasi protokol AODV, pembuatan simulasi pada NS2, SUMO, dan perhitungan metrik analisis.
5. Bab V. Uji Coba dan Evaluasi
   Bab ini berisikan hasil uji coba dan evaluasi dari implementasi yang telah dilakukan untuk menyelesaikan masalah yang dibahas pada Tugas Akhir.
6. Bab VI. Kesimpulan dan Saran
   Bab ini merupakan bab yang menyampaikan kesimpulan dari hasil uji coba yang dilakukan, masalah-masalah yang dialami pada proses pengerjaan Tugas Akhir, dan saran untuk pengembangan solusi ke depannya.
7. Daftar Pustaka
   Bab ini berisi daftar pustaka yang dijadikan literatur dalam Tugas Akhir.
8. Lampiran
   Dalam lampiran terdapat tabel-tabel data hasil uji coba dan kode sumber program secara keseluruhan.

BAB II
TINJAUAN PUSTAKA

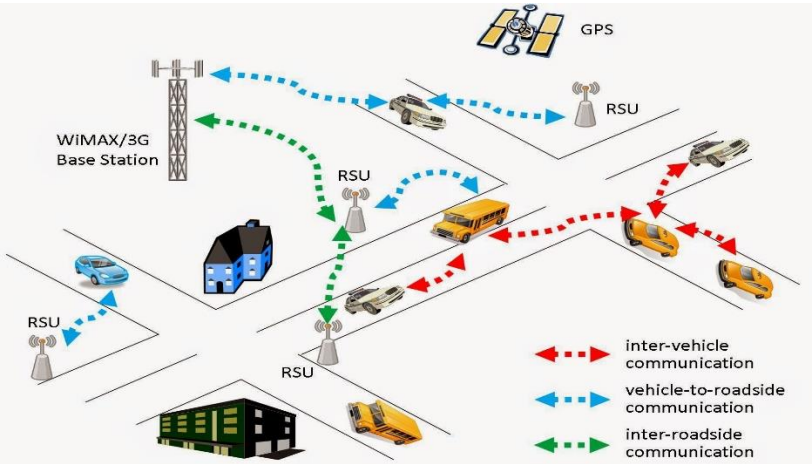
Bab ini berisi pembahasan mengenai teori-teori dasar atau penjelasan dari metode dan *tools* yang digunakan dalam Tugas Akhir.

2.1 VANETs

Vehicular Ad-hoc Network (VANETs) merupakan sekumpulan *node* yang bergerak secara dinamis (*mobile*) sebagai pengembangan dari *Mobile Ad Hoc Network* (MANET) yang mempertimbangkan semua kendaraan dalam jaringan sebagai node tersebut untuk berkomunikasi dengan kendaraan lainnya pada radius tertentu [2]. Pada VANET maupun MANET, *node* yang bergerak tergantung kepada *ad hoc routing protocol* untuk menentukan bagaimana proses pengiriman data dari node asal ke node tujuan. Meskipun menggunakan *routing protocol* yang sama, namun VANET memiliki karakter yang berbeda dengan MANET karena VANET memiliki batasan pergerakan dan kecepatan yang tinggi yang menciptakan keunikan karakteristik dari VANET. VANET merupakan pengembangan dari *Mobile Ad-hoc Network* (MANET) dimana pengembangannya difokuskan pada kendaraan (*vehicle*) dan *infrastructure* yang dapat saling berkomunikasi maupun mengirimkan data sebagai pengembangan *Intelligent Transport System* (ITS) dengan tujuan meningkatkan keselamatan dan kenyamanan berkendara. Komunikasi Wireless ini meliputi komunikasi *Inter-Vehicle Communication* (IVC),*Vehicle to Roadside* (V2R), atau *Roadside to Roadside* (R2R)..

VANETs adalah sebuah teknologi baru yang memadukan kemampuan komunikasi nirkabel kendaraan menjadi sebuah jaringan yang bebas infrastuktur serta memiliki karakteristik mobilitas yang sangat tinggi dan terbatas pada pola pergerakannya. *Node* dalam jaringan dianggap sebagai *router* yang bebas bergerak dan bebas menentukan baik menjadi *client* maupun menjadi *router*. Protokol *routing* pada VANETs memiliki dua model yaitu protokol *reactive*

routing yang membentuk tabel *routing* hanya saat dibutuhkan dan protokol *proactive routing* yang melakukan pemeliharaan tabel *routing* secara berkala pada waktu tertentu. Pergerakan *node* pada VANETs bisa berubah setiap saat dan terbatas pada rute lalu lintas yang dapat ditentukan dari koordinat peta. Hal ini membuat setiap *node* akan terus memperbarui informasi dalam tabelnya sesuai informasi dari *node* lain. Perubahaan pergerakan pada VANETs menjadi salah satu permasalahan dalam pengiriman paket data sehingga dibutuhkan informasi jarak antar *node*, kecepatan dan *delay* transmisi [5]. Ilustrasi VANETs dapat dilihat pada Gambar 2.1.



Gambar 2.1 Ilustrasi VANETs *[2]*

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

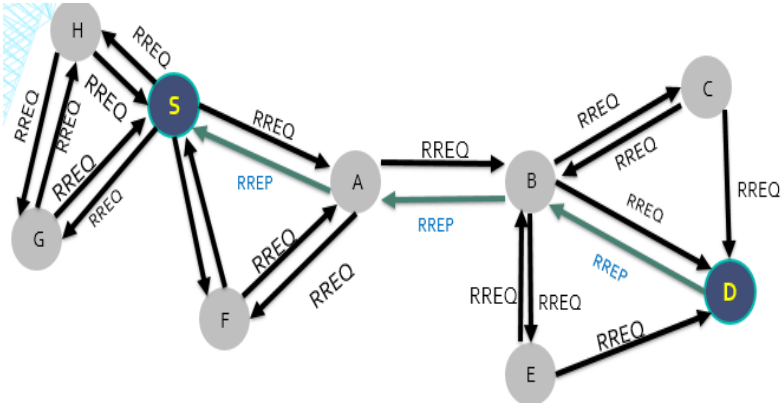
2.2 *Ad-hoc On demand Distance Vector* (AODV)

Ad-hoc On demand Distance Vector (AODV) adalah salah satu *routing* protokol yang dirancang untuk jaringan *ad-hoc mobile* dan

termasuk dalam klasifikasi *reactive routing protocol*. Dimana merupakan algoritman *routing* permintaan, yang berarti sebuah protokol yang hanya membuat sebuah rute antara node hanya saat dibutuhkan. AODV menggunakan table routing satu *entry* untuk setiap tujuan, tanpa menggunakan routing.

Ada dua tahapan dalam AODV yaitu *route discovery* dan r*oute maintenance. Route discovery* memiliki dua pesan yaitu berupa *Route Request* (RREQ) dan *Route Reply* (RREP). Sedangkan *Route maintenance* berupa *Route Error* (RERR). AODV mempercayakan pada tabel *routing* untuk menyebarkan *Route Reply* kembali ke sumber dan mengarahkan paket menuju tujuan. Ciri utama dari AODV adalah menjaga *timer-based state* pada setiap *node* sesuai dengan penggunaan tabel *routing*.

AODV adalah sebuah metode *routing* pesan antar *node* yang memungkinkan *node-node* tersebut untuk melewatkan pesan melalui lingkungannya ke *node* yang tidak dapat dihubungi secara langsung. AODV melakukan ini dengan cara menemukan rute yang bisa dilalui oleh pesan. Selain itu AODV juga memastikan rute ini tidak mengandung perulangan (*loop*), menangani perubahan rute, dan membuat rute baru apabila terjadi *error* [6]. Ilustrasi pencarian rute oleh AODV dapat dilihat pada Gambar 2.2. Struktur paket RREQ dapat dilihat pada Tabel 2.1.



Gambar 2.2 Ilustrasi pencarian rute *routing protocol* AODV *[3]*

Tabel 2.1 Struktur Paket RREQ

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

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

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

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

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

Network Simulator 2 (NS-2) merupakan sebuah network simulator yang dibuat dengan tujuan riset dan pendidikan. Pada awalnya, NS dibangun sebagai varian dari *Real Network Simulator* pada tahun 1989 di University of California Berkeley dan USC ISI sebagai bagian dari projek *Virtual INternet Testbed* (VINT). NS yang banyak dikenal dengan NS-2 (versi 2) menjadi salah satu *tool* yang sangat berguna untuk menunjukkan simulasi jaringan melibatkan *Local Area Network* , *Wide Area Network* (WAN), dan telah berkembang selama beberapa tahun belakangan untuk memasukkan jaringan nirkabel (*wireless*) dan juga jaringan *ad hoc.* NS-2 memiliki beberapa fitur kelebihan yang dapat dimanfaatkan dalam pemodelan dan pengujian VANET [7].

Pada Tugas Akhir ini, NS-2 digunakan untuk melakukan simulasi lingkungan VANETs menggunakan protokol AODV yang sudah dimodifikasi. *Trace file* yang dihasilkan oleh NS-2 untuk mengukur peforma *routing* protokol AODV yang dimodifikasi..

2.3.1 Instalasi

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

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

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

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

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

Gambar 2.4 Baris kode yang diubah pada file ls.h

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

2.3.2 *Trace File*

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

Tabel 2.2 Detail Penjelasan Trace File AODV

Kolom ke-	Penjelasan	Isi
1	*Event*	s : *sent* r : *received* f : *forwarded* D : *dropped*
2	*Time*	Waktu terjadinya *event*
3	ID *Node*	_x_ : dari 0 hingga banyak *node* pada topologi
4	*Layer*	AGT : *application* RTR : *routing* LL : *link layer* IFQ : *packet queue* MAC : *MAC* PHY : *physical*
5	*Flag*	--- : Tidak ada
6	*Sequence Number*	Nomor paket
7	*Packet Type*	AODV : paket *routing* AODV cbr : berkas paket CBR (*Constant Bit Rate*) RTS : *Request To Send* yang dihasilkan MAC 802.11 CTS : *Clear To Send* yang dihasilkan MAC 802.11 ACK : MAC ACK

		ARP : Paket *link layer address resolution protocol*
8	Ukuran	Ukuran paket pada *layer* saat itu
9	Detail MAC	[a b c d] a : perkiraan waktu paket b : alamat penerima c : alamat penerima d : IP header
10	*Flag*	-------- : Tidak ada
11	*Detail* IP *source, destination,* dan *nexthop*	[a:b c:d e f] a : IP *source node* b : *port source node* c : IP *destination node* (jika -1 berarti *broadcast*) d : *port destination node* e : IP *header* ttl f : IP *nexthop* (jika 0 berarti *node* 0 atau *broadcast*)

2.4 OpenStreetMap

OpenStreetMap (OSM) adalah sebuah proyek kolaboratif berbasis web untuk membuat dan membangun peta geografis seluruh dunia yang gratis dan terbuka, dibangun sepenuhnya oleh sukarelawan dengan melakukan survei dan mengumpulkan data menggunakan perangkat GPS, fotografi udara, yang dimanfaatkan untuk beragam kebutuhan termasuk navigasi. Kontributor OSM dapat memiliki, memodifikasi, dan membagikan data peta secara luas. Terdapat beragam jenis peta digital yang tersedia di internet, namun sebagian besar memiliki keterbatasan secara legal maupun teknis. Hal ini membuat masyarakat, pemerintah, peneliti dan akademisi, *inovator*, dan banyak pihak lainnya tidak dapat menggunakan data yang tersedia di dalam peta tersebut secara luas. Di sisi lain, baik peta dasar OSM maupun data yang tersedia di dalamnya dapat diunduh

secara gratis dan terbuka, untuk kemudian digunakan untuk diredistribusikan kembali.

OSM dapat menjadi jawaban di banyak tempat seperti ini, baik itu pengembangan ekonomi, tata kota, kontinjensi bencana, maupun untuk berbagai tujuan lainnya [8].

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

2.5 *Java OpenStreetMap Editor* (JOSM)

Java OpenStreetMap Editor (JOSM) adalah aplikasi desktop bebrbasis java dan dapat dioperasikan pada system operasi seperti Windows, MacOS, dan Linux. JOSM adalah alat penyunting data yang didapatkan dari OpenStreetMap [9].

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

2.6 *Simulation of Urban Mobility* (SUMO)

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

SUMO juga memungkinkan untuk mendefinisikan kendaraan dengan sifat tertentu seperti panjang kendaraan, kecepatan maksimum, percepatan dan perlambatannya. SUMO juga menyediakan pilihan bagi pengguna menentukan rute acak untuk kendaraan. Ada juga pilihan yang tersedia untuk model sistem

transportasi umum, dimana setiap kendaraan datang dan berangkat sesuai dengan jadwal [10].

SUMO terdiri dari beberapa *tools* yang dapat membantu pembuatan simulasi lalu lintas pada tahap-tahap yang berbeda. Berikut penjelasan fungsi *tools* yang digunakan dalam pembuatan Tugas Akhir ini:

- netgenerate
  netgenerate merupakan *tool* yang berfungsi untuk membuat peta berbentuk seperti *grid*, *spider*, dan bahkan *random network*. Sebelum proses netgenerate, pengguna dapat menentukan kecepatan maksimum jalan dan membuat *traffic light* pada peta. Hasil dari netgenerate ini berupa *file* dengan ekstensi .net.xml. Pada Tugas Akhir ini netgenerate digunakan untuk membuat peta skenario *grid*.
- netconvert
  netconvert merupakan program CLI yang berfungsi untuk melakukan konversi dari peta seperti OpenStreetMap menjadi format *native* SUMO. Pada Tugas Akhir ini penulis menggunakan netconvert untuk mengonversi peta dari OpenStreetMap.
- randomTrips.py
  Tool dalam SUMO untuk membuat rute acak yang akan dilalui oleh kendaraan dalam simulasi.
- duarouter
  Tool dalam SUMO untuk melakukan perhitungan rute berdasarkan definisi yang diberikan dan memperbaiki kerusakan rute.
- sumo
  Program yang melakukan simulasi lalu lintas berdasarkan data-data yang didapatkan dari netgenerate (skenario *grid*) atau netconvert dari randomTrips.py. Hasil simulasi dapat di-*export* ke sebuah *file* untuk dikonversi menjadi format lain.
- sumo-gui

GUI untuk melihat simulasi yang dilakukan oleh SUMO secara grafis.

- traceExporter.py
  Tool yang bertujuan untuk mengonversi *output* dari sumo menjadi format yang dapat digunakan pada *simulator* lain. Pada Tugas Akhir ini penulis menggunakan traceExporter.py untuk mengonversi data menjadi format .tcl yang dapat digunakan pada NS-2

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

2.7 AWK

AWK adalah bahasa pemrograman yang digunakan untuk melakukan *text processing* dan ekstraksi data [11]. AWK merupakan sebuah program filter untuk teks, seperti halnya perintah grep pada terminal linux. AWK dapat digunakan untuk mencari bentuk / model dalam sebuah berkas teks ke dalam bentuk teks lain. AWK dapat juga digunakan untuk melakukan proses aritmatika seperti yang dilakukan oleh perintah expr. AWK sama halnya seperti bahasa shell atau C yang memiliki karakteristik yaitu sebagai *tool* yang cocok untuk *jobs* juga sebagai pelengkap untuk *filter* standar.

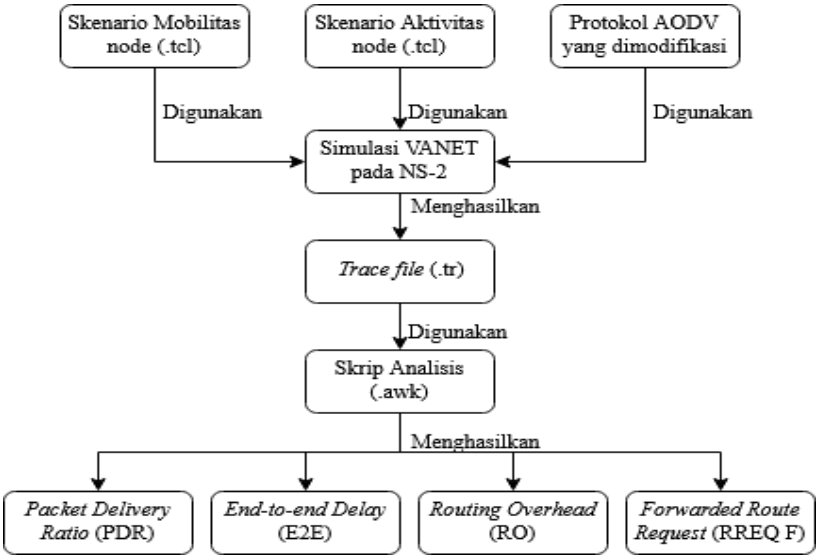
Pada Tugas Akhir ini, AWK digunakan untuk membuat script menghitung *Packet Delivery Ratio (*PDR*), End-to-end Delay*, *Routing Overhead* (RO), dan *Forwarded Route Request* (RREQ F) dari *trace file* NS2.

BAB III
PERANCANGAN

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

3.1 Deskripsi Umum

Pada Tugas Akhir ini akan diimplementasikan *routing protocol* AODV dengan memodifikasi pada bagian proses *route discovery* yang dijalankan pada simulator NS-2. Diagram dari rancangan simulasi dari AODV asli dan AODV modifikasi dapat dilihat pada Gambar 3.1.



Gambar 3.1 Diagram Alur Rancangan Simulasi

Modifikasi akan diawali dengan pencarian jumlah tetangga setiap *node* perantara (*one-hop node)*. Jumlah tetangga tiap *node* akan didapatkan dengan menggunakan HELLO *messages* yang terdapat pada AODV. Setelah jumlah tetangga setiap *node* didapatkan, maka modifikasi dilanjutkan untuk melakukan perhitungan nilai *threshold*. Hal tersebut dilakukan dengan cara menyeleksi *node – node* mana saja yang mempunyai jumlah tetangga sama dengan *threshold* yang sudah ditentukan. Jika *node* tersebut mempunyai jumlah tetangga sama dengan *threshold*, maka *node* tersebut akan menjadi *node cluster head,* lalu akan mengirimkan paket yang berisi ID nya ke seluruh node tetangganya. Dan ketika node tetangga menerima pesan broadcast yang berupa ID dari cluster head, node tetangga tersebut akan membalas untuk bergabung dengan *cluster* untuk menjadi *node Gateway*. *Gateway* dilakukan untuk menjamin komunikasi antar *cluster*.

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

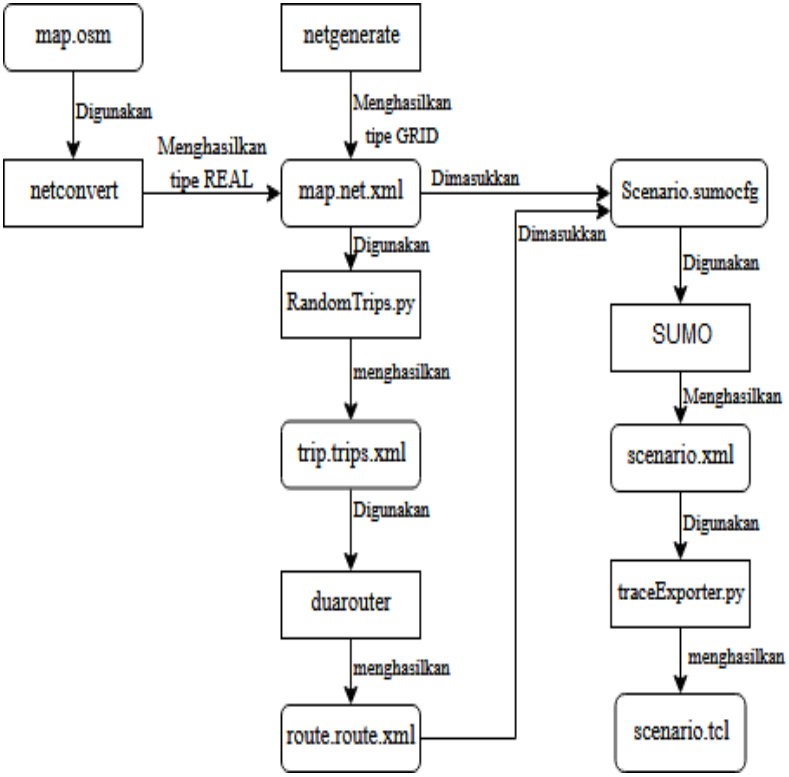
Tabel 3.1 Daftar Istilah

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

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

3.2 Perancangan Skenario Mobilitas

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



Gambar 3.2 Alur perancangan skenario *Grid* dan *Real*

3.2.1 Perancangan Skenario *Grid*

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

tiap petak sehingga mendapatkan luas area yang dibutuhkan yaitu berukuran 700 m x 700 m. Dengan 4 titik persimpangan, maka akan didapatkan 9 petak dan panjang tiap petak adalah 200 m.

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

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

3.2.2 Perancangan Skenario *Real*

Perancangan skenario mobilitas *real* diawali dengan memilih area yang akan dijadikan simulasi. Pada Tugas Akhir ini, digunakan peta dari OpenStreetMap untuk mengambil area yang dijadikan model simulasi. Setelah memilih area, dilakukan pengunduhan dengan menggunakan fitur *export* yang telah disediakan oleh OpenStreetMap. Peta hasil *export* tersebut memiliki ekstensi .osm.

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

dihasilkan tersebut dikonversi ke dalam bentuk *file* dengan ekstensi .tcl agar dapat diterapkan pada NS-2.

3.3 Perancangan Modifikasi *Routing Protocol* AODV

Protokol AODV yang diajukan pada Tugas Akhir ini merupakan modifikasi dari protokol AODV yang mengubah mekanisme *route discovery* pada protokol tersebut. Pada protokol AODV, mekanisme pencarian *node* untuk pengiriman ulang (*rebroadcast*) paket RREQ langsung dikirim begitu saja, maka pada AODV yang dimodifikasi ini akan ada proses penyeleksian *node*. Seleksi *node* dilakukan dengan cara *node* sumber mengetahui jumlah tetangga yang dimiliki *node* perantara melalui pengiriman *hello message*. Sebuah *node* akan menyimpan informasi berupa jumlah tetangganya dan *threshold* untuk dijadikan kandidat *Cluster Head* nantinya. Setelah mengetahui jumlah tetangga yang dimiliki setiap node maka terdapat fungsi yang berjalan untuk menghitung *threshold* berdasarkan jumlah tetangga tiap node.

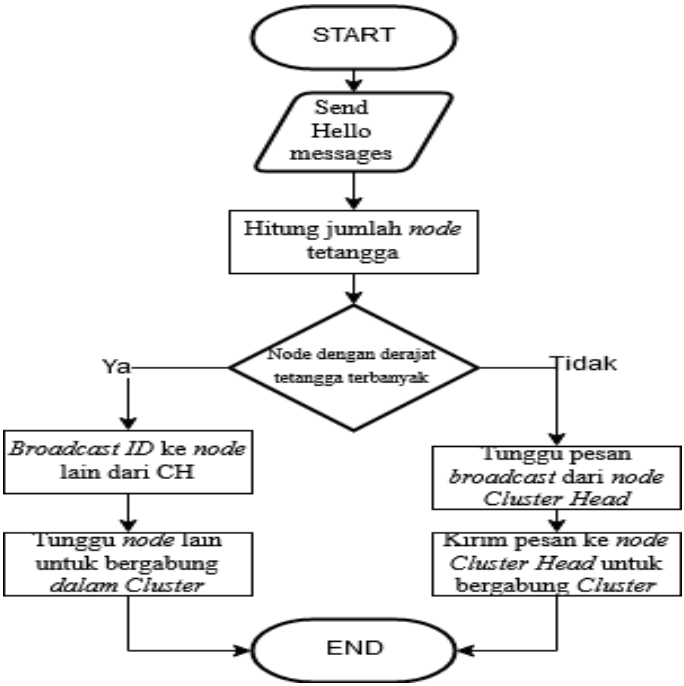
Selanjutnya, dilakukan perbandingan menggunakan *threshold*. Jika *node* tersebut mempunyai jumlah tetangga lebih dari *threshold*, maka jumlah tetangga terbanyak disimpan dalam variable *max_degree* sebagai batas untuk menjadi *node cluster head*, dan *node Cluster Head* akan mengirimkan pesan yang berisi ID nya ke seluruh node tetangganya. Dan ketika node tetangga menerima pesan yang berupa ID dari *cluster head*, node tetangga tersebut akan membalas dan bergabung dengan *cluster* untuk menjadi *node Gateway*. *Gateway* digunakan untuk menjamin komunikasi antar *cluster*. Untuk itu, dilakukan modifikasi *hello message* yang nantinya akan disimpan juga pada *entry Routing Table* sehingga mengubah struktur pada Tabel *Routing* dengan menambahkan *field NeighbourCount*, CH_Index, dan *GatewayNode* yang dapat dilihat pada Tabel 3.2.

Tabel 3.2 Modifikasi Struktur *Routing Table* AODV *Clusterng*

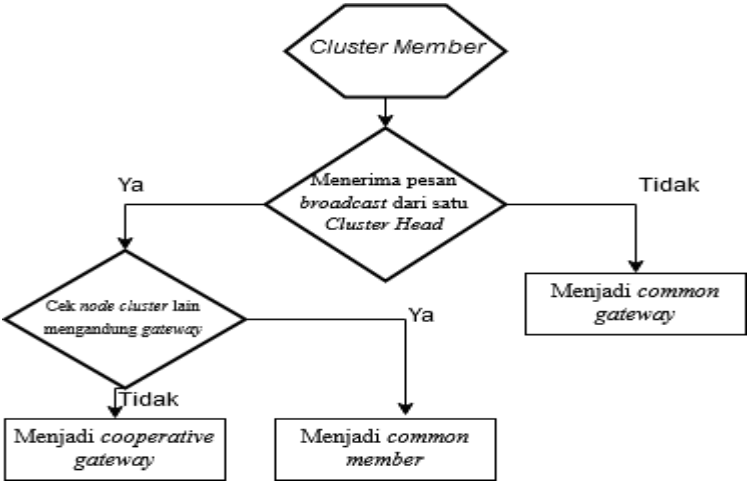
Dst IP Address	*DstSequenceNumber*	*HopCount*	*NextHop*
Lifetime	***NeighbourCount***	***CH_Index***	***GatewayNode***

Ketika pesan RREQ sudah mencapai *node* tujuan, *node* tersebut akan mengembalikannya dengan paket RREP sesuai *dengan reverse route* yang telah dibuat. Ketika paket RREP diterima, maka *node* sumber akan memulai pengiriman paket melewati *node intermediate* sesuai jalur *reverse path* nya, atau dalam hal ini, berarti hanya melewati *node Cluster Head* dan *node Gateway* sebagai node perantara antar *Cluster*.

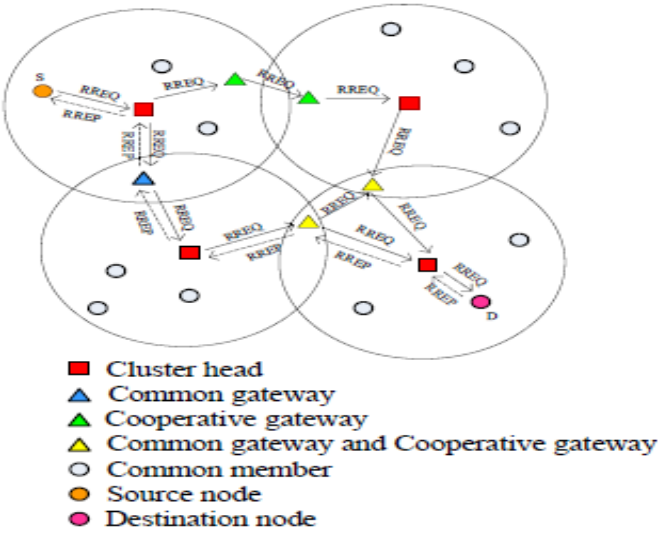
Rute tersebut tidak melakukan *rebroadcast* RREQ ke semua *node* dan, melainkan memaksa pengiriman paket hanya melewati *node – node gateway* antar *cluster* untuk sampai ke *node* destinasi. Untuk Diagram rancangan modifikasi AODV dapat dilihat pada Gambar 3.3 dan Gambar 3.4



Gambar 3.3 Alur pencarian *node Cluster Head*



Gambar 3.4 Alur penentuan *node gateway*



Gambar 3.5 Ilustrasi Rute Modifikasi AODV-*Clustering [3]*

3.3.1 Perancangan Penghitungan *Node Degree*

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

3.3.2 Perancangan Penghitungan *Threshold Cluster Head*

Pada gambar 3.3 terdapat alur dalam melakukan pencarian *Cluster Head. Threshold* akan ditentukan melalui perhitungan nilai tertinggi dari jumlah tetangga tiap *node*. *Node* dengan jumlah *node degree* tertinggi di radius transmisinya akan menjadi *node Cluster Head*. Jumlah tetangga tiap node disimpan dalam bentuk *array* sehingga diperlukan proses *sorting array* terlebih dahulu. Setelah array di sorting, maka akan dicari nilai tertingginya dan disimpan pada variable *max_degree*. Digunakan fungsi *count_neighbour* yang merupakan variable global untuk melakukan perhitungan jumlah tetangga tiap node. Dan fungsi *max_degree* untuk melakukan perbandingan batas minimal tetangga yang dimiliki tiap *node* agar dapat menjadi kandidat *Cluster Head*. *Pseudocode* untuk perhitungan *threshold* dapat dilihat pada Gambar 3.6.

```
for i=0 to node_count do
 for j=i+1 to node_count do
  if count_neighbour[i]>max_degree
   max_degree=count_neighbour[i]
  endif
  if count_neighbour[i]>count_neighbour[j]
  then
   tmp=count_neighbour[i]
   count_neighbour[i]= count_neighbour[j]
   count_neighbour[j]=tmp
  endif
 endfor
endfor
for i=0 to node_count do
 for j=0 to node_count do
  if count_neighbour[i]==count_neighbour[j]
  then count_mode[i}++
  endif
  if count_mode[i]>threshold then
  threshold=count_mode[i]
  endif
 endfor
endfor
```

Gambar 3.6 Pseudocode Perhitungan *Threshold Cluster Head*

3.3.3 Perancangan Pemilihan *Forwarding Node*

Setelah melakukan penghitungan jumlah *node* tetangga, akan dilakukan pemilihan *forwarding node*, yaitu *node* yang berhak untuk melakukan *rebroadcast* paket RREQ. Penentuan *forwarding node* dilakukan dengan cara membandingkan jumlah *node* tetangga dengan *threshold* atau batas nilai yang sudah ditentukan. Apabila *node cluster head* belum ditemukan, maka tetap lakukan pencarian saat *hello message*. Apabila *node* tersebut adalah *node* dengan *node degree* tertinggi (*Cluster Head)* ataupun *node gateway*, maka *node* tersebut berhak melakukan *rebroadcast*, jika sebaliknya, maka paket RREQ F akan di-*drop*. Jika *node* biasa menerima paket RREQ dari *node*

cluster head dan akan mengirim paket RREQ selanjutnya ke *node* biasa, maka *node* tersebut akan menjadi *cooperative gateway* yang bertugas untuk menjamin komunikasi antara dua *cluster* yang berbeda. Langkah tersebut dilakukan untuk mengurangi jumlah paket RREQ yang dikirim sehingga dapat mengurangi kemacetan yang terjadi pada jaringan. *Pseudocode* untuk pemilihan *forwarding node Cluster Head* dapat dilihat pada Gambar 3.7.

```
if node_count < threshold
      drop RREQ packet
endif
if CH_ID= -1
      calculate CH_ID
endif
for i=0 to node_count do
 for j=i+1 to node_count do
  if count_neighbour[i]>max_degree[j]
  then
   tmp= max_degree[i]
   count_neighbour[i]= max_degree[j]
   max_degree[j]=tmp
  endif
  then max_degree= cluster_head_index[j]
  rreq->rreq_broadcast= cluster_head_index
 endfor
endfor
```

Gambar 3.7 Pseudocode Pemilihan *forwarding node gateway*

3.4 Perancangan Simulasi pada NS-2

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

Kode yang diubah diantaranya adalah pencarian jumlah *node* tetangga pada *file* node.cc dan perbandingan dengan *threshold* pada

file aodv.cc. Pada saat simulasi NS-2 dijalankan, maka *routing protocol* AODV akan menyeleksi *node* mana saja yang berhak melakukan *rebroadcast* paket RREQ.

3.5 Perancangan Metrik Analisis

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

3.5.1 *Packet Delivery Ratio* (PDR)

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

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

Keterangan:

PDR = *Packet Delivery Ratio*

received = banyak paket data yang diterima

sent = banyak paket data yang dikirimkan

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

Average End-to-End Delay dihitung berdasarkan rata-rata *delay* antara waktu paket data diterima dan waktu paket dikirimkan dalam satuan detik. Delay tiap paket didapat dari rentang waktu antara

node asal saat mengirimkan paket dan *node* tujuan menerima paket. Delay tiap paket tersebut semua dijumlahkan dan dibagi dengan jumlah paket yang berhasil diterima, maka akan didapatkan rata – rata E2E, yang dapat dihitung dengan persamaan 3.2.

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

Keterangan:

E2E = *End-to-End Delay*

CBRRecvTime = Waktu *node* asal mengirimkan paket

CBRSentTime = Waktu *node* tujuan menerima paket

recvnum = Jumlah paket yang berhasil diterima

3.5.3 *Routing Overhead* (RO)

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

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

Keterangan:

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

sentnum = Jumlah total control paket yang terkirim

3.5.4 *Forwarded Route Request* (RREQ F)

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

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

Keterangan:

rreqsent = Jumlah *forwarded route request* yang terkirim

n = Jumlah paket yang dikirimkan

\

BAB IV
IMPLEMENTASI

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

4.1 Implementasi Skenario Mobilitas

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

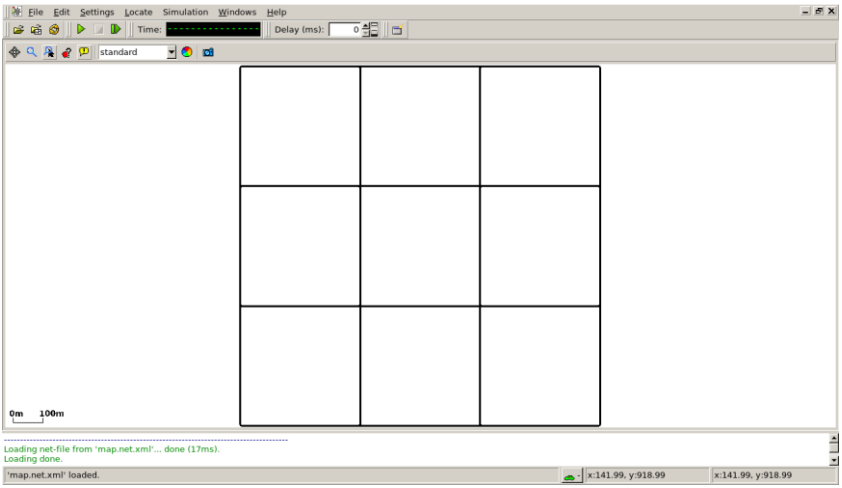
4.1.1 Skenario *Grid*

Dalam mengimplementasikan skenario *grid*, SUMO menyediakan *tools* untuk membuat peta *grid* yaitu netgenerate. Pada Tugas Akhir ini, penulis membuat peta *grid* dengan luas 700 m x 700 m yang terdiri dari titik persimpangan antara jalan vertikal dan jalan horisontal sebanyak 4 titik x 4 titik. Dengan jumlah titik persimpangan sebanyak 4 titik tersebut, makat terbentuk 9 buah petak. Sehingga untuk mencapai luas area sebesar 700 m x 700 m dibutuhkan luas per petak sebesar 200 m x 200 m. Berikut perintah netgenerate untuk membuat peta tersebut dengan kecepatan *default* kendaraan sebesar 20 m/s dapat dilihat pada Gambar 4.1.

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

Gambar 4.1 Perintah netgenerate

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



Gambar 4.2 Hasil *Generate* Peta *Grid*

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

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

Gambar 4.3 Perintah randomTrips

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

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

Gambar 4.4 Perintah duarouter

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

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

Gambar 4.5 File Skrip .sumocfg

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

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

Gambar 4.6 Perintah SUMO untuk membuat skenario .xml

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

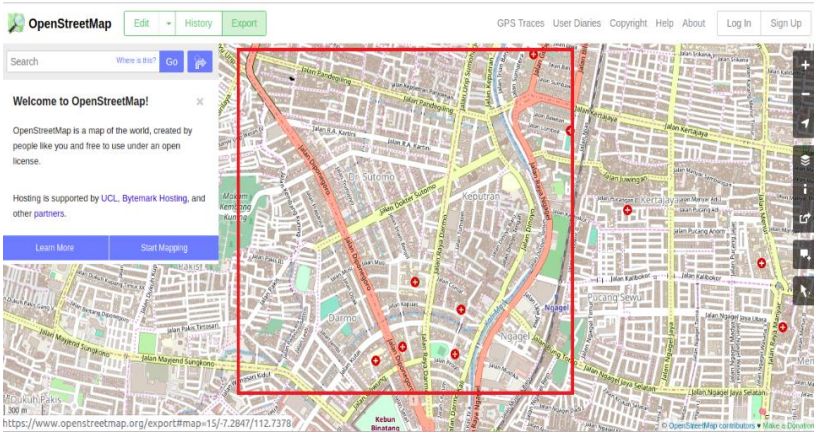
ini adalah traceExporter. Perintah untuk menggunakan traceExporter dapat dilihat pada Gambar 4.7.

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

Gambar 4.7 Perintah traceExporter

4.1.2 Skenario *Real*

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



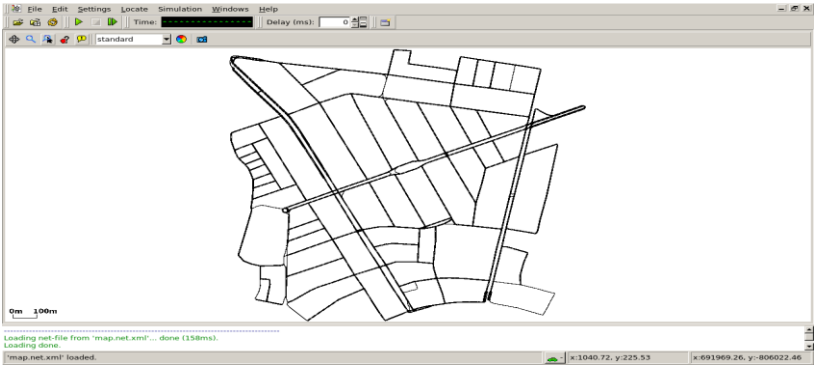
Gambar 4.8 Ekspor Peta dari OpenStreetMap

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

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

Gambar 4.9 Perintah netconvert

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



Gambar 4.10 Hasil Konversi Peta *Real*

Langkah selanjutnya sama dengan ketika membuat skenario mobilitas *grid*, yaitu membuat *node* asal dan *node* tujuan menggunakan *tool* randomTrips. Lalu membuat rute *node* untuk sampai ke tujuan menggunakan *tool* duarouter. Kemudian membuat *file* skenario berekstensi .xml menggunakan *tool* SUMO dengan bantuan *file* skrip berekstensi .sumocfg. Selanjutnya dilakukan konversi *file* skenario berekstensi .tcl untuk dapat disimulasikan pada NS-2 menggunakan *tool* traceExporter. Perintah untuk menggunakan *tools* tersebut sama dengan ketika membuat skenario *grid* di atas.

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

Pada Tugas Akhir ini dilakukan modifikasi pada *routing protocol* AODV agar dapat mengurangi jumlah *forwarding node*, yaitu *node* yang bertugas untuk melakukan *rebroadcast* paket RREQ. Hal tersebut dilakukan dengan cara memilih *forwarding node* berdasarkan *clustering* dengan cara membuat *node cluster head* yang ditentukan berdasarkan perhitungan *threshold* jumlah tetangga *node* tersebut dan dihitung berdasarkan pasangan *node* pada scenario yang bersifat statis, sehingga dapat dilihat peningkatan performa pada *routing* AODV yang telah dimodifikasi.

Implementasi modifikasi *routing protocol* AODV ini dibagi menjadi 4 bagian yaitu:

- Implementasi Penghitungan Jumlah *Node* Tetangga
- Implementasi Penghitungan *Threshold Cluster Head*
- Implementasi Pemilihan *Node Cluster Head*
- Implementasi *Forwarding Node* melalui *Gateway*

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

4.2.1 Implementasi Penghitungan *Node Degree*

Langkah yang dilakukan untuk menghitung jumlah tetangga seperti yang telah dirancang pada subbab 3.3.1 adalah dengan cara menangkap pesan HELLO *messages* ke semua *node* dari *node* yang mengirimkannya.

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

```
1. void AODV::nb_insert(nsaddr_t id)
2. {
3.   count_neighbour[index]+=1; // add neighbor
   node
4. double now = Scheduler::instance().clock();

5.
6.   AODV_Neighbor *nb = new AODV_Neighbor(id);

7.   assert(nb);
8.   nb->nb_expire = CURRENT_TIME +
9.                   (1.5 * ALLOWED_HELLO_LOSS *
    HELLO_INTERVAL);
10.   LIST_INSERT_HEAD(&nbhead,nb,nb_link);
11.     seqno += 2;
12.     assert((seqno % 2) == 0);
13.   }
```

```
14.   void AODV::nb_delete(nsaddr_t id)
15.   {
16.     count_neighbour[index]-=1;
17.
18.     AODV_Neighbor *nb = nbhead.lh_first;
19.     log_link_del(id);
20.     seqno += 2; // Set of neighbors changed

21.     assert((seqno % 2) == 0);
22.     for (; nb; nb = nb->nb_link.le_next)
23.     {
24.       if (nb->nb_addr == id)
25.       {
26.         LIST_REMOVE(nb, nb_link);
27.         delete nb;
28.         break;
29.       }
30.     }
31.     handle_link_failure(id);
32.   }
```

Gambar 4.11 Potongan Kode Modifikasi Fungsi nb_insert()

4.2.2 Implementasi Perhitungan *Threshold Cluster Head*

Pada tahap selanjutnya setelah dapat mengetahui jumlah tetangga, langkah yang harus dilakukan selanjutnya adalah menghitung *threshold* secara adaptif sebagai acuan untuk digunakan sebagai batas acuan agar suatu *node* dapat dikatakan sebagai *Cluster Head*.

Penghitungan ini dilakukan pada fungsi `count_neighbour()` dan `nodes_count` yang terletak pada kode sumber aodv.cc yang terletak pada ns2.35/aodv. Jumlah tetangga yang sudah ditemukan, disimpan dalam variabel *count-*

_neighbour. Jumlah tetangga kemudian akan dicari nilai tertinggi. Nilai tertinggi dari jumlah tetangga akan dibuat sebagai acuan untuk *Threshold* yang nilainya disimpan dalam array *threshold (th)*. Potongan kode untuk proses perhitungan *Threshold* dapat dilihat pada Gambar 4.12.

```
1.    for(int i=0;i<nodes_count;i++){
2.     for(int j=(i+1);j<nodes_count;j++){
3.       if(count_neighbour[i]>count_neighbour[j
      ]){
4.          int tmp;
5.          tmp=count_neighbour[i];
6.          count_neighbour[i]=count_neighbour[j]
      ;
7.          count_neighbour[j]=tmp;
8.         }
9.       }
10.   }
11.   for(int i=0;i<nodes_count;i++){
12.    count_mode[i]=0;
13.     for(int j=0;j<nodes_count;j++){
14.     if(count_neighbour[i] ==
15.     count_neighbour[j]){
16.         count_mode[i]++;
17.        }
18.     }
19.   }
20.
21.   for(int i=0;i<nodes_count;i++){
22.     if(count_mode[i]>th ){
23.       th=count_mode[i];
24.     }
25.   old_th = th;
26.   }
```

Gambar 4.12 Potongan Kode Perhitungan Threshold CH

4.2.3 Implementasi Pemilihan *Node Cluster Heads*

Di tahap selanjutnya, setelah mengetahui jumlah tetangga dan *Threshold*, dilakukan pemilihan *Node Cluster Head* untuk mengirimkan paket RREQ. Pemilihan ini dilakukan pada fungsi `CalculateCHID()` yang terletak pada kode sumber aodv.cc yang terletak pada ns2.35/aodv. Fungsi ini digunakan untuk menghitung nilai *threshold* pencarian kandidat *Cluster Head*. Fungsi `max_degree` merupakan fungsi penyimpanan yang nantinya digunakan sebagai pembanding minimal tetangga agar suatu node dapat menjadi *Cluster Head*. Fungsi `count_neighbour` merupakan variable global untuk melakukan perhitungan jumlah tetangga tiap node. Dan `CH_ID` digunakan untuk menyimpan ID dari *node Cluster Head* jika suatu node telah ditetapkan sebagai *Cluster Head*. Potongan kode untuk proses pemilihan *node Cluster Head* dapat dilihat pada Gambar 4.13.

```
1.    void AODV::calculateCHID()
2.    {
3.    int max_degree = -1;
4.    for (int i = 0; i < sizeof(count_neighbour
      ) / sizeof(*count_neighbour); i++) {
5.       if (count_neighbour[i] > max_degree)
6.       max_degree = count_neighbour[i];
7.       }
8.
      if (count_neighbour[index] == max_degree)
9.         {
10.        CH_ID = index;
11.        }
12.   else {
13.        CH_ID = -1;
14.          }
15.       }
16.   }
```

Gambar 4.13 Potongan Kode Pemilihan Node Cluster Head

4.2.4 Implementasi Pemilihan *Forwarding Node*

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

Penghitungan ini dilakukan pada fungsi `sendRequest()` dan fungsi `recvRequest()` yang terletak pada kode sumber aodv.cc yang terletak pada ns2.35/aodv. Apabila *node* menerima pesan dari dua *Cluster Head*, maka *node* tersebut menjadi *node gateway*. *Node Gateway* harus dilalui saat pengiriman paket dan penghubung antar *Cluster,* Potongan kode untuk proses seleksi *forwarding node* dapat dilihat pada Gambar 4.14.

```
1.   void AODV::sendRequest(nsaddr_t dst)
2.   int cluster_head = CH_ID;
3.   int GatewayNode = gateway_node
4.   if (CH_ID == -1){
5.      calculateCHID();
6.      }
7.   rq->rq_cluster_head_index = CH_ID;
8.   CH_ID = cluster_head;
9.   if (CH_ID != index){
10.  rq->rq_bcast_id = CH_ID;
11.  }
12.  void AODV::recvRequest(Packet *p)
13.  if (CH_ID != 1) {
14.      rq->rq_cluster_head_index
15.  else if (CH_ID == 1)
16.  rq->gateway_cooperative
17.  {
18.  if (rq->rq_cluster_head_index != 1){
19.  CH_ID = rq->gateway_node;
20.   }
21.  }
```

Gambar 4.14 Potongan Kode Penyeleksian *Forwarding Node*

4.3 Implementasi Simulasi pada NS-2

Implementasi simulasi VANETs diawali dengan pendeskripsian lingkungan simulasi pada sebuah *file* tcl. *File* ini berisikan konfigurasi setiap *node* dan langkah-langkah yang dilakukan selama simulasi. Potongan konfigurasi lingkungan simulasi dapat dilihat pada Gambar 4.15.

```
set val(chan)    Channel/WirelessChannel
set val(prop)    Propagation/TwoRayGround
set val(netif)   Phy/WirelessPhy
set val(mac)     Mac/802_11
set val(ifq)     Queue/DropTail/PriQueue
set val(ll)      LL
set val(ant)     Antenna/OmniAntenna
set opt(x)       1500
set opt(y)       1500
set val(ifqlen)  1000
set val(nn)      200
set val(seed)    1.0
set val(adhocRouting) AODV
set val(stop)    200
set val(cp)      "cbr200.tcl"
set val(sc)      "scenario.tcl"
```

Gambar 4.15 Implementasi Simulasi NS-2

Pada konfigurasi dilakukan pemanggilan terhadap *file traffic* yang berisikan konfigurasi *node* asal, *node* tujuan, pengiriman paket, serta *file* skenario yang berisi pergerakan *node* yang telah di*generate* oleh SUMO. Kode implementasi pada NS-2 dapat dilihat pada lampiran A.5 Kode Skenario NS-2.

Konfigurasi untuk *file traffic* bisa dilakukan dengan membuat *file* berekstensi .txt untuk menyimpan konfigurasi tersebut. Pada *file* konfigurasi lingkungan simulasi, *file traffic* tersebut dimasukkan agar

dibaca sebagai *file traffic*. Potongan konfigurasi *file traffic* dapat dilihat pada Gambar 4.16.

```
set udp_(0) [new Agent/UDP]
$ns_ attach-agent $node_(198) $udp_(0)
set null_(0) [new Agent/Null]
$ns_ attach-agent $node_(199) $null_(0)
set cbr_(0) [new Application/Traffic/CBR]
$cbr_(0) set packetSize_ 512
$cbr_(0) set interval_ 1
$cbr_(0) set random_ 1
$cbr_(0) set maxpkts_ 1000
$cbr_(0) attach-agent $udp_(0)
$ns_ connect $udp_(0) $null_(0)
$ns_ at 0 "$cbr_(0) start"
$ns_ at 200.0000000000000000 "$cbr_(0) stop"
```

Gambar 4.16 Implementasi Simulasi File Traffic

Pada konfigurasi tersebut, ditentukan *node* sumber dan *node* tujuan pengiriman paket. Pengiriman dimulai pada detik ke- 2.55. Implementasi konfigurasi *file traffic* untuk simulasi pada NS-2 dapat dilihat pada lampiran A.6 Kode Konfigurasi *Traffic*

4.4 Implementasi Metrik Analisis

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

4.4.1 Implementasi *Packet Delivery Ratio* (PDR)

Pada subbab 2.3.2 telah ditunjukkan contoh struktur data *event* yang dicatat dalam *trace file* oleh NS-2. Kemudian, pada persamaan 3.1 telah dijelaskan bagaimana menghitung PDR. Skrip awk untuk menghitung PDR berdasarkan kedua informasi tersebut dapat dilihat pada lampiran A.7 Kode Skrip AWK *Packet Delivery Ratio*.

PDR didapatkan dengan cara menghitung setiap baris terjadinya *event* pengiriman dan penerimaan paket data yang dikirim melalui agen pada *trace file*. Skrip menyaring setiap baris yang mengandung *string* AGT karena kata kunci tersebut menunjukan *event* yang berhubungan dengan paket komunikasi data. Penghitungan dilakukan dengan menjumlahkan paket yang dikirimkan dan paket yang diterima dengan menggunakan karakter pada kolom pertama sebagai *filter*. Kolom pertama menunjukan event yang terjadi dari sebuah paket. Setelah itu nilai PDR dihitung dengan cara persamaan 3.1. Pseudocode untuk menghitung PDR dapat dilihat pada Gambar 4.17.

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

Gambar 4.17 Pseudocode untuk Perhitungan PDR

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

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

Skrip awk untuk menghitung E2E dapat dilihat pada lampiran A.8 Kode Skrip AWK Rata-Rata *End-to-End Delay*.

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

Setelah mendapatkan *delay* paket, langkah selanjutnya adalah dengan mencari rata-rata dari *delay* tersebut dengan menjumlahkan semua *delay* paket dan membaginya dengan jumlah paket. *Pseudocode* untuk menghitung rata-rata E2E dapat dilihat pada Gambar 4.18.

```
sum_delay = 0
counter = 0

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

Gambar 4.18 Pseudocode untuk Perhitungan E2E

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

4.4.3 Implementasi *Routing Overhead* (RO)

Seperti yang telah dijelaskan sebelumnya, *routing overhead* merupakan jumlah dari paket kontrol *routing* baik itu RREQ, RREP, maupun RERR. Dengan begitu, untuk mendapatkan RO yang perlu dilakukan adalah menjumlahkan tiap paket dengan *filter event sent* pada kolom pertama dan *event layer* RTR pada kolom ke-4. Perhitungan RO telah dijelaskan pada persamaan 3.3. Skrip AWK untuk menghitung RO dapat dilihat pada lampiran A.9 Kode Skrip AWK *Routing Overhead*. *Pseudocode* untuk menghitung RO dapat dilihat pada Gambar 4.19.

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

Gambar 4.19 Pseudocode Perhitungan Routing Overhead

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

4.4.4 Implementasi *Forwarded Route Request*

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

pada *file* dengan ekstensi .tr. Perhitungan RREQ F telah dijelaskan pada persamaan 3.4. Skrip awk untuk menghitung RREQ F dapat dilihat pada lampiran

A.10 Kode Skrip *AWK Forwarded Route Request* . *Pseudocode* untuk menghitung RREQ F dapat dilihat pada Gambar 4.20.

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

Gambar 4.20 Pseudocode Perhitungan RREQ F

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

BAB V
UJICOBA DAN EVALUASI

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

5.1 Lingkungan Uji Coba

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

Tabel 5.1 Spesifikasi Perangkat yang Digunakan

Komponen	**Spesifikasi**
CPU	Intel(R) Core™ i7-6700HQ CPU @ 2.70GHz
Sistem Operasi	Ubuntu 18.04.2 LTS
Linux Kernel	Linux kernel 4.4
Memori	8.0 GB

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

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

Parameter lingkungan uji coba yang digunakan pada NS-2 dapat diihat pada Tabel 5.2. Pengujian dilakukan dengan menjalankan skenario yang disimulasikan pada NS-2. Dari simulasi tersebut dihasilkan sebuah *trace file* dengan ekstensi .tr yang akan dianalisis dengan bantuan skrip awk untuk mendapatkan PDR, E2E, RO, dan RREQ F menggunakan kode yang terdapat pada lampiran A.7 Kode Skrip AWK *Packet Delivery Ratio*, A.8 Kode Skrip AWK Rata-Rata

End-to-End Delay, A.9 Kode Skrip AWK *Routing Overhead*, dan A.10 Kode Skrip Awk *Forwarded Route Request*.

Tabel 5.2 Lingkungan Uji Coba

No.	Parameter	Spesifikasi
1	Network simulator	NS-2.35
2	*Routing protocol*	AODV dan AODV *Clustering*
3	Waktu simulasi	200 detik
4	Area simulasi	700 m x 700 m (Grid) 1500 m x 1500 m (Real)
5	Jumlah *Node*	50, 100, 150, 200
6	Radius transmisi	400m
7	Kecepatan maksimum	20 m/s
8	Paket Interval	1 paket/detik
9	Protokol MAC	IEEE 802.11p
10	Model Propagasi	*Two-ray ground*

5.2 Hasil Uji Coba

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

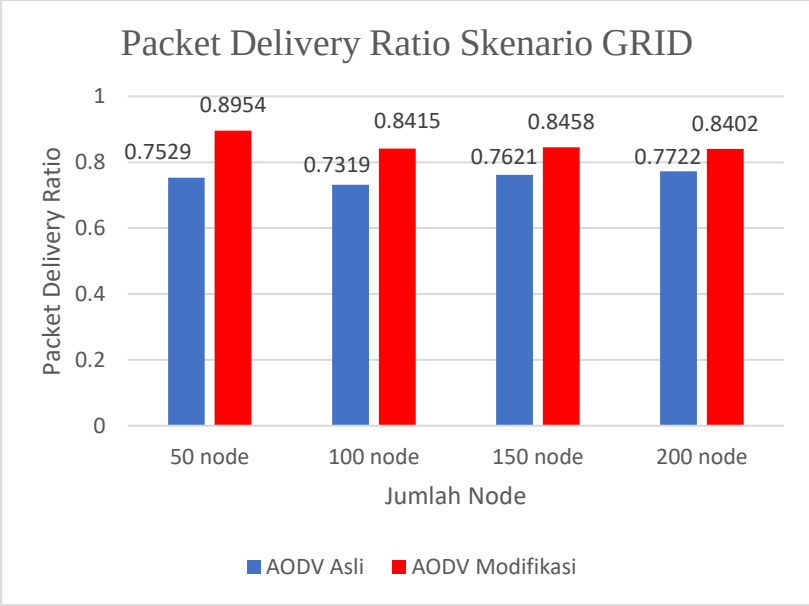
5.2.1 Hasil Uji Coba Skenario *Grid*

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

Pengambilan data uji pada skenario *grid* dilakukan sebanyak 10 kali dengan skenario mobilitas *random* pada peta *grid* dengan luas area 700 m x 700m m dan *node* sebanyak 50 untuk lingkungan jarang, 100 dan 150 *node* untuk lingkungan yang sedang, dan 200 *node* untuk lingkungan padat dilakukan pada kecepatan standar yaitu 20 m/s. Hasil analisis dapat dilihat pada Tabel 5.3, Tabel 5.4, Tabel 5.5, dan Tabel 5.6.

Tabel 5.3 Hasil Rata - Rata PDR Skenario *Grid*

Jumlah *Node*	AODV Asli	AODV Modifikasi	Perbedaan
50	0,7529	0,8954	+ 0,1424
100	0,7319	0,8415	+ 0,1095
150	0,7621	0,8458	+ 0,8363
200	0,7722	0,8402	+ 0,0679



Gambar 5.1 Grafik Packet Delivery Ratio Skenario *Grid*

Berdasarkan grafik pada Gambar 5.1 dapat dilihat bahwa *routing protocol* AODV yang telah dimodifikasi dan juga *routing protocol* AODV asli mengalami kenaikan yang signifikan pada packet delivery ratio. Pada lingkungan yang jarang dengan jumlah 50 node, menghasilkan perbedaan selisih PDR sebesar 0.1424, atau naik menjadi sekitar 18,92% dimana *routing protocol* AODV yang telah

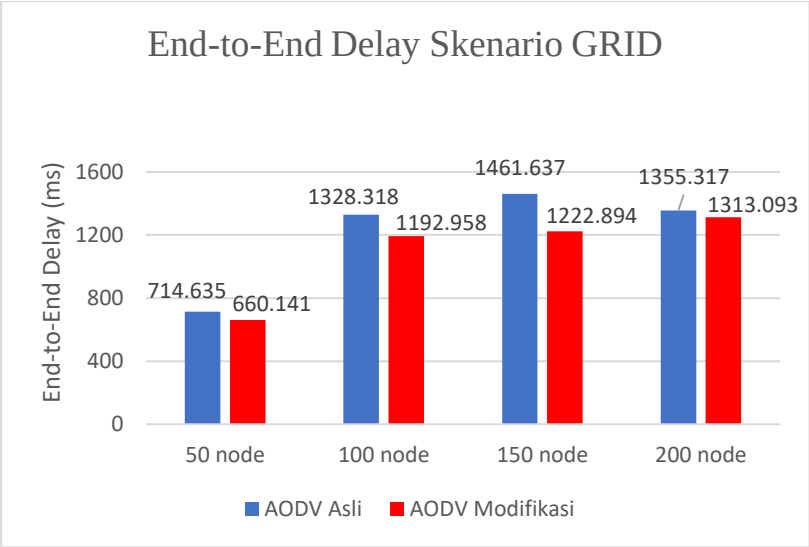
dimodifikasi unggul dalam hal PDR tersebut. Pada lingkungan yang sedang dengan jumlah 100 node, menghasilkan perbedaan selisih PDR sebesar 0.1095, atau naik menjadi sekitar 14.96% dimana *routing protocol* AODV yang telah dimodifikasi unggul dalam hal PDR tersebut dari AODV asli. Pada lingkungan yang sedang dengan jumlah 150 node, menghasilkan perbedaan selisih PDR sebesar 0.0836, atau naik menjadi sekitar 10,97% dimana *routing protocol* AODV yang telah dimodifikasi juga unggul dalam hal PDR tersebut dari AODV asli. Pada lingkungan yang padat dengan jumlah 200 node, menghasilkan perbedaan selisih PDR sebesar 0.0679, atau naik menjadi sekitar 8,79% dimana *routing protocol* AODV yang telah dimodifikasi juga unggul dalam hal PDR tersebut.

Dapat dilihat rata-rata kenaikan yang terjadi untuk PDR adalah 13,41%. Jika ketiga lingkungan tersebut dibandingkan dapat dilihat bahwa dengan jumlah *node* yang lebih sedikit atau pada lingkungan dengan *node* yang jarang, menghasilkan PDR yang lebih bagus daripada di lingkungan dengan jumlah *node* yang sedang maupun yang padat baik untuk *routing protocol* AODV asli maupun *routing protocol* AODV yang telah dimodifikasi. Dapat dilihat pula bahwa AODV yang telah dimodifikasi menghasilkan PDR yang lebih bagus daripada AODV asli dengan jumlah selisih PDR yang cukup signifikan.

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

Tabel 5.4 Hasil Rata - Rata E2E Skenario Grid

Jumlah *Node*	**AODV Asli**	**AODV Modifikasi**	**Perbedaan**
50	714.635 ms	660.141 ms	- 54.494 ms
100	1328.318 ms	1192.958 ms	- 135.360 ms
150	1461.637 ms	1222.894 ms	- 238.743 ms
200	1355.317 ms	1313.093 ms	- 42.224 ms



Gambar 5.2 Grafik End-to-end Delay Skenario *Grid*

Berdasarkan grafik pada Gambar 5.2 dapat dilihat bahwa rata-rata *end-to-end delay* antara *routing protocol* AODV asli dan AODV yang telah dimodifikasi mengalami penurunan yang signifikan. Pada lingkungan yang jarang dengan jumlah 50 node, terjadi perbedaan selisih *end-to-end delay* sebesar 54,494 ms antara *routing protocol* AODV asli dengan *routing protocol* AODV yang telah dimodifikasi atau mengalami penurunan sebesar 7,61%, dimana *routing protocol* AODV yang telah dimodifikasi unggul dalam hal *end-to-end delay* tersebut. Sedangkan pada lingkungan sedang dengan jumlah 100 node, terjadi perbedaan selisih *end-to-end delay* sebesar 135,360 ms antara *routing protocol* AODV asli dengan *routing protocol* AODV yang telah dimodifikasi atau mengalami penurunan sebesar 10,19%, dimana *routing protocol* AODV yang dimodifikasi lebih unggul dalam hal *end-to-end delay* tersebut. Pada pada lingkungan sedang dengan jumlah 150 node, terjadi perbedaan selisih *end-to-end delay* sebesar 238,743 ms antara *routing protocol* AODV asli dengan *routing protocol* AODV yang telah dimodifikasi atau

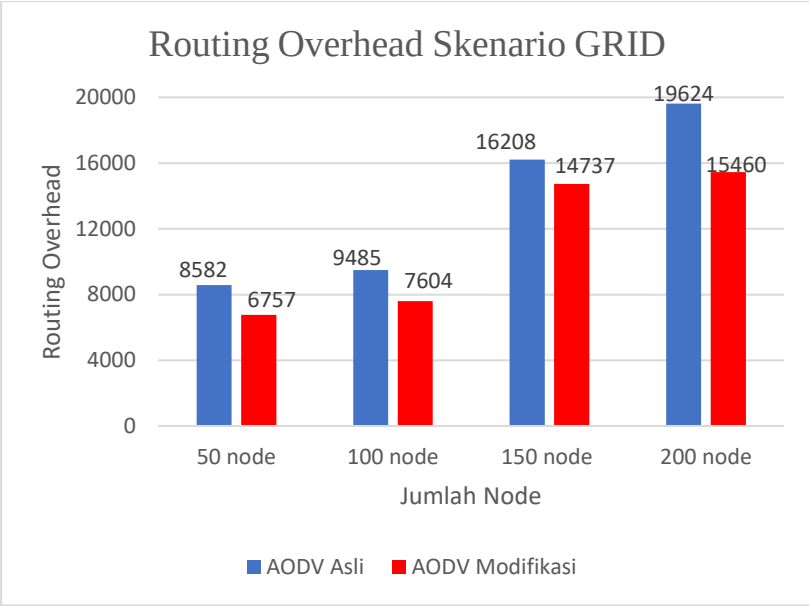
mengalami penurunan sebesar 16,33%, dimana *routing protocol* AODV yang dimodifikasi lebih unggul dalam hal *end-to-end delay* tersebut. Pada lingkungan yang padat dengan jumlah 200 node, terjadi perbedaan selisih *end-to-end delay* sebesar 42,224 ms antara ro*uting protocol* AODV asli dengan *routing protocol* AODV yang telah dimodifikasi atau mengalami penurunan sebesar 3,12%, dimana *routing protocol* AODV yang telah dimodifikasi jauh lebih unggul dalam hal *end-to-end delay* tersebut.

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

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

Tabel 5.5 Hasil Rata - Rata RO Skenario Grid

Jumlah *Node*	**AODV Asli**	**AODV Modifikasi**	**Perbedaan**
50	8582	6757	- 1825
100	9485	7604	- 1881
150	16208	14737	- 1471
200	19624	15460	- 4163



Gambar 5.3 Grafik Routing Overhead Skenario *Grid*

Berdasarkan grafik pada Gambar 5.3 dapat dilihat bahwa rata-rata *routing overhead* antara *routing protocol* AODV asli dan AODV yang telah dimodifikasi mengalami kenaikan yang signifikan. Pada lingkungan yang jarang dengan jumlah 50 node, menghasilkan perbedaan selisih *routing overhead* sebesar 1825 atau mengalami penurunan sebesar 21,27%, dimana *routing protocol* AODV modifikasi unggul dalam hal *routing overhead* tersebut karena menghasilkan *routing overhead* yang lebih rendah dari routing AODV asli. Pada lingkungan yang sedang dengan jumlah 100 node, menghasilkan perbedaan selisih *routing overhead* sebesar 1881 atau mengalami penurunan sebeasr 19.83%, dimana *routing protocol* AODV yang telah dimodifikasi unggul dalam mereduksi *routing overhead* dari AODV asli. Pada lingkungan yang sedang dengan jumlah 150 node, menghasilkan perbedaan selisih *routing overhead* sebesar 1471 atau mengalami penurunan sebesar 9,08%, dimana *routing protocol* AODV yang telah dimodifikasi mengalami

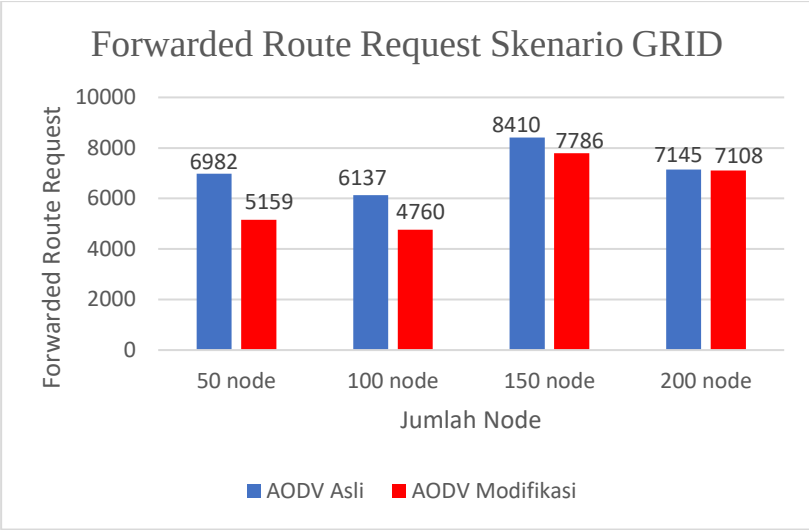
keunggulan dalam hal *routing overhead* tersebut dari AODV asli. Pada lingkungan yang padat dengan jumlah 200 node, menghasilkan perbedaan selisih *routing overhead* sebesar 4163 atau mengalami penurunan sebesar 21,22%, dimana *routing protocol* AODV yang telah dimodifikasi lebih unggul daripada AODV asli dalam hal *routing overhead* tersebut.

Dapat dilihat rata-rata penurunan yang terjadi untuk *routing overhead* adalah sebesar 17,85%. Jika ketiga lingkungan tersebut dibandingkan dapat dilihat bahwa dengan jumlah *node* yang lebih sedikit atau pada lingkungan dengan *node* yang jarang, menghasilkan *routing overhead* yang lebih bagus atau lebih sedikit daripada di lingkungan dengan jumlah *node* yang sedang maupun yang padat baik untuk *routing protocol* AODV asli maupun *routing protocol* AODV yang telah dimodifikasi. Dapat dilihat pula bahwa AODV yang telah dimodifikasi menghasilkan *routing overhead* yang lebih sedikit atau dalam hal ini AODV modifikasi lebih unggul di semua lingkungan dengan jumlah selisih *routing overhead* yang cukup signifikan. Hal ini dikarenakan dalam proses pengiriman paket, hanya melewati *node cluster head*. Serta melakukan penambahan parameter *node gateway*. Semakin rendah nilai RO maka *packet drop* semakin rendah dikarenakan pencarian rute lebih cepat atau optimal dilakukan seiring bertambahnya kepadatan node.

Untuk hasil pengambilan data *forwarded route request* (RREQ F) pada skenario *grid* 50 *node*, 100 *node*, 150 *node* dan 200 *node* dapat dilihat pada Gambar 5.4.

Tabel 5.6 Hasil Rata - Rata RREQ F Skenario Grid

Jumlah *Node*	**AODV Asli**	**AODV Modifikasi**	**Perbedaan**
50	6982	5159	- 1823
100	6137	4760	- 1377
150	8410	7786	- 624
200	7145	7108	- 37



Gambar 5.4 Grafik Forwarded Route Request Skenario *Grid*

Berdasarkan grafik pada Gambar 5.4 dapat dilihat bahwa *routing protocol* AODV yang telah dimodifikasi dan juga *routing protocol* asli mengalami perubahan *forwarded route request* (RREQ F) yang fluktuatif. Pada lingkungan yang jarang dengan jumlah 50 *node*, menghasilkan perbedaan selisih *forwarded route request* sebesar 1823 atau mengalami penurunan sebesar 26,11%, dimana *routing protocol* AODV yang telah dimodifikasi unggul dalam hal *forwarded route request* tersebut karena menghasilkan *forwarded route request* yang lebih rendah dari *routing protocol* AODV asli. Pada lingkungan yang sedang dengan jumlah 100 *node*, menghasilkan perbedaan selisih *forwarded route request* sebesar 1377 atau mengalami penurunan sebesar 22,44%, dimana *routing protocol* AODV yang telah dimodifikasi juga unggul dalam hal *forwarded route request* tersebut dari *forwarded route request* AODV asli. Pada lingkungan yang sedang dengan jumlah 150 *node*, menghasilkan perbedaan selisih *forwarded route request* sebesar 624 atau mengalami penurunan sebesar 7,42%, dimana *routing protocol* AODV modifikasi lebih unggul dalam hal *forwarded route request*

tersebut dari *forwarded route request* AODV yang asli. Pada lingkungan yang padat dengan jumlah 200 *node*, menghasilkan perbedaan selisih *forwarded route request* sebesar 37 atau mengalami penurunan sebesar 0,52%, dimana *routing protocol* AODV yang telah dimodifikasi juga unggul dalam hal *forwarded route request* tersebut.

Dapat dilihat rata-rata penurunan yang terjadi untuk *forwarded route request* adalah sebesar 14,12%. Jika ketiga lingkungan tersebut dibandingkan dapat dilihat bahwa dengan jumlah *node* yang jarang, menghasilkan *forwarded route request* yang lebih bagus atau lebih sedikit daripada di lingkungan dengan jumlah *node* yang sedang maupun yang padat baik untuk *routing protocol* AODV asli maupun *routing protocol* AODV yang telah dimodifikasi. Dapat dilihat pula bahwa AODV yang telah dimodifikasi menghasilkan *forwarded route request* yang lebih bagus atau dalam hal ini lebih rendah daripada AODV asli dengan jumlah selisih *forwarded route request* yang cukup signifikan.

Tabel 5.7 Rute dan Durasi Penggunaan Rute pada AODV

Rute	Durasi (detik)	Paket Diterima
[48]-[10]-[38]-[24]-[41]-[42]-[15]-[46]-[40]-[49]	11,93	14
[48]-[5]-[6]-[23]-[18]-[47]-[2]-[44]-[9]-[49]	5,62	6
[48]-[5]-[12]-[45]-[18]-[47]-[11]-[33]-[40]-[49]	2,49	4
[48]-[20]-[26]-[21]-[13]-[2]-[44]-[3]-[49]	10,29	10
[48]-[20]-[34]-[35]-[13]-[15]-[28]-[27]-[49]	3,64	5
[48]-[24]-[0]-[29]-[47]-[46]-[40]-[49]	1,46	4
[48]-[24]-[32]-[45]-[23]-[33]-[40]-[49]	1,21	3

[48]-[20]-[5]-[11]-[1]-[44]-[18]-[40]-[49]	4,46	4
[48]-[24]-[4]-[30]-[10]-[24]-[18]-[40]-[3]-[49]	4,27	6
[48]-[24]-[20]-[5]-[19]-[38]-[42]-[40]-[3]-[49]	1,36	3
[48]-[39]-[24]-[20]-[44]-[16]-[10]-[45]-[3]-[49]	9,43	7
[48]-[39]-[24]-[20]-[44]-[16]-[10]-[40]-[25]-[49]	2,28	3

Tabel 5.8 Durasi Penggunaan Rute pada AODV-Clustering

Rute	Durasi (detik)	Paket Diterima
[48]-[5]-[45]-[47]-[33]-[49]	14,54	16
[48]-[6]-[15]-[33]-[49]	4,02	8
[48]-[17]-[21]-[13]-[28]- [49]	1.44	2
[48]-[17]-[21]-[13]-[44]- [49]	13,10	14
[48]-[34]-[21]-[28]-[49]	6,56	7
[48]-[5]-[11]-[9]-[27]-[49]	15,25	17
[48]-[37]-[35]-[15]-[28]-[33]-[49]	3,04	7
[48]-[11]-[30]-[42]-[40]- [49]	3,25	4
[48]-[26]-[37]-[30]-[28]-[18]-[49]	6.67	8
[48]-[24]-[12]-[13]-[25]-[21]-[49]	10,11	13
[48]-[26]-[11]-[15]-[36]-[25]-[49]	8,11	9
[48]-[26]-[11]-[15]-[36]-[32]-[21]-[49]	21,85	25
[48]-[26]-[30]-[38]-[40]-[42]-[49]	45,78	44

Tabel 5.7 dan 5.8 merupakan hasil dari pengecekan rute pada salah satu skenario di lingkungan 50 *node.* Dapat dilihat bahwa durasi bertahannya rute yang terbentuk pada AODV asli cukup singkat, durasi maksimal suatu rute hanya bertahan selama 11,93 detik dengan rata-rata durasi penggunaan rute 4,87 detik ketika pengiriman paket. Sedangkan pada AODV *Clustering* bertahannya rute yang terbentuk lebih lama daripada AODV asli dengan maksimal bertahan selama 45,78 detik dan rata-rata durasi penggunaan rute 11,82 detik. AODV sering mengalami hilangnya koneksi *link* dengan node penerima karena adanya pergerakan dinamis dari tiap *node*. Sehingga saat rute terputus, harus dilakukan *route request* lagi.

Dengan menggunakan AODV *Clustering,* rute yang didapatkan cenderung stabil karena hanya melewati *Cluster Head* (*node* dengan *degree* tertinggi) dan *node gateway* yang keduanya merupakan *node* yang berhak meneruskan *route request.* Rute yang melakukan *packet drop* pada AODV asli sebanyak 95 paket sedangkan pada AODV *Clustering* hanya sebanyak 10 paket.

5.2.2 Hasil Uji Coba Skenario *Real*

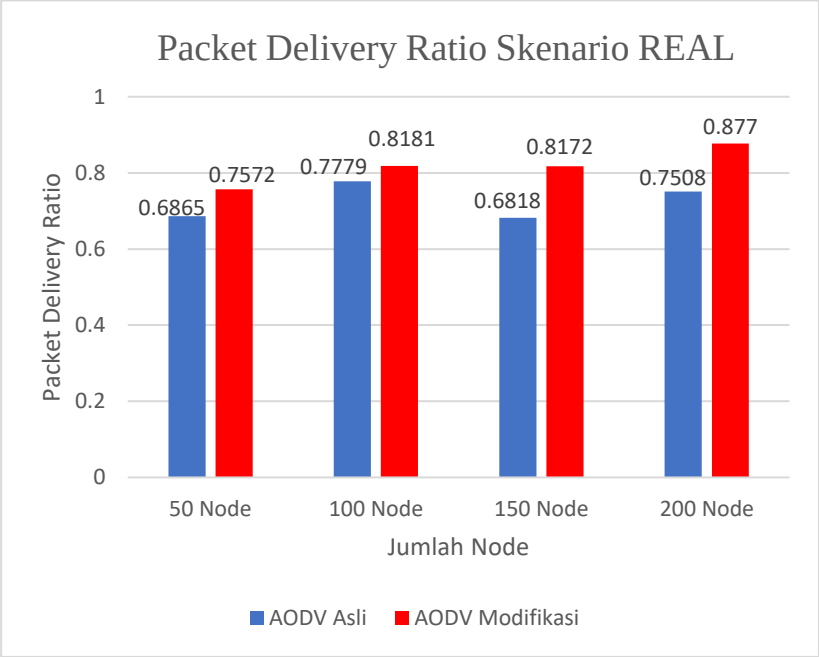
Pengujian pada skenario *real* digunakan untuk melihat perbandingan PDR, E2E, RO, dan RREQ F antara *routing protocol* AODV asli dan AODV yang telah dimodifikasi dalam pemilihan *node* yang dapat menerima paket *route request*.

Pengambilan data uji PDR, E2E, RO, dan RREQ F pada skenario *real* dilakukan sebanyak 10 kali dengan skenario mobilitas *random* pada peta *real* dengan luas area 1500 m x 1500 m dengan *range transmisi* 400 meter dan *node* sebanyak 50 untuk lingkungan yang jarang, 100 dan 150 *node* untuk lingkungan yang sedang, dan 200 *node* untuk lingkungan yang padat dilakukan pada kecepatan standar yaitu 20 m/s. Untuk hasil analisis Skenario dengan peta *Real* dengan *node* 50, 100, 150 dan 200 dapat dilihat pada Tabel 5.9, Tabel 5.10, Tabel 5.11, dan Tabel 5.12.

Tabel 5.9 Hasil Rata - Rata PDR pada Skenario Real

Jumlah *Node*	**AODV Asli**	**AODV Modifikasi**	**Perbedaan**
50	0.6865	0.7572	+ 0.0707
100	0.7779	0.8181	+ 0.0401
150	0.6818	0.8172	+ 0.1354
200	0.7508	0.8770	+ 0.1262

Dari data pada Tabel 5.9, dibuat grafik yang merepresentasikan hasil perhitungan PDR yang ditunjukkan pada Gambar 5.5.



Gambar 5.5 Grafik *Packet Delivery Ratio* Skenario *Real*

Berdasarkan grafik pada Gambar 5.5 dapat dilihat bahwa *routing protocol* AODV yang telah dimodifikasi dan juga *routing protocol* AODV asli mengalami kenaikan yang signifikan pada *packet delivery ratio*. Pada lingkungan yang jarang dengan jumlah 50 node, menghasilkan perbedaan selisih PDR sebesar 0.0707, atau naik menjadi sekitar 10,30% dimana *routing protocol* AODV yang telah dimodifikasi unggul dalam hal PDR tersebut. Pada lingkungan yang sedang dengan jumlah 100 node, menghasilkan perbedaan selisih PDR sebesar 0.0401, atau naik menjadi sekitar 5,15% dimana *routing protocol* AODV yang telah dimodifikasi unggul dalam hal PDR tersebut dari AODV asli. Pada lingkungan yang sedang dengan jumlah 150 node, menghasilkan perbedaan selisih PDR sebesar 0.1354, atau naik menjadi sekitar 19,86% dimana routing protocol AODV yang telah dimodifikasi unggul dalam hal PDR tersebut dari AODV asli. Pada lingkungan yang padat dengan jumlah 200 node, menghasilkan perbedaan selisih PDR sebesar 0.1262, atau naik menjadi sekitar 16,81 % dimana routing protocol AODV yang telah dimodifikasi juga unggul dalam hal PDR tersebut.

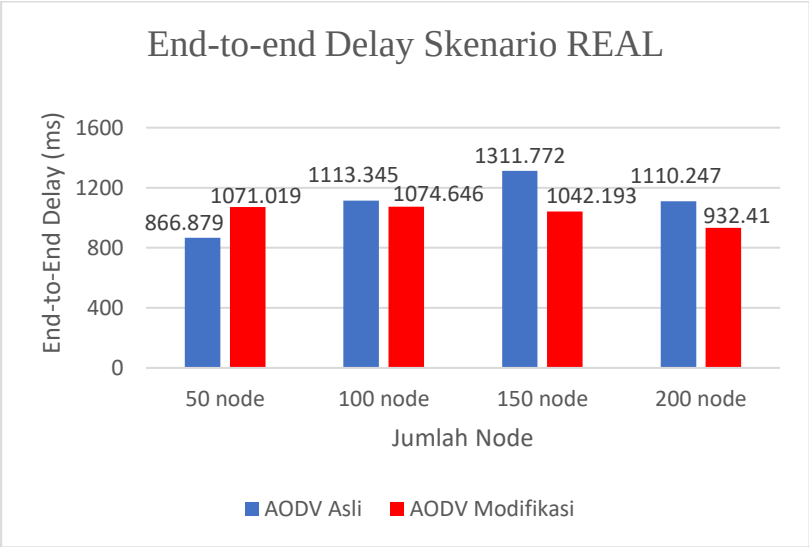
Dapat dilihat rata-rata kenaikan yang terjadi untuk PDR adalah 13,03%. Jika ketiga lingkungan tersebut dibandingkan dapat dilihat bahwa dengan jumlah *node* yang lebih banyak atau pada lingkungan dengan *node* yang padat, menghasilkan PDR yang lebih bagus daripada di lingkungan dengan jumlah *node* yang jarang maupun yang sedang baik untuk *routing protocol* AODV asli maupun *routing protocol* AODV yang telah dimodifikasi. Dapat dilihat pula bahwa AODV yang telah dimodifikasi menghasilkan PDR yang lebih bagus daripada AODV asli dengan jumlah selisih PDR yang cukup signifikan.

Hasil pengambilan data rata-rata untuk *end-to-end delay* (E2E) pada skenario *real* dengan jumlah *node* 50, 100, 150, dan 200 dapat dilihat pada Tabel 5.10.

Tabel 5.10 Hasil Rata -Rata E2E pada Skenario Real

Jumlah *Node*	**AODV Asli**	**AODV Modifikasi**	**Perbedaan**
50	866.879 ms	1071.019 ms	+ 204.140 ms
100	1113.345 ms	1074.646 ms	- 38.699 ms
150	1311.772 ms	1042.193 ms	- 269.579 ms
200	1110.247 ms	932.410 ms	- 177.837 ms

Dari data di atas, dibuat grafik yang merepresentasikan hasil perhitungan E2E yang ditunjukkan pada Gambar 5.6.



Gambar 5.6 Grafik *End-to-end Delay* pada Skenario *Real*

Berdasarkan grafik pada Gambar 5.6 dapat dilihat bahwa rata-rata *end-to-end delay* antara *routing protocol* AODV asli dan AODV yang telah dimodifikasi mengalami perubahan yang fluktuatif. Pada lingkungan yang jarang dengan jumlah 50 *node*, terjadi perbedaan selisih *end-to-end delay* sebesar 204,140 ms antara *routing protocol* AODV asli dengan *routing protocol* AODV yang telah

dimodifikasi atau mengalami kenaikan sebesar 23,55%, dimana *routing protocol* AODV asli lebih unggul dari AODV yang telah dimodifikasi dalam hal *end-to-end delay* tersebut. Sedangkan pada lingkungan sedang dengan jumlah 100 *node*, terjadi perbedaan selisih *end-to-end delay* sebesar 38,699 ms antara *routing protocol* AODV asli dengan *routing protocol* AODV yang telah dimodifikasi atau mengalami penurunan sebesar 3,48%, dimana *routing protocol* AODV yang dimodifikasi lebih unggul dalam hal *end-to-end delay*. Pada pada lingkungan sedang dengan jumlah 150 *node*, terjadi perbedaan selisih *end-to-end delay* sebesar 269,579 ms antara *routing protocol* AODV asli dengan *routing protocol* AODV yang telah dimodifikasi atau mengalami penurunan sebesar 20,55%, dimana *routing protocol* AODV yang dimodifikasi lebih unggul dalam hal *end-to-end delay* tersebut. Pada lingkungan yang padat dengan jumlah 200 *node*, terjadi perbedaan selisih *end-to-end delay* sebesar 177,837 ms antara *routing protocol* AODV asli dengan *routing protocol* AODV yang telah dimodifikasi atau mengalami penurunan sebesar 16,02%, dimana *routing protocol* AODV yang telah dimodifikasi jauh lebih unggul dalam hal *end-to-end delay* tersebut.

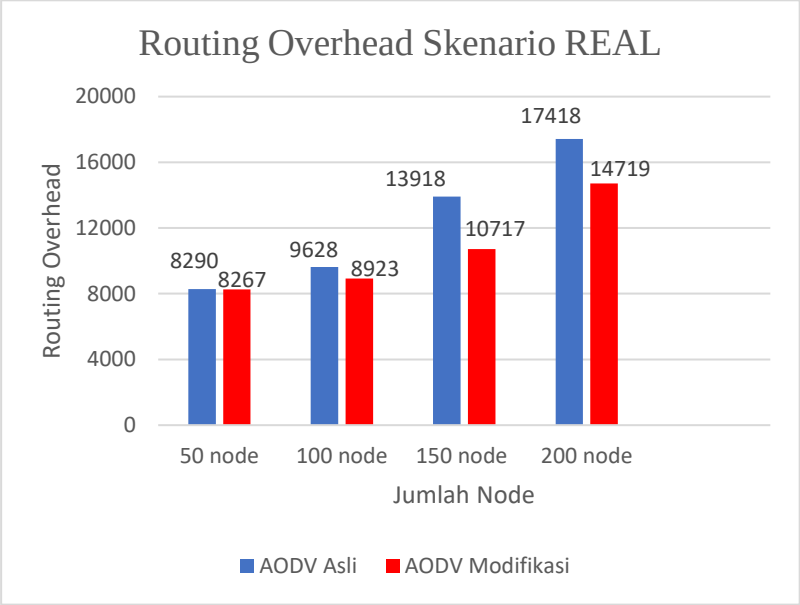
Jika ketiga lingkungan tersebut dibandingkan, memang pada lingkungan yang jarang, AODV asli lebih unggul dari AODV yang telah dimodifikasi dalam hal *end-to-end delay*. Namun, dapat dilihat juga bahwa dengan jumlah *node* sedang dan padat menghasilkan *end-to-end delay* yang lebih baik dan dapat dilihat pula bahwa AODV yang telah dimodifikasi menghasilkan *end-to end delay* yang lebih baik daripada AODV asli dengan jumlah selisih *end-to-end delay* yang cukup signifikan. Hal ini bisa terjadi karena dengan AODV modifikasi kali ini hanya melewati *cluster head* dan *node gateway* sehingga paket-paket akan lebih sedikit mengalami adanya delay dalam pengiriman. Hasil rata – rata *Delay* tidak dapat dianalisis karena terjadi fluktuasi dan tidak stabil. Hal ini dikarenakan waktu *delay* tergantung dari rata – rata waktu paket yang terkirim. Semakin banyak paket yang terkirim, maka semakin beragam *delay*nya.

Untuk hasil pengambilan data *routing overhead* pada skenario *real* 50 *node*, 100 *node*, 150 *node* dan 200 *node* dapat dilihat pada Tabel 5.11.

Tabel 5.11 Hasil Rata - Rata RO Skenario Real

Jumlah *Node*	**AODV Asli**	**AODV Modifikasi**	**Perbedaan**
50	8290	8267	- 23
100	9628	8923	- 704
150	13918	10717	- 3200
200	17418	14719	- 2699

Dari data di atas, dibuat grafik yang merepresentasikan hasil perhitungan RO yang ditunjukkan pada Gambar 5.7.



Gambar 5.7 Grafik *Routing Overhead* Skenario *Real*

Berdasarkan grafik pada Gambar 5.7 dapat dilihat bahwa rata-rata *routing overhead* antara *routing protocol* AODV asli dan AODV yang telah dimodifikasi mengalami penurunan yang signifikan. Pada lingkungan yang jarang dengan jumlah 50 *node*, menghasilkan perbedaan selisih *routing overhead* sebesar 23 atau mengalami penurunan sebesar 0,28%, dimana *routing protocol* AODV modifikasi unggul dalam hal *routing overhead* tersebut karena menghasilkan *routing overhead* yang lebih rendah dari routing AODV yang asli. Pada lingkungan yang sedang dengan jumlah 100 *node*, menghasilkan perbedaan selisih *routing overhead* sebesar 704 atau mengalami penurunan sebeasr 7,32%, dimana *routing protocol* AODV yang telah dimodifikasi mengalami keunggulan dalam hal routing overhead tersebut dari AODV asli. Pada lingkungan yang sedang dengan jumlah 150 *node*, menghasilkan perbedaan selisih *routing overhead* sebesar 3200 atau mengalami penurunan sebesar 22,99%, dimana *routing protocol* AODV yang telah dimodifikasi mengalami keunggulan dalam hal routing overhead tersebut dari AODV asli. Pada lingkungan yang padat dengan jumlah 200 *node*, menghasilkan perbedaan selisih *routing overhead* sebesar 2699 atau mengalami penurunan sebesar 15,50%, dimana *routing protocol* AODV yang telah dimodifikasi juga lebih unggul daripada AODV yang asli dalam hal routing overhead tersebut.

Dapat dilihat rata-rata penurunan yang terjadi untuk *routing overhead* adalah sebesar 11,52%. Jika ketiga lingkungan tersebut dibandingkan dapat dilihat bahwa dengan jumlah *node* yang lebih sedikit atau pada lingkungan dengan *node* yang jarang, menghasilkan *routing overhead* yang lebih bagus atau lebih sedikit daripada di lingkungan dengan jumlah *node* yang sedang maupun yang padat baik untuk *routing protocol* AODV asli maupun *routing protocol* AODV yang telah dimodifikasi. Dapat dilihat pula bahwa AODV yang telah dimodifikasi menghasilkan *routing overhead* yang lebih sedikit atau dalam hal ini AODV modifkikasi lebih unggul di semua lingkungan dengan jumlah selisih *routing overhead* yang cukup signifikan. Hal ini dikarenakan dalam proses pengiriman paket, diinisiasi dengan pendefinisian dan melakukan pencarian *node cluster head*. Serta

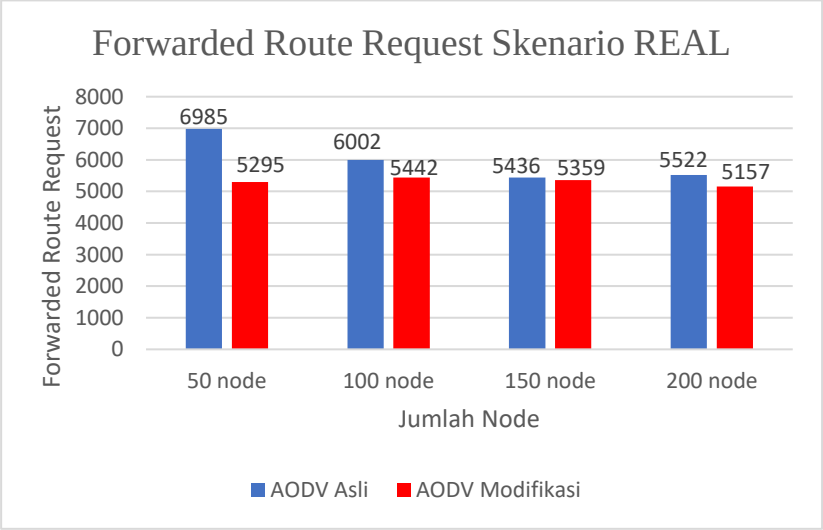
melakukan penambahan parameter *node gateway*. Semakin rendahnya nilai RO maka *packet drop* semakin rendah dikarenakan pencarian rute lebih cepat atau optimal dilakukan seiring bertambahnya kepadatan *node*.

Untuk hasil pengambilan data *forwarded route request* (RREQ F) pada scenario *real* 50 *node*, 100 *node*, 150 *node* dan 200 *node* dapat dilihat pada Tabel 5.12.

Tabel 5.12 Hasil Rata - Rata RREQ F pada Skenario Real

Jumlah *Node*	**AODV Asli**	**AODV Modifikasi**	**Perbedaan**
50	6985	5295	- 1690
100	6002	5442	- 560
150	5436	5359	- 77
200	5522	5157	- 365

Dari data di atas, dibuat grafik yang merepresentasikan hasil perhitungan RREQ F yang ditunjukkan pada Gambar 5.8.



Gambar 5.8 Grafik *Forwarded Route Request* Skenario *Real*

Berdasarkan grafik pada Gambar 5.8 dapat dilihat bahwa *routing protocol* AODV yang telah dimodifikasi dan juga *routing protocol* asli mengalami perubahan *forwarded route request* (RREQ F) yang fluktuatif. Pada lingkungan yang jarang dengan jumlah 50 *node*, menghasilkan perbedaan selisih *forwarded route request* sebesar 1690 atau mengalami penurunan sebesar 24,19%, dimana *routing protocol* AODV yang telah dimodifikasi unggul dalam hal *forwarded route request* tersebut karena menghasilkan *forwarded route request* yang lebih rendah dari *routing protocol* AODV asli. Pada lingkungan yang sedang dengan jumlah 100 *node*, menghasilkan perbedaan selisih *forwarded route request* sebesar 560 atau mengalami penurunan sebesar 9,33%, dimana *routing protocol* AODV yang telah dimodifikasi juga unggul dalam hal *forwarded route request* tersebut dari *forwarded route request* AODV asli. Pada lingkungan yang sedang dengan jumlah 150 *node*, menghasilkan perbedaan selisih *forwarded route request* sebesar 77 atau mengalami penurunan sebesar 1,42%, dimana *routing protocol* AODV modifikasi lebih unggul dalam hal *forwarded route request* tersebut dari *forwarded route request* AODV asli. Pada lingkungan yang padat dengan jumlah 200 *node*, menghasilkan perbedaan selisih *forwarded route request* sebesar 365 atau mengalami penurunan sebesar 6,61%, dimana *routing protocol* AODV yang telah dimodifikasi juga unggul dalam hal *forwarded route request* tersebut.

Dapat dilihat rata-rata penurunan yang terjadi untuk *forwarded route request* adalah sebesar 10,39% untuk hasil rata-rata dari 50 *node*, 100 *node*, 150 *node*, dan 200 *node*. Jika ketiga lingkungan tersebut dibandingkan dapat dilihat bahwa dengan jumlah *node* yang jarang, menghasilkan *forwarded route request* yang lebih bagus atau lebih sedikit daripada di lingkungan dengan jumlah *node* yang sedang maupun yang padat baik untuk *routing protocol* AODV yang telah dimodifikasi. Dapat dilihat pula bahwa AODV yang telah dimodifikasi menghasilkan *forwarded route request* yang lebih bagus atau dalam hal ini lebih rendah daripada AODV asli.

Tabel 5.13 Rute dan Durasi Penggunaan Rute pada AODV

Rute	Durasi (detik)	Paket Diterima
[48]-[42]-[9]-[12]-[1]-[24]-[49]	4,41	5
[48]-[37]-[11]-[32]-[1]-[24]-[49]	1,41	2
[48]-[14]-[39]-[4]-[28]-[24]-[49]	2,55	3
[48]-[42]-[39]-[11]-[32]-[2]-[24]-[49]	6,.37	8
[48]-[9]-[33]-[11]-[2]-[24]-[49]	5,18	6
[48]-[34]-[14]-[0]-[2]-[24]-[49]	5,39	6
[48]-[4]-[33]-[39]-[2]-[24]-[49]	6,32	7
[48]-[9]-[0]-[42]-[32]- [22]-[24]-[49]	5,33	6
[48]-[9]-[4]-[40]-[25]- [2]-[24]-[49]	1.60	3
[48]-[9]-[0]-[30]- [38]-[24]-[49]	4,24	7
[48]-[9]-[4]-[32]- [41]-[35]-[43]-[49]	12,38	14
[48]-[31]-[32]-[7]-[3]-[24]-[49]	5,71	7
[48]-[31]-[0]-[18]- [34]-[41]-[25]-[35-[43]-[49]	19,44	21
[48]-[26]-[30]-[39]- [10]-[3]-[24]-[49]	41,70	42

Tabel 5.13 merupakan hasil dari pengecekan rute pada salah satu skenario *real* di lingkungan 50 *node.* Dapat dilihat bahwa durasi bertahannya rute yang terbentuk pada AODV asli cukup singkat, durasi maksimal suatu rute hanya bertahan selama 41,70 detik.

Dengan rata-rata durasi bertahannya suatu rute selama 8,71 detik ketika pengiriman paket.

Tabel 5.14 Durasi Pengunaan Rute pada AODV-Clustering

Rute	Durasi (detik)	Paket Diterima
[48]-[37]-[39]-[35]-[1]-[24]-[49]	38,54	39
[48]-[34]-[25]-[22]-[36]-[38]-[24]-[49]	5,59	7
[48]-[40]-[44]-[39]-[32]-[35]-[24]-[49]	8,89	10
[48]-[40]-[25]-[11]-[41]-[24]-[49]	2,84	4
[48]-[31]-[18]-[32]-[36]-[38]-[24]-[49]	3,28	5
[48]-[31]-[30]-[26]-[41]-[35]-[49]	3,48	4
[48]-[31]-[18]-[32]-[36]-[38]-[24]-[49]	6,40	6
[48]-[26]-[32]-[36]-[25]-[35]-[43]-[49]	3,39	4
[48]-[26]-[30]-[39]-[38]-[24]-[49]	61,51	61

Tabel 5.14 merupakan hasil dari pengecekan rute pada salah satu skenario *real* di lingkungan 50 *node.* Dapat dilihat bahwa durasi bertahannya rute yang terbentuk pada AODV *Clustering* lebih lama daripada AODV asli dengan maksimal bertahan selama 61,51 detik dan rata-rata durasi penggunaan rute 14,49 detik ketika melakukan pengiriman paket. Dengan menggunakan AODV *Clustering,* rute yang didapatkan cenderung stabil karena hanya melewati *Cluster Head* (*node* dengan *degree* tertinggi) dan *node gateway* yang berhak meneruskan *route request.* Rute yang melakukan *packet drop* pada AODV asli sebanyak 38 paket sedangkan pada AODV *Clustering* hanya sebanyak 26 paket.

BAB VI
KESIMPULAN DAN SARAN

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

6.1 Kesimpulan

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

1. Dengan menggunakan metode *Clustering* berdasarkan *node degree*, AODV modifikasi sudah berhasil membatasi *forwarding node* yang bertugas untuk melakukan *rebroadcast* RREQ dilihat dari *Forwarded Route Request* dan durasi rute pengiriman.
2. Dampak metode AODV-*Clustering* terhadap performa protokol AODV pada skenario *grid* adalah rata – rata kenaikan *Packet Delivery Ratio* sebesar 13,41%, rata – rata penurunan *Routing Overhead* sebesar 17,85%, dan rata – rata penurunan *Forwarded Route Request* sebesar 16,36%.
3. Dampak metode AODV-*Clustering* terhadap performa protokol AODV pada skenario *real* adalah rata – rata kenaikan *Packet Delivery Ratio* sebesar 13,03%, rata – rata penurunan *Routing Overhad* sebesar 11,52%, dan rata – rata penurunan *Forwarded Route Request* sebesar 10,39%.

6.2 Saran

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

1. Menambahkan aspek lain untuk pembatasan *forwarding node* yang meneruskan paket RREQ seperti energi, tingkat kepadatan *node*, kecepatan, arah,, dan lain-lain.
2. Lebih banyak uji coba yang dilakukan untuk mendapatkan hasil yang lebih akurat,

(Halaman ini sengaja dikosongkan)

DAFTAR PUSTAKA

[1] "VANET - Vehicle Ad hoc Network," [Online]. Available: http://comp.ist.utl.pt/~rmr/WSN/CaseStudies2007-no/WSN_Transportation/. [Diakses 15 Desember 2018].

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

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

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

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

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

[7] "OpenStreetMap," [Online]. Available: https://www.openstreetmap.org/. [Diakses 18 Desember 2018].

[8] "JOSM," [Online]. Available: https://josm.openstreetmap.de/. [Diakses 18 Desember 2018].

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

[10] “AWK,” [Online]. Available: http://tldp.org/LDP/abs/html/awk.html. [Diakses 20 Desember 2018].

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

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

LAMPIRAN

A.1 Kode Fungsi CountThreshold()

```
1. void CountThreshold::handle(Event *)
2. {
3.    double now = Scheduler::instance().clock
   ();
4.    FILE *fp;
5.    double interval = 5.0;
6.    int run = now / interval;
7.    if (masuk[run]==0) masuk[run]=0;
8.    if(now > 0.000000 && masuk[run]==0)
9.    {
10.     masuk[run]=100;
11.
12.     for(int i=0;i<nodes_count;i++)
13.     {
14.      for(int j=(i+1);j<nodes_count;j++)
15.       {
16.        if(count_neighbour[i]>count_neighbo
   ur[j])
17.         {
18.          int tmp;
19.          tmp=count_neighbour[i];
20.     count_neighbour[i]=count_neighbour[j]
21.          count_neighbour[j]=tmp;
22.         }
23.       }
24.     }
25. for(int i=0;i<nodes_count;i++)
26.     {
27.      count_mode[i]=0;
```

```
28.       for(int j=0;j<nodes_count;j++)
29.       {
30.       if(count_neighbour[i]==count_neighbo
   ur[j]){
31.           count_mode[i]++;
32.        }
33.     for(int i=0;i<nodes_count;i++)
34.     {
35.       if(count_mode[i]>th)
36.       {
37.         th=count_mode[i];
38.       }
39.     }
40.     old_th = th;
41.   }
42.   Scheduler::instance().schedule(this, &in
   tr, interval);
43. }
```

A.2 Kode Fungsi nb_insert()

```
1.  void AODV::nb_insert(nsaddr_t id)
2.  {
3.    count_neighbour[index]+=1;
4.    AODV_Neighbor *nb = new AODV_Neighbor(id);
5.    assert(nb);
6.    nb->nb_expire = CURRENT_TIME +
7.                    (1.5 * ALLOWED_HELLO_LOSS *
   HELLO_INTERVAL);
8.    LIST_INSERT_HEAD(&nbhead, nb, nb_link);
9.    seqno += 2; // set of neighbors changed
10.   assert((seqno % 2) == 0);
11. }
```

A.3 Kode Fungsi nb_remove()

```
1. void AODV::nb_delete(nsaddr_t id)
2. {
3.   count_neighbour[index]-=1;
4.
5.   AODV_Neighbor *nb = nbhead.lh_first;
6.
7.   log_link_del(id);
8.   seqno += 2; // Set of neighbors changed

9.   assert((seqno % 2) == 0);
10.
11.  for (; nb; nb = nb->nb_link.le_next)
12.  {
13.    if (nb->nb_addr == id)
14.    {
15.      LIST_REMOVE(nb, nb_link);
16.      delete nb;
17.      break;
18.    }
19.  }
20.
21.  handle_link_failure(id);
22.}
23.
24.      sendReply(rq->rq_dst,
25.                rq->rq_hop_count,
26.                rq->rq_src,
27.                rq->rq_src_seqno,
28.                (u_int32_t)(rt-
   >rt_expire - CURRENT_TIME),
```

```
29.                 rq->rq_timestamp);
30.  #endif
31.
32.      Packet::free(p);
33. else
34.     {
35.       ih->saddr() = index;
36.       ih->daddr() = IP_BROADCAST;
37.       rq->rq_hop_count += 1;
38.         if (rt)
39.         rq->rq_dst_seqno = max(rt-
   >rt_seqno, rq->rq_dst_seqno);
40.
41.       forward((aodv_rt_entry *)0, p, DELAY
   );
42.     }
43.
44.}
```

A.4 Kode Fungsi CalculateCHID()

```
1. void AODV::calculateCHID()
2.   int max_degree = -
   1;
3.
4.   for (int i = 0; i < sizeof(count_neighbo
   ur) / sizeof(*count_neighbour); i++) {
5.     if (count_neighbour[i] > max_degree)
6.       max_degree = count_neighbour[i];
7.   }
8.
9.   if (count_neighbour[index] == max_degree
    {
10.    CH_ID = index;
11.  }
12.  else {
13.    CH_ID = -1;
14.  }
15.
16.  FILE *fp = fopen("debug.txt", "a");
17.  fprintf(fp, "\n %f function AODV::calcul
   ateCHID on node %d with degree: %d, max_de
   gree: %d, CH_ID: %d", CURRENT_TIME, index,
    count_neighbour[index], max_degree, CH_ID
   );
18.  fclose(fp);
19.}
20.
21.
22.int cluster_head = CH_ID;
```

```
23.if (CH_ID == -1){
24.calculateCHID();
25.     }
26.      rq->rq_cluster_head_index = CH_ID;
27.      CH_ID = cluster_head;
28.       if (CH_ID != index) {
29.rq->rq_bcast_id = CH_ID;
30.Scheduler::instance().schedule(target_, p,
    0.);
31.}
32.
33.ch->ptype() = PT_AODV;
34.       ch->size() = IP_HDR_LEN + rp-
   >size();
35.       ch->iface() = -2;
36.       ch->error() = 0;
37.       ch->addr_type() = NS_AF_INET;
38.       ch->next_hop_ = rt->rt_nexthop;
39.       ch->prev_hop_ = index; //
40.       ch->direction() = hdr_cmn::DOWN;
41.
42.sender_node->addNeighbor(receiver_node);
43.     receiver_node-
   >addNeighbor(sender_node);
44.
45.         nb_insert(rp->rp_dst);
46.
47.ih->saddr() = index;
48.ih->daddr() = IP_BROADCAST;
49.ih->sport() = RT_PORT;
50.ih->dport() = RT_PORT;
51.ih->ttl_ = 1;
```

A.5 Kode Skenario NS-2

```
set val(chan)   Channel/WirelessChannel;
set val(prop)   Propagation/TwoRayGround;
set val(netif)  Phy/WirelessPhy;
set val(mac)    Mac/802_11;
set val(ifq)    Queue/DropTail/PriQueue;
set val(ll)     LL;
set val(ant)    Antenna/OmniAntenna;
set opt(x)      1500;
set opt(y)      1500;
set val(ifqlen) 1000;
set val(nn)     200;
set val(seed)   1.0;
set val(adhocRouting)   AODV;
set val(stop)   200;
set val(cp)      "cbr200.tcl";
set val(sc)      "scenarioreal.tcl";

set ns_          [new Simulator]

# setup topography object

set topo   [new Topography]

# create trace object for ns and nam

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

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

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

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

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

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

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

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

# Define node initial position in nam

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

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

    $ns_ initial_node_pos $node_($i) 20
}

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

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

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

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

A.6 Kode Konfigurasi *Traffic*

```
set udp_(0) [new Agent/UDP]
$ns_ attach-agent $node_(198) $udp_(0)
set null_(0) [new Agent/Null]
$ns_ attach-agent $node_(199) $null_(0)
set cbr_(0) [new Application/Traffic/CBR]
$cbr_(0) set packetSize_ 512
$cbr_(0) set interval_ 1
$cbr_(0) set random_ 1
$cbr_(0) set maxpkts_ 1000
$cbr_(0) attach-agent $udp_(0)
$ns_ connect $udp_(0) $null_(0)
$ns_ at 0 "$cbr_(0) start"
$ns_ at 200.0000000000000000 "$cbr_(0)
stop"
```

A.7 Kode Skrip AWK *Packet Delivery Ratio*

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

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

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

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

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

A.8 Kode Skrip AWK Rata-Rata *End-to-End Delay*

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

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

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

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

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

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

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

A.9 Kode Skrip AWK *Routing Overhead*

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

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

A.10 Kode Skrip AWK *Forwarded Route Request*

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

B.1 Tabel Hasil Skenario *Grid* 50 *node*

		Send	Receive	Loss	PDR	Delay	RO	RREQ F
1	Original	198	142	56	0.71717	952.793	9065	5450
	Modifikasi	205	170	35	0.82927	925.919	7573	5260
2	Original	197	175	22	0.88833	1204.68	9547	5340
	Modifikasi	201	184	17	0.91542	711.621	6231	4840
3	Original	205	182	23	0.88781	481.305	9376	8700
	Modifikasi	199	180	19	0.90452	636.289	6782	5000
4	Original	206	177	29	0.85922	168.825	8411	8970
	Modifikasi	200	167	33	0.835	605.518	6748	5240
5	Original	203	168	35	0.82759	927.26	7112	6940
	Modifikasi	197	185	12	0.93909	995.366	6539	5290
6	Original	202	162	40	0.80198	1072.18	5491	6720
	Modifikasi	208	184	24	0.88462	466.985	6866	5220
7	Original	200	149	51	0.745	507.055	5643	7940
	Modifikasi	206	181	25	0.87864	851.555	6983	4980
8	Original	196	164	32	0.83674	206.077	9841	8150
	Modifikasi	201	178	23	0.88557	277.681	6769	5180
9	Original	203	108	95	0.53202	798.812	7851	5080
	Modifikasi	193	180	13	0.93264	645.236	6578	5100
10	Original	203	88	115	0.4335	827.361	13485	6530
	Modifikasi	198	188	10	0.9495	485.236	6501	5480
Rata-rata			Original		0.75293	714.635	8582.2	6982
			Modifikasi		0.89543	660.141	6757	5159

B.2 Tabel Hasil Skenario *Grid* 100 *node*

		Send	Receive	Loss	PDR	Delay	RO	RREQ F
1	Original	196	137	59	0.69898	1115.94	10020	5070
	Modifikasi	196	134	62	0.68367	2516.5	9490	4070
2	Original	205	172	33	0.83902	1002.8	8302	5260
	Modifikasi	195	172	23	0.88205	863.676	7865	4840
3	Original	197	155	42	0.7868	1900.35	12840	3820
	Modifikasi	197	172	25	0.8731	1232.02	9541	5100
4	Original	199	129	70	0.64824	1934.76	10334	7670
	Modifikasi	204	172	32	0.84314	1307.43	7774	4670
5	Original	192	93	99	0.48438	2645.18	8666	3390
	Modifikasi	194	170	24	0.87629	1317.06	8500	4450
6	Original	207	161	46	0.77778	288.476	7652	12610
	Modifikasi	201	167	34	0.83085	1006.5	7269	4980
7	Original	200	146	54	0.73	875.395	7670	5110
	Modifikasi	208	169	39	0.8125	654.171	6490	4460
8	Original	195	166	29	0.85128	973.345	11274	6240
	Modifikasi	204	169	35	0.82843	1313.59	8428	5260
9	Original	198	165	33	0.83333	410.633	8716	6250
	Modifikasi	204	193	11	0.94608	241.089	4381	4890
10	Original	200	134	66	0.67	2136.3	9380	5950
	Modifikasi	205	172	33	0.83902	1477.54	6302	4880
Rata-rata		Original			0.73198	1328.32	9485.4	6137
		Modifikasi			0.84151	1192.96	7604	4760

B.3 Tabel Hasil Skenario *Grid* 150 *node*

		Send	Receive	Loss	PDR	Delay	RO	RREQ F
1	Original	195	155	40	0.79487	1793.75	20704	7730
	Modifikasi	204	176	28	0.86275	1056.13	18719	7650
2	Original	206	173	33	0.83981	1532.02	13478	7460
	Modifikasi	194	173	21	0.89175	304.376	8477	7150
3	Original	207	140	67	0.67633	1524.86	16460	7500
	Modifikasi	206	164	42	0.79612	1383.72	16644	7630
4	Original	198	155	43	0.78283	1370.17	15372	8500
	Modifikasi	199	172	27	0.86432	1236.75	9908	7260
5	Original	197	172	25	0.8731	1090.1	18642	7690
	Modifikasi	202	176	26	0.87129	1419.8	15324	7430
6	Original	197	166	31	0.84264	656.772	17750	8370
	Modifikasi	197	165	32	0.83756	1886.03	18234	7710
7	Original	201	173	28	0.8607	515.31	14924	7360
	Modifikasi	204	179	25	0.87745	936.361	15775	7900
8	Original	196	152	44	0.77551	845.623	17134	8440
	Modifikasi	200	162	38	0.81	1838.82	15245	8110
9	Original	209	78	131	0.37321	4626.05	14870	14300
	Modifikasi	206	170	36	0.82524	1138.77	15292	8460
10	Original	203	163	40	0.80296	661.714	12748	6750
	Modifikasi	202	166	36	0.82178	1028.18	13752	8560
Rata-rata			Original		0.76219	1461.64	16208.2	8410
			Modifikasi		0.84583	1222.89	14737	7786

B.4 Tabel Hasil Skenario *Grid* 200 *node*

		Send	Receive	Loss	PDR	Delay	RO	RREQ F
1	Original	205	165	40	0.80488	577.033	18896	8170
	Modifikasi	194	170	24	0.87629	599.936	18884	7840
2	Original	205	175	30	0.85366	1503.32	24988	7990
	Modifikasi	200	160	40	0.8	1225.85	11665	6230
3	Original	201	165	36	0.8209	763.965	20524	8050
	Modifikasi	201	183	18	0.91045	875.954	12832	7700
4	Original	202	186	16	0.92079	1141.42	17490	7560
	Modifikasi	202	174	28	0.86139	1332.76	18096	8440
5	Original	204	147	57	0.72059	1983.52	18072	4350
	Modifikasi	201	152	49	0.75622	1830.7	16857	5860
6	Original	207	132	75	0.63768	1379.67	18732	7060
	Modifikasi	202	167	35	0.82673	1129.43	17731	8010
7	Original	193	118	75	0.6114	2272.67	21554	5370
	Modifikasi	200	186	14	0.93	902.809	9376	6500
8	Original	203	161	42	0.7931	1533.63	23174	6690
	Modifikasi	198	174	24	0.87879	1785.99	16571	7220
9	Original	200	146	54	0.73	1264.98	18460	7420
	Modifikasi	199	148	51	0.74372	1121.83	17311	6180
10	Original	200	166	34	0.83	1132.96	14358	8790
	Modifikasi	204	167	37	0.81863	2325.67	15286	7100
Rata-rata			Original		0.7723	1355.32	19624.8	7145
			Modifikasi		0.84022	1313.09	15460.9	7108

B.5 Tabel Hasil Skenario *Real* 50 *node*

		Send	Receive	Loss	PDR	Delay	RO	RREQ F
1	Original	202	168	34	0.83168	838.448	7789	6530
	Modifikasi	197	160	37	0.81218	624.308	8433	5840
2	Original	199	69	130	0.34673	248.218	5150	1030
	Modifikasi	202	166	36	0.82178	323.995	8150	5440
3	Original	195	174	21	0.89231	97.0136	7327	5200
	Modifikasi	205	191	14	0.93171	150.949	7964	5650
4	Original	196	158	38	0.80612	569.018	7455	6780
	Modifikasi	196	170	26	0.86735	896.024	7878	4720
5	Original	210	72	138	0.34286	2019.41	10585	15900
	Modifikasi	199	101	98	0.50754	2681.35	8097	5430
6	Original	204	88	116	0.43137	2343.32	5447	4200
	Modifikasi	200	95	105	0.475	2670.7	8626	4180
7	Original	202	117	85	0.57921	1634.76	13940	12820
	Modifikasi	199	133	66	0.66834	1679.13	7923	3680
8	Original	201	187	14	0.93035	308.148	7615	5180
	Modifikasi	210	189	21	0.9	353.951	8564	6390
9	Original	198	167	31	0.84343	146.589	8656	6730
	Modifikasi	199	148	51	0.74372	798.552	8653	5010
10	Original	201	173	28	0.8607	463.865	8945	5480
	Modifikasi	205	173	32	0.8439	531.226	8389	6610
Rata-rata			Original		0.68648	866.879	8290.9	6985
			Modifikasi		0.75715	1071.02	8267.7	5295

B.6 Tabel Hasil Skenario *Real* 100 *node*

		Send	Receive	Loss	PDR	Delay	RO	RREQ F
1	Original	199	160	39	0.80402	546.642	10224	4710
	Modifikasi	194	149	45	0.76804	807.421	13565	6630
2	Original	198	144	54	0.72727	1080.88	13162	4780
	Modifikasi	202	153	49	0.75743	1746.71	11084	5940
3	Original	202	142	60	0.70297	1330.82	8518	5790
	Modifikasi	199	151	48	0.75879	2279.36	11088	6050
4	Original	200	169	31	0.845	824.139	10264	5840
	Modifikasi	199	165	34	0.82915	533.927	6665	4830
5	Original	195	163	32	0.8359	836.951	8904	5400
	Modifikasi	204	174	30	0.85294	784.941	8799	5540
6	Original	203	162	41	0.79803	430.654	10788	10790
	Modifikasi	204	176	28	0.86275	945.545	9382	5480
7	Original	201	142	59	0.70647	1821.35	8716	6500
	Modifikasi	199	154	45	0.77387	1299.51	5833	4580
8	Original	195	142	53	0.72821	1558.15	8534	4460
	Modifikasi	196	172	24	0.87755	235.479	6653	5380
9	Original	195	161	34	0.82564	775.115	9734	6390
	Modifikasi	201	171	30	0.85075	1172.47	6204	4210
10	Original	201	162	39	0.80597	1928.75	7444	5360
	Modifikasi	199	169	30	0.84925	941.101	9966	5780
Rata-rata			Original		0.77795	1113.35	9628.8	6002
			Modifikasi		0.81805	1074.65	8923.9	5442

B.7 Tabel Hasil Skenario *Real* 150 *node*

		Send	Receive	Loss	PDR	Delay	RO	RREQ F
1	Original	201	112	89	0.55721	966.295	12798	5780
	Modifikasi	197	166	31	0.84264	681.201	6318	4800
2	Original	199	155	44	0.77889	1602.72	17428	4310
	Modifikasi	194	150	44	0.7732	2466.58	10369	5190
3	Original	201	167	34	0.83085	901.226	14176	6440
	Modifikasi	195	171	24	0.87692	944.44	10351	5660
4	Original	205	103	102	0.50244	1315.03	15626	5380
	Modifikasi	204	165	39	0.80882	1141.29	14162	6550
5	Original	202	158	44	0.78218	913.809	16030	9070
	Modifikasi	202	187	15	0.92574	491.029	8591	5490
6	Original	206	164	42	0.79612	614.236	13252	3470
	Modifikasi	198	157	41	0.79293	1416.83	10957	5650
7	Original	198	142	56	0.71717	2004.31	14148	4740
	Modifikasi	193	165	28	0.85492	800.417	12895	5290
8	Original	200	150	50	0.75	2192.72	9418	4420
	Modifikasi	209	179	30	0.85646	928.235	10476	5310
9	Original	199	74	125	0.37186	1140.52	13140	6030
	Modifikasi	208	141	67	0.67789	461.654	14204	4000
10	Original	197	144	53	0.73096	1466.85	13168	4720
	Modifikasi	202	154	48	0.76238	1090.25	8856	5650
Rata-rata			Original		0.68177	1311.77	13918.4	5436
			Modifikasi		0.81719	1042.19	10717.9	5359

B.8 Tabel Hasil Skenario *Real* 200 *node*

		Send	Receive	Loss	PDR	Delay	RO	RREQ F
1	Original	204	166	38	0.81373	474.872	21498	5200
	Modifikasi	200	166	34	0.83	1104.28	11109	4470
2	Original	197	165	32	0.83756	748.521	10588	6650
	Modifikasi	209	184	25	0.88038	1203.36	13567	4610
3	Original	201	180	21	0.89552	949.353	14872	4990
	Modifikasi	202	165	37	0.81683	1555.46	17220	4520
4	Original	197	144	53	0.73096	681.538	14658	5900
	Modifikasi	195	173	22	0.88718	782.007	14878	5510
5	Original	198	138	60	0.69697	1757.33	19848	5790
	Modifikasi	202	183	19	0.90594	422.738	19374	6450
6	Original	201	136	65	0.67662	1043.11	20990	6600
	Modifikasi	192	176	16	0.91667	985.054	16643	5810
7	Original	200	170	30	0.85	1691.78	21108	5620
	Modifikasi	199	174	25	0.87437	237.252	9414	4840
8	Original	202	107	95	0.5297	1357.18	14922	5110
	Modifikasi	201	164	37	0.81592	866.539	13138	4650
9	Original	201	143	58	0.71144	1214.32	12528	4350
	Modifikasi	198	178	20	0.89899	1204.02	16510	5310
10	Original	196	150	46	0.76531	1184.47	23170	5010
	Modifikasi	194	183	11	0.9433	963.39	15337	5400
	Rata-rata		Original		0.75078	1110.25	17418.2	5522
			Modifikasi		0.87696	932.41	14719	5157

BIODATA PENULIS

Naufal Pranasetyo Fodensi, lahir di Jakarta, 4 Desember 1997. Penulis adalah anak pertama dari lima bersaudara. Penulis menempuh pendidikan sekolah dasar di SD Putra 1 Jakarta lalu melanjutkan pendidikan sekolah menengah pertama di SMP Islam PB. Soedirman Jakarta dan penulis menempuh pendidikan menengah atas di SMA Negeri 62 Jakarta. Selanjutnya penulis melanjutkan pendidikan sarjana di Departemen Informatika, Fakultas Teknologi Informasi dan Komunikasi, Institut Teknologi Sepuluh Nopember Surabaya. Selama kuliah, penulis aktif dalam berbagai organisasi baik tingkat jurusan maupun universitas.

Dalam menyelesaikan pendidikan S1, penulis mengambil bidang minat Arsitektur dan Jaringan Komputer (AJK). Sebagai mahasiswa, penulis berperan aktif dalam beberapa organisasi kampus seperti staf ahli Hubungan Luar Himpunan Mahasiswa Teknik-Computer (HMTC) ITS, staf ahli *Student Resource Development* Badan Eksekutif Mahasiswa FTIf *ITS* , staf Wahana Teknologi pada acara ITS EXPO 2016, dan *Committee* dalam acara *International Conference on Information & Communication Technology and System* (ICTS) 2017. Selain itu, penulis juga menjadi staf Perlengkapan dan Transportasi SCHEMATICS 2016 dan koordinator Kamzin pada acara SCHEMATICS 2017. Penulis pernah melakukan kerja praktik di Otoritas Jasa Keuangan Jakarta pada Juni – Agustus 2018 dan membuat aplikasi berbasis *web* FTP Monitoring Pertukaran Data SRO. Penulis dapat dihubungi melalui nomor *handphone*: 081266268097 atau *email*: naufalpf@gmail.com.