TUGAS AKHIR - KI141502

Implementasi Preferred Group Broadcasting pada protokol Ad hoc On-Demand Distance Vector untuk mengatasi Broadcast Storm dalam Proses Route Discovery

RAGA KRILIDO OKTANANDA
NRP 5112100064

Dosen Pembimbing I
Dr.Eng. RADITYO ANGGORO, S.Kom., M.Sc.

Dosen Pembimbing II
TOHARI AHMAD, S.Kom., MIT., Ph.D.

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2016

[Halaman ini sengaja dikosongkan]

TUGAS AKHIR - KI1502

IMPLEMENTASI PREFERRED GROUP BROADCASTING PADA PROTOKOL AD HOC ON-DEMAND DISTANCE VECTOR UNTUK MENGATASI BROADCAST STORM DALAM PROSES ROUTE DISCOVERY

RAGA KRILIDO OKTANANDA
NRP 5112100064

Dosen Pembimbing I
Dr.Eng. RADITYO ANGGORO, S.Kom., M.Sc.

Dosen Pembimbing II
TOHARI AHMAD, S.Kom., MIT., Ph.D.

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2016

[Halaman ini sengaja dikosongkan]

UNDERGRADUATE THESIS - KI1502

IMPLEMENTATION OF THE PREFERRED GROUP BROADCASTING ON PROTOCOL AD HOC ON-DEMAND DISTANCE VECTOR TO OVERCOME THE BROADCAST STORM IN THE PROCESS OF ROUTE DISCOVERY

RAGA KRILIDO OKTANANDA
NRP 5112100064

Supervisor I
Dr.Eng. RADITYO ANGGORO, S.Kom., M.Sc.

Supervisor II
TOHARI AHMAD, S.Kom., MIT., Ph.D.

DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY

[Halaman ini sengaja dikosongkan]

LEMBAR PENGESAHAN

IMPLEMENTASI PREFERRED GROUP BROADCASTING PADA PROTOKOL AD HOC ON-DEMAND DISTANCE VECTOR UNTUK MENGATASI BROADCAST STORM DALAM PROSES ROUTE DISCOVERY
TUGAS AKHIR

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

Oleh
RAGA KRILIDO OKTANANDA
NRP : 5112 100 064

Disetujui oleh Dosen Pembimbing Tugas Akhir:

1. Dr.Eng. Radityo Anggoro, S.Kom., M.Sc. .....
   NIP: 19841016200812100 (Pembimbing 1)

2. Tohari Ahmad, S.Kom., MIT., Ph.D .....
   NIP: 197505252003121002 (Pembimbing 2)

SURABAYA
JUNI, 2016

[Halaman ini sengaja dikosongkan]

IMPLEMENTASI PREFERRED GROUP BROADCASTING PADA PROTOKOL AD HOC ON-DEMAND DISTANCE VECTOR UNTUK MENGATASI BROADCAST STORM DALAM PROSES ROUTE DISCOVERY

Nama Mahasiswa	**: RAGA KRILIDO OKTANANDA**
NRP	**: 5112100064**
Jurusan	**: Teknik Informatika FTIF-ITS**
Dosen Pembimbing 1	**: Dr.Eng. Radityo Anggoro, S.Kom., M.Sc**
Dosen Pembimbing 2	**: Tohari Ahmad, S.Kom., MIT., Ph.D.**

Abstrak

Saat ini perkembangan teknologi Vehicular Ad Hoc Networks (VANETs) sudah semakin pesat. Mulai dari penggunaan pada kendaraan, maupun beberapa peralatan yang lain. Namun sampai saat ini penggunaan VANETs masih terkendala dengan metode routing mana yang paling efektif dan paling efisien. Salah satu metode routing yang paling sering digunakan adalah metode routing AODV. Namun terdapat kelemahan pada metode ini yakni RREQ dan RREP akan meningkat jika dalam kondisi mobilitas tinggi hingga akan terjadi broadcast storm.

Penambahan algoritma PGB pada protokol AODV bertujuan untuk mengatasi permasalahan broadcast storm pada proses RREQ. Hasil menunjukkan bahwa protokol AODV+PGB membuat nilai dari jumlah paket dari protokol yang dikirim lebih sedikit dari jumlah paket dari protokol AODV dengan nilai 17427 untuk AODV dan 6529 untuk AODV+PGB.

Kata kunci: AODV, PGB, VANET, routing protokol

[Halaman ini sengaja dikosongkan]

IMPLEMENTATION OF THE PREFERRED GROUP BROADCASTING ON PROTOCOL AD HOC ON-DEMAND DISTANCE VECTOR TO OVERCOME THE BROADCAST STORM IN THE PROCESS OF ROUTE DISCOVERY

Student's Name : **RAGA KRILIDO OKTANANDA**
Student's ID : **5112100064**
Department : **Teknik Informatika FTIF-ITS**
First Advisor : **Dr.Eng. Radityo Anggoro, S.Kom., M.Sc**
Second Advisor : **Tohari Ahmad, S.Kom., MIT., Ph.D.**

Abstract

Nowadays, Vehicular Ad Hoc Networks (VANETs) Technology has been developed incredibly fast. Starting from the implementation in vechicles and on other tools. Unfortunately, there is a problem in using VANETs about choosing the most effective and efficient routing methode. One of routing method that is commonly used is AODV routing methode. But there are some weakness found in this method, the value of RREQ and RREP will increase if it running in high-mobility condition until it occurs broadcast storm.

The addition of PGB's algorithm on AODV protocol is designed to solve the problem of broadcast storm in RREQ process. The result show that the protocol of AODV+PGB has less value of package than AODV's protocol with value 17427 for AODV and 6529 for AODV+PGB.

Keyword: AODV, PGB, VANET, routing protokol

[Halaman ini sengaja dikosongkan]

KATA PENGANTAR

Alhamdulillahirabbil'alamin, segala puji bagi Allah SWT, yang telah melimpahkan rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul ***"IMPLEMENTASI PREFERRED GROUP BROADCASTING PADA PROTOKOL AD HOC ON-DEMAND DISTANCE VECTOR UNTUK MENGATASI BROADCAST STORM DALAM PROSES ROUTE DISCOVERY".*** Shalawat serta salam selalu senantiasa saya tujukan kepada Rasulullah SAW, selaku inspirator dunia nomor satu saat ini, hingga akhir zaman.

Pengerjaan Tugas Akhir ini merupakan salah satu dari sekian banyak kesempatan yang saya dapatkan, untuk mendapatkan ilmu dan pengalaman berharga selama saya berada di kampus Teknik Informatika ITS ini. Dengan pengerjaan Tugas Akhir ini, saya menjadi semakin bisa untuk memanajemen waktu dan memanajemen diri sendiri, sehingga Tugas Akhir ini dapat selesai tepat waktu.

Selesainya Tugas Akhir ini tidak lepas dari bantuan dan dukungan beberapa pihak. Sehingga pada kesempatan ini penulis mengucapkan syukur dan terima kasih kepada:

1. Allah SWT dan Nabi Muhammad SAW.
2. Ibu, Ibu, Ibu, Ayah dan Adik yang selalu memberikan do'a, dukungan, serta motivasi, sehingga penulis selalu termotivasi untuk menyelesaikan Tugas Akhir.
3. Bapak Dr.Eng. Radityo Anggoro, S.Kom., M.Sc, selaku pembimbing I yang selalu menyemangati dan memotivasi dengan ilmu-ilmu yang diluar dugaan saya.
4. Bapak Tohari Ahmad, S.Kom., MIT., Ph.D selaku pembimbing II yang telah mengoreksi buku ini dengan cermat.
5. Ipul, Icang, Aried, Indra, Hafiz, Djuned, Fadrian, Miftah, Ardhana, dan semua teman-teman penghuni Lab. DTK atas suntikan semangatnya tiap hari.
6. Reyhan, Bila, Prad, Randy, mas Dimyati dan semua anak bimbing Pak Onggo atas bantuan dalam pengerjaan TA.

7. Nuning Septiana yang telah membantu dalam pengerjaan Buku Tugas Akhir ini.
8. Serta semua pihak yang yang telah turut membantu penulis dalam menyelesaikan Tugas Akhir ini.

Penulis menyadari bahwa Tugas Akhir ini masih memiliki banyak kekurangan. Sehingga dengan kerendahan hati, penulis mengharapkan kritik dan saran dari pembaca untuk perbaikan ke depannya.

Surabaya, Juni 2016

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 *Vehicular Ad-Hoc Networks* (VANETs) sudah semakin pesat. Mulai dari penggunaan pada kendaraan, maupun beberapa peralatan yang lain. Namun sampai saat ini penggunaan jaringan VANETs masih terkendala dengan metode *routing* mana yang paling efektif dan paling efisien. Salah satu metode routing yang paling sering digunakan adalah AODV. Namun terdapat kelemahan pada metode ini yakni RREQ dan RREP akan meningkat jika dalam kondisi mobilitas tinggi hingga akan terjadi *broadcast storm*. Kondisi tersebut akan mempengaruhi kinerja dari protokol routing akibat adanya *packet redundancy*, *contention*, dan *collision*.

Solusinya adalah dengan membatasi *node* yang akan melakukan *broadcasting*. *Preferred Group Broadcasting* (PGB) adalah sebuah mekanisme *broadcasting* yang bisa digunakan untuk mengatasi permasalahan *broadcast storm.* PGB akan melakukan pembatasan pada *node* mana yang akan melakukan *broadcast*. Pembatasan dan pemilihan dilakukan berdasarkan jarak *node* dengan *node* pengirim dan kuat sinyal *node* yang ditangkap oleh *node* pengirim. PGB akan membuat tiga grup yaitu *Preferred group* (PG), *IN group*, dan *OUT group.* PG adalah kumpulan *node* yang akan melakukan *broadcasting*. *IN group* adalah grup yang memiliki sinyal yang lebih kuat dari pada PG dan *OUT group* adalah grup yang memiliki sinyal yang lebih lemah dari pada PG.

Tugas akhir ini mengimplemantasikan metode *routing* AODV dengan metode *broadcasting* PGB sehingga metode *broadcasting* yang telah dimiliki oleh AODV akan digantikan oleh PGB.

1.2 Rumusan Masalah

Rumusan masalah yang diangkat dalam tugas akhir ini dapat dipaparkan sebagai berikut:

1. Bagaimana cara untuk mencegah *Broadcasting Storm* pada AODV?
2. Bagaimana mengimplementasikan metode PGB kedalam metode routing AODV pada jaringan VANETs.

1.3 Batasan Masalah

Permasalahan yang dibahas dalam tugas akhir ini memiliki beberapa batasan antara lain:

1. Simulator yang digunakan adalah simulator NS2.
2. Metode routing yang digunakan adalah metode AODV.
3. Metode yang digunakan untuk mengatasi *Broadcast Storm problem* adalah PGB.

1.4 Tujuan

Tujuan dari Tugas Akhir ini adalah mengimplementasi metode routing AODV dengan metode *broadcasting* PGB dalam simulator NS2.

1.5 Manfaat

Manfaat dari hasil pembuatan Tugas Akhir ini yaitu untuk pengembangan metode *routing* AODV dengan mengimplementasikan metode *routing* AODV dengan metode *broadcasting* PGB pada simulator NS2

1.6 Metodologi

1. Penyusunan proposal tugas akhir
   Penyusunan Proposal Tugas Akhir ini merupakan tahap awal dalam proses pengerjaan Tugas Akhir. Dalam proposal ini mengimplementasikan algoritma *broadcasting Preferred Group Broadcasting* pada metode *routing Ad hoc On-Demand Distance Vector.*
2. Studi literatur
   Tugas Akhir ini menggunakan literatur *paper*. *Paper* yang digunakan adalah *"An Evaluation of Inter-Vehicle Ad Hoc Networks Based on Realistic Vehicular Traces"*. *Paper* tersebut menjelaskan mengenai implementasi algoritma

broadcasting Preferred Group Broadcasting pada metode *routing Ad hoc On-Demand Distance Vector.*

3. Implementasi
   Implementasi merupakan tahap untuk membangun algoritma yang akan digunakan. Pada tahap ini dilakukan implementasi dari metode PGB dan metode *routing* AODV untuk mengatasi masalah *broadcasting strom* dengan menggunakan simulator NS2 [1].
4. Pengujian dan evaluasi
   Pengujian dilakukan dengan cara menjalankan simulasi dengan beberapa paremeter pada NS2 dengan metode *routing* AODV dan AODV + PGB. Evaluasi dilakukan dengan membandingkan hasil simulasi AODV dan AODV + PGB. Beberapa parameter yang digunakan adalah:
   1. PDR (*packet delivery ratio*)
      PDR adalah jumlah paket data yag diterima dibagi dengan jumlah paket data yang dikirimkan
   2. *End-to-end delay*
      End-to-end delay adalah waktu kedatangan paket data pada destination dikurangi waktu paket data yang dikirimkan oleh source
   3. *Routing Overhead*
      Routing Overhead adalah jumlah pesan AODV yang ditransmisikan.
5. Penyusunan Buku Tugas Akhir
   Pada tahap ini dilakukan penyusunan laporan yang menjelaskan dasar teori dan metode yang digunakan dalam tugas akhir ini serta hasil dari implementasi aplikasi perangkat lunak yang telah dibuat.

1.7 Sistematika Penulisan Laporan Tugas Akhir

Buku Tugas Akhir ini bertujuan untuk mendapatkan gambaran dari pengerjaan Tugas Akhir ini. Selain itu, diharapkan dapat berguna untuk pembaca yang tertarik untuk melakukan

pengembangan lebih lanjut. Secara garis besar, buku Tugas Akhir terdiri atas beberapa bagian seperti berikut ini:

Bab I Pendahuluan

Bab yang berisi mengenai latar belakang, tujuan, dan manfaat dari pembuatan Tugas Akhir. Selain itu permasalahan, batasan masalah, metodologi yang digunakan, dan sistematika penulisan juga merupakan bagian dari bab ini.

Bab II Dasar Teori

Bab ini berisi penjelasan secara detail mengenai dasar-dasar penunjang dan teori-teori yang digunakan untuk mendukung pembuatan Tugas Akhir ini.

Bab III Perancangan Perangkat Lunak

Bab ini berisi tentang desain sistem yang disajikan dalam bentuk pseudocode.

Bab IV Implementasi

Bab ini membahas implementasi dari desain yang telah dibuat pada bab sebelumnya. Penjelasan berupa code yang digunakan untuk proses implementasi.

Bab V Uji Coba Dan Evaluasi

Bab ini menjelaskan kemampuan perangkat lunak dengan melakukan pengujian kebenaran dan pengujian kinerja dari sistem yang telah dibuat.

Bab VI Kesimpulan Dan Saran

Bab ini merupakan bab terakhir yang menyampaikan kesimpulan dari hasil uji coba yang dilakukan dan saran untuk pengembangan perangkat lunak ke depannya.

BAB II
TINJAUAN PUSTAKA

Bab ini berisi penjelasan teori-teori yang berkaitan dengan rancangan alat yang diajukan pada pengimplementasian program. Penjelasan ini bertujuan untuk memberikan gambaran secara umum terhadap alat yang dirancang dan berguna sebagai penunjang dalam pengembangan perangkat lunak.

2.1. VANET (*Vehicular Ad hoc Network*)

Mobile ad hoc network (MANET) adalah sekumpulan mobile *node*s yang secara dinamis lokasinya sering berubah sedemikian rupa hingga interkoneksi antar *node*s mampu mengatasi perubahan yang terjadi secara terus menerus. Routing protocol digunakan untuk menemukan rute antar *node*s agar kominikasi antar jaringan bisa terfasilitasi. Tujuan utama dalam ad-hoc network routing protocol adalah membetulkan dan meng-efisisensikan rute yang sudah ada diantara pasangan *node*s sehingga pesan bisa dikirim dengan waktu yang lebih singkat. Konstruksi route didesain dengan overhead dan konsumsi bandwith minimal [2].

VANET berasal dari MANET. Namun VANET dikussuskan pada penggunaan di kendaraan. Sebuah jaringan terorganisir yang dibentuk dengan menghubungkan kendaraan dan RSU (Roadside Unit) disebut Vehicular Ad Hoc Network (VANET), dan RSU lebih lanjut terhubung ke jaringan backbone berkecepatan tinggi melalui koneksi jaringan. Kepentingan peningkatan baru-baru ini telah diajukan pada aplikasi melalui V2V (Vehicle to Vehicle) dan V2I (Vehicle to Infrastructure) komunikasi, bertujuan untuk meningkatkan keselamatan mengemudi dan manajemen lalu lintas sementara menyediakan driver dan penumpang dengan akses Internet. Dalam vanets, rsus dapat memberikan bantuan dalam menemukan fasilitas seperti restoran dan pompa bensin, dan melakukan *broadcast* pesan yang terkait seperti (maksimum

kurva kecepatan) pemberitahuan untuk memberikan pengendara informasi. Sebagai contoh, sebuah kendaraan dapat berkomunikasi dengan lampu lalu lintas cahaya melalui V2I komunikasi, dan lampu lalu lintas dapat menunjukkan ke kendaraan ketika keadaan lampu ke kuning atau merah. Ini dapat berfungsi sebagai tanda pemberitahuan kepada pengemudi, dan akan sangat membantu para pengendara ketika mereka sedang berkendara selama kondisi cuaca musim dingin atau di daerah asing. Hal ini dapat mengurangi terjadinya kecelakaan. Melalui komunikasi V2V, pengendara bisa mendapatkan informasi yang lebih baik dan mengambil tindakan

Awal untuk menanggapi situasi yang abnormal. Untuk mencapai hal ini, suatu OBU (On-Boards Unit) secara teratur menyiarkan pesan yang terkait dengan informasi dari posisi pengendara, waktu saat ini, arah mengemudi, kecepatan, status rem, sudut kemudi, 6 lampu sen, percepatan / perlambatan, kondisi lalu lintas.

2.2. AODV (*Ad hoc On-Demand Distance Vector*)

Ad hoc On-Demand Distance Vector merupakan sebuah metode penentu jalur (*routing*) pada jaringan *ad hoc mobile.* AODV dapat digunakan pada metode *routing unicast* maupun *multicast*. AODV menggunakan *on demand routing* dimana pembuatan jalur hanya dilakukan jika diminta oleh *node* asal. AODV memiliki mekanisme *route discovery* yang terdiri dari *Route Request* (RREQ) dan *Route Replay* (RREP) dan *route maintenance* yang terdiri dari Data, *Route update* dan *Route Error* (RRER) [3].

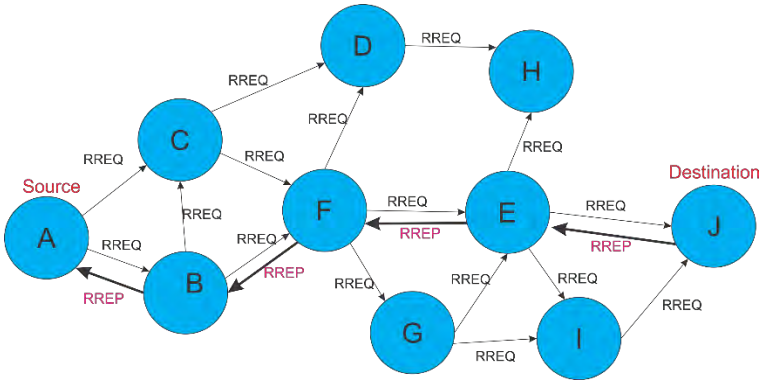
AODV memiliki sebuah tabel *routing* yang berisi:

- *Destination IP Address*: berisi alamat IP dari *node* tujuan yang digunakan untuk menentukan rute.
- *Destination Sequence Number*: destination sequence number bekerjasama untuk menentukan rute

- *Next Hop*: 'Loncatan' (hop) berikutnya, bisa berupa tujuan atau *node* tengah, field ini dirancang untuk meneruskan paket ke *node* tujuan.
- *Hop Count*: Jumlah hop dari alamat IP sumber sampai ke alamat IP tujuan.
- *Lifetime*: Waktu dalam milidetik yang digunakan untuk *node* menerima RREP.
- *Routing Flags*: Status sebuah rute; up (valid), down (tidak valid) atau sedang diperbaiki.

Protokol ini akan membuat sebuah *route* hanya jika diminta oleh *source*. *Source* akan menyebarkan RREQ pada *node* terdekat. *Node* yang menerima RREQ akan meneruskan RREQ hingga ditemukan *node Destination*. Setelah ditemukan, *Destination* akan mengirim kan RREP melalui *node-node* yang dilalui oleh RREQ hingga sampai pada *Source* [3].

Pada proses *route discovery* paket RREQ yang diterima pada suatu *node* akan dilakukan pengecekan terlebih dahulu apakah pernah disebar apa belum. Jika sudah maka paket RREQ akan di *drop*. Namun jika belum maka akan dilakukan proses *broadcast.* Gambaran proses *route discovery* bisa dilihat pada Gambar 2.1.



Gambar 2.1 Proses *route discovery* pada AODV

Pada Gambar 2.1 hampir semua *node* melakukan penyebaran. Hal ini membuat satu *node* bisa menerima lebih dari satu paket RREQ yang sama. Dalam satu area dengan jumlah kendaraan/*node* yang banyak tentunya akan menciptakan lalu lintas paket RREQ yang lebih banyak. Kondisi dimana setiap *node* melakukan peneybaran paket RREQ dan satu *node* memperoleh lebih dari satu paket RREQ yang sama disebut sebagai *broadcast storm*. Dalam tugas akhir ini, akan diimplementasikan metode untuk mengatasi permasalahan *broadcast storm* tersebut.

2.3. PGB (*Preferred Group Broadcasting*)

Preferred Group Broadcasting atau disingkat dengan PGB adalah sebuah *broadcasting mechanism* yang bertujuan untuk mereduksi *broadcast overhead* [4]. PGB akan mengurangi tranmisi yang redundan untuk menemukan *route* yang stabil dengan kemampuan *auto-correct* [1]. PGB membuat *node* penerima menentukan apakah akan melakukan *broadcast* ataukah membiarkan saja. PGB akan membuat tiga grup yaitu *Preferred group* (PG), *IN group*, dan *OUT group.* PG adalah kumpulan *node* yang akan melakukan *broadcasting*. *IN group* adalah grup yang memiliki sinyal yang lebih kuat dari pada PG dan *OUT group* adalah grup yang memiliki sinyal yang lebih lemah dari pada PG. Ilustrasi pembagian grup bisa dilihat pada Gambar 2.2 [1].

Pembagian ini bertujuan untuk menentukan besaran delay yang akan digunakan kepada masing-masing paket. Setiap grup akan memiliki delay masing-masing bergantung pada posisi dan *signal strength* yang diterima oleh *node* penerima dari *node* pengirim. Penghitungan delay setiap grup berbeda sesuai dengan posisi mereka.

Persamaan untuk penghitungan delay bisa dilihat pada

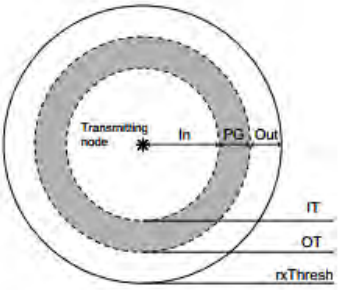
$$\boldsymbol{hoTime = T_i + jitterT_i} \quad (2.1)$$

$$\Delta \boldsymbol{P_i} = \left| |\boldsymbol{rxTh - rxP}| - \boldsymbol{f_i} \right| \quad (2.2)$$

$$\boldsymbol{T_i = \Delta P_i . dt_i + minT_i} \quad (2.3)$$

$$jitterT_i \in [0, dt_i)$$

Pada Tugas Akhir ini algoritma PGB akan digunakan untuk mengatasi masalah *broadcast storm* dalam proses *route discovery*. Metode yang digunakan yaitu dengan mengatur delay pada setiap paket yang akan dikirim. Hal ini memanfaatkan sifat dari NS2 dimana setiap paket atau setiap aktivitas yang dilakukan akan dianggap sebagai sebuah even.

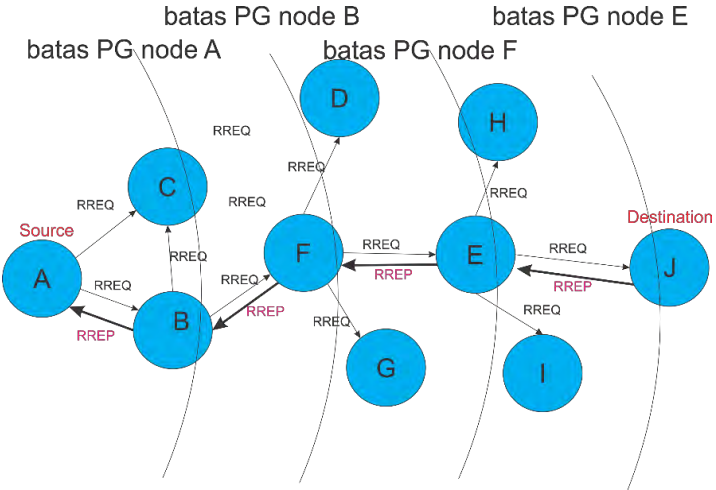


Gambar 2.2 Pembagian grup pada PGB [1]

2.4. AODV + PGB

AODV+PGB merupakan modifikasi dari *routing protocol* AODV yang digabung dengan mekanisme *broadcasting mechanism* PGB. Perubahan yang dilakukan akan diterpkan pada mekanisme broadcast RREQ dengan menerapkan pembagian grup seperti grup PGB dimana bertujuan untuk membatasi *node* mana yang akan melakukan proses *rebroadcast*. Proses *rebroadcast* RREQ akan dilakukan oleh *Preferred group* (PG). Gambaran proses *route discovery* protokol AODV+PGB bisa dilihat pada Gambar 2.3.

Proses pembagian berdasarkan kekuatan signal yang didapat oleh *node*. Setiap grup akan diberikan *delay* yang berbeda. Pemberian delay berdasarakan persamaan 2.1 [1]. *Delay* yang diberikan juga berasal dari kekuatan signal *node* tersebut. Pada persamaan 2.2 telah dijabarkan mengena pengaruh kekuatan signal dengan nilai *delay* yang diberikan.



Gambar 2.3 Proses *route discovery* pada AODV+PGB

Dalam kondisi tertentu, kemungkinan tidak terdapat satu *node* pun yang berada pada posisi PG dan hanya terdapat pada area IN dan OUT. Dalam kondisi seperti itu, nilai delay yang ada akan menuntukan *node* pada posisi mana yang akan melakukan *rebroadcast* menggantikan *node* yang berasal dari area PG. Pada implementasi ini, prioritas setelah *node* PG adalah *node* yang berasal dari area OUT lalu baru area IN.

2.5. Simulation of Urban Mobility (SUMO)

SUMO adalah program aplikasi simulator yang digunakan untuk menciptakan lingkungan bergerak yang bersifat dinamis [5]. SUMO merupakan program yang bersifat *free* dan *open-source*. SUMO dikembangkan pertama kali pada tahun 2000 dengan tujuan untuk membantu penelitian yang memerlukan pergerakan kendaran pada jalan raya.

SUMO meliki banyak sekali tools yang memiliki fungsi masing-masing. Pada tugas akhir ini terdapat bebrpa tools yang digunakan antara lain:

- Netgenerate
  Netgenerate merupakan tools untuk membuat peta dengan bentuk grid, spider, atau random. Netgenarte akan menntukan kecepatan kendaraan serta membuat *traffic light* pada map yang dibentuk. Hasil dari negenerate adalah sebuah berkas berekstensi .net.xml. Pada Tugas Akhir ini, netganerate digunakan untuk membuat skenario dengan bentuk peta grid.
- Netconvert
  Netconvert merupakan *tools* dari sumo untuk membantu mengkonversi peta dari bentuk *.osm kedalam bentuk *.map.net.xml. Berkas hasil edit dari JOSM akan dikonfersi kedalam bentuk *.map.net.xml oleh netconvert untuk selanjutnya dikonversi menjadi tcl. Pada tugas akhir ini, Netconvert digunakan untuk mengubah *.osm kedalam bentuk *.map.net.xml.
- randomTrips.py
  tools ini digunakan untuk membuat *node* awal dan akhir pada peta. Hasil dari randomTrips.py berupa berkas berekstensi *.trips.xml. Pada tugas akhir ini tools ini digunakan untuk membat *node* pada peta grid dan Surabaya.
- duarouter
  duarouter digunakan untuk memberikan arah pada *node* hasil dari randomTrips.py. Duarouter secara default menggunakan algoritm Djikstra. Hasil dari duarouter befupa berkas dengan ekstensi *.rou.xml.
- sumo-gui
  sumo-gui berfungsi untuk menampilkan pergerakan setiap *node* yang telah dibuat sebelumnya. Sumo-gui menggabungkan peta yang memiliki ekstensi .net.xml dengan routenya .rou.xml dalam sebuah berkas berkestensi *.sumocfg.

- Sumo
  Sumo digunakan untuk membuat berkas berekstensi .xml gabungan dari dua berkas sebelumnya yakni berkas berkestensi *.net.xml dan *.rou.xml.
- traceExporter.py
  traceExporter.py digunakan untuk menghasilkan berkas berkestensi *.tcl dari berkas .xml yang telah dihasilkan oleh sumo sebelumnya.

2.6. OpenStreetMap

Openstreetmap (OSM) adalah proyek nirlaba yang dilakukan oleh OpenStreetMap Foundation yang berada di United Kungdom. OpenStreetMap Project berbasis di OpenStreetMap.org, adalah upaya pemetaan seluruh dunia yang mencakup lebih dari dua juta relawan di seluruh dunia. Sifat dari openstreetmap adalah Open Data sehingga setiap data yang terdapat pada OpensStreetMap dapat digunakan dan diedut secara bebas untuk keperluan apapun termasuk keperluan navigasi [6].

Melalui Open Data Commons Open Database License 1.0, 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 bebas. Di sisi lain, baik peta dasar OSM maupun data yang tersedia di dalamnya dapat diunduh secara gratis dan terbuka, untuk kemudian digunakan dan diredistribusikan kembali. Pada Tugas akhir ini, OpenStreetMap digunakan untuk membentuk peta Surabaya.

2.7. JOSM

JOSM (Java OpenStreetMap Editor) adalah alat penyunting untuk data hasil dari OpenStreetMap. JOSM tidak membutuhkan koneksi internet. Penggunaan JOSM

membutuhkan installasi Java sebelumnya. Pada tugas akhir ini JOSM digunakan untuk mengedit peta yang didapat dari OpenStreetMap [7].

2.8. AWK

Awk adalah bahasa pemrograman yang digunakan untuk melakukan manipulasi data dan membuat laporan [8]. Awk adalah sebuah perograman filter untuk teks, seperti halnya perintah "grep". Awk dapat digunakan untuk mencari bentuk/model dalam sebuah berkas teks. Awk juga dapat mengganti bentuk satu teks ke dalam bentuk teks lain. Awk dapat juga digunakan untuk melakukan proses aritmatika seperti yang dilakukan oleh perintah "expr". Awk adalah sebuah pemrograman seperti pada shell atau C yang memiliki karakteristik yaitu sebagai tool yang cocok untuk jobs juga sebagai pelengkap (complicated) untuk filter standard. Pada tugas akhir ini AWK digunakan untuk membuat script untuk menghitung PDR, delay, dan routing overhead dari hasil trace NS-3.

2.9. NS2 dan Proses Instalasi NS2

NS adalah simulator diskrit yang digunakan pada penelitian jaringan. NS memberikan dukungan substansial untuk simulasi TCP, routing, dan protokol multicast melalui jaringan kabel dan nirkabel (lokal dan satelit) [9]. NS@ adalah varian dari produk NS dan penerus dari versi sebelumnya yaitu NS1. NS2 dibuat dengan menggunakan bahasa pemrograman C++ dan untuk melakukan konfigurasi menggunakan O Tcl.

NS2 telah digunakan dalam banyak penelitihan, telah tervalidasi dan terverifikasi pada banyak paper internasional. NS2 dikembangkan oleh University of California Berkeley dan USC ISI sebagai bagian dari projek Virtual INternet Testbed (VINT). Network Simulator pertama kali dibangun sebagai varian dari REAL Network Simulator pada tahun 1989 di University of California Berkeley (UCB). Pada tahun 1995 pembangunan Network Simulator didukung oleh Defense Advanced Research Project Agency (DARPA) melalui VINT

Project, yaitu sebuah tim riset gabungan yang beranggotakan tenaga-tenaga ahli dari beberapa instansi ternama. NS juga bersifat open source dibawah Gnu Public License (GPL), sehingga NS dapat di-download dan digunakan secara gratis melalui web site NS yaitu http://www.isi.edu/nsnam/. Sifat open source juga mengakibatkan pengembangan NS menjadi lebih dinamis. Struktur NS. Ns dibangun menggunakan metoda object oriented dengan bahasa C++ dan OTcl (variant object oriented dari Tcl).

Sebelum melakukan instalasi NS2, terlebih dahulu dilakukan instalasi dependensi untuk mendukung. Salah satu dependensi yang dibutuhkan yakni gcc-4.3.

```
sudo apt-get install gcc-4.4 build-essential
autoconf automake libxmu-dev
```

Gambar 2.4 Perintah untuk dependensi NS2

Namun karena gcc-4.3 sudah tidak ada maka bias diganti dengan gcc-4.4. berikut potongan *command* diterminal untuk melakukan instalasi dependensi NS2. Setelah instalasi semua dependensi lengkap silahkan unduh *berkas* NS2 pada laman yang tertera pada Gambar 2.5

```
https://sourceforge.net/projects/nsnam/fil
es/latest/download
```

Gambar 2.5 Laman unduh untuk *berkas* NS2

Setelah semua selesai terunuduh buat direktori sebagai tempat *berkas* NS2 nanti dan pindahkan *berkas* hasil unduhan tadi ke dalam direktori tersebut. Disini direktori yang dibuat bernama "workplace"

```
$ cd workplace
$ tar xjf ns-allinone-2.35.tar.gz
```

Gambar 2.6 Perintah untuk mengekstrak berkas NS2

Sebelum dilakukan beberapa modifikasi pada dua berkas yakni berkas "ls.h" pada direktori "linkstate" dan berkas "Makefile.in" pada direktori "otcl-1.13".
Berikut langkah untuk melakukan edit berkas ls.h pada Gambar 2.7.

```
$ cd ns-allinone-2.35/ns-2.35/linkstate
$ gedit ls.h
```

Gambar 2.7 Perintah untuk melakukan edit pada ls.h

Setelah tampil kotak dialog gedit silahkan menuju baris ke 137 dan lakukan perubahan pada kata "erase" ubah menjadi "this -> erase". Contoh berubahan bias dilihat pada Gambar 2.8

```
  void eraseAll() { this->erase(baseMap::begin(), baseMap::end()); }
  T* findPtr(Key key) {
      iterator it = baseMap::find(key);
      return (it == baseMap::end()) ? (T *)NULL : &((*it).second);
  }
;
```

Gambar 2.8 Contoh tampilan ls.h yang diedit

Setelah itu, ubah berkas "Makefile.in" pada direktori "otcl-1.13" untuk memberikan informasi pada NS2 mengenai versi gcc yang digunakan. Berikut perintahnya.

```
Sudo gedit ns-allinone-2.34/otcl-
1.13/Makefile.in
```

Gambar 2.9 Perintah untuk melakukan edit pada Makefile.in

Setelah kotak dialog tampil, silahkan ganti CC= @CC@ menjadi CC=gcc-4.4 seperti Gambar 2.10. Setelah proses edit kedua berkas selesai, maka dilakukan proses instalasi dengan perintah pada Gambar 2.11. Proses instalasi memakan waktu sekitar 10 – 15 menit. Proses instalasi berhenti jika sudah muncul gambar seperti Gambar 2.12. Langkah selanjutnya yang harus dilakukan adalah melakukan konfigurasi pada *environment path* melalui berkas ".bashrc". Pada berkas

tersebut harus ditambahkan pengaturan mengenai *path* yang akan digunakan oleh NS2.

```
CC=         gcc-4.4
CFLAGS=     @CFLAGS@
RANLIB=     @RANLIB@
INSTALL=    @INSTALL@

#
# how to compile, link, and name shared libraries
#

SHLIB_LD=   @SHLIB_LD@
SHLIB_CFLAGS=   @SHLIB_CFLAGS@
SHLIB_SUFFIX=   @SHLIB_SUFFIX@
SHLD_FLAGS= @DL_LD_FLAGS@
DL_LIBS=    @DL_LIBS@

SHLIB_LD_LIBS  = @SHLIB_LD_LIBS@
```

Gambar 2.10 Tampilan dalam berkas Makefile.in

```
sudo su cd ~/ns-allinone-2.35/./install
```

Gambar 2.11 Perintah install NS2

Perintah untuk pengaturan pada berkas ".bashrc" dapat dilihat pada Gambar 2.13. Pada baris "home/krilido/workplace/" bisa diganti sesuai dengan letak direktori yang digunakan untuk meletakkan berkas dari NS2. Misal jika berkas diletakkan pada "home/abc/ns2/" maka konfigurasi harus diganti dengan "/home/abc/ns2/ns-allinone-2.35/otcl-1.14".

Gambar 2.12 Hasil setelah melakukan instalasi pada NS2

```
# LD_LIBRARY_PATH
OTCL_LIB=/home/krilido/workplace/ns-allinone-
2.35/otcl-1.14/
NS2_LIB=/home/krilido/workplace/ns-allinone-
2.35/lib/
USR_Local_LIB=/usr/local/lib/
export
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$OTCL_LIB:$NS2_
LIB:$USR_Local_LIB
# TCL_LIBRARY
TCL_LIB=/home/krilido/workplace/ns-allinone-
2.35/tcl8.5.10/library/
USR_LIB=/usr/lib/
export
TCL_LIBRARY=$TCL_LIBRARY:$TCL_LIB:$USR_LIB
# PATH
XGRAPH=/home/krilido/workplace/ns-allinone-
2.35/xgraph-12.2/:/home/krilido/workplace/ns-
allinone-2.35/bin/:/home/krilido/workplace/ns-
allinone-
2.35/tcl8.5.10/unix/:/home/krilido/workplace/ns-
allinone-2.35/tk8.5.10/unix/
NS=/home/krilido/workplace/ns-allinone-2.35/ns-
2.35/
NAM=/home/krilido/workplace/ns-allinone-
2.35/nam-1.15/
export PATH=$PATH:$XGRAPH:$NS:$NAM
```

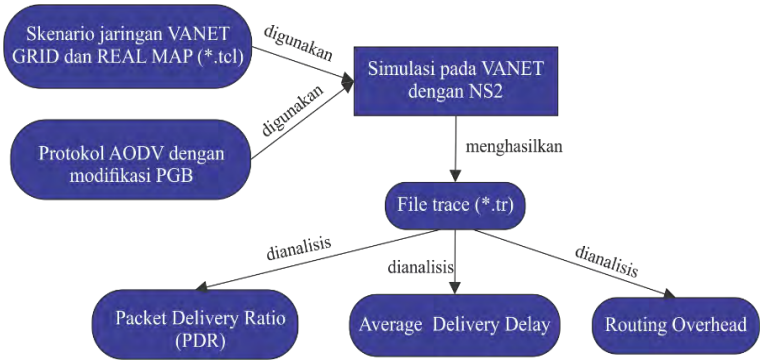
Gambar 2.13 Konfigurasi *Path* pada berkas .bashrc

BAB III
PERANCANGAN

Perancangan merupakan bagian penting dari pembuatan sistem secara teknis. Sehingga bab ini secara khusus akan 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 dilakukan analisis pengaruh algoritma PGB pada protokol AODV di jaringan VANET. Dimana metode routing AODV akan menerapkan algoritma PGB pada proses meneruskan RREQ. Ilustrasi rancangan sistem bisa dilihat pada Gambar 3.1.Gambar 3.1 Diagram rancangan simulasi



Gambar 3.1 Diagram rancangan simulasi

PGB pada protokol AODV di jaringan VANET. Dimana metode routing AODV akan menerapkan Proses pengujian melibatkan dua jenis skenario yang berbeda yakni skenario dengan menggunakan peta grid dan skenario dengan menggunakan peta real Surabaya. Perancangan dengan peta berbentuk grid menggunakan alat bantu berupa SUMO sedangkan perancangan

dengan menggunakan peta real Surabaya menggunakan alat bantu SUMO, peta digital OpenStreetMap, dan aplikasi JOSM.

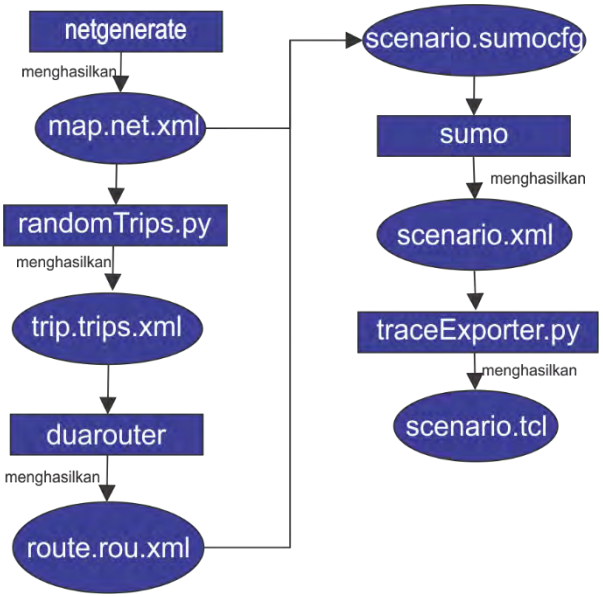
Simulasi pada NS2 dilakukan dengan melakukan beberapa modifikasi pada modul AODV, *Scheduler*, *Packet*, dan pada *Wireless-phy*. Setiap hasil dari simulasi akan dihitung PDR, d*elivery delay*, dan *routing overhead*. Dari hasil tersebut dianalisis pengaruh dari algoritma PGB pada protocol AODV.

3.2 Perancangan Skenario

Perencangan skenario simulasi pada VANET dibagi menjadi dua yakni skenario menggunakan peta berbentuk grid dan peta real Surabaya. Peta grid berupa peta yang hanya terdapat jalan dan saling berpotongan hingga membentuk persegi. Peta dengan bentuk grid dipilih sebagai tes awal simulasi karena bentuknya yang menggambarkan kondisi yang sebenarnya dan kondisi peta yang seimbang dan stabil. Sedangkan untuk peta riil menggunakan peta dari Surabaya. Peta riil didapat dengan cara mengimpor dari OpenStreetMap.

3.2.1 Pembuatan Peta Grid

Peta grid untuk pengujian pada tugas ini berukuran 800 m x 800 m atau 0,64 km^2 dan jumlah garis horizontal dan vertikalnya masing-masing sebanyak 6 (enam). Ilustrasi dapat dilihat pada Gambar 3.2. Pembuatan peta konfigurasi menggunakan *tools* netgenerate yang menghasilkan peta lengkap dengan jalan dan *traffic light*. Peta yang dibuat berjumlah tiga buah dengan variasi kecepatan 10 m/s, 15 m/s, dan 20 m/s. Hasil peta yang telah dibuat akan ditambah dengan *node* beserta *node* awal dan tujuan oleh modul randomTrips.py. Sedang untuk arah setiap *node* akan dibuat secara acak oleh modul duarouter. Kedua berkas yakni peta grid dan pergerakan disimulasikan menjadi satu di dalam sebuah konfigurasi simulasi dan dengan menggunakan tools sumo dan modul traceExporter menghasilkan skenario yang dapat digunakan pada NS2.



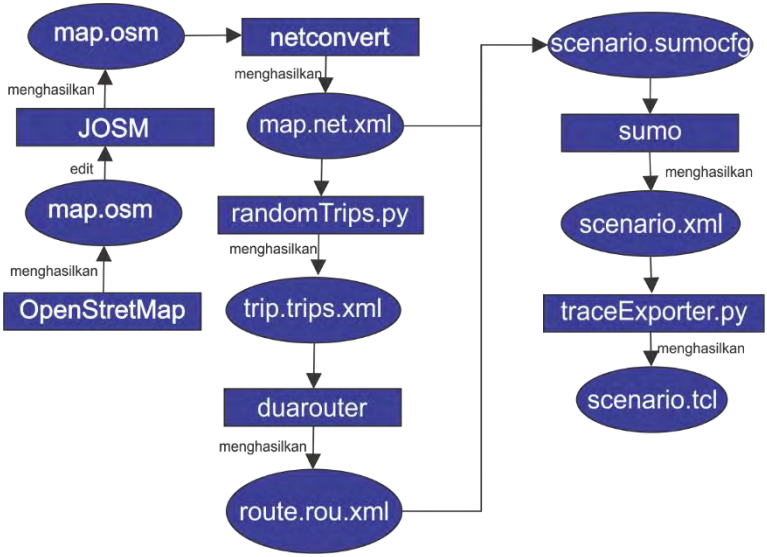
Gambar 3.2 Alur pembuatan peta grid

3.2.2 Pembuatan Peta Riil Surabaya

Pembuatan peta riil Surabaya menggunakan peta yang diambil dari OpenStreetMap. Kemudian diimpor kedalam berkas berformat *.osm. Setelah itu akan dilakukan proses edit pada aplikasi JSOM. Pada aplikasi ini, akan dilakukan penghapsan objek yang diarasa tidak diperlukan dan penambahan *traffic light* pada peta. Setelah peta dirasa sudah cukup maka akan dilakukan proses pengkonversian dengan menggunakan *tools* netconvert hinga menghasilkan berkas dengan ekstensi *.net.xml. Langkah selanjutnya sama persis dengan langkah pada proses pembuatan peta grid. Alur proses secara lengkap bisa dilihat pada Gambar 3.3.

3.3 Perancangan Sumulasi pada NS2

Simulasi VANET pada NS-2 dilakukan dengan menggunakan skenario mobilitas dan digabungkan dengan skrip TCL yang berisikan konfigurasi mengenai lingkungan simulasi.



Gambar 3.3 Alur pembuatan peta riil Surabaya

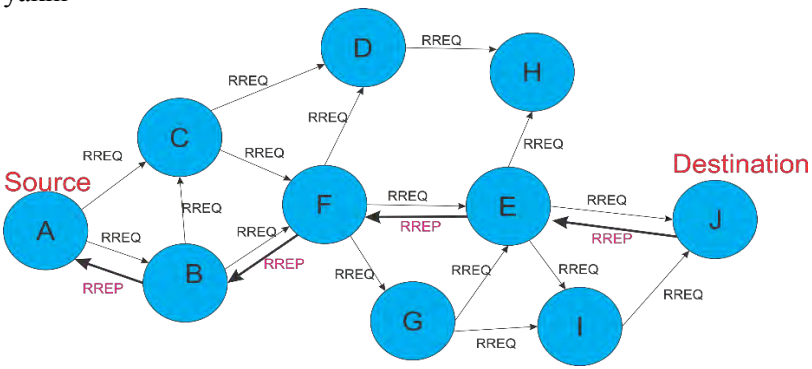
3.4 Perancangan Protokol AODV+PGB

Permasalah yang sering muncul pada protokol AODV yaitu kondisi ketika sebuah *node* menerima sebuah paket bertipe RREQ maka *node* tersebut akan melakukan *forwarding* paket RREQ kepada semua *node* tetangganya. Sehingga akan terjadi *broadcast storm* paket RREQ seperti pada Gambar 3.4.

Setiap *node* dapat memperoleh lebih dari satu RREQ yang sama namun dari sumber yang berbeda. Hal ini mengakibatkan *traffic* menjadi tinggi dan kemungkinan *collision* semakin tinggi. Sehingga diperlukan algoritma baru untuk mengatasi masalah tersebut

Algoritma *Preferred Group Broadcasting* (PGB) mengubah prinsip pada proses *forwarding* RREQ pada protokol AODV dimana hanya *node-node* tertentu yang akan melakukan proses *forwarding* RREQ bukan semua *node* dengan cara membatasi *node* yang berhak melakukan *rebroadcasting* hanya pada *node* yang

berada pada area tertentu. PGB membagi area menjadi tiga bagian yakni



Gambar 3.4 Proses *route discovery* pada AODV

- Area dalam (*Inner*/IN) = *node* dengan signal lebih kuat dari PG
- Area *Preferred group* (PG) = *node* yang akan digunakan untuk melakukan *rebroadcasting*
- Area luar (*Outer*/OUT) = *node* dengan signal lebih lemah dari PG

Proses pengelompokan tersebut berdasarkan *signal strength* yang diterima oleh masing-masing *node*. Pengelompokan *signal strength* menggunakan *thresholds* yang yang telah ada [3]. Pengelompokan juga berguna menentukan nilai delay yang akan diberikan kepada masing-masing *node*. Delay akan digunakan sebagai pengatur jadwal pada *scheduler*. Setiap RREQ paket yang diterima akan diberikan delay sesuai dengan posisi *node* asal yang mengirimkan terhadap penerima. Delay paling sedikit berasal dari *node* pada posisi PG, sedangkan delay terlama berasal dari *node* pada posisi IN. Alur dari proses AODV+PGB bisa dilihat pada Gambar 3.6.

Ketika ada paket datang pertama-tama dilakukan pengecekan apakah paket RREQ pernah diterima sebelumnya atau tidak. Jika ya paket RREQ yang sama, baik itu telah masuk

kedalam antrian *scheduler* atau belum maka akan dilakukan *drop* pada kedua paket. Jika belum pernah menerima maka akan dilakukan pengecekan posisi *node* pengirim apakah *node* pengirim masuk dalam area PG dari *node* penerima atau tidak. Prinsip yang digunakan adalah *signal strength* dari *node* penerima terhadap *node* pengirim sama dengan *signal strength* yang diterima *node* pengirim terhadap *node* penerima. Sehingga jika *node* penerima masuk dalam area PG *node* pengirim maka pengerim juga masuk kedalam *node* PG dari penerima.

Setelah dikelompokan kedalam salah satu kelompok, penghitungan delay dilakukan. Penghitungan delay berdasarkan *signal strength* yang diperoleh. Dalam beberapa kasus, akan ada dua atau lebih *node* yang memiliki nilai *signal strength* yang sama. Dalam hal ini otomatis nilai dari delay mereka berdua akan sama. Maka untuk mengatasi hal tersebut akan ditambahkan *jitter* pada setiap paket yang ada. *Jitter* berasal dari nilai random untuk mengatasi dua atau lebih *node* melakukan *broadcast* secara bersamaan.

Setelah penambahan selesai maka paket akan masuk kedalam daftar even dalam *scheduler* dan informasi mengenai RREQ pada tabel *broadcast* AODV. Jika selama proses antrian *node* menerima paket yang sama dan dari pengirim asal yang sama, maka otomatis paket akan di drop dan paket yang berada di dalam *scheduler* juga akan di drop. Sehingga *node* penerima tidak akan melakukan *broadcasting* paket tersebut sama sekali. Namun jika selama proses antrian tidak ada paket yang sama, maka paket akan di *broadcast.* Ilustrasi untuk kondisi pada AODV+PGB bisa dilihat pada Gambar 3.5. Pada gambar tersebut, *node* yang melakukan proses penyebaran paket RREQ hanya terbatas dan tidak semua *node*.

3.5 Perancangan Metrik Analisis

Berikut ini merupakan beberapa parameter yang dianalisis dalam Tugas Akhir ini:

3.5.1 *Packet Delivery Ratio* (PDR)

Packet delivery ratio adalah prosensatse jumlah antara paket yang dikirim dan diterima. Penghitungan melibatkan perbandingan dari paket yang dikirim dan diterima. Rumus untuk menghitung PDR dapat dilihat pada persamaan dibawah ini dimana $received$ adalah banyaknya paket data yang diterima dan *sent* adalah banyaknya paket data yang dikirimkan.

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

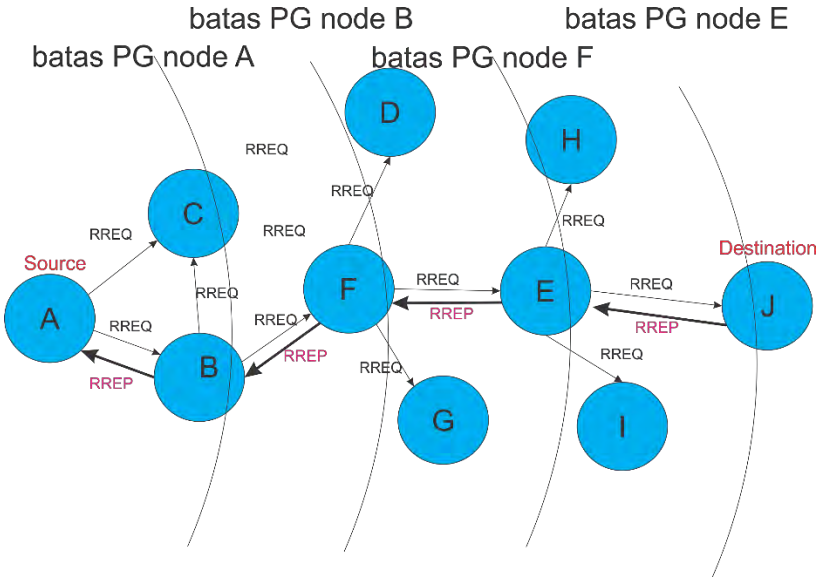
3.5.2 *Average Delivery Delay*

Average delivery delay dihitung berdasarkan rata-rata *delay* antara waktu paket diterima dan waktu paket dikirim. *Average delivery delay* dihitung dengan menggunakan rumus dibawah, di $t_{\text{received[i]}}$ adalah waktu penerimaan paket dengan urutan / id ke-*i*, $t_{\text{sent[i]}}$ adalah waktu pengiriman paket dengan urutan / id ke-*i*, dan *sent* adalah banyaknya paket data yang dikirimkan.

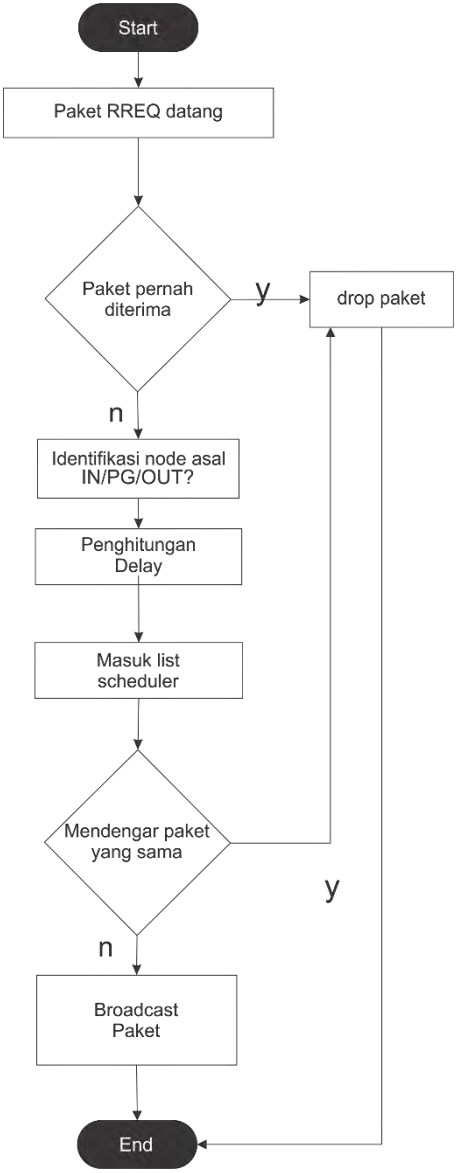
$$\boldsymbol{Delay = \frac{\sum_{i \le sent}^{i=0} t_{received[i]} - t_{sent[i]}}{sent}} \qquad (3.2)$$

3.5.3 *Routing Overhead* (RO)

Routing overhead adalah jumlah paket kontrol routing yang ditransimisikan per data paket yang terkirim ke tujuan selama simulasi terjadi. Routing overhead dihitung berdasarkan paket routing yang ditransmisikan baik itu *Route Request* (RREQ), *Route Reply* (RREP), dan *Route Error* (RERR).



Gambar 3.5 Proses *route discovery* pada AODV+PGB



Gambar 3.6 *flow chart* AODV+PGB

[Halaman ini sengaja dikosongkan]

BAB IV
IMPLEMENTASI

Pada bab ini akan dibahas mengenai implementasi yang dilakukan berdasarkan rancangan yang telah dijabarkan pada bab sebelumnya. Sebelum penjelasan implementasi akan ditunjukkan terlebih dahulu lingkungan untuk melakukan implementasi.

4.1 Lingkungan Implementasi

Pada implementasi sistem, digunakan beberapa perangkat pendukung sebagai berikut:

4.1.1 Lingkungan Perangkat Lunak

Adapun perangkat lunak yang digunakan untuk pengembangan sistem sebagai berikut:

- Sistem operasi Ubuntu 14.04 64 bit
- Google Chrome versi 50.0.2661.102 m (64-bit) untuk mengoperasikan web OpenSteetMap, dan
- JOSM versi 9979 sebagai editor peta hasil ekspor dari OpenStreetMap.

4.1.2 Lingkungan Perangkat Keras

Spesifikasi perangkat keras yang digunakan sebagai lingkungan implementasi perangkat lunak Tugas Akhir adalah sebagai berikut:

- P*rocessor* Intel(R) Core(TM) i3-3110M CPU @ 2.4GHz,
- RAM sebesar 4 GB DDR3, dan
- Media penyimpanan sebesar 500 GB.

4.2 Implementasi Skenario

Pada subbab ini akan dijelaskan implementasi yang telah dilakukan pada sistem. Implementasi skenario dibagi menjadi dua bagian yakni skenario dengan peta grid dan skenario dengan peta riil Surabaya.

4.2.1 Skenario Grid

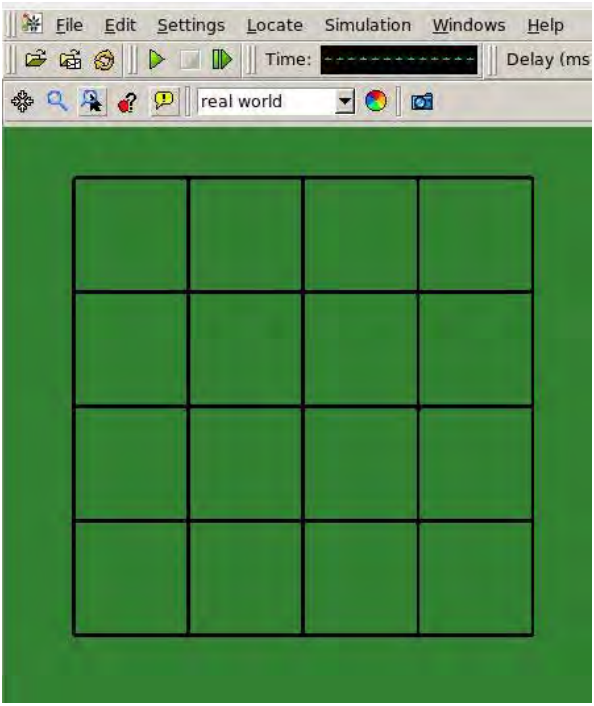
Dalam implementasi skenario grid, dibuat peta melalui *tools* netgenerate dari aplikasi SUMO dengan luas 800 m x 800 m dan jumlah garis vertikal dan horisontal sebanyak lima buah. Setiap petak berukuran 200 m x 200 m dan terdapat 5 titik x 5 titik serta terdapat 9 perempatan dengan kecepatan 10 m/s, 15 m/s, 20 m/s. Proses pembuatan peta grid pada terminal di Ubuntu dengan menggunakan netgenerate menggunakan perintah seperti pada Gambar 4.1.

```
netgenerate --grid --grid.number=5 --
grid.length=200 --default.speed=10 --
tls.guess=1 --output-file=map5-6.net.xml
```

Gambar 4.1 Perintah untuk membuat peta

Hasil peta jika dibuka dengan sumo-gui akan terlihat seperti pada Gambar 4.2. Setelah petak terbentuk, maka akan dilakukan pembuatan *node* beserta tujuan dan nasal secara random dengan menggunakan modul randomTrips.py. Perintah untuk modul randomTrips.py bisa dilihat pada Gambar 4.3. Hasil dari proses tersebut akan dilanjutkan oleh *tools* duarouter dimana setiap *node* akan diberikan arah gerak. Algoritma yang digunakan secara normal adalah algoritma Djikstra. Setelah proses tersebut selesai akan menghasilkan berkas berekstensi *.xml. Perintah untuk proses duarouter bisa dilihat pada Gambar 4.4. Lalu hasil dari proses netgenerate dan hasil dari proses duarouter akan digabungkan menjadi satu pada berkas berekstensi sumocfg. *Script* untuk menggabungkan bisa dilihat pada Gambar 4.5.

Script yang telah dibuat sebelumnya akan dijalankan oleh sumo untuk menggabungkan berkas peta dan berkas pergerakan nya. Hal yang perlu diperhatikan yakni berkas map.net.xml dan route.rou.xml harus disimpan dalam direktori yang sama dengan berkas .sumocfg. Untuk melakukan pengecekan, berkas sumocfg bisa dibuka melalui tools sumo-gui.



Gambar 4.2 Peta hasil dari netgenerate

```
randomTrips.py -n map5-6.net.xml -e 100 -l -
-trip-attributes="departLane=\"best\"
departSpeed=\"max\"
departPos=\"random_free\"" -o trip.trips5-
6.xml
```

Gambar 4.3 Perintah untuk membuat *node* beserta asal dan tujuannya

```
duarouter --trip-files trips.trips.xml --net-
file kapal.xml --output-file result.rou.xml -
-ignore-errors --repair
```

Gambar 4.4 Perintah untuk membuat rute

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration
      xmlns:xsi="http://www.w3.org/2001/XMLSch
ema-instance"
      xsi:noNamespaceSchemaLocation="http://su
mo.dlr.de/xsd/sumoConfiguration.xsd">
      <input>
            <net-file value="map5-6.net.xml"/>
            <route-files
value="route.rou.xml"/>
      </input>
      <time>
            <begin value="0"/>
            <end value="20"/>
      </time>
```

Gambar 4.5 Contoh isi berkas sumocfg

Untuk dikonversi menjadi sebuah berkas xml yang berisi peta grid dan pergerakan kendaraan dijalankan perintah pada Gambar 4.6.

```
sumo -c jajal.sumocfg --fcd-output scan.xml
```

Gambar 4.6 Perintah untuk mengkonversi peta dan pergerakan *node* pada sumo menjadi xml

Berkas scenario.xml kemudian dikonversi menjadi berkas pergerakan dalam format mobilitas NS-2 dengan bantuan modul traceExporter.py dengan perintah seperti pada Gambar 4.7.

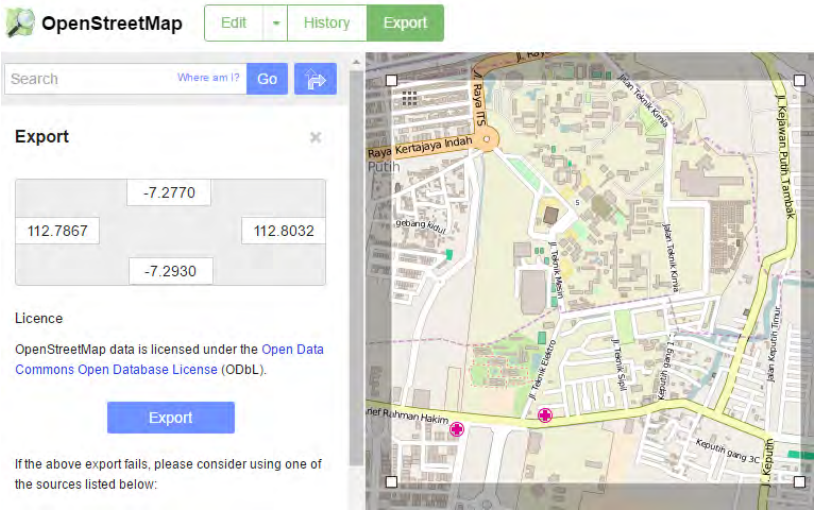
```
traceExporter.py    --fcd-input=scan.xml    --
ns2mobility-output=ns25_6.tcl
```

Gambar 4.7 Perintah untuk mengkonversi berkas xml dari SUMO ke dalam format mobilitas NS-2

4.2.2 Skenario Riil

Pada dasarnya, pembuatan skenario dengan peta riil hampir sama dengan pembuatan skenario pada peta grid. Yang

menjadi pembeda hanyalah pada bagian awalnya saja. Jika pada peta grid peta langsung dibuat secara otomatis oleh netgenerate, pada peta riil, berkas peta terlebih dahulu diambil dari OpenStreetMap dengan cara menandai wilayah kemudian diekspor dan secara otomatis akan ter-download berkas berekstensi .osm melalui browser. Contoh proses *capture* bisa dilihat pada Gambar 4.8.



Gambar 4.8 Proses capture peta Surabaya dengan OpenStreetMap

Hasil dari proses tersebut kemudian diedit menggunakan JOSM. JOSM berfungsi untuk mengisikan lampu merah pada tiap perempatan. Jalan yang tidak digunakan dihapus serta bentuk dan ukuran peta dibuat supaya hampir sama dengan peta grid. Beberapa jalan baru juga ditambahkan agar kepadatan jalan tetap stabil sehingga tidak akan ada daerah pada peta yang jarang dikunjungi kendaraan. Jalan baru tersebut juga berfungsi agar bentuk peta mendekati persegi agar mirip dengan skenario grid.

Setelah dirasa cukup, lalu berkas dikonversi agar berbentuk .net.xml. Konversi dilakukan dengan bantuan netconvert

dari SUMO. Perintah yang dijalankan untuk mengkonversi bisa dilihat pada Gambar 4.9.

```
netconvert --osm-files map-1.osm --output-file
map.net.xml
```

Gambar 4.9 Perintah mengkonversi berkas osm dari OpenStreetMap menjadi format SUMO

Langkah selanjutnya sama persis dengan langkah pembuatan berkas skenario peta grid sampai berhasil dikonnversi kedalam format pergerakan NS-2.

4.3 Implementasi algoritma PGB pada *routing protocol* AODV

Protokol AODV PGB adalah turunan dari protocol AODV. Untuk melakukan modifikasi pada *routing protocol* AODV agar dapat menerapkan algoritma PGB harus ada perubahan pada beberapa bagian antara lain:

- Header dari paket
- Penentuan *signal strength* untuk memperhitungkan delay
- Penanganan paket RREQ
- Penanganan ketika terjadi *hop splitting* dan *hop marging*

Maka pertama-tama harus dilakukan proses pengeditan dulu pada beberapa berkas di NS-2. Terdapat 7 berkas yang diedit yakni `packet.h`, `aodv.cc`, `aodv.h`, `scheduler.cc`, `scheduler.h`, `wireless-phy.cc`, dan `wireless-phy.h`.

4.3.1 Modifikasi packet.h

Packet.h merupakan *class* yang digunakan sebagai acuan untuk isi dari semua paket di semua kelas. Baik pada protokol AODV maupun protokol yang lain. Dalam Tugas Akhir ini, pada packet.h akan ditambahkan variabel untuk menampung nilai delay pada paket tersebut. Penambahan dilakukan pada `struct`

`hdr_cmn` dikarenakn *struct* tersebut merupakan *header* bagi paket yang akan dikirim. Penambahan seperti pada Gambar 4.10.

```
struct hdr_cmn {
       enum dir_t { DOWN= -1, NONE= 0, UP= 1 };
       packet_t ptype_;    // packet type (see above)
       int    size_;       // simulated packet size
       int    uid_;        // unique id
       int    error_;             // error flag
       int     errbitcnt_;     // # of corrupted bits
jahn
       int     fecsize_;
       double ts_;          // timestamp: for q-delay
measurement
       int    iface_;              //         receiving
interface (label)
       dir_t  direction_;  //    direction:    0=none,
1=up, -1=down
       double dly;         //variable delay
```

Gambar 4.10 Modifikasi paket.h

4.2.2 Modifikasi wireless-phy.cc

Kelas wireless-phy merupakan kelas yang berada pada *physical* layer dan berguna untuk menerima dan mengirim paket dari atau untuk suatu *node*. *Class* wireless-phy berasal dari *class* Phy dimana wireless-phy melakuakn *overwrite* pada dua fungsi yakni fungsi *sendDown*(p) dan *sendUp*(p). Dalam tugas akhir ini, pada *class* wireless-phy akan ditambahkan fungsi baru untuk melakukan kalkulasi beberapa delay yang akan diberikan dalam suatu header paket. Bentuk fungsi bisa dilihat pada Gambar 4.11. Selain itu, juga terdapat perubahan pada fungsi *sendUp*(p). Perubahan bisa dilihat pada Gambar 4.12. Ketika header paket diterima, langsung dilakukan pengecekan kekuatan sinyal oleh fungsi *sendUp*(p) lalu menimpannya pada variabel Pr. Jika kurang dari *thresholds* (RXThresh) maka paket akan di drop.

Pada fungsi `WirelessPhy::cal()` dilakukan penghitungan delay berdasarkan kekuatan signal yang ada. Selain itu juga diberikan nilai random untuk mencegah terjadinya

broadcast paket secara bersamaan dari dua atau lebih *node* yang memiliki nilai kekuatan sinyal yang sama.

```
double
WirelessPhy::cal(double  pr,double rxTh,double  txP){
double top=1.92e-06;  double mxT3=0.100;
double bwh=2.33e-10;  double random;
double OT=1.45e-9;    double xx3;
double IT=5.79e-9;
double dT=0.010;
double mnT1=dT/3;
double mxT1=0.015;

if (pr>OT && pr<IT){
              double fot=OT;
              double delP1 =rxTh-pr;

              if (delP1<0) delP1=delP1*-1;
              double delP0 = delP1-fot;
              if (delP0<0) delP0=delP0*-1;

              double delT = mxT1-mnT1;
              double delFot=IT-OT;

              random=Random::uniform(0.0001, 0.0009);

              double T1=(delP0*delT/delFot)+mnT1;
              double waktu=T1+random;
              return waktu;
       }
              return waktu;
       }
       else if (pr>bwh && pr<OT)
       {
              double fot=OT;
              double delP1 =rxTh-pr;
              if (delP1<0) delP1=delP1*-1;
              double delP0 = delP1-fot;
              if (delP0<0) delP0=delP0*-1;
              double delT = mxT2-mnT2;
              double delFot=OT;
              random= Random::uniform(0.001, 0.009);
```

```
                double T2=(delP0*delT/delFot)+mnT2;
                double waktu=T2+random;
                return waktu;
        }
        else if (pr>IT && pr<top)
        {
                double fot=IT;
                double delP1 =rxTh-pr;
                if (delP1<0) delP1=delP1*-1;
                double delP0=delP1-fot;
                if (delP0<0) delP0=delP0*-1;
                double delT = mxT3-mnT3;
                double delFot=(txP-rxTh)-IT;
                random= Random::uniform(0.01, 0.09);
                double T3=(delP0*delT/delFot)+mnT3;
                double waktu=T3+random;
                return waktu;
        }
        else
        {
                return 0;
        }
}
```

Gambar 4.11 Modifikasi wireless-phy.cc

```
p->txinfo_.RxPr = Pr;
        p->txinfo_.CPThresh = CPThresh_;
        struct hdr_cmn *ch = HDR_CMN(p);
        ch->dly=cal(Pr,RXThresh_,Pt_);
```

Gambar 4.12 Modifikasi fungsi sendUp

4.3.2 Modifikasi wireless-phy.h

Karena melakukan penambahan fingsi baru yakni `WirelessPhy::cal()` pada wireless-phy.cc maka harus dilakukan perubahan pada wireless-phy.h untuk melakukan pendefinisian dari fungsi `WirelessPhy::cal()` karena struktur dari NS-2

menggunakan *class*. Perubahan kode bisa dilihat pada Gambar 4.13.

```
void node_on();
void node_off();
double cal(double  pr,double rxTh,double  txP);
```

Gambar 4.13 Modifikasi wireless-phy.h

4.3.3 Modifikasi scheduler.cc

Pada NS-2, setiap pengiriman paket akan dianggap sebagai sebuah even. Dan setiap even tersebut akan diproses pada *class* ini. Setiap even akan diberikan jadwal tersendiri untuk melakukan pengiriman pada kelas ini. Pada Tugas Akhir ini, kunci dari semua proses berada pada penggunaan *class* ini. Pada *class* ini dibuat dua fungsi yakni fungsi `Scheduler::schedulex(Handler* h, Event* e, double delay)` dan `Scheduler::scheduleDel(scheduler_uid_t uid)`. Fungsi `Scheduler::scheduleDel(scheduler_uid_t uid)` dapat dilihat pada
Gambar 4.14 dan fungsi `Scheduler::schedulex(Handler* h, Event* e, double delay)` dapat dilihat pada Gambar 4.15.

Pada fungsi `scheduleDel` akan dilakukan penghapusan pada daftar *scheduler*. Proses ini dilakukan jika pada sebuah *node*, *node* tersebut memperoleh paket *broadcast* yang sama dan dari sumber yang sama untuk mengurangi proses *broadcast* ulang. Untuk fungsi `schedulex` diambil dari fungsi `schedule` karena fungsi dasaranya hampir sama, namun nilai balikan dan kegunaan dari fungsi berbeda. Fungsi `schedule` digunakan untuk memberikan *schedule* pada semua paket. Sedangkan fungsi `schedulex` digunakan hanya untuk paket AODV dengan menggunakan algoritma PGB.

4.3.4 Modifikasi scheduler.h

Karena melakukan penambahan fingsi baru yakni `Scheduler::schedulex ()` dan `Scheduler::scheduleDel` pada schedule.cc maka harus dilakukan perubahan pada schedule.h

```
bool
Scheduler::scheduleDel(scheduler_uid_t uid){
       Event* p = lookup(uid);
       if (p != 0) {
              /*XXX  make  sure  it  really  is  an
atevent*/
              cancel(p);
              //AtEvent* ae = (AtEvent*)p;
              //delete ae;
              return true; } return false;}
```

Gambar 4.14 Modifikasi pada fungsi schedulerDel

```
int
Scheduler::schedulex(Handler*  h,  Event*  e,  double
delay)
{
       int uuuu=0;
              if (!h) {
              fprintf(stderr,
                     "Scheduler: attempt to schedule
an event with a NULL handler."
                     "  Don't DO that at time %f\n",
clock_);
              abort();
       };
       if (e->uid_ > 0) {
              printf("Scheduler:   Event   UID   not
valid!\n\n");
              abort();
       }
       if (delay < 0) {
              fprintf(stderr,
                     "warning:                      ns
Scheduler::schedule: scheduling event\n\t"
                     "with  negative  delay  (%f)  at
time %f.\n", delay, clock_);
```

```
}
       if (uid_ < 0) {
              fprintf(stderr, "Scheduler: UID space
exhausted!\n");
              abort();
       }

e->uid_ = uid_++;
       e->handler_ = h;
       double t = clock_ + delay;
       e->time_ = t;
       insert(e);
       uuuu=e->uid_;
       return uuuu;}
```

Gambar 4.15 Modifikasi pada fungsi schedulex

untuk melakukan pendefinisian dari fungsi `Scheduler::schedulex ()` dan `Scheduler::scheduleDel` karena struktur dari NS-2 menggunakan *class*. Perubahan kode bsia dilihat pada Gambar 4.16.

```
class Scheduler : public TclObject {
public:
       static Scheduler& instance() {
              return (*instance_);              //
general access to scheduler
       }
       void   schedule(Handler*,   Event*,   double
delay);      // sched later event
       int   schedulex(Handler*,   Event*,   double
delay);      // tambah
       bool scheduleDel(scheduler_uid_t uid);
```

Gambar 4.16 Modifikasi scheduler.h

4.3.4 Modifikasi aodv.cc

Pada *class* ini dilakukan modifikasi paling banyak. Beberapa modifikasi berbentuk fungsi baru. Beberapa fungsi baru adalah `AODV::id_lookupx`, `AODV::forwardx`, `AODV::id_update` dan fungsi yang dilakukan diubah yakni

`AODV::recvRequest`. Daftar fungsi yang diubah dapat dilihat pada Gambar 4.17

```
int
AODV::forwardx(aodv_rt_entry *rt, Packet *p, double
delay) {
struct hdr_cmn *ch = HDR_CMN(p);
struct hdr_ip *ih = HDR_IP(p);
int cc=-1;
 if(ih->ttl_ == 0) {
#ifdef DEBUG
  fprintf(stderr,     "%s:     calling     drop()\n",
__PRETTY_FUNCTION__);
#endif // DEBUG
  drop(p, DROP_RTR_TTL);
  return cc;
 }
 if ((( ch->ptype() != PT_AODV && ch->direction() ==
hdr_cmn::UP ) &&
       ((u_int32_t)ih->daddr() == IP_BROADCAST))
              || (ih->daddr() == here_.addr_)) {
       dmux_->recv(p,0);
       return cc; }
 if (rt) {
   assert(rt->rt_flags == RTF_UP);
   rt->rt_expire       =       CURRENT_TIME       +
ACTIVE_ROUTE_TIMEOUT;
   ch->next_hop_ = rt->rt_nexthop;
   ch->addr_type() = NS_AF_INET;
   ch->direction() = hdr_cmn::DOWN;
 }
 else { // if it is a broadcast packet
   // assert(ch->ptype() == PT_AODV); // maybe a diff
pkt type like gaf
   assert(ih->daddr() == (nsaddr_t) IP_BROADCAST);
   ch->addr_type() = NS_AF_NONE;
   ch->direction() = hdr_cmn::DOWN;        }
if (ih->daddr() == (nsaddr_t) IP_BROADCAST) {
 // If it is a broadcast packet
   assert(rt == 0);
if (ch->ptype() == PT_AODV) {
ch->dly;
```

```
cc=Scheduler::instance().schedulex(target_,   p,ch-
>dly);
   } else {
     cc=Scheduler::instance().schedulex(target_,
p, ch->dly);  // No jitter
   }
 }
 else { // Not a broadcast packet
   if(delay > 0.0) {
     Scheduler::instance().schedule(target_,    p,
delay);
   }
   else {
   //  Not  a  broadcast  packet,  no  delay,  send
immediately
     Scheduler::instance().schedule(target_,    p,
0.);
   }
 }

 return cc;
}
int
AODV::id_lookupx(nsaddr_t   id,   u_int32_t   bid,
nsaddr_t d, u_int8_t h) {
BroadcastID *b = bihead.lh_first;
int pcc=-1;
       for( ; b; b = b->link.le_next) {
   if ((b->src == id) && (b->id == bid) && (b->dst
== d) && (b->hop_count == h)){
     pcc=b->uid_sche;
     return pcc;
   }
 }
 return pcc;
}
void
AODV::id_update(nsaddr_t    id,    u_int32_t    bid,
nsaddr_t d, u_int8_t h, int x) {
BroadcastID *b = bihead.lh_first;
// Search the list for a match of source and bid
```

```
for( ; b; b = b->link.le_next) {
   if ((b->src == id) && (b->id == bid))
     b->updateUid(x, d, h);
 }
}
```

Gambar 4.17 Modifikasi aodv.cc

Sedang pada fungsi `AODV::recvRequest` bisa dilihat pada Gambar 4.18.

```
 if (id_lookup(rq->rq_src, rq->rq_bcast_id)) {
       //tambah
       printf(" -- Drop");
       int       yy=id_lookupx(rq->rq_src,       rq-
>rq_bcast_id,rq->rq_dst,rq->rq_hop_count);
       if(yy>-1){
              printf(" with uid= %d ",yy);
       if(Scheduler::instance().scheduleDel(yy)){
                     printf(" Droped ");
              }else{
                     printf(" Fail ");}  }
    printf("\n");
uidx=forwardx((aodv_rt_entry*) 0, p, DELAY);
   id_update(rq->rq_src,     rq->rq_bcast_id,     rq-
>rq_dst, rq->rq_hop_count, uidx);
```

Gambar 4.18 Modifikasi fungsi recvRequest

Pada fungsi `AODV::id_lookupx`, `AODV::forwardx` memiliki sifat yang sama seperti fungsi `AODV::id_lookup`, `AODV::forward`, namun terdapat beberapa perbedaan yakni fungsi `AODV::id_lookupx` digunakan untuk mencari paket yang telah diterima. Jika paket yang sama telah diterima, maka paket akan didrop dari daftar *schedule*. Sedang pada fungsi `AODV::forwardx` sama dengan fungsi `AODV::forward` namun perbedaannya terletak pada delay yang diberikan. Jika `pada AODV::forward` delay yang diberikan bergantung pada *scheduler* maka pada fungsi

`AODV::forward` delay yang diberikan sesuai dengan perhitungan PGB yang ada.

4.4 Implementasi Metrik Analisis

Hasil dari simulasi skenario dalam NS-2 adalah sebuah trace berkas. Trace berkas digunakan sebagai bahan analisis untuk pengukuran performa dari protokol yang disimulasikan. Dalam Tugas Akhir ini, terdapat empat metrik yang akan menjadi parameter analisis, yaitu *packet delivery ratio, average end-to-end delay, dan routing overhead*.

4.4.1 Implementasi *Packet Delivery Ratio*

Packet delivery ratio didapatkan dengan cara menghitung setiap baris terjadinya event pengiriman dan penerimaan paket data yang dikirim melalui agen pada trace berkas. Pada lampiran A 3 ditunjukkan cara menyaring data yang dibutuhkan untuk perhitungan PDR. Pada skrip tersebut dilakukan penyaringan pada setiap baris yang mengandung string AGT karena event tersebut berhubungan dengan paket data. Paket yang dikirim dan diterima dibedakan dari kolom pertama pada baris yang telah disaring. Setelah pembacaan setiap baris selesai dilakukan, selanjutnya dilakukan perhitungan PDR dengan persamaan 3.1.

Contoh perintah untuk memanggil skrip AWK untuk menganalisis trace berkas dan contoh keluarannya dapat dilihat pada Gambar 4.19 dan Gambar 4.20.

```
awk -f pdr.awk s-aodv1.tr
```

Gambar 4.19 Perintah untuk menjalankan skrip AWK penghitungan PDR

```
Sent: 150 Recv: 109 Ratio: 0.7267
```

Gambar 4.20 Keluaran dari Skrip pdr.awk

4.4.2 Implementasi *Average End-to-End Delay*

Dalam pembacaan baris trace berkas untuk perhitungan *average end-to-end delay* terdapat lima kolom yang harus diperhatikan, yaitu kolom pertama yang berisi penanda event

pengiriman atau penerimaan, kolom kedua yang berisi waktu terjadinya event, kolom keempat yang berisi informasi layer komunikasi paket, kolom keenam yang berisi ID paket, dan tipe paket pada kolom ketujuh.

Delay dari setiap paket dihitung dengan cara mengurangi waktu penerimaan dengan waktu pengiriman berdasarkan ID paket. Hasil pengurangan waktu dari masing-masing paket dijumlahkan dan dibagi dengan jumlah paket CBR yang ID-nya terlibat dalam perhitungan pengurangan waktu. Kode implementasi dari perhitungan *average end-to-end delay* dapat dilihat pada lampiran A 4.

Contoh perintah untuk memanggil skrip AWK untuk menganalisis trace berkas dan contoh keluarannya dapat dilihat pada Gambar 4.21 dan Gambar 4.22.

```
awk -f endtoend.awk s-aodv1.tr
```

Gambar 4.21 Perintah untuk menjalankan skrip AWK penghitungan endt to end delay

```
Average End-to-End Delay    = 224.818 ms
```

Gambar 4.22 Keluaran dari Skrip endtoend.awk

4.4.2 Implementasi *Routing overhead*

Routing overhead didapatkan dengan cara menyaring setiap baris yang mengandung string REQUEST, REPLY, dan ERROR. Setiap ditemukannya string tersebut, dilakukan increment untuk menghitung jumlah paket *routing* yang tersebar di jaringan. Kode implementasi dari perhitungan *routing overhead* dapat dilihat pada lampiran A 5.

Contoh perintah untuk memanggil skrip AWK untuk menganalisis trace berkas dan contoh keluarannya dapat dilihat pada Gambar 4.23 dan Gambar 4.24.

```
awk -f ro.awk s-aodv1.tr
```

Gambar 4.23 Perintah untuk menjalankan skrip AWK penghitungan RO

```
routing packets 37.195, data = 185, NRL = 37.195
```

Gambar 4.24 Keluaran dari Skrip ro.awk

4.5 Implementasi Skenario Simulasi pada NS2

Untuk melakukan simulasi VANET pada NS-2, dibutuhkan sebuah berkas OTcl yang berisi deskripsi dari lingkungan simulasi. Berkas tersebut berisikan pengaturan untuk setiap *node* dan beberapa even yang perlu diatur agar berjalan pada waktu tertentu. Contoh potongan skrip pengaturan *node* dapat dilihat pada Gambar 4.26. Penjelasan dari pengaturan *node* dapat dilihat pada Tabel 4.1. Pengaturan lainnya yang dilakukan pada berkas tersebut antara lain lokasi penyimpanan trace berkas, lokasi berkas mobilitas *node*, konfigurasi *node* sumber dan *node* tujuan, dan konfigurasi event pengiriman paket data. Kode implementasi skenario yang digunakan pada Tugas Akhir ini dapat dilihat pada lampiran A 1 dan lampiran untuk *pattern* skenario. Skenario simulasi dijalankan dengan perintah pada Gambar 4.25. Setelah simulasi selesai akan ada keluaran berupa trace berkas hasil simulasi yang akan digunakan untuk analisis. Isi dari traceberkas adalah catatan seluruh event yang dari setiap paket yang tersebar di dalam lingkungan simulasi.

```
ns aodv.tcl
```

Gambar 4.25 Perintah Untuk Menjalankan 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 val(ifqlen)           50                   ;
set val(nn)         50                         ;
set val(rp)         AODV                       ;
```

```
set val(energymodel)       EnergyModel         ;
set val(initialenergy)     100                 ;
set val(lm)         "off"                      ;
set val(x)        1000                         ;
set val(y)           1000                      ;
set val(stop)        200                       ;
set val(cp)             "traffic1"           ;
set val(sc)              "ns2.tcl"
set ns_      [new Simulator]
set tracefd [open s-aodv2.tr w]
set namtrace [open s-aodv2.nam w]
set topo [new Topography]
$topo load_flatgrid $val(x) $val(y)
create-god $val(nn)

# Create nn mobile node s [$val(nn)] and attach them
to channel
$ns_ node-config    -adhocRouting $val(rp) -llType
$val(ll) -macType $val(mac) -channel $chan_1_ -
fqType $val(ifq) -ifqLen $val(ifqlen) -antType
$val(ant) - ropType $val(prop)    -phyType
$val(netif) -opoInstance $topo -agentTrace ON -
routerTrace ON -macTrace OFF -movementTrace ON
```

Gambar 4.26 Potongan Skrip Pengaturan *Node*

Tabel 4.1 Penjelasan dari Parameter Pengaturan *Node*

Parameter	Value	Penjelasan
adhocRouting	AODV	Menggunakan *routing protocol* AODV
llType	LL	Menggunakan *link layer* standar
macType	Mac/802_11	Menggunakan tipe MAC 802.11 karena komunikasi data bersifat *wireless*
channel	Channel/WirelessChannel	*Channel* komunikasi yang digunakan
ifqType	Queue/DropTail/PriQueue	Menggunakan *priority queue* sebagai antrian paket dan paket yang dihapus saat antrian

Parameter	Value	Penjelasan
		penuh adalah paket yang paing baru
ifqLen	50	Jumlah maksimal paket pada antrian
antType	Antenna/OmniAntenna	Jenis antena yang digunakan adalah *omniantenna*
propType	Propagation/TwoRayGround	Tipe propagasi sinyal *wireless* adalah *tworay ground*
phyType	Phy/WirelessPhy	Komunikasi menggunakan media *wireless*
topoInstance	$topo	Topologi yang digunakan daat menjalankan skenario
agentTrace	ON	Aktifkan pencatatan aktifitas dari agen *routing protocol*
routerTrace	ON	Aktifkan pencatatan pada aktifitas *routing protocol*
macTrace	OFF	Matikan pencatatan MAC *layer* pada *trace berkas*
movementTrace	ON	Aftifkan pencatatan pergerakan *node*

BAB V
UJI COBA DAN EVALUASI

Pada bab ini, dibahas uji coba dan evaluasi dari skenario NS3 yang telah dibuat.

5.1 Lingkungan Uji Coba

Uji coba dilakukan pada laptop dengan sistem operasi Ubuntu Linux dengan NS2 telah terpasang didalamnya. Untuk spesifikasi laptop dapat dilihat pada

Tabel 5.1 Spesifikasi laptop yang digunakan

Komponen	Spesifikasi
CPU	P*rocessor* Intel(R) Core(TM) i3-3110M CPU @ 2.4GHz,
Sistem Operasi	Ubuntu 14.04 64 bit
Memori	4 GB DDR3
Grafis	Intel HD 4000
Penyimpanan	500 GB HDD

Pengujian dilakukan dengan menjalankan program yang sudah dibuat pada NS2 tadi. Lalu akan didapatkan sebuah berkas berekstensi .tr yang kemudian akan dianalisis dengan script AWK yang sudah dibuat sebelumnya. Parameter pengujian bisa dilihat pada Tabel 5.2.

Tabel 5.2 Parameter Pengujian

No	Parameters	Spesifikasi
1	Netwok Simulator	NS-2, 2.35
2	Routing Protocol	AODV + PGB dan AODV
3	Waktu simulasi	200 ms
4	Area simulasi	800 m x 800 m untuk grid, 1000 x 1000 untuk riil
5	Banyak kendaraan	50
6	Radius Tranmisi	290 m
7	Kecepatan maksimal	10 m/s, 15 m/s, 20 m/s
8	Tipe data	Constant Bit Rate (CBR)

No	Parameters	Spesifikasi
9	Source / destination	Statik
10	Ukuran paket data	512 kb
11	Protokol	MAC dan IEEE 802.11p
12	Mode propagasi	Two way ground
13	Mobility model	Peta grid dan peta real Surabaya
14	Tipe kanal	Wireless channel

5.2 Hasil Uji Coba

Hasil uji coba dari skenario grid dan skenario surabaya dapat dilihat sebagai berikut :

5.2.1 Hasil Uji Coba Grid

Hasil pengujian data PDR, delay, dan RO pada skenario grid ditunjukkan pada tabel di bawah ini:

Tabel 5.3 Hasil PDR skenario Grid

Kecepatan	AODV	AODV+PGB
10 m/s	87.498 %	91.094 %
15 m/s	85.478 %	91.986 %
20 m/s	85.076 %	88.996 %

Tabel 5.4 Hasil *delay* skenario grid

Kecepatan	AODV	AODV+PGB
10 m/s	389.97 ms	797.177 ms
15 m/s	494.98922 ms	558.5974 ms
20 m/s	195.435 ms	486.54 ms

Tabel 5.5 Hasil RO skenario grid

Kecepatan	AODV	AODV+PGB
10 m/s	17364.2 paket	6484 paket
15 m/s	16978.8 paket	6373 paket

Kecepatan	AODV	AODV+PGB
20 m/s	17427.6 paket	6529.8 paket

Tabel 5.6 Hasil penyebaran paket RREQ skenario grid

Kecepatan	AODV	AODV+PGB
10 m/s	14448 paket	4999 paket
15 m/s	14340 paket	4930 paket
20 m/s	14684 paket	5094 paket

Dari data diatas, dibuat sebuah grafik yang merepresentasikan PDR, Delay dan RO yang ditunjukkan pada Gambar 5.1 Gambar 5.2 Gambar 5.3. Dari hasil skenario diatas PDR pada kecepatan 10 m/s memiliki nilai AODV+PGB lebih baik 3,6 % dari pada AODV. Pada kecepatan 15 m/s dan 20 m/s, terjadi penurunan nilai namun AODV+PGB cenderung menghasilkan nilai yang lebih baik dari pada AODV yakni dengan selisih nilai 6,5% dan 3,92% lebih baik.

Kesimpulan dari hasil skenario grid pada pengujian PDR, nilai PDR pada AODV+PGB lebih tinggi dari pada nilai PDR dari AODV. Kecepatan pada kendaraan tidak memberikan efek yang signifikan pada hasil pengujian dikarenakan hasil yang tidak terlalu terpaut jauh.

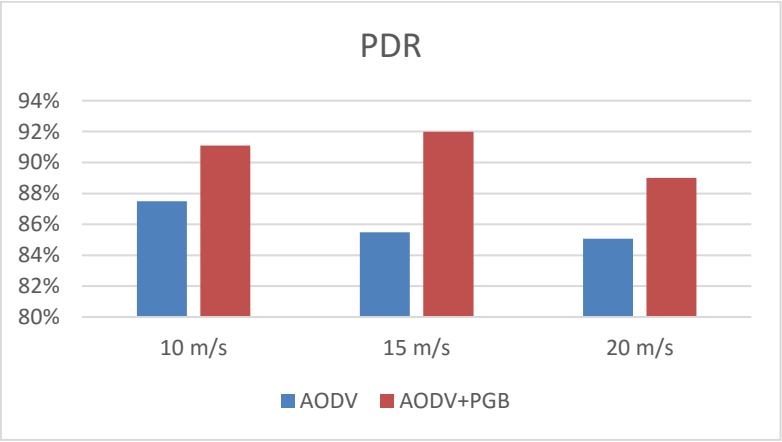
Hasil dari pengujian *routing overhead* pada skenario grid dengan kecepatan 10 m/s menunjukkan adanya pengurangan paket *routing control* antara AODV dan AODV+PGB sebesar 10880 paket dimana AODV+PGB lebih unggul dari pada AODV. Hal tersebut juga terjadi pada kecepatan 15 m/s dan 20 m/s dimana AODV+PGB lebih unggul dari pada AODV dengan selisi poin 10605 paket dan 10897 paket.

AODV+PGB lebih unggul pada *routing overhead* jika dibandingkan dengan AODV biasa dikarenakan *node* yang melakukan penyebaran paket RREQ menjadi lebih sedikit dari pada AODV saja karena dilakukan pembatasan pada siapa saja yang berhak melakukan proses penyebaran paket. Selain itu, jumlah paket bertipe RTR juga jauh berkurang. Paket bertipe RTR

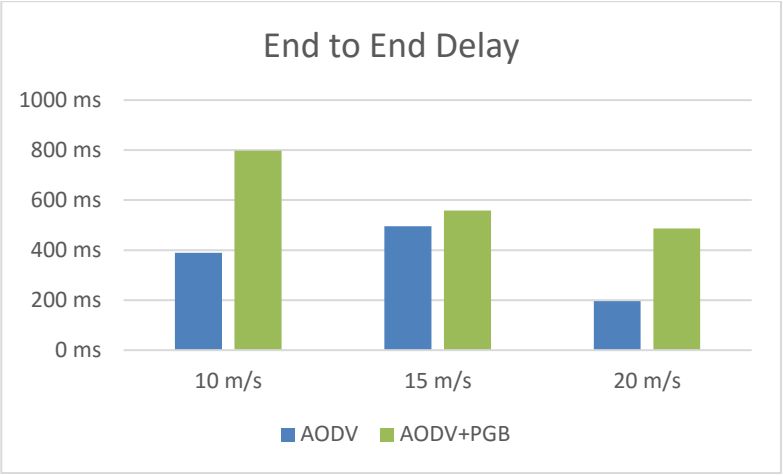
merupakan tipe paket yang dikirim *routing protocol* untuk melakukan operasi pencarian *node* dan sebagainya.

Pada jumlah penyebaran paket RREQ (Tabel 5.6), AODV+PGB lebih sedikit menyebarkan paket RREQ pada jaringan. Pada kecepatan 10 m/s, selisih paket yang disebarkan antara AODV dengan AODV+PGB sebesar 9449 paket RREQ. Sedangkan pada kecepatan 15 m/s dan 20 m/s sebesar 9410 paket dan 9590 paket. Dari hasil data tersebut terbukti bahwa AODV+PGB berhasil mengurangi jumlah paket RREQ yang tersebar saat proses *route discovery*.

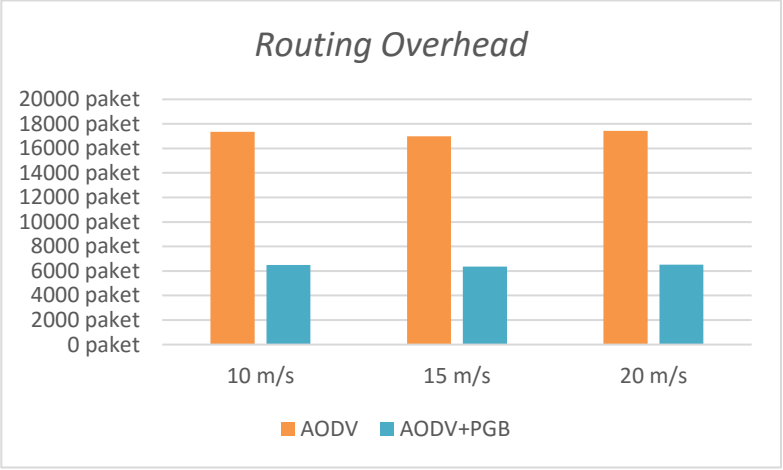
Namun pada nilai *End to end delay* hasil yang dihasilkan dari AODV jauh lebih baik dari pada hasil yang diberikan oleh AODV+PGB. Pada kecepatan 10 m/s AODV lebih cepat 407 ms dari pada AODV+PGB. Sedang pada kecepatan 15 m/s terjadi kenaikan delay terkecil yakni 63 ms. Sedangkan pada kecepatan 20 m/s kenaikan delay antara AODV dan AODV+PGB sebanyak 291 ms.



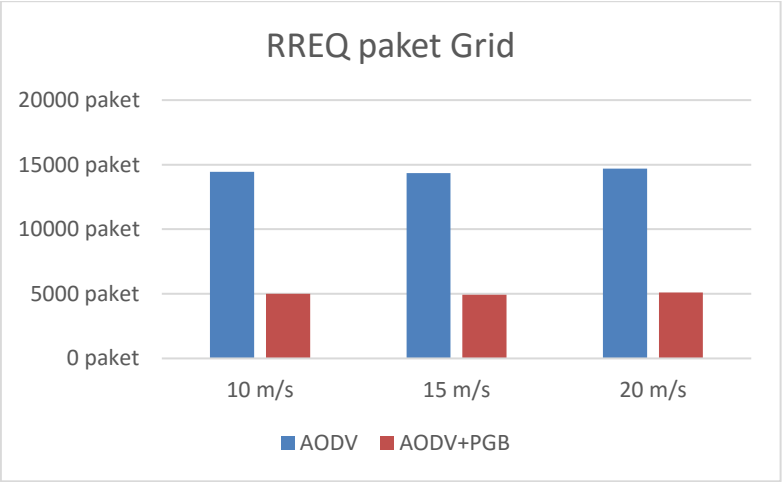
Gambar 5.1 Grafik PDR skenario grid.



Gambar 5.2 Grafik End to End Delay skenario grid



Gambar 5.3 Grafik perbedaan *routing overhead*



Gambar 5.4 Grafik perbedaan jumlah RREQ paket

5.2.2 Hasil Uji Coba Peta Riil Surabaya

Hasil pengujian data PDR, delay, dan RO pada skenario riil Surabaya ditunjukkan pada tabel di bawah ini:

Tabel 5.7 Hasil PDR skenario riil

Kecepatan	AODV	AODV+PGB
10 m/s	59.64%	70.62%
15 m/s	55.73%	60.66%
20 m/s	57.42%	63.03%

Tabel 5.8 Hasil *delay* skenario riil

Kecepatan	AODV	AODV+PGB
10 m/s	1560.668 ms	2457.448 ms
15 m/s	1768.841 ms	3482.02 ms
20 m/s	955.48 ms	3124.97 ms

Tabel 5.9 Hasil RO skenario riil

Kecepatan	AODV	AODV+PGB
10 m/s	10647.6 paket	4091.2 paket
15 m/s	10349 paket	4374 paket
20 m/s	8396 paket	3247.8 paket

Tabel 5.10 Hasil penyebaran paket RREQ skenario riil

Kecepatan	AODV	AODV+PGB
10 m/s	8872.20 paket	3219.40 paket
15 m/s	8555.80 paket	3509.20 paket
20 m/s	6841.40 paket	2554.80 paket

Dari data diatas, dibuat sebuah grafik yang merepresentasikan PDR, Delay dan RO yang ditunjukkan pada Gambar 5.5, Gambar 5.6, dan Gambar 5.7. Hasil pengujian secara keseluruhan hampir sama dengan pengujian menggunakan peta grid. PDR pada kecepatan 10 m/s memiliki peningkatan paling besar dari pada hasil PDR dari kecepatan yang lain yakni sebesar 10.98 % lebih unggul AODV+PGB dari pada PGB. Sedangkan pada kecepatan 15 m/s dan 20 m/s terjadi penurunan peningkatan pada AODV+PGB namun masih tetap unggul yakni dengan poin 4,9% dan 5,6%. Kecepatan kendaraan memberikan pengaruh namun tidak signifikan.

Hal yang sama juga terjadi pada *routing overhead*. Kenaikan juga hampir sama dengan kenaikan pada skenario grid. Pada kecepatan 10 m/s terjadi kenaikan sebanyak 6556 poin, unggul AODV+PGB dari pada AODV. Pada kecepatan 15 m/s dan 20 m/s juga terjadi kenaikan cukup besar yaitu 5975 poin dan 5148 poin. Kecepatan berpengaruh pada penurunan selisih poin maupun tidak terlalu signifikan

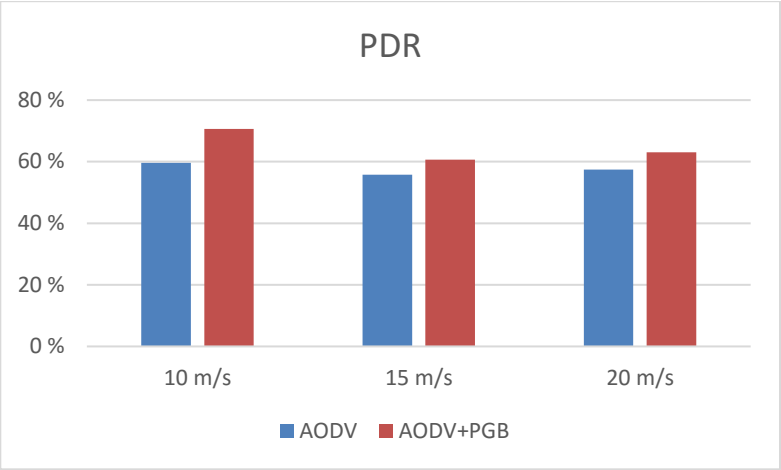
Paket RREQ yang disebar pada jaringan dari AODV+PGB lebih sedikit dari pada AODV jika dilihat dari hasil pengujian pada Tabel 5.10. pada kecepatan 10 m/s selisih paket RREQ yang disebar antara AODV+PGB dan AODV sebanyak 5653 paket dan lebih

sedikit AODV+PGB. Sedang pada kecepatan 15 m/s selisih yang terjadi sebesar 5064 paket. Pada kecaptan 20 m/s sebesar 4287 paket. Dari hasil tersebut terbukti bahwa AODV+PGB dapat mengurangi jumlah paket RREQ yang tersebar pada jaringan.

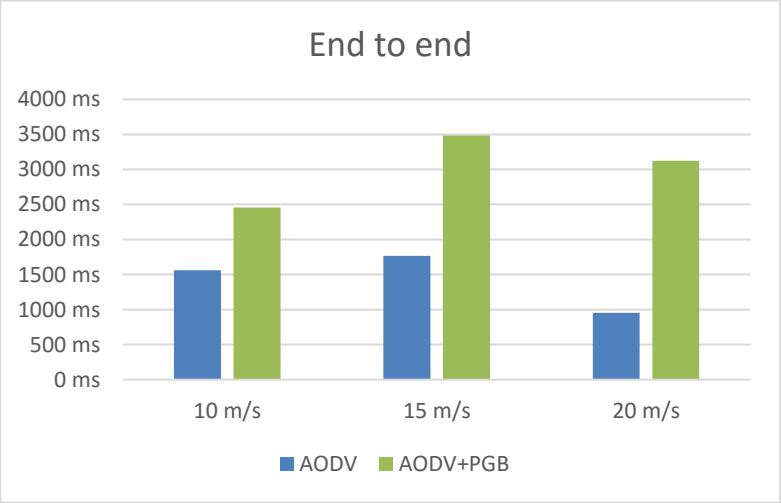
Pada pengujian *End to end delay* menunjukkan bahwa semakin meningkat kecepatan, selisih dari *delay* yang diberikan oleh AODV+PGB semakin meningkat. Pada kecepatan 10 m/s, selisih *delay* antara AODV dengan AODV+PGB sebesar 896 ms. Pada kecepatan 15 m/s, kenaikan selisih *delay* hampir satu detik dengan kecepatan 10 m/s yakni 1713 ms. Sedang pada kecepatan 20 m/s selisih *delay* antara AODV dan AODV+PGB telah mencapai 2 detik yakni 2169 ms.

Hasil pada skenario riil hampir sama dengan hasil dari skenario grid. Perbedaan hanya berada pada selisih nilai yang dihasilkan dari perbedaan hasil. Pada pengujian grid, hasil penghitungan PDR menunjukkan kecenderungan menurun jika kecepatan dinaikkan. Pada kecepatan 10 m/s, peningkatan nilai PDR sebanyak 18,4 %, sedangkan pada kecapatan 15 dan 20 m/s berturut-turut adalah 8,8 % dan 9,8 %. Pada skenario riil, hasil yang ditunjukkan tidak jauh berbeda. Pada kecepatan 10 m/s peingkatan hanya 4,11 %. Sedangkan pada kecepatan 15 m/s dan 20 m/s berturut-turut 7,6 % dan 4,6 %. Perbedaan yang terlihat jauh pada skenario grid dan skenario riil dikarenakan kondisi peta grid yang mendakati ideal sedangkan kondisi peta riil yang tidak dapat ditentukan.

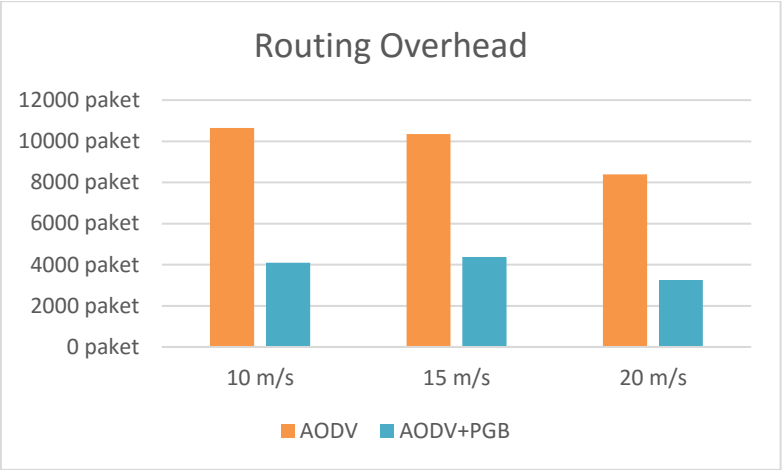
Kondisi yang berbeda terjadi pada pengujian *end to end delay*. Semakin tinggi kecepatan semakin tinggi pula nilai dari *end to end delay*. Pada peta grid, kenaikan terbesar terjadi pada kecepatan 20 m/s dimana terjadi kenaikan sebanyak 227 % pada AODV+PGB dari AODV. Sedangkan pada riil skenario, kenaikan yang terjadi sebesar 149 % pada AODV+PGB dari AODV.



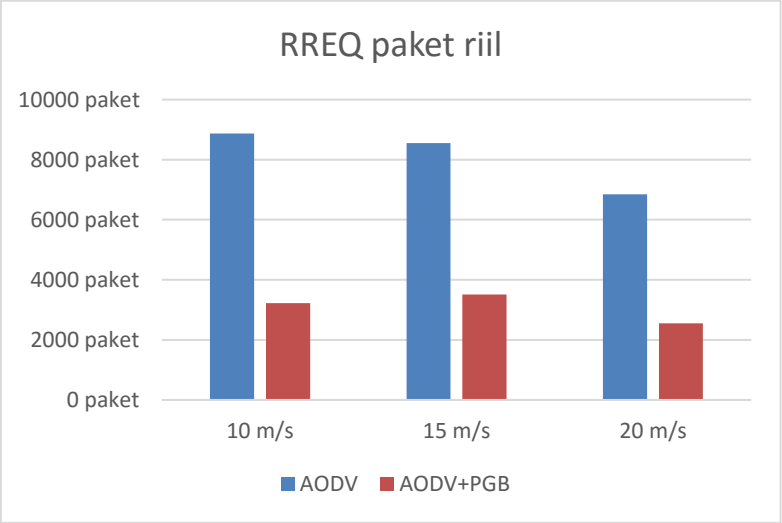
Gambar 5.5 grafik PDR skenario Riil



Gambar 5.6 grafik *End to end delay* skenario Riil



Gambar 5.7 grafik *Routing Overhead* skenario Riil



Gambar 5.8 Grafik perbedaan jumlah RREQ paket

Pada salah satu percobaan hasil *map* yang didapat bisa dilihat pada Gambar 5.9. Ketika dijalankan dengan menggunakan AODV dan dilakukan analisa pada berkas *trace route* yang telah dihasilkan, maka *node* yang melakukan penyebaran RREQ sebanyak 45 *node*. Sedangkan ketika dijalankan menggunakan AODV+PGB *node* yang melakukan penyebaran paket RREQ hanya berjumpa 19 *node*. Dari hasil percobaan telah dibuktikan bahwa *node* yang melakukan proses penyebaran RREQ jauh berkurang pada AODV+PGB. Hal ini membuktikan bahwa proses pembatasan pada mekanisme PGB telah berjalan secara semestinya.

```
r 2.557827335 _33_ RTR  --- 0 AODV 48 [0 ffffffff 30
800] ------- [48:255 -1:255 30 0] [0x2 1 1 [49 0]
[48 4]] (REQUEST)
r 2.557827578 _45_ RTR  --- 0 AODV 48 [0 ffffffff 30
800] ------- [48:255 -1:255 30 0] [0x2 1 1 [49 0]
[48 4]] (REQUEST)
r 2.557827627 _47_ RTR  --- 0 AODV 48 [0 ffffffff 30
800] ------- [48:255 -1:255 30 0] [0x2 1 1 [49 0]
[48 4]] (REQUEST)
s 2.567356382 _33_ RTR  --- 0 AODV 48 [0 ffffffff 30
800] ------- [33:255 -1:255 29 0] [0x2 2 1 [49 0]
[48 4]] (REQUEST)
```

Gambar 5.9 contoh *trace file*

[Halaman ini sengaja dikosongkan]

LAMPIRAN

A 1. 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 val(ifqlen)           50            ;
set val(nn)         50                  ;
set val(rp)         AODV                ;
set val(energymodel)      EnergyModel   ;
set val(initialenergy)    100           ;
set val(lm)         "off"               ;
set val(x)       1000                   ;
set val(y)          1000                ;
set val(stop)       200                 ;
set val(cp)             "traffic1"      ;
set val(sc)             "ns2.tcl"
set ns_      [new Simulator]
set tracefd [open s-aodv2.tr w]
set namtrace [open s-aodv2.nam w]
#$ns_ use-newtrace
$ns_ trace-all $tracefd
$ns_ namtrace-all-wireless $namtrace $val(x)
$val(y)
Agent/AODV set num_nodes $val(nn)

# Setting up Topography Object

set topo [new Topography]
$topo load_flatgrid $val(x) $val(y)
create-god $val(nn)

set chan_1_ [new $val(chan)]

# Configure the nodes
```

```
$ns_ node-config    -adhocRouting $val(rp) \
                    -llType $val(ll) \
                    -macType $val(mac) \
                    -channel $chan_1_ \
                    -ifqType $val(ifq) \
                    -ifqLen $val(ifqlen) \
                    -antType $val(ant) \
                    -propType $val(prop) \
                    -phyType $val(netif) \
                    -topoInstance $topo \
                    -agentTrace ON \
                    -routerTrace ON \
                    -macTrace OFF \
                    -movementTrace ON \

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

# Provide Initial Location of Mobile Nodes
puts "Loading connection pattern..."
source $val(cp)
puts "Loading scenario file..."

source $val(sc)

for {set i 0} {$i < $val(nn)} {incr i} {
$ns_ initial_node_pos $node_($i) 30
}
for {set i 0} {$i < $val(nn)} { incr i} {
$ns_ at $val(stop) "$node_($i) reset"
}
$ns_ at 200.01 "puts \"end simulation\" ; $ns_
halt"
proc stop {} {
global ns_ tracefd namtrace
$ns_ flush-trace
close $tracefd
close $namtrace
#exec nam aodv.nam &
}
$ns_ run
```

A 2 *Script Pattern* Skenario

```
#
# nodes: 50, max conn: 1, send rate: 1, seed: 1
#
#
# 1 connecting to 2 at time 2.5568388786897245
#
set udp_(0) [new Agent/UDP]
$ns_ attach-agent $node_(48) $udp_(0)
set null_(0) [new Agent/Null]
$ns_ attach-agent $node_(49) $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_ 10000
$cbr_(0) attach-agent $udp_(0)
$ns_ connect $udp_(0) $null_(0)
$ns_ at 2.5568388786897245 "$cbr_(0) start"
#
#Total sources/connections: 1/1
#
```

A 3 Kode awk Perhitungan *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 4 Kode awk Perhitungan *Average End-to-End Delay*

```
BEGIN {

    seqno = -1;

    count = 0;}
{
    if($4 == "AGT" && $1 == "s" && seqno < $6) {

          seqno = $6;

    }
    #end-to-end delay

    if($4 == "AGT" && $1 == "s") {

          start_time[$6] = $2;

    } else if(($7 == "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;
    print "\n";
    print "Average End-to-End Delay   = " n_to_n_delay
* 1000 " ms";
    print "\n";
}
```

A 5 Kode awk Perhitungan *Routing Overhead*

```
BEGIN{
recvs = 0;
routing_packets = 0;
}

{
if (( $1 == "r") &&  ( $7 == "cbr" ) && ( $19=="4" ))
recvs++;
if (($1 == "s" || $1 == "r") && $4 == "RTR" && $7
=="AODV")    routing_packets++;
}

END{
printf("\nrouting packets %d", routing_packets);
printf("\ndata = %d", recvs);
printf("\nNRL = %.3f\n", routing_packets/recvs);
}
```

A 6 Kode awk perhitungan paket RREQ

```
BEGIN {
sent=0;
received=0;
}
$0 ~/^r.*REQUEST)/ {
        sent ++ ;
}
END{
 printf " Packet Sent:%d\n",sent;

}
```

BAB VI
KESIMPULAN DAN SARAN

Pada bab ini akan diberikan kesimpulan yang diambil selama pengerjaan Tugas Akhir ini serta saran-saran tentang pengembangan yang dapat dilakukan terhadap tugas akhir ini dimasa yang akan datang.

6.1 Kesimpulan

1. Hasil pengujian PDR pada skenario grid AODV+PGB lebih baik dengan hasil 90,7% dari pada AODV dengan nilai 86,02%. Sedangkan pada skenario riil AODV+PGB unggul dengan hasil 64,77% sedang pada AODV hasil yang dicapai hanya 67,60%. Dari hasil tersebut metode PGB terbukti mampu menurunkan jumlah paket yang dikirm dan memberikan peningkatan pada hasil PDR pada skenario grid dan skenario riil.
2. Hasil pengujian *average end to end delay* menunjukkan hasil yang berbeda yakni pada skenario riil AODV lebih unggul dengan hasil 1428,33 ms. Lebih baik 1593,15 ms dari pada AODV+PGB dengan hasil 3021,48 ms. Sedangkan pada skenario grid hasil yang diberikan tidak berbeda dengan hasil skeraio riil yakni AODV lebih unggul dengan hasil 360,13 ms dari pada AODV+PGB dengan hasil 614,15 ms. Hal ini depengaruhi oleh kondisi *path* atau jalur yang dilalui untuk mengirimkan RREP berbeda pada AODV dan AODV+PGB.
3. Hasil pengujian *Routing Overhead* pada skenario grid menunjukkan penurunan jumlah paket *routing control* yang tersebar menggalami penurunan pada AODV+PGB dengan jumlah paket yang tersebar sebanyak 6462 paket sedang pada AODV paket yang tersebar sebanyak 17257 paket. Pada skenario riil, AODV+PGB lebih sedikit mengirmkan paket dengan 3904 paket sedangkan AODV menyebarkan 9797 paket.

4. Hasil dari penghitungan penyebaran paket RREQ menunjukkan penurunan pada AODV+PGB. Pada skenario riil jumlah paket yang disebar rata-rata dari ketiga kecepatan sebanyak 3094 paket untuk AODV+PGB sedangkan pada AODV paket yang disebar sebanyak 8090 paket. Sedang pada skenario grid AODV+PGB hanya menyebar paket RREQ sebanyak 5008 paket dan pada AODV menyebar sebanyak 14491 paket. Kesimpulan dari percobaan RREQ bahwa penyebaran paket RREQ jauh berkurang pada AODV+PGB dari pada pada AODV dan kondisi tersebut dapat menanggulangi permasalahan *broadcast storm* pada proses *route discovery* di AODV.
5. Kecepatan kendaraan / *node* memberikan pengaruh pada protokol AODV dan AODV+PGB. Hal ini berdasarkan dari ketiga skenario pengujian.
6. Hasil pada skenario grid lebih bagus dari pada skenario riil dikarenakan kondisi grid yang stabil berbeda dengan kondisi riil yang memiliki kondisi lebih tak beraturan.

6.2 Saran

Saran yang diberikan untuk pengembangan aplikasi ini adalah :

1. Untuk selanjutnya bisa dilakukan penelitian lebih lanjut pada protokol berbasis *position base,* atau bersifat proaktif.
2. Perlu diadakan penelitian lebih lanjut untuk optimasi waktu *hold* pada *scheduler.*

DAFTAR PUSTAKA

[1] V. Naumov, R. Baumann and T. Gross, "An Evaluation of Inter-Vehicle Ad Hoc Networks Based on Realistic Vehicular Traces," *MobiHoc '06 Proceedings of the 7th ACM international symposium on Mobile ad hoc networking and computing,* pp. 108-119, 2006.

[2] D. P. I. I. Ismail and M. H. F. Ja'afar, "Mobile Ad Hoc Network Overview," in *Asia-Pacific Conference On Applied Electromagnetics Proceedings*, Malaka, 2007.

[3] R. F. Sari, A. Syarif and B. Budiardjo, "Analisis Kinerja Protokol Routing Ad Hoc On-Demand Distance Vector (AODV) Pada Jaringan Ad Hoc Hybrid: Perbandingan Hasil Simulasi dengan NS-2 Dan Implementasi Pada Testbed dengan PDA," *MAKARA, TEKNOLOGI, VOLUME 12, NO. 1,* pp. 7-18, 2008.

[4] M. Sangari and Dr.K.Baskaran, "A Comprehensive Survey on Efficient Routing Protocols And Simulation Tools For VANET," *International Journal of Computer Science and Information Technologies, ,* vol. 5 (3), pp. 2729-2737, 2014.

[5] Wiki, "SUMO User Documentation," SUMO, 6 June 2016. [Online]. Available: http://sumo.dlr.de/wiki/SUMO_User_Documentation. [Accessed 20 June 2016].

[6] OpenStreetMap Foundation, "OpenStreetMap," OpenStreetMap Foundation, [Online]. Available: https://www.openstreetmap.org. [Accessed 20 June 2016].

[7] Wiki, "JOSM," Wiki, 13 June 2016. [Online]. Available: http://wiki.openstreetmap.org/wiki/JOSM. [Accessed 20 June 2016].

[8] Wikipedia, "AWK," Wikipedia, 10 June 2016. [Online]. Available: https://en.wikipedia.org/wiki/AWK. [Accessed 20 June 2016].

[9] K. Fall and K. Varadhan, "The Manual (formerly Notes and Documentation)," UC Berkeley, LBL, USC/ISI, and Xerox PARC., 5 November 2011. [Online]. Available: http://www.isi.edu/nsnam/ns/doc/index.html. [Accessed 17 Desember 2015].

BIODATA PENULIS

Raga Krilido Oktananda lahir pada tanggal 11 Oktober 1994 dari pasangan Hari Krido Prasetyo dan Lisminingsih sebagai anak pertama. Penulis menempuh pendidikan mulai dari SDN Tambakagung 1 (2000 - 2006), SMPN 1 Puri (2006 - 2009), SMAN 1 Sooko (2009 - 2012) dan S1 Teknik Informatika ITS (2012 - 2016).

Selama masa kuliah, penulis aktif dalam organisasi Himpunan Mahasiswa Teknik Computer (HMTC). Selama menempuh pendidikan penulis juga aktif dalam organisasi kemahasiswaan antara lain staff Kementrian Dalam Negeri Badan Eksekutif Mahasiswa ITS pada tahun ke-2 dan anggota Dewan Perwakilan Mahasiswa ITS pada tahgun ke-3.

Selama kuliah di teknik informatika ITS, penulis mengambil rumpun mata kuliah Arsitektur dan Jaringan Komputer (AJK). Komunikasi dengan penulis dapat melalui email : **raga.krilido@gmail.com.**