TUGAS AKHIR – KI141502

IMPLEMENTASI RELIABLE MULTIPOINT RELAY PADA OPTIMIZED LINK STATE ROUTING PROTOCOL (OLSR) DENGAN MENGGUNAKAN SIMULATOR NS2

MOHAMAD INDRA CAHYA
NRP 5121100028

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

Dosen Pembimbing II
Ir. FX. ARUNANTO, M.Sc.

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2015

[Halaman ini sengaja dikosongkan]

UNDERGRADUATE THESIS – KI1502

IMPLEMENTATION RELIABLE MULTIPOINT RELAY ON OPTIMIZED LINK STATE ROUTING PROTOCOL (OLSR) USING NS2 SIMULATOR

MOHAMAD INDRA CAHYA
NRP 5121100028

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

Supervisor II
Ir. FX. ARUNANTO, M.Sc.

DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY

[Halaman ini sengaja dikosongkan]

LEMBAR PENGESAHAN

IMPLEMENTASI RELIABLE MULTIPOINT RELAY PADA OPTIMIZED LINK STATE *ROUTING* PROTOCOL (OLSR) DENGAN MENGGUNAKAN SIMULATOR NS2

TUGAS AKHIR

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

Oleh
MOHAMAAD INDRA CAHYA
NRP: 5121 100 028

Disetujui oleh Dosen Pembimbing Tugas Akhir:

1. Dr.Eng. Radityo Anggoro, S.Kom., M.Sc. .....
   NIP: 19841016 200812 1 002 (Pembimbing 1)

2. Ir. F.X. Arunanto, M.Sc. .....
   NIP: 19570101 198303 1 004 (Pembimbing 2)

SURABAYA
JUNI, 2015

[Halaman ini sengaja dikosongkan]

IMPLEMENTASI RELIABLE MULTIPOINT RELAY PADA OPTIMIZED LINK STATE *ROUTING* PROTOCOL (OLSR) DENGAN MENGGUNAKAN SIMULATOR NS2

Nama Mahasiswa	**: MOHAMAD INDRA CAHYA**
NRP	**: 5121100028**
Jurusan	**: Teknik Informatika FTIF-ITS**
Dosen Pembimbing 1	**: Dr.Eng. Radityo Anggoro, S.Kom.,M.Sc**
Dosen Pembimbing 2	**: Ir. F.X Arunanto, M.Sc**

Abstrak

OLSR (Optimized Link State Routing Protocol) adalah sebuah protocol link-state routing yang proactif. OLSR mempunyai 2 buah kontrol, yaitu "HELLO" dan "TC" dan mengandalkan pemilihan node MPR (Multi Point Relay) dalam pengiriman sebuah pesan. MPR adalah beberapa node yang telah dipilih dengan kriteria tertentu yang bertugas untuk meneruskan sebuah pesan. Namun posisi dari MPR tersebut tersebar dengan acak, sehingga ada kemungkinan sebauh pesan akan drop atau tidak dapat sampai ke node tujuan dikarenakan node MPR tersebt terlalu jauh dengan node asal maupun node tujuan.

Berdasarkan permasalahan tersebut, maka dilakukan penelitian terhadap kinerja protokol OLSR dengan menambahkan algoritma PGB (preferred Group Broadcasting). PGB adalah metode pengiriman sebuah pesan hanya kepada node yang berada pada area tertentu saja. Diharapkan dengan ditambahkannya PGB pada OLSR akan menjamin sebuah pesan akan sampai kepada node tujuan dan tidak mengalami storm broadcasting.

Simulator yang digunakan pada uji coba adalah NS2. Diharapkan dari uji coba yang dilakukan akan diketahui bagaimana efek yang diberikan oleh PGB pada OLSR dengan menghitung nilai PDR (packet Delivery Ratio), End-to-End Delay dan TC (Topology Control).

Kata kunci: VANETs, OLSR, PGB, Signal Strength, **NS2**

[Halaman ini sengaja dikosongkan]

IMPLEMENTATION RELIABLE MULTIPOINT RELAY ON OPTIMIZED LINK STATE *ROUTING* PROTOCOL (OLSR) USING NS2 SIMULATOR

Student's Name	**: MOHAMAD INDRA CAHYA**
Student's ID	**: 5121100028**
Department	**: Teknik Informatika FTIF-ITS**
First Advisor	**: Dr.Eng. Radityo Anggoro, S.Kom.,M.Sc**
Second Advisor	**: Ir. F.X Arunanto, M.Sc**

Abstract

OLSR (Optimized Link State Routing Protocol) proactive protocol. OLSR has 2 kind of control,"HELLO" and "TC" and relied on MPR (Multi Point Relay) to sending a message. MPR is some nodes that have been selected with certain criteria which served to pass on a message. However, the position of the MPR is scattered with randomly, so there is a possibility of obstruction message will drop or can not get to the destination node because the node MPR too far with the source node or with the destination node.

Based on these problems, then do research on the performance of OLSR protocol by adding algorithms PGB (preferred Group Broadcasting). PGB is a method of sending a message only to nodes that are in a particular area. PGB is expected with the addition of the OLSR will guarantee a message will get to the destination node and prevent storm broadcasting problem.

Network simulator that is used to conduct this study was NS-2. It is expected from the experiments performed will be known how the effect given by the PGB on OLSR by calculating the value of the PDR (Packet Delivery Ratio), End-to-End Delay and TC (Topology Control).

Keyword: VANETs, OLSR, PGB, Signal Strength, NS2

[Halaman ini sengaja dikosongkan]

DAFTAR ISI

[Halaman ini sengaja dikosongkan]

DAFTAR GAMBAR

DAFTAR TABEL

[Halaman ini sengaja dikosongkan]

BAB I
PENDAHULUAN

1.1 Latar Belakang

Teknologi *wireless* saat ini telah didukung oleh berbagai perangkat *mobile*. Sehingga memungkinkan suatu perangkat mobile mengirimkan data ke perangkat lainnya. Perangkat *mobile* disini mencakup *netbook, tablet, smartphone* dan berbagai perangkat yang terhubung menggunakan teknologi. Dengan adanya kemampuan pengiriman data antara perangkat *mobile* tersebut, maka memungkinkan perangkat-perangkat tersebut saling terhubung satu sama lain untuk saling bertukar data pada jaringan *MANET (Mobile Ad hoc Network)* ataupun *VANET (Vehicular Ad-hoc Networks).*

Pada penelitian kali ini akan digunakan jaringan VANET, karena VANET merupakan jenis khusus dari MANET karena memiliki mobilitas yang sangat tinggi [1]. Namun dalam implementasinya VANETs tersebut akan membentuk banyak route yang menghubungkan tiap *node* yang akan membuat beban *routing* menjadi besar, selain itu VANETs juga dapat mengalami terjadinya *Storm Broadcasting* [2]. Untuk mengatasi permasalahan tersebut solusi yang dapat ditawarkan adalah dengan menggunakan teknik *routing* OLSR *(Optimized Link State Routing Protocol)*. OLSR adalah optimasi protokol *routing* IP untuk jaringan mobile *ad hoc*, yang juga dapat digunakan pada jaringan nirkabel. OLSR adalah *link-state routing* protokol *proaktif*, menggunakan *path forwarding hop* terpendek untuk mengirimkan data dan haya memilih beberapa *node* tetangga untuk meneruskan paket data yang disebut *node* MPR, sehingga dapat mengurangi beban *routing* dan dapat mencegah terjadinya *Storm Broadcasting* [3]. Namun sebelum dapat melakukan *routing* OLSR akan dilakukan penghitungan kekuatan signal dari *node* yang ada dengan menggunakan metode broadcast protokol PGB *(Preferred Group Broadcasting)*. Dengan menggabungkan PGB dan protokol OLSR diharapkan akan didapatkan *node* MPR yang lebih bagus/terpercaya dalam meneruskan pengiriman paket data yang disebut sebagai *Reliable Multipoint Relay*. Dengan didapatkannya

node relay yang lebih bagus diharapkan akan didapatkan satu rute pengiriman paket data yang lebih efisien dan stabil. Dari implementasi tersebt akan menghasilkan protokol baru yaitu OLSR+PGB. Protokol OLSR+PGB akan diuji pada simulator NS2 *(Network Simlator 2)* agar paket data yang dikirim hanya melewati *node* yang terbaik dan mendapatkan hasil yang lebih optimal.

1.2 Rumusan Masalah

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

1. Bagaimana mengimplementasikan Protokol PGB kedalam metode *routing* OLSR pada jaringan VANETs dengan menggunakan NS2?
2. Bagaimana mencegah terjadinya *storm broadcasting* pada OLSR?

1.3 Batasan Masalah

Batasan pada permasalahan yang dibahas dalam tugas akhir ini adalah:

1. Simulator yang digunakan adalah NS versi 2.35 (NS2.35).

1.4 Tujuan

Mencegah terjadinya *Storm Broadcasting* dengan cara mengimplementasikan metode *routing* PGB dengan protokol OLSR sehingga diharapkan akan mampu mengrangi jmlah pesan TC, *end-to-end Delay* dan meningkatkan *Packet Delivery Ratio*.

1.5 Manfaat

Stabilisasi koneksi antar node dan mempercepat proses pengiriman paket data.

1.6 Metodologi

Penyusunan proposal tugas akhir

Penyusunan Proposal Tugas Akhir ini merupakan tahap awal dalam proses pengerjaan Tugas Akhir. Dalam proposal ini akan dilakukan pengimplementasian algoritma PGB *(Preferred Group*

Broadcasting) pada metode *routing* OLSR *(Optimized Link State Routing Protocol).*

1. **Studi Literatur**
   Tugas Akhir ini menggunakan literatur *paper* dan buku. *Paper* yang digunakan adalah *"An Evaluation of Inter-Vehicle Ad Hoc Networks Based on Riilistic Vehicular Traces"* dan "*Optimized Link State Routing Protocol for Ad Hoc Network"*. *Paper* tersebut menjelaskan mengenai implementasi algoritma *broadcasting Preferred Group Broadcasting* dan menjelaskan tentang protocol OLSR
2. **Implementasi**
   Pada tahap ini dilakukan implementasi dari metode PGB (*Preferred Group Broadcasting*) dan metode *routing* OLSR (*Optimized Link State Routing Protocol)* untuk mengatasi masalah *broadcasting strom* dengan menggunakan simulator NS2.
3. **Pengujian dan Evaluasi**
   Pengujian dilakukan dengan menjalankan skenario simulasi dengan beberapa paremeter pada NS2 dengan metode *routing* OLSR + PGB. Evaluasi dilakukan dengan melihat hasil dari implementasi metode *routing* AODV + *PGB.* Beberapa parameter yang digunakan adalah:

   1. PDR (*packet delivery ratio*)
      PDR adalah jumlah paket data yang 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. TC *(Topology Control)*
      Topology Control merupakan jenis kontrol pada protocol OLSR yang dapat dikirim ke seluruh jaringan secara broadcasting.

4. **Penysunan Buku Tgas 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 bagi 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 yang digunakan sebagai bahan 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.

[Halaman ini sengaja dikosongkan]

BAB II
TINJAUAN PUSTAKA

Bab ini berisi penjelasan beberapa teori yang berkaitan dengan pengimplementasian protokol yang akan dibuat. Penjelasan ini bertujuan untuk memberikan gambaran umum tentang skenario uji coba yang akan dilakukan dan berguna sebagai penunjang dalam pengembangan protokol yang akan dibuat.

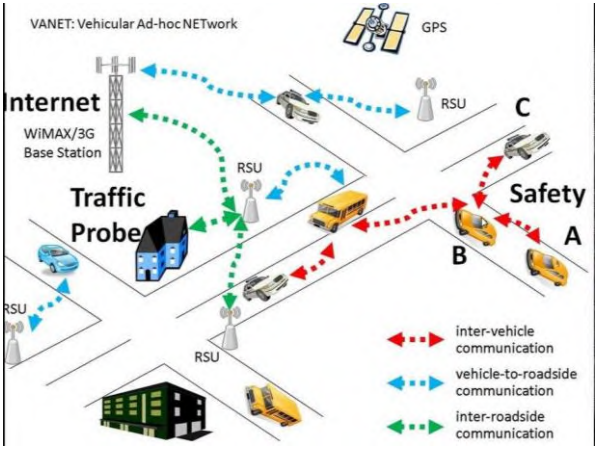
2.1 VANET (*Vehicular Ad hoc Network*)

Vanet merupakan sebuah jaringan yang terorganisir yang terbentuk dengan menghubungkan kendaraan dan RSU (*Roadside Unit)*, dan RSU lebih lanjut terhubung ke jaringan *backbone* berkecepatan tinggi melalui koneksi sebuah jaringan. Kepentingan peningkatan komunikasi baru-baru ini telah diajukan pada aplikasi melalui V2V (*Vehicle to Vehicle*) dan V2I (*Vehicle to Infrastructure)*, yang bertujjuan untuk meningkatkan keselamatan dalam mengemudi dan mengatur lalu lintas.

Dalam VANETs, RSUs dapat membantu dalam menemukan fasilitas-fasilitas umum, seperti restoran, Stasiun Pengisian Bahan Bakar Umum, selain itu RSUs juga dapat melakukan *broadcast* pesan yang berhubungan dengan informasi lain yang dibutuhkan oleh pengendara, seperti maksimum kurva kecepatan pada suatu wilayah. Sebagai contoh, sebuah kendaraan dapat mengetahui informasi / berkomunikasi dengan lampu lalu lintas melalui V21, dan lampu lalu lintas tersebut mampu memberikan informasi tentang status dari lampu lalu lintas tersebut.

Hal tersebut dapat sangat membantu pengguna jalan dalam berkendara dalam berbagai situasi, misalnya pada saat cuaca buruk ataupun saat berkendara di tempat yang belum pernah dilewati sebelumnya. Dengan adanya teknologi seperti diatas diharapkan akan mampu mengurangi terjadinya kecelakaan. Dengan menggunakan komunikasi V2V, pengendara bisa mendapatkan informasi yang lebih baik dalam mengambil keputusan untuk menanggapi situasi yang tidak diharapkan.

Untuk dapat mencapai hal tersebut, suatu OBU *(On-Broadrs Unit)* secara teratur menyiarkan pesan yang terkait dengan informasi dari posisi pengendara, waktu saat ini, arah mengemudi, kecepatan, status rem, sudut kemudi, lampu sein, percepatan / perlambatan dan kondisi lalu lintas.



Gambar 2.1 Ilustrasi VANETs [4]

Pada tugas akhir ini VANETs digunakan sebagai jenis jaringan nirkabel dalam menganalisis kinerja *routing protocol* OLSR *(Optimized Link State Routing Protocol)* dan PGB *(Prefered Group Broadcasting).*

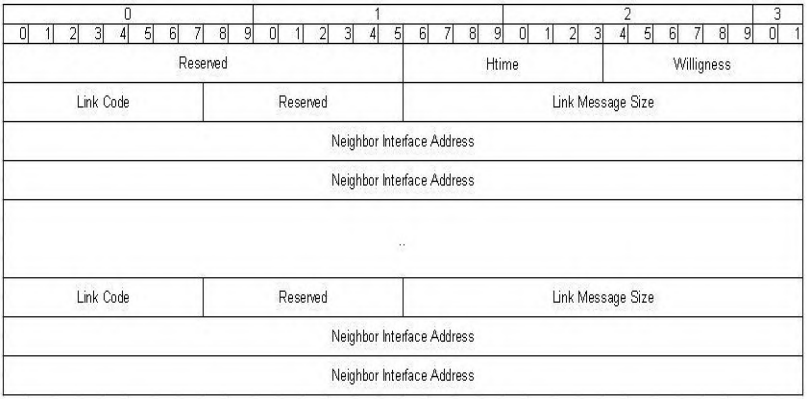
2.2 OLSR (*Optimized Link State Routing Protocol)*

OLSR adalah sebuah protokol *routing* untuk IP yang di optimisasi untuk jaringan *mobile ad-hoc*, yang juga dapat digunakan pada jaringan *wireless ad-hoc*. OLSR adalah sebuah *protocol link-state routing* yang *proactif*, yang artinya dia akan selalu memperbarui tabel *routing* secara otomatis tanpa ada permintaan terlebih dahulu [5].

Perubahan topologi pada suatu jaringan dapat menyebabkan luapan informasi topologi yang berujung *overhead* pada semua *node* yang ada di jaringan. Untuk mencegah terjadinya *over head* digunakan teknik *Multi Point Relay* (MPR).tujuan utama dari

MPR adalah untuk mengurangi luapan dengan cara memilih beberapa *node* yang bertindak sebagai MPR. Sehingga hanya *node* MPR saja yang akan meneruskan pesan yang diterima dengan kata lain juga dapat membuat rute menjadi semakin pendek. Pemilihan MPR adalah dengan cara memilih node tetangga yang dapat menjangkau banyak *node* tetangga selanjutnya.

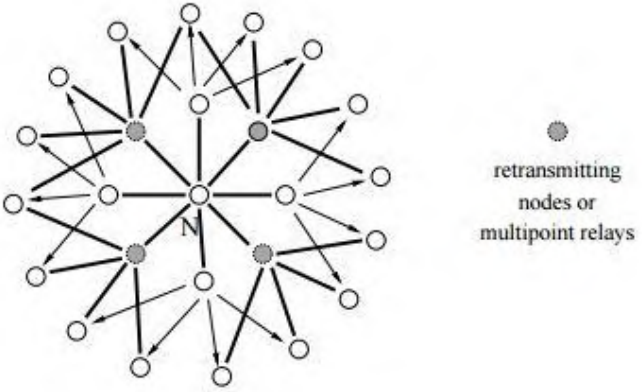
OLSR menggunakan 2 jenis kontrol, yaitu pesan "HELLO" dan "*Topology Control* (TC)". Pesan HELLO digunakan untuk menemukan info tentang kondisi link dan *node* tetangga. Pesan HELLO juga digunakan untuk memilih MPR *selection set*. Tugas dari MPR *selection set* adalah memilih *node* tetangga untuk bertindak sebagai *node* MPR. Sehingga dengan adanya pesan hello ini *node* pengirim dapat menentukan *node* MPR-nya. Pesan hello hanya dapat dikirim sejauh 1 *hop*, tetepi pesan TC dapat dikirim ke seluruh jaringan secara *broadcasting*.fungsi dari pesan TC ini adalah untuk menyebar informasi tentang *node* tetanga mana saja yang telah digunakan sebagain *node* MPR. Pesan TC hanya disebar satu periodik dan hanya *node* MPR yang dapat meneruskan pesan TC tersebut.



Gambar 2.2 Tabel *Roting* HELLO

0										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
ANSN										Reserved																					
Advertised Neighbor Main Address																															
Advertised Neighbor Main Address																															

Gambar 2.3 Tabel *Routing* TC



Gambar 2.4 Contoh Pemilihan *Node* MPR

2.3 PGB (*Preferred Group Broadcasting*)

PGB bukanlah sebuah protokol penyiaran yang terbaik tetapi merupakan solusi untuk mencegah masalah *storm broadcasting*. *Storm broadcasting* merupakan permasalahan dimana satu paket data berputar-putar pada *node-node* tetangga atau terjadi *looping* karena tidak menemukan node tujuan. Semakin banyak *node* tetangga yang meneruskan paket data, maka semakin besar pula kemungkinan terjadinya *storm broadcasting*. Tujuan utama pada PGB adalah agar rute yang akan dilewati dapat lebih stabil [6]. Setiap *node* di PGB akan merasakan tingkat kekuatan sinyal dari *node* terdekat/tetangga. Kekuatan sinyal digunakan untuk mendapatkan lama waktu *timeout*. Hanya *node* dengan batas waktu terpendek akan mengirim ulang/meneruskan pesan yang dikirim. PGB dapat mengurangi jumlah RREQ [7].

PGB mengklasifikasikan setiap *node* yang menerima paket *broadcast* kedalam salah satu dari tiga kelompok, berdasarkan tingkat *signal strength* dan juga waktu *delay* yang didapatkan dari node tetangga. Kelompok tersebut adalah IN, PG dan OUT.

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.

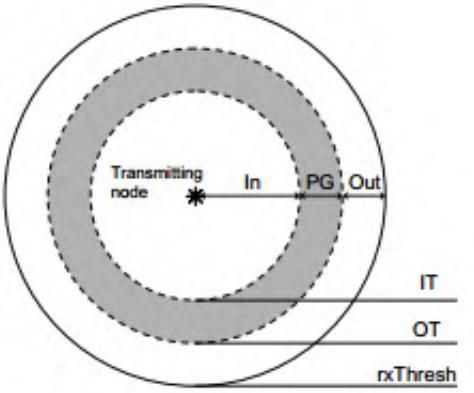
Persamaan untuk penghitungan *delay*:

$$\boldsymbol{hoTime = T_i + jitterT_i}$$

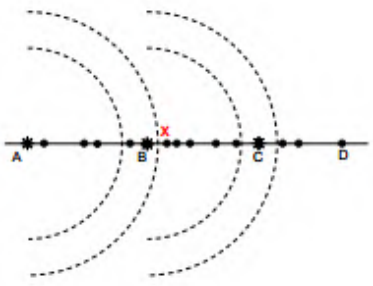
$$\boldsymbol{\Delta P_i = ||rxTh - rxP| - f_i|}$$

$$\boldsymbol{T_i = \Delta P_i . dt_i + minT_i}$$

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



Gambar 2.5 Pembagian *Range* PGB



Gambar 2.6 Contoh PGB pada Jalan Lurus

Pada Tugas Akhir ini algoritma PGB akan digunakan untuk mengatasi masalah *broadcast storm*. Metode yang digunakan yaitu dengan mengatur jarak atau *signal* tertentu dan juga melihat nilai *delay* dari setiap node dalam menerima *broadcast* pesan HELLO. Hal ini memanfaatkan sifat dari NS2 dimana setiap *node* memiliki kekuatan *signal* yang dapat dignakan sebagai acuan pengiriman pesan.

Dengan cara menentukan *range signal*, membandingkan waktu *delay* antar *node* tetangga dan membagi *node* tetangga kedalam beberapa area sesuai dengan algoritma PGB yang sudah diterapkan, maka akan didapatkan *node* tetangga yang lebih baagus/terpercaya kualitasnya untuk meneruskan paket data. Node tersebtlah yang disebut sebagai *Reliable Multipoint Relay*.

2.4 SUMO *(Simulation of Urban Mobility)*

SUMO adalah program aplikasi simulator yang digunakan untuk menciptakan lingkungan bergerak yang bersifat dinamis. SUMO merupakan program yang bersifat *free* dan *open-source*. SUMO dikembangkan pertama kali pada tahun 2001 dengan tujuan untuk membantu penelitian yang memerlukan pergerakan kendaran pada jalan raya [8].

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

- *Netgenerate*

  Netgenerate merupakn tools untuk membuat peta dengan bentuk grid, spider, atau random. Netgenarte akan menentukan kecepatan kendaraan serta membuat *traffic light* pada map yang dibentuk. Hasil dari negenerate adalah sebuah file berekstensi *.net.xml*.

- *Netconvert*

  Netconvert.exe merupakan *tools* yang berfungsi untuk merubah peta yang didapat dari *Open street map*s yang berekstensi *.osm* menjadi berkestensi *.net.xml*, sehingga sesuai dengan ekstensi yang dapat dibaca oleh SUMO.

- *randomTrips.py*

  Tools ini digunakan untuk membuat *node* awal dan akhir pada peta. Hasil dari *randomTrips.py* berupa file berekstensi **.trips.xml*. Pada tugas akhir ini tools ini digunakan untuk membat *node* pada peta riil kota Tuban.

- *duarouter*

  Duarouter digunakan untuk memberikan arah pada *node* hasil dari *randomTrips.py*. Duarouter secara *default* menggunakan algoritm *Djikstra*. Hasil dari *duarouter* befupa file 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 file berkestensi **.sumocfg*.

- sumo

  sumo digunakan untuk membuat file berekstensi **.xml* gabungan dari dua file sebelumnya yakni file berkestensi **.net.xml* dan **.rou.xml*.

- *traceExporter.py*

TraceExporter.py digunakan untuk menghasilkan file berkestensi **.tcl* dari file *.xml* yang telah dihasilkan oleh SUMO sebelumnya.

2.5 *Open Street Map*

Open Street Map adalah proyek nirlaba yang dilakukan oleh *Open Street Map Foundation* yang berada di *United Kungdom*. *Open Street Map Project* berbasis di *Openstreetmap*.org, adalah upaya pemetaan seluruh dunia yang mencakup lebih dari dua juta relawan di seluruh dunia. Sifat dari *Open Street Map* adalah *open* data sehingga setiap data yang terdapat pada *Open Street Map* dapat digunakan dan diedit secara bebas untuk keperluan apapun termasuk keperluan navigasi [9].

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, *Open street map* digunakan untuk membentuk peta riil kota Tuban.

2.6 AWK

AWK adalah bahasa pemrograman yang digunakan untuk melakukan manipulasi data dan membuat laporan. AWK adalah sebuah perograman filter untuk teks, seperti halnya perintah "*grep*". AWK dapat digunakan untuk mencari bentuk/model dalam sebuah file 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 melakukan perintah dalam melakkan satu pekerjaan maupun sebagai pelengkap (*complicated*) untuk filter standard.

Pada tugas akhir ini AWK digunakan untuk membuat *script* yang digunakan untuk menghitung PDR, *delay*, dan jumlah TC dari *trace file* yang telah didapatkan dari proses menjalankan protokol OLSR dan OLSR+PGB pada NS-2.

2.7 TCL

TCL *(Tool Command Language)* atau sering diucapkan sebagai "*tickle*" adalah bahasa pemrograman yang diciptakan oleh John Ousterhout, pendiri Electric Cloud.Inc (bersama John Graham Cumming) dan seorang professor komputer di Universitas Stanford. Sebelumnya Ousterhout adalah professor komputer di Berkeley California, disanalah dia pertama sekali menciptakan bahasa pemrograman Tcl dan platform Tk sebagai *widget toolkit independen* untuk mengembangkan aplikasi desktop. Pada saat ini, penggunaan Tcl sebagai alat untuk melakukan pengecekan/ujicoba secara otomatis pada hardware ataupun software cukup populer.

2.8 NS2

Network Simulator (NS) pertama kali dibangun sebagai varian dan RIIL *Network Simulator* pada tahun 1989 di University of California Berkeley. Pada tahun 1995 pembangunan *Network Simulator* didukung oleh DARPA *(Defense Advanced research Project Agency)* melalui proyek VINT *(Virtual Internet Testbed)*, yaitu sebuah tim riset gabungan yang beranggotakan tenaga ahli dari LBNL (Lawrence Berkeley of National Laboratory), Xerox PARC, UCB dan USC/ISI (University of Southern California school of Engineering/ information Science Institute). Tim gabungan ini membangun sebuah perangkat lunak simulasi jaringan internet untuk kepentingan riset interaksi-interaksi antar protokol dalam konteks pengembangan protokol internet pada saat ini dan masa yang akan datang.

Network simulator (NS2) merupakan aplikasi *open source* yang dapat mensimulasikan jaringan berbasis TCP/IP dengan berbagai macam media seperti pada jaringan (TCP/UDP/RTP),

traffik beheavior (FTP, *Telnet* CBR, dll.), *Queue management* (RED, FIFO, CBQ), algoritma *routing* unicast (DSDV dan *Link State*), aplikasi multimedia berupa *layer video,quality of Service,audio-video* dan *transcoding*. Selain itu, NS juga mengimplementasikan berberapa MAC *(Medium Access Controler)* seperti IEEE 802.11 WiFi, 802.16 WiMax diberbagai kondisi jaringan misalnya *Point to Point, Point to Molti Point dan celleluler*, bahkan untuk *media Satelite*.

Beberapa keuntungan menggunakan *network simulator* sebagai perangkat lunak simulasi adalah: *network simulator* dilengkapi dengan *tool* validasi, pembuatan simulasi dengan menggunakan *network simulator* jauh lebih mudah daripada menggunakan *software* develover seperti Delphi atau C++. *Network simulator* dapat digunakan pada sistem operasi windows dan sistem operasi linux [10].

2.8.1 Instalasi NS2.35 Normal

Sebelum melakukan instalasi NS2, terlebih dahulu dilakukan instalasi *dependensi* yang dibutuhkan. Salah satu *dependensi* yang dibutuhkan yaitu gcc-4.4. Berikut potongan *command* diterminal untuk melakukan instalasi *dependensi* NS2.

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

Gambar 2.7 Cara *Install Dependensi*

Setelah instalasi semua *dependensi* lengkap silahkan unduh *file* NS2 pada laman.

```
https://sourceforge.net/projects/nsnam/files/latest/download
```

Gambar 2.8 Laman Unduh *file* NS2

Setelah semua selesai terunduh buat direktori sebagai tempat *file* NS2 nanti dan pindahkan *file* hasil unduhan tadi ke dalam direktori tersebut. Disini direktori yang dibuat bernama “Z”.

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

Gambar 2.9 Perintah *Ekstraksi* NS2

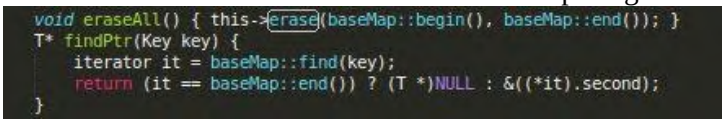
Sebelum dilakukan beberapa modifikasi pada dua berkas yakni berkas "ls.h" pada direktori "linkstate" dan berkas "Makefile.in" pada direktori "otcl-1.14".

Berikut langkah untuk melakukan edit berkas ls.h

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

Gambar 2.10 Perintah Untuk Edit pada *ls.h*

Buat perubahan pada baris 137, ubah kata "*erase*" menjadi *"this -> erase".* Contoh berubahan bias dilihat pada gambar.



```
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);
}
```

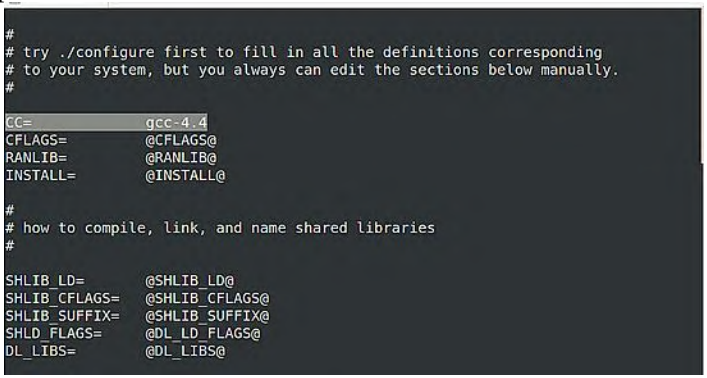
Gambar 2.11 Tampilan ls.h

Setelah itu, ubah file "Makefile.in" pada direktori "otcl-1.14" untuk memberikan informasi pada NS2 mengenai versi gcc yang digunakan. Berikut perintahnya.

```
$sudo gedit ns-allinone-2.34/otcl-1.14/Makefile.in
```

Gambar 2.12 Perintah Untuk Eedit pada *Makefile*.in

Edit teks dari CC= @CC@ menjadi CC=gcc-4.4 dengan cara seperti berikut.



```
#
# try ./configure first to fill in all the definitions corresponding
# to your system, but you always can edit the sections below manually.
#

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@
```

Gambar 2.13 Tampilan dalam berkas Makefile.in

Setelah proses edit kedua berkas selesai, maka dilakukan proses instalasi dengan perintah sebagai berikut

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

Gambar 2.14 Install NS2

Waktu untuk *instalasi* NS2 tergantung dari prosessor yang digunakan. Setelah proses *instalasi* selesai, kemudian lakukan konfigurasi pada *environment path* melalui berkas ".bashrc". Pada berkas tersebut harus ditambahkan pengaturan mengenai *path* yang akan digunakan oleh NS2.

Perintah untuk pengaturan pada berkas *".bashrc"* dapat dilihat pada contoh gambar dibawah pada baris "home/cahya/Z/" bisa diganti sesuai dengan letak direktori yang dimana tempat NS2 berada.

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

Gambar 2.15 Konfigurasi *file .bashrc*

2.8.2 Instalasi NS2.35 Dengan *Patch* OLSR

Pada NS2.35 standar, tidak tersedia protocol OLSR didalamnya, maka untuk dapat melakkan implementasi OLSR pada NS2.35 harus dilakukan proses *patch.* Sebelum melakukan *patching* NS2.35, terlebih dahulu dilakukan instalasi *dependensi* yang dibutuhkan seperti saat kita melakukan instalasi *dependensi* ntuk NS2.35 normal sebelumnya.

Setelah semua *dependensi* berhasil terinstal, yang perlu dilakukan adalah mengunduh file NS2.35 pada laman berikut.

https://drive.google.com/file/d/0B7S255p3kFXNSGJCZ2YzUGJDVk0/view?usp=sharing

Gambar 2.16 Laman Unduh File *NS2.35*

Setelah file berhasil terunduh, *ekstrak* file tersebut sehingga menjadi sebuah folder dengan nama "ns-allinone-2.35" dengan cara seperti berikut.

```
$tar xvf ns-allinone-2.35_gcc482.tar.gz
```

Gambar 2.17 Cara *Ekstraksi* File NS2.35

Lalu masuk kedalam direktori folder NS2.35 dengan cara menuliskan perintah seperti berikut.

```
$cd ns-allinone-2.35/
```

Gambar 2.18 Masuk ke *Folder* NS2.35

Langkah selanjutnya adalah mengunduh file *patch* yang akan digunakan untk memasukkan protokol OLSR kedalam NS2.35, file *patch* tersebut dapat diunduh pada laman berikut.

https://drive.google.com/file/d/0B7S255p3kFXNeVZhWFVVZlJnUEU/view?usp=sharing

Gambar 2.19 Laman Unduh File *Patch* OLSR

Setelah file berhasil terunduh, masukkan file tersebut kedalam folder "ns-allinone-2.35" dan lakukan proses *patching* seperti berikut.

```
$ patch -p0 < umolsr-ns235_v1.0-2014.patch
```

Gambar 2.20 Proses *Patching* OLSR ke dalam NS2

Setelah proses *patching* selesai, lakukan proses instalasi dengan cara seperti berikut.

```
$ ./install
```

Gambar 2.21 Proses Instalasi

Tunggu beberapa saat sampai proses instalasi selesai, setelah proses instalasi selesai dan berhasil dilakukan, masuk ke folder "ns-2.35" dengan cara seperti berikut.

```
$cd ns-2.35/
```

Gambar 2.22 Masuk ke Folder NS2.35

Proses selanjtnya adalah membuat file untuk eksekusi dalam menjalankan protokol kita sesuai dengan *patching* yang telah dilakukan. Lakukan cara seperti berikut.

```
$cp ns ns-olsr
$sudo cp ns-olsr /usr/local/bin/
```

Gambar 2.23 Membuat File untuk Eksekusi

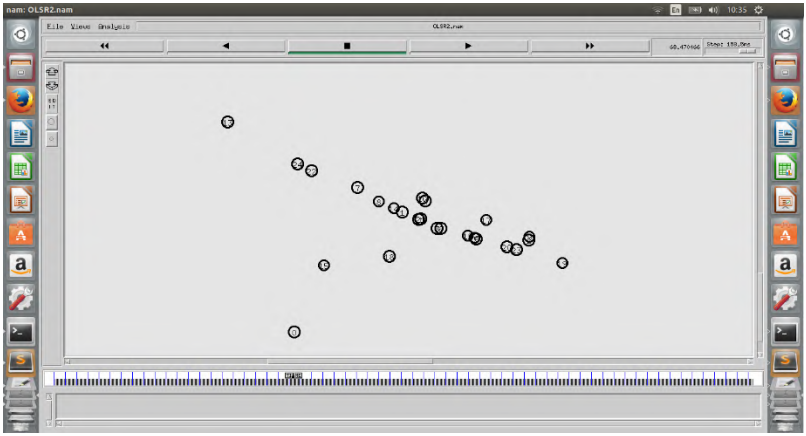
Tujuan membat file "ns-olsr" adalah untuk digunakan dalam menjalankan protokol yang akan kita gunakan. Jika pada NS2.35 normal menggunakan perintah "ns namaFile.tcl" untuk menjalankan protokol, pada NS2.35 ini menggunakan perintah "ns-olsr namaFile.tcl" untuk menjalankan protokol. Hal tersebut terjadi karena kita telah melakukan *patching* pada NS2.35 normal.

Untuk memastikan bahwa proses patching pada NS2.35 telah berhasil, dapat dicoba dengan melakukan pengujian protokol OLSR dengan menuliskan perintah seperti berikut.

```
$ns-olsr <namaFile.tcl>
```

Gambar 2.24 Cara Menjalankan NS

Dan pada NS2.35 ini tidak perlu merubah isi dari file *ls.h* ataupun pada file *bashrc* karena hal tersebut sudah ditangani oleh *patch* yang telah kita masukkan sebelumnya.



Gambar 2.25 Contoh Visualisasi Hasil *Running*

BAB III
PERANCANGAN

Perancangan merupakan bagian penting dari pembuatan sistem secara teknis. Sehingga pada bab ini secara khusus akan dijelaskan bagaimana 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 protokol yang digunakan adalah OLSR *(Optimized Link State Routing Protocol)*. Selanjtnya akan dilakukan analisis pengaruh dari implementasi algoritma PGB *(Prefered Group Broadcasting)* untuk protokol OLSR pada jaringan VANET. Dimana metode *routing* OLSR akan menerapkan algoritma PGB pada proses pemilihan *node* MPR. *Node* MPR yang terpilih merupakan node MPR yang lebih bagus/terpercaya dalam meneruskan paket data, hal tersebut dilihat dari *signal strength* dan juga waktu *delay* yang dimiliki node tersebut. Node MPR yang sudah dipilih dan dianggap terbaik hasil dari penerapan algoritma PGB inilah yang dinamakan sebagai *Reliable Multipoint Relay*.

Dimana metode *routing* OLSR+PGB akan menerapkan skenario pengujian dengan menggunakan peta riil kota Tuban. Perancangan dengan menggunakan peta riil Tuban menggunakan file *.osm* yang bisa didapatkan dari *Open street map*, lalu file *.osm* yang telah didapat akan diolah dengan menggunakan alat bantu SUMO.

Simulasi pada NS2 dilakukan cukup dengan melakukan beberapa modifikasi pada modul *Wireless-phy.cc, Wireless-phy.h, packet.h, skenario.cc, skenario.h dan olsr.cc.* Setiap hasil dari simulasi akan dihitung PDR, d*elivery delay*, dan jumlah TC. Dari hasil tersebut dianalisis pengaruh dari algoritma PGB pada protocol OLSR.

3.2 Perancangan Skenario

Perencangan skenario simulasi pada VANET akan dilakukan dengan menggunakan peta riil. Peta riil yang akan digunakan

merupakan peta dari kota Tuban yang didapat dengan cara *mengeksport* file *.osm* dari *Open Street Map*.

3.2.1 Pembuatan Peta Riil Kota Tuban

Pembuatan peta riil kota Tuban menggunakan peta yang diambil dari *Open street map*. Kemudian *diimpor* kedalam file berformat **.osm*. Setelah itu dilakukan proses pengkonversian dengan menggunakan *tools netconvert* hinga menghasilkan file dengan ekstensi **.net.xml*. Setelah terbentuk file **.net.xml*, dilakukan proses *mengimpor* bentuk geometris (poligon atau tempat tujuan) dengan nama *typemap.xml*, dari proses ini akan tercipta file dengan ekstensi **.poly.xml*.

Langkah selanjutnya adalah membuat file *trip.trips.xml* dengan menggunakan *tools randomTrips.py*. Pada *Tools randomTrips.py* dapat ditambahkan atribut *–r* untuk untuk menetapkan jumlah *node* dan juga digunakan untuk mengatur pergerakan random dari *node* yang telah ditentukan jumlahnya jumlah *node* yang akan menghasilkan file dengan ekstensi **trip.xml* dan **.rou.xml*. pembuatan file dengan ekstensi **rou.xml* juga dapat dilakukan dengan menggunakan *tools douaroter*. Tahap selanjutnya adalah membuat file dengan ekstensi **.sumo.cfg*.

Lakukan konfigurasi pada file **sumo.cfg* dengan memanggil file **net.xml*, **rou.xml*, dan **poly.xml*, selain itu pada file **sumo.cfg* dapat dilakukan pengaturan waktu mulai dan waktu berhentinya simulasi. Setelah terbentuk file **sumo.cfg*, kita dapat mendapatkan file dengan ekstensi **xml*. Dengan menggunakan *tools traceExporter.py* kita dapat merubah file **xml* yang sudah dibuat menjadi file dengan ekstensi **tcl*, file inilah yang nantinya akan menjadi skenario jalannya *node* yang dapat dipanggil pada file **tcl* yang akan kita jalankan pada protokol OLSR di NS2.

3.3 Perancangan Simulasi pada NS2

Simulasi VANET pada NS-2 dilakukan dengan menggunakan skenario mobilitas dan digabungkan dengan skrip TCL yang sudah berisikan konfigurasi mengenai lingkungan simulasi. Perencangan skenario uji coba dapat dilihat pada tabel berikut:

Tabel 3.1 Skenario Uji Coba

No.	Parameter	Spesifikasi
1	Simulator	NS2.35
2	*Routing* protocol	OLSR OLSR +PGB
3	Waktu simulasi	300
4	Jumlah *node*	25, 50, dan 75 kendaraan
5	Luas area	Real : 10000 m x 10000 m
6	Radius Transmisi	290 m
7	Kecepatan Maksimal	10, m/s, 15 m/s dan 20 m/s
8	*Node* asal dan tujuan	Dinamis
9	Tipe data	Constant Bit Rate (CBR)
10	Kecepatan generasi paket data	1 detik
11	Ukuran paket data	512 byte
12	Protocol MAC	IEEE 802.11p
13	Mode propagasi	Two-ray ground reflection model
14	Tipe mobilitas	NS-2
15	Model mobilitas	Peta riil kota Tuban
16	Tipe Kanal	Channel/WirelessChannel

3.4 Perancangan Protokol AODV+PGB

Algoritma *Preferred Group Broadcasting* (PGB) mengubah prinsip pada proses *forwarding* dan pemilihan MPR pada protokol OLSR dimana hanya *node-node* tertentu yang akan terpilih sebagai *Reliable Multipoint Relay*. PGB membagi area menjadi tiga bagian yaitu:

- 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 [11].

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.

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* OLSR. 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.*

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.

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

3.5.2 End-to-End Delay

End-to-End Delay merupakan sebutan untuk waktu rata-rata yang dibutuhkan sebuah paket untuk sampai pada *node* tujuan. Hal tersebut juga dipengaruhi oleh keterlambatan yang disebabkan oleh proses penemuan rute/jalan untuk mengirimkan paket data dari *node* awal ke *node* tujuan. Lamanya waktu yang dibutuhkan sebuah paket untuk dapat terkirim dari *node* awal ke *node* tujuan.

PDR = $\sum$ ($arrive\ time - \ send\ time$) / $\sum Number\ of\ connections$

3.5.3 Topology Control (TC)

Topology Control merupakan salah satu jenis kontrol pada protocol OLSR yang dapat dikirim ke seluruh jaringan secara *broadcasting*. Fungsi dari pesan TC ini adalah untuk menyebar informasi tentang *node* tetangga mana saja yang telah digunakan sebagain *node* MPR. *Node* TC hanya disebar satu periodik dan hanya *node* MPR yang dapat meneruskan pesan TC tersebut.

Jumlah dari pesan TC dapat dihitung melalui *trace file* yang didapatkan setelah pengujian protokol dijalankan. Pesan TC yang dihitung hanya pesan TC yang berasal dari *node* yang melakukan aktifitas *sending* paket data.

[Halaman ini sengaja dikosongkan]

BAB IV
IMPLEMENTASI

Pada bab ini akan dibahas mengenai proses pengujian dan implementasi yang dilakukan berdasarkan rancangan yang telah dijabarkan pada bab sebelumnya. Sebelum menjelaskan implementasi akan ditunjukkan terlebih dahulu lingkungan dilakukannya 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
- Network Simulator 2.35 menggunakan Patch UM-OLSR
- SUMO.0.26.0, dan
- Network Animator 1.14

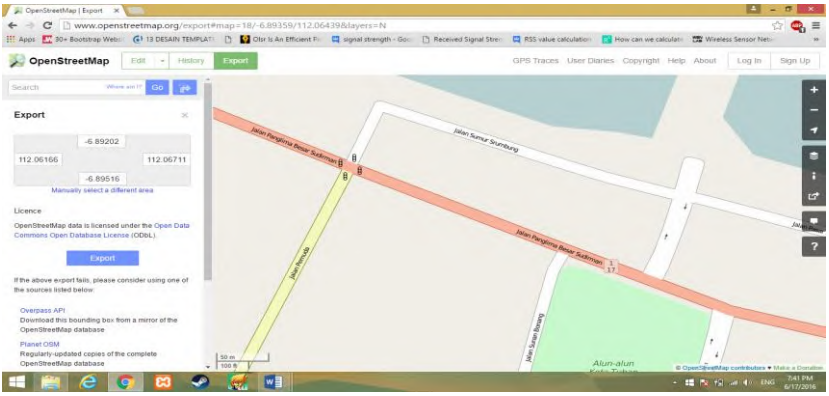
4.1.2 Lingkungan Perangkat Keras

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

- P*rocessor* Intel(R) Core(TM) i5-2410M CPU @ 2.30GHz (4 CPUs), ~2.3GHz,
- RAM sebesar 4 GB DDR3, dan
- Media penyimpanan sebesar 500 GB.

4.2 Implementasi Skenario

Skenario riil pada Tugas Akhir ini menggunakan peta Kota Tuban, lebih tepatnya daerah Alun-alun kota Tuban. Peta diambil dari *Open street map* dengan cara menandai wilayah yang ingin diambil, kemudian melakukan *eksport*. Secara otomatis *Open Street Map* akan memberikan *file* dengan *ekstensi .osm* untuk diunduh.



Gambar 4.1 Unduh *Map* pada *open street map*

File yang telah terunduh dikonversi agar berbentuk *.net.xml*. Konversi dilakukan dengan bantuan *tools netconvert* dari SUMO.

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

Gambar 4.2 Perintah *Konversi* file *.osm* dari *Open street map* menjadi format SUMO

Langkah selanjutnya adalah memasukkan bentuk geometris pada peta dengan cara memasukkan file *typemap.xml*. Bentuk geometris dapat digunakan untuk mengetahi keadaan riil lokasi tersebut (sungai, hutan, gunung). Untuk mendapatkan file **poly.xml* lakukan perintah seperti berikut.

```
$polyconvert --net-file alun2.net.xml --osm-files alun.osm --type-file typemap.xml -o alun2.poly.xml
```

Gambar 4.3 Memasukkan Bentuk Geometris pada *MAP* dan Membentuk File **.poly.xml*

Setelah itu gunakan *tools randomTrips.py* untuk mengatur banyaknya *node* dan pergerakannya dalam simulasi.

```
$python /usr/local/src/sumo-0.26.0/tools/randomTrips.py -n alun2.net.xml -e 25 -l
$python /usr/local/src/sumo-0.26.0/tools/randomTrips.py -n alun2.net.xml -r alun.rou.xml -e 25 -l
```

Gambar 4.4 Proses Mengatur Jumlah *Node* dan Pergerakan *Node*

Proses selanjutnya adalah membuat file **.sumo.cfg* dan lakukan konfigurasi pada file tersebut. Berikut potongan file **.sumo.cfg* yang dibuat.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://sumo.sf.net/xsd/sumoConfiguration.
xsd">
   <input>
      <net-file value="alun.net.xml"/>
      <route-files value="alun.rou.xml"/>
            <additional-files value="alun.poly.xml"/>
   </input>
   <time>
      <begin value="0"/>
      <end value="300"/>
            <step-length value="0.1"/>
   </time>
</configuration>
```

Gambar 4.5 *Konfigurasi* File **.sumo.cfg*

Langkah selanjutnya adalah membuat file dengan ekstensi **.xml* dengan cara seperti berikut.

```
$sumo -c alun.sumo.cfg --fcd-output alun.xml
```

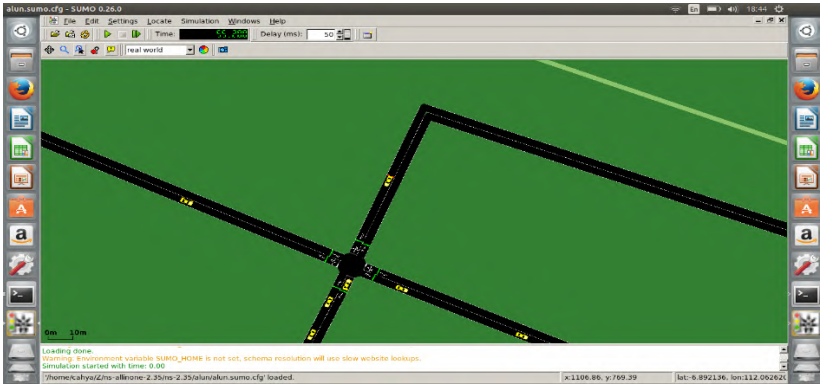
Gambar 4.6 Membuat File **.xml*

Langkah selanjutnya adalah menggunakan *tools traceExporter.py* untk merubah file **.xml* yang telah dibuat menjadi file dengan ekstensi **.tcl*, file ini yang nantinya akan dijadikan sebagai sumber skenario yang akan dijalankan saat menjalankan simulasi pada protokol yang akan diuji.

```
$/home/asus/sumo-0.26.0/tools/traceExporter.py --fcd-input
manhattan1.xml --ns2mobility-output manhattan1.tcl
```

Gambar 4.7 Merubah File *.xml* Menjadi *.tcl*

Setelah file **.tcl* yang digunakan untuk skenario berhasil dibuat, makan file tersebut siap untuk dipanggil pada file **.tcl* yang akan dijalankan pada protokol yang akan diuji.



Gambar 4.8 Contoh Tampilan Map pada SUMO

Dengan menggunakan file **.tcl* yang didalamnya sudah ada konfigurasi skenario seperti diatas, maka seperti inilah tampilan hasil running file **.tcl* pada NAM.

Gambar 4.9 Contoh Tampilan Visalisasi Running Protokol OLSR+PGB pada NAM

4.3 Implementasi Algoritma PGB + OLSR

Untuk melakukan modifikasi pada *routing protocol* OLSR agar dapat menerapkan algoritma PGB maka harus dilakukan proses pengeditan terlebih dahulu pada beberapa file di NS-2. Beberapa file yang perlu diedit yaitu *packet.h*, *olsr.cc*, *wireless-phy.cc*, *wireless-phy.h* dan *packet.cc*.

Penjelasan singkat menegenai pembagian *range* dan mekanisme implementasi algotitma PGB pada OLSR di NS2 adalah sebagai berikut:

1. Mendapatkan/menentukan kekuatan signal/*signal strength* dari sebuah node.
2. Mengeset nilai *delay* toleran yang nantinya akan ditambahkan dengan waktu *delay* yang didapatkan dari setiap node.
3. Membagi wakt delay kedalam 3 bagian, minimal, optimal maksimal.
4. Menentukan area pengelompokan node, dengan membagi *signal strength* kedalam beberapa area (IN, PG dan OUT) dan melihat waktu *delay* yang didapatkan dari sebuah node.
5. *Node* dengan *signal strength* pada area PGB dan waktu *delay* optimal sajalah yang selanjutnya dipilih sebagai *Reliable Multipoint Relay*.

Berikut potongan kode program yang telah dirubah ataupun yang telah ditambahkan untuk implementasi uji coba protokol OLSR+PGB.

4.3.1 Modifikasi packet.h

Packet.h merupakan *class* yang akan dipanggil semua kelas karena berisi semua informasi paket data yang akan dikirim. Sehingga *packet.h* digunakan pada semua protokol. Pada modifikasi kali ini, pada *packet.h* akan ditambahkan variabel *dly* untuk menampung nilai *delay* pada paket tersebut. Penambahan dilakukan pada *struct hdr_cmn* dikarenakn *struct* tersebut merupakan *header* bagi paket yang akan dikirim.

```
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 *packet.h*

4.3.2 Modifikasi wireless-phy.cc

Dalam tugas akhir ini, pada file *wireless-phy* akan ditambahkan *class* baru untuk melakukan kalkulasi *delay* yang akan diberikan dalam suatu *header paket, class* tersebut diberi nama *call().*

Inti dari *class call()* adalah menentukan kekuatan/*range signal* sebuah *node*, selain itu juga dibuat batas atas dan batas bawah dari area luar "OT" dan area dalam "IT" yang nantinya akan digunakan sebagai acuan dalam menentukan *range* PGB. Selain itu juga dilakukan proses mengeset nilai *delay* toleran, yaitu nilai *random* yang ditambahkan dengan waktu sebuah *node* dalam menerima sebuah paket, yang digunakan untuk menentukan bahwa sebuah *node* berada pada *range* PGB. Jika sebuah *node* mempunyai nilai *delay* toleran yang telah ditentukan, maka node tersebut berada pada area PGB. Dari pembuatan *class call()* ini, akan didapatkan posisi dari sebuah node, apakah berada pada area IN, OUT ata PGB [12].

Selain itu pada fungsi *sendUp(p)* juga akan dilakukan pengeditan untuk melakukan pengecekan *signal strength.* Pada NS2, *signal strength* disimpan pada variabel *Pr*. Pengecekan disini berhubungan dengan *class call(),* karena saat melakukan pengecekan dibutuhkan pemanggilan nilai *delay* dari *class call().*

```
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>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;
        }
}
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);   //tambah
        ch->dly= cal(Pr,RXThresh_,Pt_);    //tambah
```

Gambar 4.12 pemanggilan *delay*

4.3.3 Modifikasi wireless-phy.h

Wireless-phy.h merupakan *include* dari *wireless-phy.cc*, sehingga jika kita membuat *class* baru pada *wireless-phy.cc*, maka *class* tersebut juga harus dideklarasikan pada *wireless-phy.h* agar dapat dipanggil. Disini akan ditambahkan *class call* yang sebelumnya telah dibuat pada *wireless.cc.*

```
double cal(double  pr,double rxTh,double  txP);//tambah
```

Gambar 4.13 Deklarasi *class call*

4.3.4 Modifikasi scheduler.cc

Scheduler.cc merupakan sebuah file yang berfungsi sebagai tempat dimana semua kegiatan atau *event* sebuah pengiriman disimpan. Pada modifikasi kali ini, akan dilakukan sebuah perubahan pada *class Scheduler::schedulex()* yang berfungsi untuk mengirim paket data hanya ke *node* MPR yang berada pada area PGB, jika tidak ditemukan *node* MPR pada area PGB maka pencarian akan dilakukan pada area IN, jika tidak ditemukan juga *node* MPR pada area IN, pencarian akan dilakkan pada area OUT. Selain itu pada *scheduler.cc* juga dibuat *class* baru dengan nama *Scheduler::scheduleDel()* yang berfungsi menangani jika ada sebuah *node* yang mendapatkan paket *broadcast* atau data yang sama dari beberapa *node* tetangga.

```
int
Scheduler::schedulex(Handler* h, Event* e, double delay)
{
        int uuuu=0;
        // handler should ALWAYS be set... if it's not, it's a bug in the caller
        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) {
                // You probably don't want to do this
                // (it probably represents a bug in your simulation).
                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 4.14 *class schedulex*

```
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 4.15 *class schedulDel*

4.3.5 Modifikasi scheduler.h

Sama seperti file *wireless-phy.h*, file *scheduler.h* juga merupakan *include* dari file *schedule.cc*, sehingga pada file *schedule.h* harus dideklarasikan juga nama *class* yang sudah ditambahkan pada file *schedule.cc*. disini akan ditambahkan *class* dengan nama *schedulex()* dan *scheduleDel()* agar dapat dipanggil pada file *scheduler.cc.*

```
int schedulex(Handler*, Event*, double delay); //tambah
bool scheduleDel(scheduler_uid_t uid);          //tambah
```

Gambar 4.16 Deklarasi *class schedulex* dan class *scheduleDel*

4.3.6 Modifikasi olsr.cc

Olsr.cc merupakan file utama pada protokol OLSR yang ada di NS2.35 ini, file inilah yang akan memanggil semua file yang berhubungan dengan protokol OLSR. Pada modifikasi kali ini, dilakukan perubahan pada *OLSR::recv_olsr(Packet* p)*. Tujuan perubahan tersebut adalah untuk mekanisme *forwarding* paket data. Paket data hanya akan dikirimkan oleh *node* yang berada pada area PGB dengan mengimplementasikan *signal strength* dan *delay* yang didapat dari *class call* sebelumnya. Jika *delay* node tetangga terlalu besar/terlalu kecil dari syarat yang sudah ditentukan sebelumnya, maka bisa dipastikan bahwa node tetangga tersebut bukan berada pada area PGB.

```
struct hdr_cmn *ch = HDR_CMN(p);                    //tambah
for (addr_list_t::iterator it = duplicated->iface_list().begin();
        it != duplicated->iface_list().end();
        it++) {
                if (*it == ra_addr() && ch->dly > 0.01f) { //tambah
                    do_forwarding = false;
                    break;
}
```

Gambar 4.17 Perubahan *class OLSR::recv_olsr(Packet* p)*

[Halaman ini sengaja dikosongkan]

BAB V
UJI COBA DAN EVALUASI

Pada bab ini akan dijelaskan uji coba yang dilakukan pada *routing protocol* yang telah dikerjakan serta analisa dari uji coba yang telah dilakukan. Pembahasan pengujian meliputi lingkungan uji coba dan hasil uji coba pada peta riil kota Tuban.

5.1 Lingkungan Uji Coba

Uji coba dilakukan pada laptop yang memiliki spesifikasi perangkat keras dan perangkat lunak sebagai berikut:

1. Perangkat keras
   a. Processor Intel(R) Core(TM) i5-2410M CPU @ 2.30GHz (4 CPUs), ~2.3GHz
   b. RAM sebesar 4 GB DDR3
   c. Media Penyimpanan sebesar 500 GB
   d. Tipe sistem 64-bit
2. Perangkat lunak
   a. Sistem Operasi berupa Linux 14.04
   b. Network Simulator 2.35
   c. GNU AWK versi 4.0.1 untuk analisa trace file

5.2 Skenario Uji Coba

Akan dilakukan 2 kali jenis pengujian, pengujian yang pertama adalah skenario pengujian dijalankan sebanyak 3 kali. Pada setiap skenario pengujian dilakukan 3 kali perubahan jumlah *node*, yaitu sebanyak 25, 50 dan 75 *node*. Sedangkan pengujian yang kedua adalah dengan menggunakan jumlah node yang tetap yaitu 50 node, namun akan ada perubahan pada kecepatan pergerakan node yaitu 10 m/s, 15 m/s dan 20 m/s dan pada setiap kecepatan akan dilakukan 5 kali uji coba dengan menggunakan pergerakan yang acak yang nantinya akan didapatkan hasil rata-rata dari pengujian tersebut.

Tabel 5.1 Skenario Uji Coba

No.	Parameter	Spesifikasi
1	Simulator	NS2.35
2	*Routing* protocol	OLSR OLSR +PGB

No.	Parameter	Spesifikasi
3	Waktu simulasi	300
4	Jumlah *node*	25, 50, dan 75 kendaraan
5	Luas area	Real : 10000 m x 10000 m
6	Radius Transmisi	290 m
7	Kecepatan Maksimal	10, m/s, 15 m/s dan 20 m/s
8	*Node* asal dan tujuan	Dinamis
9	Tipe data	Constant Bit Rate (CBR)
10	Kecepatan generasi paket data	1 detik
11	Ukuran paket data	512 byte
12	Protocol MAC	IEEE 802.11p
13	Mode propagasi	Two-ray ground reflection model
14	Tipe mobilitas	NS-2
15	Model mobilitas	Peta riil kota Tuban
16	Tipe Kanal	Channel/WirelessChannel

5.3 Hasil Uji Coba

Berikut merupakan hasil dari uji coba dan perbandingan performa antara protokol OLSR dengan OLSR+PGB. Parameter yang digunakan untuk pengujian adalah sebagai berikut:

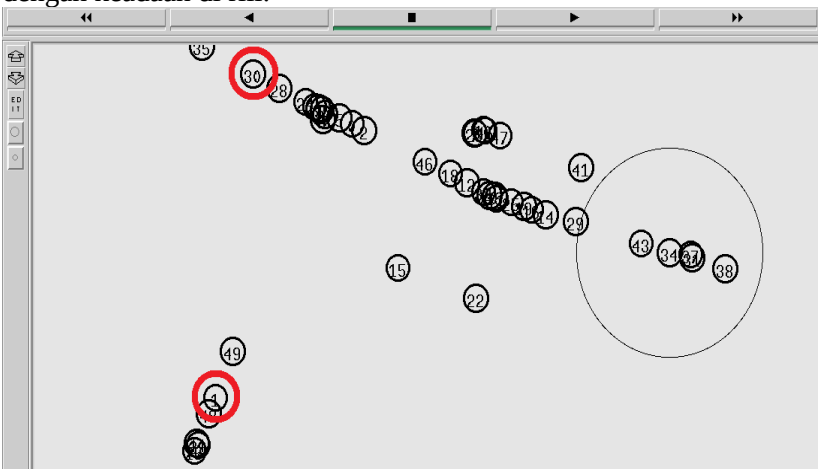
- Start : *Node* awal pengirim paket data.
- End : N*ode* tujuan akhir penerima paket data.
- Send : Banyaknya jumlah *packet send* yang didapat dari *trace file* (*output* pengujian).
- Receive: Banyaknya jumlah *packet recv* yang didapat dari *trace file* (*output* pengujian).
- PDR : Nilai *packet delay ratio* yang didapat berdasarkan nilai *receive* dan *send* yang telah didapat.
- Forward : Jumlah *node* yang meneruskan paket data sampai ke *node* tujuan.
- E2E-Delay: Waktu rata-rata pngiriman data.
- TC : Jumlah paket TC *(Topologi Control)* yang didapat dalam satu kali skenario pengujian.

5.3.1 Skenario Pertama

Skenario dengan menggunakan perubahan jumlah node pada area uji coba dan kecepatan yang sama. Posisi *node* pengirim dan *node* penerima dipilih secara *random* agar sesuai dengan keadaan di lapangan.

5.3.1.1 Skenario dengan 50 *Node*

Skenario dengan menggunakan 50 *node* dan posisi *node* pengirim dan *node* penerima dipilih secara *random* agar sesuai dengan keadaan di riil.



Gambar 5.1 *Node* sumber dan *node* tujan pada salah satu percobaan skenario pertama

Hasil uji skenario pertama dengan menggunakan protokol OLSR+PGB:

Tabel 5.2 Hasil Performa OLSR+PGB dengan 50 *Node*

Start	End	Send	Receive	PDR (%)	Forward	E2E Delay(detik)	TC
1	49	198	187	94.44	9	**5.99422**	4984
1	10	195	78	40.00	129	**10.8039**	4937
1	20	192	94	48.96	133	**8.33779**	5219
1	30	195	84	**43.08**	17	**6.87615**	5085

Hasil rata-rata:

Rata-rata	PDR (%)	Forward	E2E Delay(detik)	TC
	56.62	72	8.003015	5056.25

Hasil uji skenario pertama dengan menggunakan protokol OLSR:

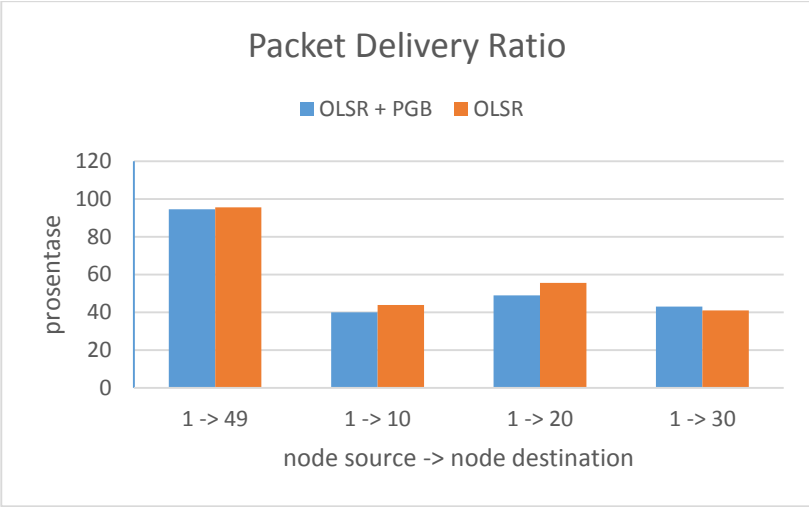
Tabel 5.3 Hasil Performa OLSR dengan 50 *Node*

Start	End	Send	Receive	PDR (%)	Forward	E2E Delay(detik)	TC
1	49	203	194	**95.57**	15	6.18661	**4052**
1	10	196	86	**43.88**	142	11.6057	**4348**
1	20	200	111	**55.50**	105	8.76511	**4079**
1	30	195	80	41.03	19	6.95489	**4246**

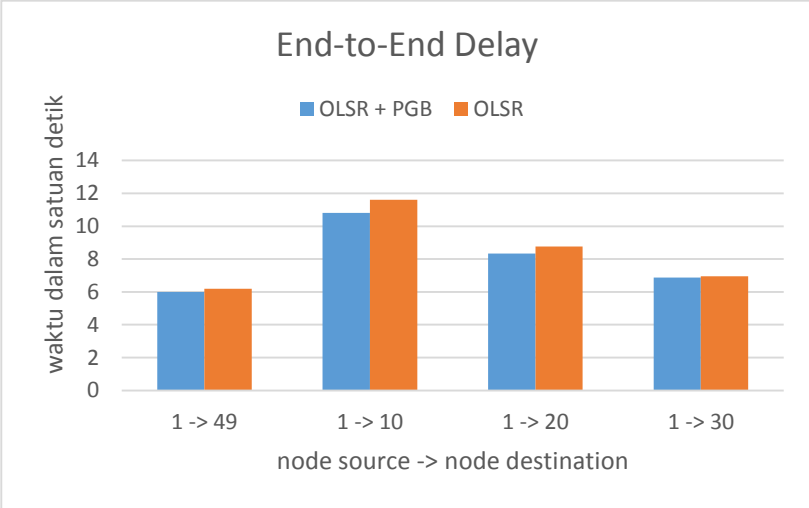
Hasil rata-rata:

Rata-rata	PDR (%)	Forward	E2E Delay(detik)	TC
	58.995	70.25	8.3780775	4181.25

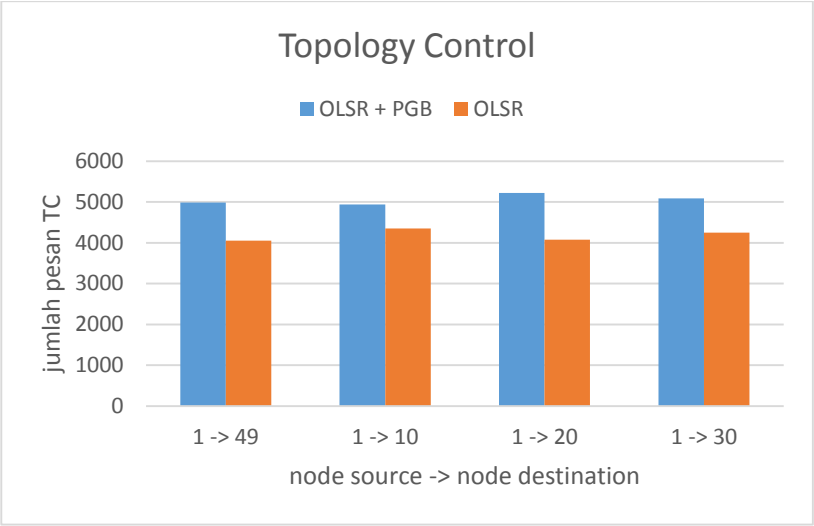
Dari data diatas dapat terlihat bahwa pada saat skenario menggunakan node/kendaraan sebanyak 50 node dengan protokol OLSR memiliki nilai *Packet Delivery Ratio* yang lebih tinggi sebesar 4.017% dibandingkan dengan menggunakan protokol OLSR+PGB. Untuk *End-to-End Delay*, OLSR+PGB memiliki nilai yang lebih optimal sebesar 4.47% dibandingkan dengan menggunakan protokol OLSR. Sedangkan untuk jumlah pesan *Topology Control* OLSR memiliki jumlah pesan *Topology Control* yang lebih kecil sebesar 17.30% dibandingkan dengan menggunakan protokol OLSR+PGB. Berikut grafik yang merepresentasikan data yang telah didapatkan diatas.



Gambar 5.2 Perbandingan Performa PDR dari OLSR+PGB dengan OLSR pada Skenario Pertama



Gambar 5.3 Perbandingan Performa *End-to-End Delay* dari OLSR+PGB dengan OLSR pada Skenario Pertama



Gambar 5.4 Perbandingan Performa TC dari OLSR+PGB dengan OLSR pada Skenario Pertama

5.3.1.2 Skenario dengan 25 *Node*

Skenario dengan menggunakan 25 *node* dan posisi *node* pengirim dan *node* penerima dipilih secara *random* agar sesuai dengan keadaan di riil.

Gambar 5.5 *Node* sumber dan *node* tujan pada salah satu percobaan skenario pertama

Hasil uji skenario pertama dengan menggunakan protokol OLSR+PGB:

Tabel 5.4 Hasil Performa OLSR+PGB dengan 25 *Node*

Start	End	Send	Receive	PDR (%)	Forward	E2E Delay(detik)	TC
1	5	197	10	05.08	38	**3.6278**	1120
1	10	195	41	**21.03**	31	8.67584	1151
1	15	193	27	13.99	13	5.62821	1097
1	24	194	28	14.43	17	**5.44363**	1042

Hasil rata-rata:

Rata-rata	PDR (%)	Forward	E2E Delay(detik)	TC
	13.6325	24.75	5.84387	1102.5

Hasil uji skenario pertama dengan menggunakan protokol OLSR:

Tabel 5.5 Hasil Performa OLSR dengan 25 *Node*

Start	End	Send	Receive	PDR (%)	Forward	E2E Delay(detik)	TC
1	5	196	14	**07.14**	12	9.32308	**976**
1	10	198	36	18.18	28	**7.32184**	**975**
1	15	194	28	**14.43**	0	**3.98579**	**971**
1	24	193	30	**15.54**	17	7.04449	**964**

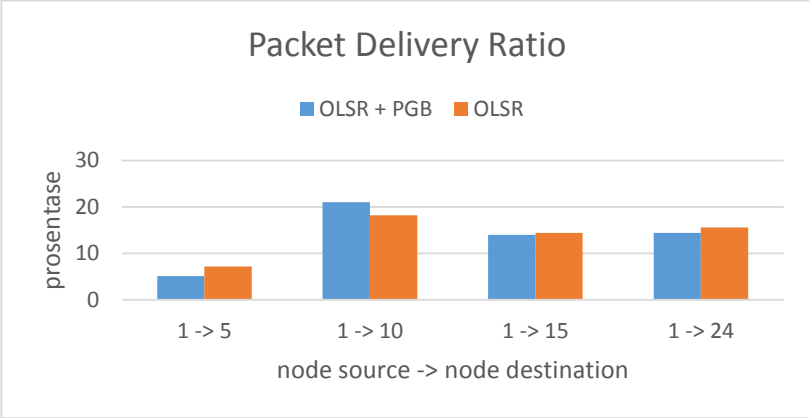
Hasil rata-rata:

Rata-rata	PDR (%)	Forward	E2E Delay(detik)	TC
	13.8225	14.25	6.9188	971.5

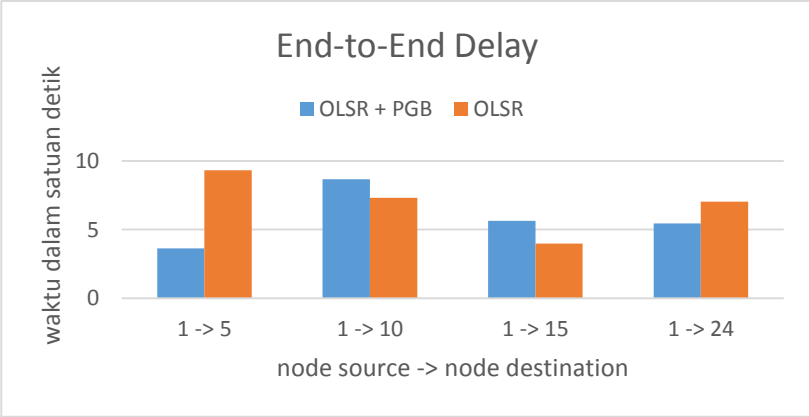
Catatan: pada baris yang berwarna hitam terlihat bahwa tidak ada *node* yang meneruskan paket data, yang berarti pengiriman gagal.

Dari data diatas dapat terlihat bahwa pada saat skenario menggunakan node/kendaraan sebanyak 50 node dengan protokol OLSR memiliki nilai *Packet Delivery Ratio* yang lebih tinggi sebesar 13.7% dibandingkan dengan menggunakan protokol OLSR+PGB. Untuk *End-to-End Delay*, OLSR+PGB memiliki nilai yang lebih

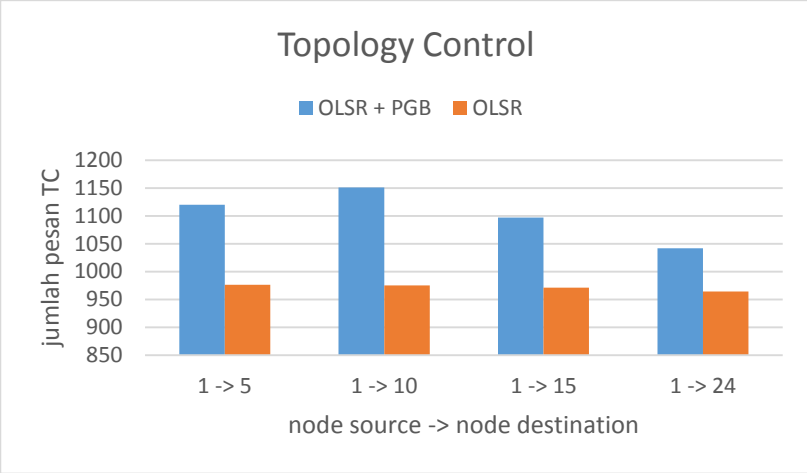
optimal sebesar 15.53% dibandingkan dengan menggunakan protokol OLSR. Sedangkan untuk jumlah pesan *Topology Control* OLSR memiliki jumlah pesan *Topology Control* yang lebih kecil sebesar 13.48% dibandingkan dengan menggunakan protokol OLSR+PGB. Berikut grafik yang merepresentasikan data yang telah didapatkan diatas.



Gambar 5.6 Perbandingan Performa PDR dari OLSR+PGB dengan OLSR pada Skenario Kedua



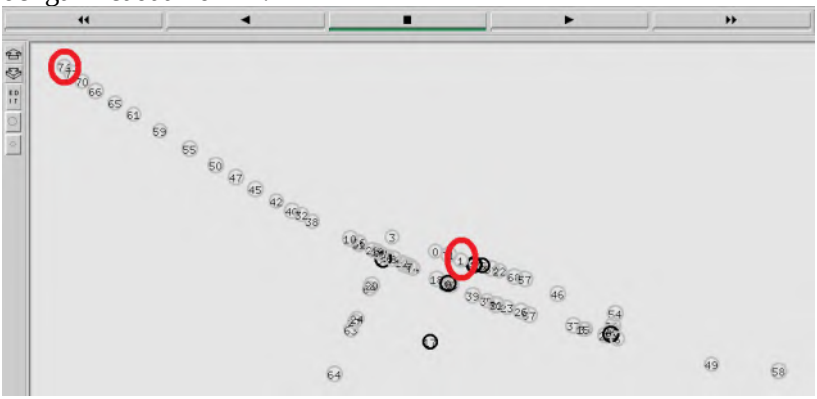
Gambar 5.7 Perbandingan Performa End-to-End Delay dari OLSR+PGB dengan OLSR pada Skenario Kedua



Gambar 5.8 Perbandingan Performa TC dari OLSR+PGB dengan OLSR pada Skenario Kedua .

5.3.1.3 Skenario dengan 75 *Node*

Skenario dengan menggunakan 75 *node* dan posisi *node* pengirim dan *node* penerima dipilih secara *random* agar sesuai dengan keadaan di riil.



Gambar 5.9 *Node* sumber dan *node* tujan pada salah satu percobaan skenario pertama

Hasil uji skenario pertama dengan menggunakan protokol OLSR+PGB:

Tabel 5.6 Hasil Performa OLSR+PGB dengan 75 *Node*

Start	End	Send	Receive	PDR (%)	Forward	E2E Delay(detik)	TC
1	25	198	43	21.72	64	**9.34129**	16262
1	45	192	134	69.79	88	**11.241**	15433
65	50	199	155	**77.89**	113	6.84132	16163
1	74	197	86	43.65	120	57.6517	15434

Hasil rata-rata:

Rata-rata	PDR (%)	Forward	E2E Delay(detik)	TC
	53.2625	96.25	21.2688275	15823

Hasil uji skenario pertama dengan menggunakan protokol OLSR:

Tabel 5.7 Hasil Performa OLSR+PGB dengan 75 *Node*

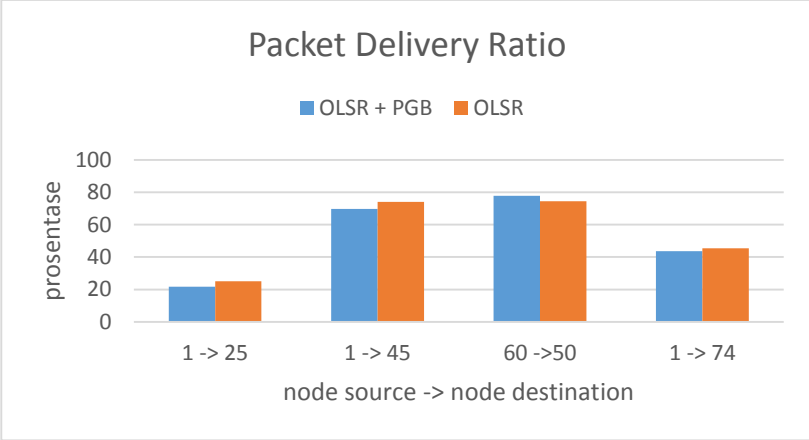
Start	End	Send	Receive	PDR (%)	Forward	E2E Delay(detik)	TC
1	25	187	47	**25.13**	44	9.6736	**12134**
1	45	186	136	**74.12**	110	11.9752	**11910**
65	50	195	145	74.36	42	**6.3593**	**11852**
1	74	198	90	**45.45**	222	**11.9227**	**11353**

Hasil rata-rata:

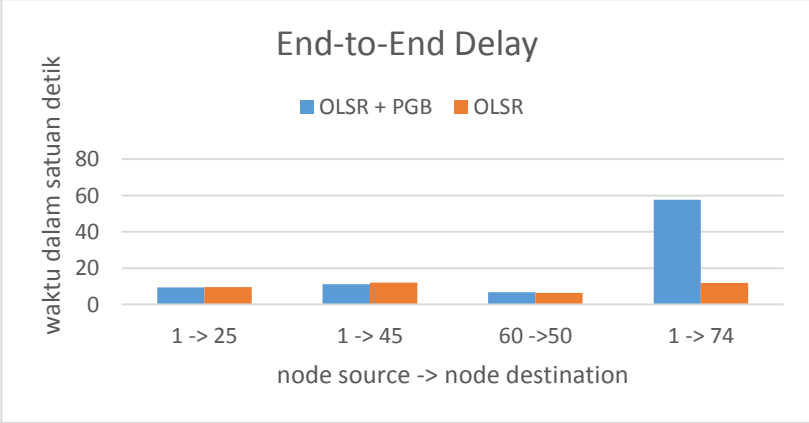
Rata-rata	PDR (%)	Forward	E2E Delay(detik)	TC
	54.765	104.5	9.9827	11812.25

Dari data diatas dapat terlihat bahwa pada saat skenario menggunakan node/kendaraan sebanyak 50 node dengan protokol OLSR memiliki nilai *Packet Delivery Ratio* yang lebih tinggi sebesar 2.74% dibandingkan dengan menggunakan protokol OLSR+PGB. Untuk *End-to-End Delay*, OLSR+PGB memiliki nilai yang lebih optimal sebesar 53.06% dibandingkan dengan menggunakan protokol

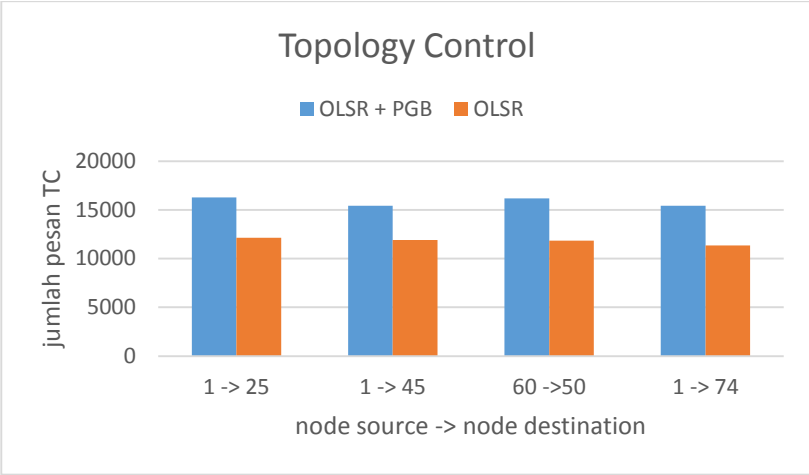
OLSR. Sedangkan untuk jumlah pesan *Topology Control* OLSR memiliki jumlah pesan *Topology Control* yang lebih kecil sebesar 25.34% dibandingkan dengan menggunakan protokol OLSR+PGB. Berikut grafik yang merepresentasikan data yang telah didapatkan diatas.



Gambar 5.10 Perbandingan Performa PDR dari OLSR+PGB dengan OLSR pada Skenario Ketiga



Gambar 5.11 Perbandingan Performa End-to-End Delay dari OLSR+PGB dengan OLSR pada Skenario Ketiga

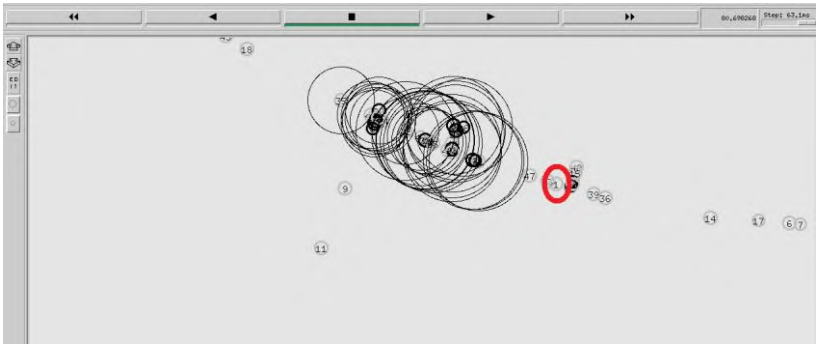


Gambar 5.12 Perbandingan Performa TC dari OLSR+PGB dengan OLSR pada Skenario Ketiga

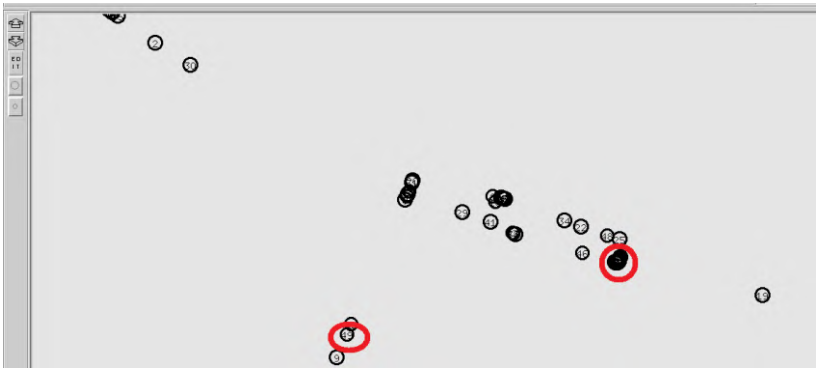
Dari grafik diatas dapat terlihat bahwa pada saat skenario menggunakan 25, 50 dan 75 node, prosentase *Packet Delivery Ratio* pada saat menggunakan protokol OLSR cenderng sedikit lebih tinggi daripada saat menggunakan protokol OLSR+PGB. Sedangkan untuk *End-to-End Delay*, hasil lebih optimal didapatkan saat menggunkan protokol OLSR+PGB (semakin sedikit waktu *delay*, maka semakin optimal), meskipun pada salah satu *case* mendapatkan hasil yang lebih jelek. Namun untuk jumlah *Topology Contro*l jumlah lebih sedikit didapatkan saat menggunakan protokol OLSR standar.

5.3.2 Skenario Kedua

Skenario dengan jumlah node yang sama yaitu 50 node, namun kecepatan node pada setiap pengujian akan berbeda. Pada setiap satu kecepatan akan dilakukan 5 kali pengujian dengan pergerakan node yang acak/ berubah-ubah pada setiap pengujiannya. Posisi *node* pengirim dan *node* penerima dipilih secara *random* agar sesuai dengan keadaan di lapangan.



Gambar 5.13 *Node* sumber



Gambar 5.14 *Node* tujan pada salah satu percobaan skenario pertama

5.3.2.1 Skenario dengan Kecepatan 10 m/s

Hasil uji skenario pada kecepatan 10 m/s dengan menggunakan protokol OLSR+PGB:

Tabel 5.8 Performa OLSR+PGB pada Kecepatan 10 m/s

OLSR+PGB	Send	Recv	PDR(%)	Forward	E2E Delay(detik)	TC
Percobaan 1	205	47	22.93	264	6.72559	5918
Percobaan 2	199	8	04.02	42	20.271	5664
Percobaan 3	194	20	10.31	5	4.59642	5538

Percobaan 4	194	138	71.13	302	14.0781	5781
Percobaan 5	190	80	42.11	66	7.13876	5233
Rata-rata			**30.10**	135.8	**10.561974**	**5626.8**

Hasil uji skenario pada kecepatan 10 m/s dengan menggunakan protokol OLSR:

Tabel 5.9 Performa OLSR pada Kecepatan 10 m/s

OLSR+PGB	**Send**	**Recv**	**PDR(%)**	**Forward**	**E2E Delay(detik)**	**TC**
Percobaan 1	191	47	24.61	125	6.86248	4407
Percobaan 2	197	14	07.11	51	18.8748	5162
Percobaan 3	197	23	11.68	56	5.08412	4698
Percobaan 4	194	142	73.20	230	259.371	4488
Percobaan 5	195	85	43.59	77	7.25756	4072
Rata-rata			**32.038**	107.8	**59.489992**	**4565.4**

5.3.2.2 Skenario dengan Kecepatan 15 m/s

Hasil uji skenario pada kecepatan 15 m/s dengan menggunakan protokol OLSR+PGB:

Tabel 5.10 Performa OLSR+PGB pada Kecepatan 15 m/s

OLSR+PGB	**Send**	**Recv**	**PDR(%)**	**Forward**	**E2E Delay(detik)**	**TC**
Percobaan 1	203	33	16.26	27	5.80584	6187
Percobaan 2	197	42	21.32	246	5.54273	6574
Percobaan 3	192	22	11.46	215	6.5784	5781
Percobaan 4	194	67	34.54	99	7.42201	5736
Percobaan 5	197	12	06.09	62	12.5661	5354
Rata-rata			**17.934**	129.8	**7.583016**	**5926.4**

Hasil uji skenario pada kecepatan 15 m/s dengan menggunakan protokol OLSR:

Tabel 5.11 Performa OLSR pada Kecepatan 15 m/s

OLSR+PGB	**Send**	**Recv**	**PDR(%)**	**Forward**	**E2E Delay(detik)**	**TC**

Percobaan 1	192	33	17.19	57	7.63274	5156
Percobaan 2	197	46	23.35	289	5.30353	5447
Percobaan 3	191	28	14.66	152	6.71689	4679
Percobaan 4	198	67	33.84	194	6.91333	4590
Percobaan 5	202	11	05.45	79	13.9386	4729
Rata-rata			**18.898**	154.2	**8.101018**	**4920.2**

5.3.2.3 Skenario dengan Kecepatan 20 m/s

Hasil uji skenario pada kecepatan 20 m/s dengan menggunakan protokol OLSR+PGB:

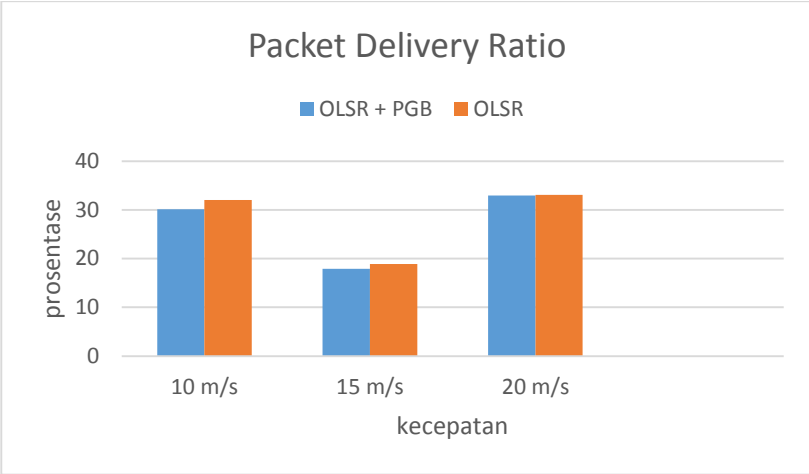
Tabel 5.12 Performa OLSR+PGB pada Kecepatan 20 m/s

OLSR+PGB	**Send**	**Recv**	**PDR(%)**	**Forward**	**E2E Delay(detik)**	**TC**
Percobaan 1	196	101	51.53	78	8.78925	5836
Percobaan 2	197	20	10.15	163	10.8847	6527
Percobaan 3	198	24	12.12	139	9.21357	6063
Percobaan 4	201	180	89.55	32	7.07174	6554
Percobaan 5	202	3	01.49	288	7.8174	6634
Rata-rata			**32.968**	140	**8.755332**	**6322.8**

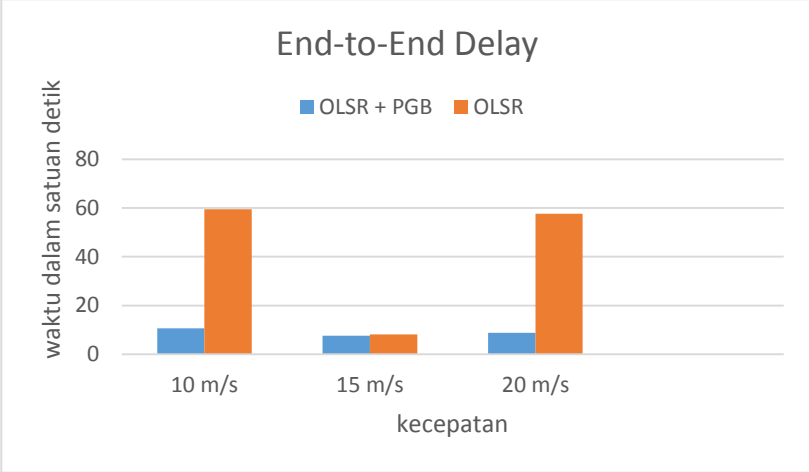
Hasil uji skenario pada kecepatan 20 m/s dengan menggunakan protokol OLSR:

Tabel 5.13 Performa OLSR pada Kecepatan 20 m/s

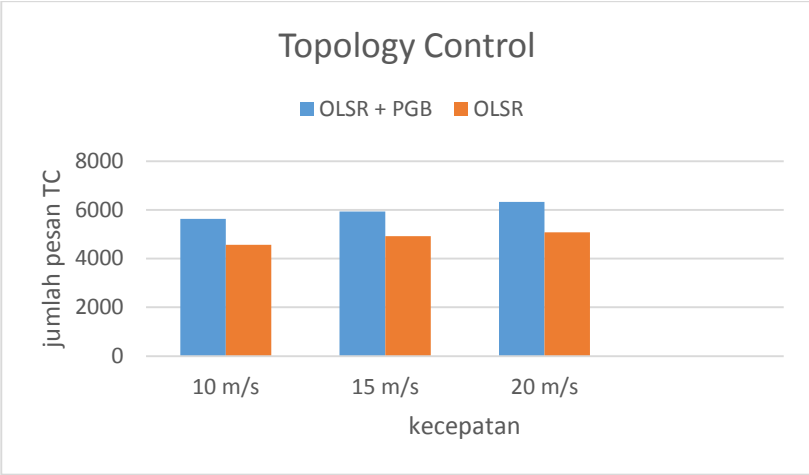
OLSR+PGB	**Send**	**Recv**	**PDR(%)**	**Forward**	**E2E Delay(detik)**	**TC**
Percobaan 1	198	106	53.54	83	7.66233	4949
Percobaan 2	199	19	09.55	223	251.917	5367
Percobaan 3	203	27	13.30	88	11.9898	4622
Percobaan 4	194	169	87.11	23	6.52437	5419
Percobaan 5	195	4	02.05	387	10.2473	5066
Rata-rata			**33.11**	160.8	**57.66816**	**5084.6**



Gambar 5.15 Perbandingan Performa PDR dari OLSR+PGB dengan OLSR



Gambar 5.16 Perbandingan Performa End-to-End Delay dari OLSR+PGB dengan OLSR



Gambar 5.17 Perbandingan Performa TC dari OLSR+PGB dengan OLSR

Dari data dan grafik diatas dapat terlihat bahwa pada saat skenario menggunakan kecepatan 10 m/s, 15 m/s dan 20 m/s dengan protokol OLSR memiliki nilai *Packet Delivery Ratio* yang lebih besar sebanyak 6.04%, 5.10% dan 0.42% dibandingkan dengan menggunakan protokol OLSR+PGB. Untuk *End-to-End Delay*, pada saat skenario menggunakan kecepatan 10 m/s, 15 m/s dan 20 m/s dengan protokol OLSR+PGB memiliki nilai *End-to-End Delay* yang lebih optimal sebesar 82.24%, 6.39% dan 84.81% dibandingkan dengan menggunakan protokol OLSR. Sedangkan untuk jumlah pesan *Topology Control*, pada saat skenario menggunakan kecepatan 10 m/s, 15 m/s dan 20 m/s dengan protokol OLSR+PGB memiliki jumlah pesan *Topology Control* yang lebih banyak sebesar 18.86%, 16.97% dan 19.58% dibandingkan dengan menggunakan protokol OLSR.

[Halaman ini sengaja dikosongkan]

BAB VI
KESIMPULAN DAN SARAN

1.1 Kesimpulan

Dari hasil pengamatan dan percobaan selama perancangan, implementasi, dan uji coba aplikasi, maka dapat diambil kesimpulan sebagai berikut:

1. Nilai PDR *(Packet Delivery Ratio)* dengan menggunakan OLSR+PGB cenderung lebih kecil dibandingkan dengan menggunakan OLSR.
2. Nilai *End-to-End Delay* dengan menggunakan OLSR+PGB lebih kecil daripada menggunakan OLSR.
3. Jumlah pesan TC *(Topology Control)* dengan menggunakan OLSR+PGB lebih besar daripada menggunakan OLSR, Hal tersebut terjadi karena dengan menggunakan OLSR+PGB, *node* MPR yang meneruskan paket dan pesan TC lebih banyak daripada dengan menggunakan OLSR.
4. Penerapan algoritma PGB pada protokol OLSR tidak dapat memberikan hasil yang maksimal. Hal tersebut dikarenakan keunggulan OLSR terletak pada pemilihan *node* MPR-nya, sedangkan dengan menggunakan PGB hanya *node* MPR yang berada di *range* tertentu yang dapat meneruskan pengiriman pesan, sedangkan kita tidak mengetahui kualitas *node* MPR pada *range* tersebut. Dengan kata lain penggunaan algoritma PGB tidak dapat memberi jaminan bahwa *node* MPR yang melanjutkan pengiriman pesan adalah *node* MPR yang benar-benar terbaik kualitasnya.
5. Dengan jumlah pesan TC yang lebih banyak, maka tujuan utama untuk mencegah terjadinya *strom broadcasting* belum dapat tercapai dengan menggunakan algoritma PGB pada protokol OLSR.
6. Meskipun tujuan utama tidak terpenuhi, namun dengan menggunakan OLSR+PGB didapatkan nilai *End-to-End Delay*

yang lebih kecil yang berarti proses pengiriman paket data dapat terkirim lebih capat dan stabil.

7. Semakin cepat proses pengiriman suatu paket data, semakin banyak pula jumlah paket data yang dapat terkirim dalam satu waktu.

1.2 Saran

Saran yang diberikan untuk pengembangan aplikasi ini adalah:

1. Pemilihan *node* MPR pada protokol OLSR mungkin akan lebih efektif jika berdasarkan kepada kualitas *node* tetangga, tidak hanya berdasarkan *range* tertentu atapun *signal strength node* tetangga yang didapat.
2. Menambahkan algoritma pemilihan kalitas *node* MPR pada area PGB.
3. Karena protokol OLSR pada NS2 masih merupakan protokol hasil dari *patching* yang berarti bukan protokol resmi dari NS2, maka masih terdapat beberapa *bug* pada protokol tersebut, sehingga perlu diujicobakan performansi protokol OLSR dalam simulator yang berbeda untuk menambah tingkat validasi performansi protokol yang dikembangkan.

DAFTAR PUSTAKA

[1] O. A. Wahab, H. Otrok and A. Mourad, "VANET QoS-OLSR: QoS-based clustering protocol for Vehicular Ad hoc," *Computer Communications*, vol. 36, no. 13, p. 1422–1435, 2013.

[2] F. Ryandi, "Routing Protocol in VANET (Ad Hoc Network)," [Online]. Available: http://farhanryandi.student.telkomuniversity.ac.id/2015/02/12/routing-protocol-in-vanet-ad-hoc-network/.

[3] "Optimized Link State Routing Protocol," [Online]. Available: https://en.wikipedia.org/wiki/Optimized_Link_State_Routing_Protocol.

[4] "Security and Privacy in Vehicular Ad Hoc Networks (VANETs)," [Online]. Available: https://ece.uwaterloo.ca/~kan.yang/security_bbcr/vanet.html.

[5] A. Tønnesen, T. Lopatic, H. Gredler, B. Petrovitsch, A. Kaplan and S.-O. Tücke, "OLSRD an ad hoc wireless mesh routing daemon," [Online]. Available: http://www.olsr.org/.

[6] T. R. Gross and V. Naumov, "Connectivity-Aware Routing (CAR) in Vehicular Ad Hoc Network," in *IEEE INFOCOM 2007 - 26th IEEE International Conference on Computer Communications*, Anchorage, AK, 2007.

[7] U. Nagaraj and P. Dhamal, "Broadcasting Route Protocols in vanet," *Network and Complex Systems*, vol. 1, p. 19, 2011.

[8] "SUMO – Simulation of Urban MObility," [Online]. Available: http://www.dlr.de/ts/en/desktopdefault.aspx/tabid-9883/16931_read-41000/.

[9] "OpenStreetMap," [Online]. Available: http://openstreetmap.id/about/tentang-openstreetmap/.

[10] T. Issariyakul and E. Hossain, Introduction to Network Simulator NS2, Bangkok: Springer US, 2012.

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

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

[13] H. Ghafoor, I. Koo and N.-u.-D. Gohar, "Neighboring and Connectivity-Aware Routing in VANETs," *the Scientific World Journal*, vol. 2014, p. 10, 2014.

[14] 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.

[15] P. Jacquet, P. Muhlethaler, T. Clausen, A. Louiti and L. V. A. Qayyum, "Optimized Link State Routing Protocol for Ad Hoc Network," in *Multi Topic Conference, 2001. IEEE INMIC 2001. Technology for the 21st Century. Proceedings. IEEE International*, 2001.

[16] A.Preethi and Dr.Mrs.G.Kalpana, "AN IMPROVED OLSR PROTOCOL USING MULTIPOINT RELAY SELECTION IN MANET," *International Journal of Advanced Research in Computer Engineering & Technology*, vol. 4, no. 1, pp. 255-261, 2015.

[17] D. Katsaros, L. Maglaras and L. Tassiulas, "VANET Packet Scheduling/Routing and Information Dissemination," 2012.

[18] M. Mohit and M. S. Pal, "Stable MPR Selection in OLSR for Mobile Ad-Hoc Network," *(IJCSIT) International Journal of Computer Science and Information Technologies*, vol. 6(6), pp. 5121-5125, 2015.

LAMPIRAN

A1. Kode Skenario NS2

```
# Simulation with AODV Routing Protocol
# Define Options

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)                OLSR;
set val(energymodel)       EnergyModel;
set val(initialenergy)     100;
set val(lm)                "off";
set val(x)                         5000;
set val(y)                         5000;
set val(stop)              300;
set val(sc)                    "alun.tcl";
set val(cp)                    "traffic1;

set ns_   [new Simulator]
set tracefd [open olsrAlun1.tr w]
set namtrace [open olsrAlun1.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)

# Create nn mobilenodes [$val(nn)] and attach them to channel

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

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

# Set a TCP connection between node_(0) and node_(1) sumo

#Setup a TCP connection

# Define node initial position in nam

for {set i 0} {$i < $val(nn)} {incr i} {
# 30 defines the node size for nam
$ns_ initial_node_pos $node_($i) 50
}

# Telling nodes when the simulation ends

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

# Ending nam and the simulation

# This method of calling print-stats should not be used as it should be
called everytime with a node's id
# However it shall do the same work
# $ns_ at 100.00 "[$node_(1) agent 255] print-stats"

$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_(1) $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, End-to-End Delay dan Topology Control

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

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

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

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

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



{
		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;

              printf "Packet sendLine \t= %d \n", sendLine;
              printf "Packet recvLine \t= %d \n", recvLine;
              printf "Packet PDR Ratio \t= %.4f \n", (recvLine/sendLine);
              printf "Packet forwardLine\t= %d \n", fowardLine;
              printf "End-to-End Delay \t= " n_to_n_delay * 1000 " ms \n";
              printf "Topology Control \t= %d \n", TC;
}
```

BAB VI
KESIMPULAN DAN SARAN

1.1 Kesimpulan

Dari hasil pengamatan dan percobaan selama perancangan, implementasi, dan uji coba aplikasi, maka dapat diambil kesimpulan sebagai berikut:

1. Nilai PDR *(Packet Delivery Ratio)* dengan menggunakan OLSR+PGB cenderung lebih kecil dibandingkan dengan menggunakan OLSR.
2. Nilai *End-to-End Delay* dengan menggunakan OLSR+PGB lebih kecil daripada menggunakan OLSR.
3. Jumlah pesan TC *(Topology Control)* dengan menggunakan OLSR+PGB lebih besar daripada menggunakan OLSR, Hal tersebut terjadi karena dengan menggunakan OLSR+PGB, *node* MPR yang meneruskan paket dan pesan TC lebih banyak daripada dengan menggunakan OLSR.
4. Penerapan algoritma PGB pada protokol OLSR tidak dapat memberikan hasil yang maksimal. Hal tersebut dikarenakan keunggulan OLSR terletak pada pemilihan *node* MPR-nya, sedangkan dengan menggunakan PGB hanya *node* MPR yang berada di *range* tertentu yang dapat meneruskan pengiriman pesan, sedangkan kita tidak mengetahui kualitas *node* MPR pada *range* tersebut. Dengan kata lain penggunaan algoritma PGB tidak dapat memberi jaminan bahwa *node* MPR yang melanjutkan pengiriman pesan adalah *node* MPR yang benar-benar terbaik kualitasnya.
5. Dengan jumlah pesan TC yang lebih banyak, maka tujuan utama untuk mencegah terjadinya *strom broadcasting* belum dapat tercapai dengan menggunakan algoritma PGB pada protokol OLSR.
6. Meskipun tujuan utama tidak terpenuhi, namun dengan menggunakan OLSR+PGB didapatkan nilai *End-to-End Delay*

yang lebih kecil yang berarti proses pengiriman paket data dapat terkirim lebih capat dan stabil.

7. Semakin cepat proses pengiriman suatu paket data, semakin banyak pula jumlah paket data yang dapat terkirim dalam satu waktu.

1.2 Saran

Saran yang diberikan untuk pengembangan aplikasi ini adalah:

1. Pemilihan *node* MPR pada protokol OLSR mungkin akan lebih efektif jika berdasarkan kepada kualitas *node* tetangga, tidak hanya berdasarkan *range* tertentu atapun *signal strength node* tetangga yang didapat.
2. Menambahkan algoritma pemilihan kalitas *node* MPR pada area PGB.
3. Karena protokol OLSR pada NS2 masih merupakan protokol hasil dari *patching* yang berarti bukan protokol resmi dari NS2, maka masih terdapat beberapa *bug* pada protokol tersebut, sehingga perlu diujicobakan performansi protokol OLSR dalam simulator yang berbeda untuk menambah tingkat validasi performansi protokol yang dikembangkan.

DAFTAR PUSTAKA

[1] O. A. Wahab, H. Otrok and A. Mourad, "VANET QoS-OLSR: QoS-based clustering protocol for Vehicular Ad hoc," *Computer Communications,* vol. 36, no. 13, p. 1422–1435, 2013.

[2] F. Ryandi, "Routing Protocol in VANET (Ad Hoc Network)," [Online]. Available: http://farhanryandi.student.telkomuniversity.ac.id/2015/02/12/routing-protocol-in-vanet-ad-hoc-network/.

[3] "Optimized Link State Routing Protocol," [Online]. Available: https://en.wikipedia.org/wiki/Optimized_Link_State_Routing_Protocol.

[4] "Security and Privacy in Vehicular Ad Hoc Networks (VANETs)," [Online]. Available: https://ece.uwaterloo.ca/~kan.yang/security_bbcr/vanet.html.

[5] A. Tønnesen, T. Lopatic, H. Gredler, B. Petrovitsch, A. Kaplan and S.-O. Tücke, "OLSRD an ad hoc wireless mesh routing daemon," [Online]. Available: http://www.olsr.org/.

[6] T. R. Gross and V. Naumov, "Connectivity-Aware Routing (CAR) in Vehicular Ad Hoc Network," in *IEEE INFOCOM 2007 - 26th IEEE International Conference on Computer Communications*, Anchorage, AK, 2007.

[7] U. Nagaraj and P. Dhamal, "Broadcasting Route Protocols in vanet," *Network and Complex Systems,* vol. 1, p. 19, 2011.

[8] "SUMO – Simulation of Urban MObility," [Online]. Available: http://www.dlr.de/ts/en/desktopdefault.aspx/tabid-9883/16931_read-41000/.

[9] "OpenStreetMap," [Online]. Available: http://openstreetmap.id/about/tentang-openstreetmap/.

[10] T. Issariyakul and E. Hossain, Introduction to Network Simulator NS2, Bangkok: Springer US, 2012.

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

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

[13] H. Ghafoor, I. Koo and N.-u.-D. Gohar, "Neighboring and Connectivity-Aware Routing in VANETs," *the Scientific World Journal,* vol. 2014, p. 10, 2014.

[14] 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.

[15] P. Jacquet, P. Muhlethaler, T. Clausen, A. Louiti and L. V. A. Qayyum, "Optimized Link State Routing Protocol for Ad Hoc Network," in *Multi Topic Conference, 2001. IEEE INMIC 2001. Technology for the 21st Century. Proceedings. IEEE International*, 2001.

[16] A.Preethi and Dr.Mrs.G.Kalpana, "AN IMPROVED OLSR PROTOCOL USING MULTIPOINT RELAY SELECTION IN MANET," *International Journal of Advanced Research in Computer Engineering & Technology,* vol. 4, no. 1, pp. 255-261, 2015.

[17] D. Katsaros, L. Maglaras and L. Tassiulas, "VANET Packet Scheduling/Routing and Information Dissemination," 2012.

[18] M. Mohit and M. S. Pal, "Stable MPR Selection in OLSR for Mobile Ad-Hoc Network," *(IJCSIT) International Journal of Computer Science and Information Technologies,* vol. 6(6), pp. 5121-5125, 2015.

BIODATA PENULIS

Mohamad Indra Cahya, lahir di Tuban pada 11 Mei 1994. Penulis menempuh pendidikan mulai dari TK Bhayangkari Soko, Tuban (1998 - 2000), SDN Sokosari 1 (2000 - 2006), SMPN 01 Soko (2006 - 2009), SMAN 01 Rengel (2009 - 2012) dan S1 Teknik Informatika ITS (2012 - 2016).

Selama masa kuliah, penulis aktif dalam organisasi Himpunan Mahasiswa Teknik Computer (HMTC). Diantaranya adalah menjadi staff Departemen Kewirausahaan dan Minat Bakat HMTC 2013 – 2015. Penulis juga aktif dalam kegiatan institute, diantaranya dengan menjadi OC (Organitation Commite) dari acara GERIGI ITS dan ITS EXPO. Penulis juga aktif dalam kegiatan UKM (Unit Kegiatan Mahasiswa) sebagai anggota UKM Futsal.

Selama kuliah di teknik informatika ITS, penulis mengambil rumpun mata kuliah Komputasi Arsitektur Jaringan Kompter (AJK) Komunikasi dengan penulis dapat melalui

e-mail :**cahya49indra @gmail.com.**