TUGAS AKHIR - KI141502

IMPLEMENTASI ADAPTIF CONTROL MESSAGE PADA OPTIMIZED LINK STATE ROUTING PROTOCOL

DIMAS AULIA RAHMAN
NRP 5112100103

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

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

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

TUGAS AKHIR – KI141502

IMPLEMENTASI ADAPTIF CONTROL MESSAGE PADA OPTIMIZED LINK STATE ROUTING PROTOCOL

DIMAS AULIA RAHMAN
NRP 5112100103

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

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

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

[Halaman ini sengaja dikosongkan]

UNDERGRADUATE THESES – KI141502

IMPLEMENTATION OF ADAPTIVE CONTROL MESSAGE ON OPTIMIZED LINK STATE ROUTING PROTOCOL

DIMAS AULIA RAHMAN
NRP 5112100103

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

Supervisor II
Ir. F.X. Arunanto, M.Sc.

DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA 2016

[Halaman ini sengaja dikosongkan]

LEMBAR PENGESAHAN

IMPLEMENTASI ADAPTIF CONTROL MESSAGE PADA OPTIMIZED LINK STATE ROUTING PROTOCOL

TUGAS AKHIR

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

Oleh
DIMAS AULIA RAHMAN
NRP : 5112 100 103

Disetujui oleh Dosen Pembimbing Tugas Akhir:

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

2. Ir. F.X. Arunanto, M.Sc. .....
   NIP: 195701011983031004 (Pembimbing 2)

SURABAYA
JUNI, 2016

[Halaman ini sengaja dikosongkan]

IMPLEMENTASI ADAPTIF CONTROL MESSAGE PADA OPTIMIZED LINK STATE ROUTING PROTOCOL

Nama Mahasiswa	**: DIMAS AULIA RAHMAN**
NRP	**: 5112100103**
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 merupakan routing protocol yang bersifat proaktif, protokol ini mengirimkan hello message secara terus-menerus untuk mengetahui node tetangga di sekitarnya. Sementara untuk mengatasi node yang tidak terjangkau oleh hello message, digunakan TC message untuk menjangkaunya. Salah satu atribut yang dapat dimodifikasi adalah selang waktu pengiriman TC message. Pengiriman dua pesan control tersebut menyebabkan routing overhead yang tinggi pada OLSR.

Berdasarkan permasalahan tersebut, maka dilakukan penelitian terhadap kinerja OLSR yang dapat mengurangi routing overhead dan tetap menjaga kualitas dari OLSR itu sendiri. Penelitian tersebut adalah menggunakan jumlah node tetangga di sekitar node tersebut untuk mengatur selang waktu pengiriman TC message.

Simulator yang digunakan untuk uji coba adalah NS-3. Dari hasil uji coba yang dilakukan dengan protokol yang diusulkan, peforma mengalami kenaikan pada packet delivery ratio dan penurunan pada routing overhead.

Kata kunci: VANET, OLSR, Selang Waktu TC Message, NS-3

[Halaman ini sengaja dikosongkan]

IMPLEMENTATION OF ADAPTIVE CONTROL MESSAGE ON OPTIMIZED LINK STATE ROUTING PROTOCOL

Student's Name	**: DIMAS AULIA RAHMAN**
Student's ID	**: 5112100103**
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 is a proactive routing protocol, this protocol sends hello message continuously to determine the node neighbors. Meanwhile, to overcome node which is not covered by the hello message, used TC message to reach them. One of the attributes that can be modified is the time interval of TC message delivery. These control messages led to the routing overhead is high on OLSR.

Based on these problems will do research of the performance of OLSR which can reduce the routing overhead while still maintaining the quality of OLSR itself. The study is using a number of neighboring nodes around the node to set the time interval of TC message delivery.

Network simulator that is used to conduct this study was NS-3. From the results of experiments performed with the proposed protocol, a performance increase in packet delivery ratio and a performance decrease in routing overhead.

Keyword: VANET, OLSR, TC Message Interval, NS-3

[Halaman ini sengaja dikosongkan]

KATA PENGANTAR

Alhamdulillahirabbil'alamin, segala puji bagi Allah SWT, yang telah melimpahkan rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudulu "**IMPLEMENTASI ADAPTIF CONTROL MESSAGE PADA OPTIMIZED LINK STATE ROUTING PROTOCOL**". Shalawat serta salam senantiasa saya tujukan kepada Rasulullah SAW, selaku panutan utama bagi penulis.

Dalam pelaksanaan dan pembuatan Tugas Akhir ini, tentunya penulis mendapat banyak bantuan dari berbagai pihak. Tanpa mengurangi rasa hormat penulis ingin mengucapkan terima kasih sebanyak-banyaknya kepada:

1. Allah SWT dan Nabi Muhammad SAW.
2. Ayah, Ibu dan kedua kakak dan keluarganya yang selalu memberikan do'a, dukungan, dan motivasi sehingga penulis dapat menyelesaikan Tugas Akhir.
3. Bapak Dr.Eng. Radityo Anggoro, S.Kom., M.Sc. selaku pembimbing I dan koordinatior TA yang selalu membantu dan meluang waktunya untuk penyelesaian Tugas Akhir ini.
4. Bapak Ir. F.X. Arunanto, M.Sc. selaku pembimbing II yang banyak membantu penulis dalam pengerjaan Tugas Akhir.
5. Bapak Dr.Eng Darlis Herumurti, S.Kom.,M.Kom. selaku kepala jurusan Teknik Informatika dan segenap dosen Teknik Informatika yang telah memberikan ilmunya.
6. Teman-teman yang mengambil Tugas Akhir bertema Vanet yang telah berbagi informasi, infromasi, dan pengalaman yang berharga.
7. Teman-teman seangkatan 2012 yang selalu membantu ketika penulis mengalami kesulitan.
8. Teman-teman Demand yang setia menemani penulis untuk berpetualang bersama menjalani derasnya lautan kehidupan sampai saat ini.

9. Serta semua pihak yang telah turut membantu penulis dalam menyelesaikan Tugas Akhir dan kuliah di Teknik Informatika ITS sampai saat ini.

Penulis menyadari bahwa Tugas Akhir ini masih memiliki banya kekurangan dan jauh dari kesempurnaan. Sehingga dengan kerendahan hati, penulis mengharapkan kritik dan saran yang membangun dari pembaca untuk memperbaiki segala kesalahan yang ada.

Surabaya, Juni 2016

DIMAS AULIA RAHMAN

DAFTAR ISI

DAFTAR GAMBAR

[Halaman ini sengaja dikosongkan]

DAFTAR TABEL

[Halaman ini sengaja dikosongkan]

BAB I
PENDAHULUAN

Bab ini membahas garis besar penyusunan Tugas Akhir yang meliputi latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, metodologi penyusunan Tugas Akhir, dan sistematika penulisan.

1.1 Latar Belakang

Dewasa ini perkembangan dunia internet semakin berkembang dengan pesat, tidak hanya *browsing*, *chatting*, dan lain sebagainya. Perkembangan teknologi internet sekarang sudah memasuki pemilihan jalur untuk kendaraan. Penggunaan teknologi internet dalam pemilihan jalur kendaraan dapat membantu manusia dalam mengatasi berbagai permasalahan saat berada di jalan. Proses penentuan rute berhubungan dengan proses pengiriman paket data dalam jaringan internet. Proses pencarian rute untuk pengiriman data dari sumber ke tujuan disebut dengan *routing*. Pembangunan infrastruktur yang digunakan untuk mendukung proses pencarian rute tersebut membutuhkan banyak sumber daya. Salah satu cara untuk mengatasi masalah tersebut adalah dengan memanfaatkan jaringan Ad-Hoc yang mendasari pembuatan Vehicular Ad-Hoc Network (VANET).

VANET merupakan pengembangan dari Mobile Ad-Hoc Network (MANET) yang diaplikasikan dalam kendaraan. Implementasi dari VANET telah menciptakan beberapa teknik *routing protocol*. Secara garis besar, *routing protocol* dalam VANET dapat dibedakan menjadi dua, yaitu *routing protocol* bersifat proaktif dan reaktif. *Routing protocol* proaktif menggunakan basis data untuk menyimpan *node* dan rute yang berada dalam jaringan, sedangkan *routing protocol* reaktif tidak menggunakan basis data melainkan melakukan pencarian *node* berikutnya pada setiap *node* untuk membentuk sebuah rute.

Tugas Akhir ini akan menggunakan *Optimized Link State Routing Protocol* (OLSR) yang diimprovisasi untuk meningkatkan

kinerjanya. OLSR merupakan routing protocol yang bersifat proaktif. OLSR mengirimkan *control message* untuk menentukan rute pengiriman paket data berikutnya. Pengiriman paket data terus menerus seperti ini dapat menimbulkan routing overhead yang membebani jaringan. Pada Tugas Akhir ini, selang waktu pengiriman *control message* akan diatur bersifat adaptif yang mengacu pada jumlah *node* disekitarnya dengan tujuan untuk meningkatkan peforma OLSR dan mengurangi *routing overhead* dalam jaringan.

1.2 Rumusan Masalah

Rumusan masalah yang diangkat dalam Tugas Akhir dapat dipaparkan sebagai berikut:

1. Bagaimana menentukan selang waktu *control message* secara adaptif terhadap jumlah *node* tetangga di sekitarnya yang ada dalam jaringan?
2. Seberapa besarkah pengaruh selang waktu *control message* yang adaptif terhadap peforma *routing protocol* OLSR?

1.3 Batasan Masalah

Permasalahan yang dibahas dalam Tugas Akhir ini memiliki beberapa batasan antara lain:

1. Menentukan selang waktu *control message* mengacu pada jumlah *node* tetangga (*1-hop nodes*) di sekitarnya.
2. Uji coba menggunakan Network Simulator 3 (NS-3) versi 3.22.
3. Pembuatan skenario uji coba menggunakan Simulation of Urban Mobility (SUMO).

1.4 Tujuan

Tujuan dari pembuatan Tugas Akhir ini adalah untuk meningkatkan peforma *routing protocol* OLSR.

1.5 Manfaat

Manfaat dari hasil pembuatan Tugas Akhir ini yaitu sebagai penambah wawasan mengenai protokol proaktif dalam lingkungan VANET, terutama OLSR. Diharapkan hasil Tugas Akhir ini dapat

digunakan untuk meningkatkan implementasi VANET di masa depan.

1.6 Metodologi

1. Penyusunan proposal tugas akhir

   Penyusunan Proposal Tugas Akhir ini merupakan tahap awal dalam proses pengerjaan Tugas Akhir. Dalam proposal ini mengimplementasikan selang waktu yang bersifat adaptif pada *routing protocol* OLSR. Pendahuluan pada proposal Tugas Akhir ini terdiri dari latar belakang diajukanya usulan Tugas Akhir, rumusan masalah yang diangkat, batasan masalah untuk Tugas Akhir, tujuan dari pembuatan Tugas Akhir, dan manfaat hasil dari pembuatan Tugas Akhir. Selain itu dijulaskan pula tinjauan pustaka yang digunakan sebagai referensi pendukung implementasi Tugas Akhir. Pada proposal ini juga terdapat perencanaan jadwal pengerjaan Tugas Akhir.

2. Studi literatur

   Tugas Akhir ini menggunakan literatur *paper*. *Paper* yang digunakan adalah "Performance Improvement of Optimized Link State Routing (OLSR) Protocol" dan "Tuning OLSR". Kedua *paper* tersebut menjelaskan mengenai perubahan selang waktu dari *control message* OLSR mempengaruhi peforma dari OLSR tersebut.

3. Analisis dan desain perangkat lunak

   Tugas Akhir ini akan mengubah selang waktu dari *control message* dari *node* asal dan *multipoint relay* bergantung pada jumlah *node* yang ada di sekitar *node* asal. Semakin banyak jumlah *node*, maka *control message* dari *multipoint relay* akan dikurangi. Sedangkan semakin sedikit jumlah *node*, maka *control message* dari *multipoint relay* akan dinormalkan atau dipercepat.

4. Implementasi perangkat lunak

   Implementasi dari Tugas Akhir ini adalah sebuah simulasi keadaan dimana *routing protocol* yang diusulkan akan digunakan di dalamnya. Kakas bantu yang akan digunakan

untuk simulasi adalah Simulation of Urban Mobility (SUMO) dan Network Simulator 3 (NS3).

5. Pengujian dan evaluasi

   Tugas Akhir ini akan diuji dan dievaluasi dengan perbandingan *routing protocol* OLSR di lingkungan perkotaan. Hasil dari pengujian dapat dilihat dari perbedaan *Packet Delivery Ratio* (PDR) dan *routing overhead* (RO) dari kedua *routing protocol*.

6. Penyusunan Buku Tugas Akhir

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

1.7 Sistematika Penulisan Laporan Tugas Akhir

Buku Tugas Akhir ini bertujuan untuk mendapatkan gambaran dari pengerjaan Tugas Akhir ini. Selain itu, diharapkan dapat berguna untuk pembaca yang tertarik untuk melakukan pengembangan lebih lanjut. Secara garis besar, buku Tugas Akhir terdiri atas beberapa bagian seperti berikut ini:

BAB I PENDAHULUAN

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

BAB II TINJAUAN PUSTAKA

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

BAB III PERANCANGAN

Bab ini berisi perancangan metode yang nantinya akan diimplementasikan dan dilakukan uji coba.

BAB IV IMPLEMENTASI

Bab ini membahas implementasi dari desain yang telah dibuat pada bab sebelumnya. Penjelasan berupa implementasi mobilitas kendaraan, konfigurasi system, dan skrip analisis yang digunakan untuk menguji performa *routing protocol*.

BAB V PENGUJIAN 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.

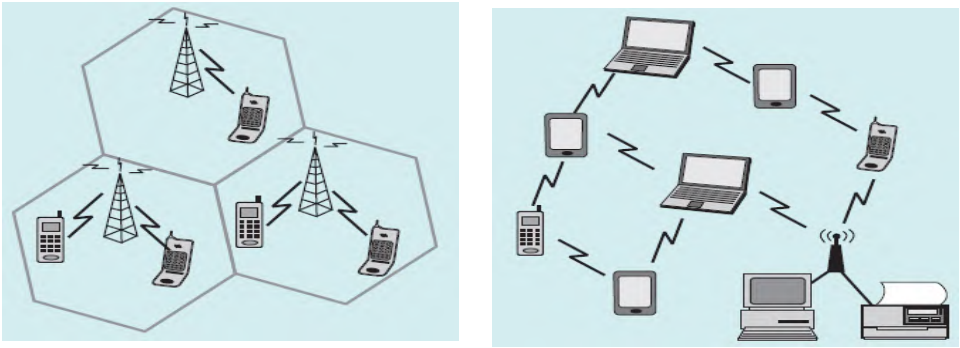
[Halaman ini sengaja dikosongkan]

BAB II
TINJAUAN PUSTAKA

Bab ini berisi penjelasan teori-teori yang berkaitan dengan pengimplementasian perangkat lunak. Penjelasan ini bertujuan untuk memberikan gambaran secara umum terhadap *routing protocol* yang dirancang, kakas bantu yang dgunakan, dan berguna sebagai penunjang dalam pengembangan *routing protocol*.

2.1 Mobile Ad-Hoc Network (VANET)

Mobile Ad-Hoc Network adalah sebuah jaringan yang didalamnya terdapat kumpulan dari *mobile node* yang saling berhubungan menggunakan media komunikasi *wireless* dengan cara berkomunikasi dinamis dan sesuai kebutuhan masing-masing *mobile node* [1]. Komunikasi yang terjadi tidak bergantung kepada infrastruktur saja, tetapi dapat terjadi antar *mobile* devices. Perbedaan dari cellular network dan Mobile Ad-Hoc Network dapat dilihat pada Gambar 2-1.



(a)Cellular Network (b)Mobile Ad-Hoc Network

Gambar 2-1. Perbedaan Cellular Network dengan Mobile Ad-Hoc Network [2]

2.2 Vehicular Ad-Hoc Network (VANET)

Vehicular Ad-Hoc Network adalah penerapan dari MANET pada kendaraan. VANET terbentuk dari komunikasi antara kendaraan dengan kendaraan disekitarnya dan infrastruktur pendukung VANET, umumnya disebit sebagai *roadside unit* (RSU). Implementasi dari VANET dapat terjadi dengan penggunaan radio spesial dan sensor khusus yang ditempatkan pada kendaraan, misal pada mobil [3]. Sebuah jaringan VANET bertujuan untuk meningkatkan keselamatan pengemudi dan manajemen lalu lintas serta menyediakan akses internet untuk pemngemudi dan penumpang di dalam kendaraan. Dalam VANET, RSU dapat memberikan bantuan dalam menemukan fasilitas seperti restoran dan pom bensin, dan melakukan *broadcast* pesan yang terkait, seperti pemberitahuan untuk memberikan pengendara mengenai informasi yang ada di jalan. Untuk memnuhi fungsionalitas tersebut, suatu *On-Boards Unit* (OBU) secara teratur menyiarkan pesan yag terkait dengan informasi dari pengendara, waktu saat itu, arah mengemudi, kecepatan, status rem, sudut kemudi, lampu sen, akselerasi, dan kondisi lalu lintas.

2.3 Optimized Link State Routing Protocol (OLSR)

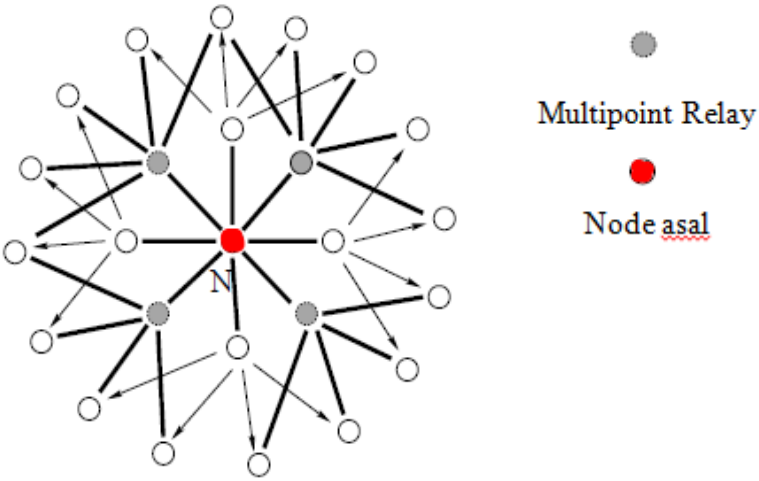
Optimized Link State Routing Protocol merupakan sebuah routing protocol yang berdasarkan pada algoritma *link state routing* yang bersifat proaktif (*table-driven*). *Link state routing* pertama kali diimplemetasukan sekita tahun 1978 oleh John McQuillan pada jaringan Advanced Research Project Agency Network (ARPANET) [4]. Algoritma link state routing menggunakan Algoritma Dijkstra untuk menghitung rute terbaik. Langkah-langkah dari algoritma link state routing adalah sebagai berikut [5]:

- Menentukan tetangga dari setiap *node*
- Setiap mengirimkan control message yaitu HELLO message pada setiap *node* tetangga (*node* dengan jarak 1-hop dalam jangkauan sebuah *node*), sedangkan *multipoint relay* akan

mengirimkan TC message ke setiap tetangga kecuali *node* asal

- Memasukan data ke routing table ditambah dengan sequence number untuk mengatasi perubahan yang terjadi pada jaringan
- Menghitung rute terbaik ke tujuan dengan Algoritma Dijkstra.

Optimized Link State Routing Protocol menggunakan langkah kerja yang sama dengan *link state algorithm*. Perbedaanya yaitu OLSR menggunakan *multipoint relay* untuk mengurangi kepadatan pengiriman paket data dalam jaringan. *Multipoint relay* merupakan *node*s pilihan yang digunakan untuk mengirim *control message*, agar tidak setiap *node* yang mendapat *control message* pertama dari *node* asal mengirimkan *control message* ke *node* yang sama. Implementasi dari *multipoint relay* dapat dilihat pada Gambar 2-2.



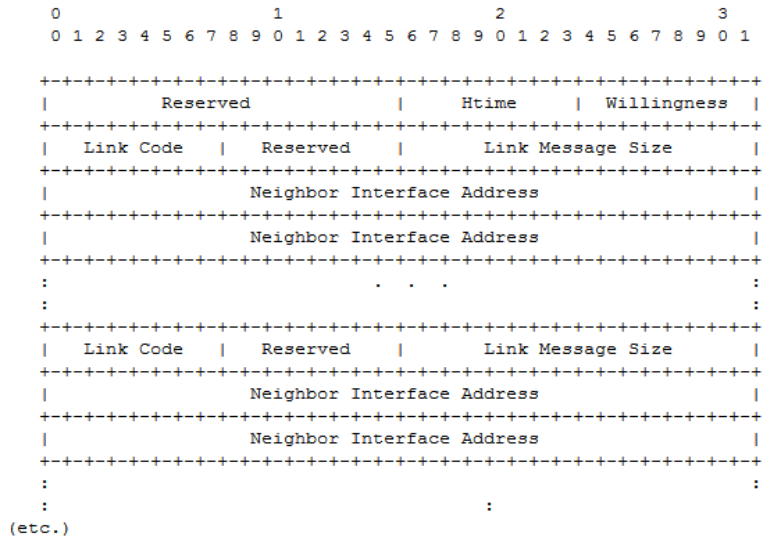
Gambar 2-2. Penerapan Multipoint Relay pada Optimized Link State Routing Protocol [6]

2.3.1 Control Messages

Control messages yang digunakan untuk proses *routing* pada OLSR ada dua, yaitu:

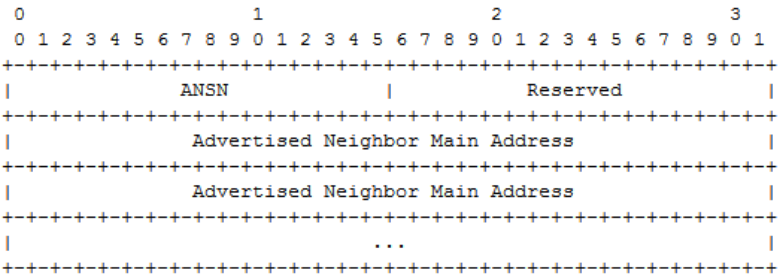
a) *Hello Message*

Hello Message adalah *control message* yang dikirimkan secara berkala (*default*=2 detik) untuk *link sensing*, mendeteksi keberadaan *node* tetangga di sekitar, dan pemilihan *multi point relay*. Pengiriman dilakukan dengan cara *broadcast* dan setiap *node* mengirim *hello message*, sehingga informasi didapatkan dengan pertukaran *hello message* tanpa menggunakan ACK. TTL dari *hello message* adalah 1, yang berarti *hello message* tidak di-*broadcast* ulang oleh *node* penerimanya. Agar menjaga informasi tetap *valid*, setiap pesan memiliki *valid time* selama tiga kali dari selang waktu pengiriman *hello message* (*default*=6 detik). Format dari *hello message* dapat dilihat pada Gambar 2-3.



***Gambar 2-3. Format hello message* [7]**

b) *Topology Control Message* (*TC Message*)
Topology Control Message adalah *control message* yang dikirimkan secara berkala (*default*=5 detik) dan di-*broadcast* ulang oleh node MPR-nya. *Control message* ini digunakan untuk menyebarkan informasi rute yang dapat digunakan untuk pengiriman paket data dengan cara melampirkan alamat dari *node* yang direkomendasikan untuk menjadi *next-hop* dari *node* yang memilih *node* pengirim *TC Message* sebagai MPR-nya. Pengiriman dilakukan dengan cara *broadcast* dan setiap MPR mengirim *TC Message*, kemudian di-*broadcast* ulang oleh MPR dari *node* MPR tersebut. Informasi didapatkan dengan pertukaran *TC Message* tanpa menggunakan ACK. TTL dari *TC message* adalah 255, yaitu nilai TTL maksimum agar *TC Message* dapat menyebar luas dalam jaringan. Agar menjaga informasi tetap *valid*, setiap pesan memiliki *valid time* selama tiga kali dari selang waktu pengiriman *TC message* (*default*=15 detik). Format dari *TC message* dapat dilihat pada Gambar 2-4.



***Gambar 2- 4. Format topology control message* [7]**

ANSN adalah *Advertised Neighbor Sequence Number* yang berubah setiap kali terjadi perubahan pada *Adevertised Neighbor Set* (*node* yang direkomndasikan menjadi *next-hop*) dari suatu *node*.

2.4 Simulation of Urban Mobility (SUMO)

Simulation of Urban Mobility (SUMO) adalah aplikasi simulator yang digunakan untuk membuat simulasi pergerakan kendaraan pada daerah dan jalur tertentu. SUMO bersifat *free*, *open-source*, dengan ukuran yang kecil dan mencakup berbagai bentuk dari jalan raya. SUMO dikembangkan pada tahun 2000 yang bertujuan untuk membantu penelitian-penelitian ang melibatkan pergerakan kendaraan di jalan raya, terutama pada daerah perkotaan. SUMO dikembangkan pertama kali oleh Daniel Krajzewicz, Eric Nikolay, dan Michael Behrisch.

SUMO terdiri dari banyak perkakas yang memiliki berbagai fungsi. Berikut merupakan penjelasan dari perkakas bantu dari SUMO yang digunakan untuk pembuatan Tugas Akhir ini:

- netgenerate
  netgenerate merupakan perkakas bantu yang berfungsi untuk membuat peta jalan raya. Terdapat beberapa jenis jalan yang dapat dibuat dengan netgenerate, antara lain grid, *spider*, dan *random network*. Perkakas ini juga dapat menentukan kecepatan maksimal dan lampu lalu lintas dari peta jalan raya. Hasil netgenerate berupa file dengan ekstensi .net.xml. Pada Tugas Akhir ini, perkakas netgenerate digunakan untuk mebuat peta jalan raya untuk scenario grid.
- netconvert
  netconvert merupakan perkakas bantu yang berfungsi untuk mengkonversi peta riil menjadi sebuah peta jalan raya dengan ekstensi .net.xml sesuai dengan format SUMO. Pada Tugas Akhir ini, peta riil didapatkan dari OpenStreetMap, dengan ekstensi .osm.
- randomTrips.py
  randomTrips.py merupakan perkakas bantu python pada SUMO yang berfungsi untuk membuat *node* beserta titik awal, dan tujuannya secara acak. Hasil dari randomTrips.py adalah sebuah file dengan ekstensi .trips.xml. Pada Tugas

Akhir ini randomTrips.py digunakan untuk membuat *node* pada skenario grid dan riil.

- duarouter
  duarouter berfungsi untuk membuat rute pergerakan tiap *node* sesuai dengan titik asal menuju tujuan dari masing-masing *node* yang dibuat oleh randomTrips.py. Secara default duarouter menggunakan algoritma Djikstra untuk membuat rutenya. Hasil dari duarouter berupa dua buah *file* berekstensi .rou.xml dan .rou.alt.xml. Kedua *file* memiliki isi yang sama dan berfungsi sebagai *file* cadangan. Pada Tugas Akhir ini, duarouter digunakan untuk menentukan rute tiap *node* di skenario grid dan scenario riil.
- sumo-gui
  sumo-gui berfungsi untuk menampilkan pergerakan setiap *node* pada peta scenario yang telah dibuat secara grafikal. Cara kerja sumo-gui adalah dengan menggabungkan *file* peta scenario .net.xml dan *file* rute tiap *node* .rou.xml dalam sebah file dengan format .sumo.cfg. Pada Tugas Akhir ini sumo-gui digunakan untuk menghasilkan skenario yang akan digunakan pada skenario grid dan skenario riil.
- sumo
  perkakas bantu sumo berfungsi untuk membuat *file* berekstensi .xml yang dapat diekspor ke dalam bebrbagai Bahasa simulasi, pada Tugas Akhir ini akan diekspor kedalam bahasa simulasi NS-3 dengan perkakas traceExporter.py. Pada dasarnya perkakas ini sama dengan sumo-gui tetapi tanpa menampilkan pergerakan skenario secara grafikal.
- traceExporter.py
  traceExporter.py merupakan perkakas bantu python ang digunakan untuk mengeskpor *file* SUMO berekstensi .xml agar dapat digunakan pada perkakas simulasi seperti NS-3. Pada Tugas Akhir ini, *file* SUMO akan diekspor menggunakan fungsi ns2mobility-output pada traceExporter.py agar dapat digunakan pada NS-3.

2.5 OpenStreetMap

OpenStreetMap adalah proyek yang mengumpulkan data spasial dan digunakan secara bebas (Open Data). Data tersebut digunakan untuk membangun peta dunia dan turunanya yang dimanfaatkan untuk beragam kebutuhan termasuk navigasi. OpenStreetMap memungkinkan siapa saja untuk melihat, mengubah, dan menggunakan data geografs yang telah dibangun secara kolaboratf dari berbagai tempat di permukaan Bumi.

Inti dari proyek ini adalah sebuah database geografis yang dapat dimanfaatkan secara bebas berdasarkan Lisensi Open Database. Pemanfaatan untuk keperluan di cetak, website, atau aplikasi perangkat lunak seperti navigasi tidak dibatasi atau ditarik baya, asal menyebutan OpenStreetMap sebagai sumber datanya. Pada Tugas Akhir ini OpenStreetMap digunakan sebagai penyedia peta untuk skenario riil, yaitu pada daerah Surabaya.

2.6 Java OpenStreetMap Editor (JOSM)

JOSM adalah alat untuk mengedit data yang didapatkan dari OpenStreetMap. JOSM awalnya dikembangkan oleh Immanuel Scholz dan saat ini dikelola oleh Dirk Stockeer. Untuk mengoprasikan perkakas ini, java harus terpasang terlebih dahulu. Pada Tugas Akhir ini, JOSM digunakan untuk mengedit peta riil yang didapatkan dari OpenStreetMap.

2.7 AWK

Awk adalah bagasa pemrograman yang digunakan untuk mengelola data, terutama yang bertipe *string* untuk dimanipulasi dan menghasilkan laporan yang diinginkan. Awk menggunakan pemrograman filter untuk teks, seperti perintah “grep” pada linux. Awk digunakan untuk mengganti bentuk satu teks ke bentuk teks lain, memporses perintah aritmatika, dan melakukan filter dari sejumlah data bertipe *string*. Awk adalah sebuah perograman sepeti pada shell atau C yang memiliki karakterstik yaitu sebuah perkakas

yang cocok digunakan sebagai pelengkap untuk filter standar. Pada Tugas Akhir ini, jenis AWK yang digunakan adalah GAWK dan berfungsi untuk membuat script penghitung *Packet Delivery Ratio* (PDR) dan *Routing Overhead* (RO) dari hasi *tracing* pada NS-3.

2.8 Network Simulator 3 (NS-3)

Perkakas simulasi NS-3 merupakan sebuah *network simulator* yang memiliki tujuan utama untuk menggantikan posisi Network Simulator 2 (NS-2) pada bidang riset dan pendidikan. Proyek NS-3, dimulai pada tahun 2006, adalah sebuah proyek *open source* yang diatur oleh komunitas peneliti dan pengembang Network Simulator. NS-3 biasanya digunakan pada system operasi Linux, namun dapat digunakan dengan FreeBSD, Cygwin (Windows), dan dukungan dari Visual Studio (masih dalam pengembangan). NS-3 dikembangkan dalam Bahasa C++ di lapisan inti dan sedikit skrip python. Fitur-fitur NS-3 diantaranya adalah system atribut NS-3 terdokumentasi dengan baik dengan menggunakan doxygen. Setiap objek NS-3 memiliki seperangkat atribut *(name, type, initial value)* dan NS-3 dikembangkan semirip mungkin dengan system pada dunia nyata.

Pada VANET, NS-3 memiliki beberapa fitur yang dapat dimanfaatkan untuk memodelkan dan menguji protocol VANET. VANET disimulasikan dengan membuat skenario jaringan dan penggunaan *routing protocol* yang akan diujikan. Pada NS-3 terdapat *source* untuk emlakukan percobaan lingkungan VANET, pembuatan topologi, *node* dan protokol yang digunakan untuk VANET. Dengan NS-3 fungsi-fungsi baru didalam *core* NS-3 dapat ditambahkan atau diubah, karena NS-3 bersifat *open-source*. Simulasi VANET membutuhkan model yang bisa menangkap jaringan *wireless* bersifat dinamis, dengan propogasi dan karakter mobilitas dari skenario. NS-3 juga terintegrasi dengan beberapa perkakas lain seperti SUMO untuk pembuatan scenario atau Wireshark untuk *tracing* hasil simulasi. Pada Tugas Akhir ini NS-3 digunakan untuk mensimulasikan skenario VANET pada skenario grid dan skenario riil.

2.8.1 Instalasi NS-3

Instalasi NS-3 pada Tugas Akhir ini dipasang pada system operasi Linux Ubuntu 14.04 LTS dengan persyaratan RAM lebih dari 1GB, karena ada kemungkinan *out of memory* saat instalasi menggunakan RAM kurang dari atau sama dengan 1GB. Langkah pertama instalasi NS-3 yaitu melakukan instalasi dependensi yang diperlukan dalam penggunaan NS-3. Terdapat banya dependensi yang dibutuhkan oleh NS-3, oleh karena itu agar mengurangi kemungkinan *error* selama penggunaan NS-3 lebih baik memasang semua dependensi yang diperlukan. Perintah yang diguakan untuk memasang dependensi yang diperlukan ada pada Gambar 2-5.

```
sudo apt-get install gcc g++ python python-dev qt4-
dev-tools libqt4-dev cmake libc6-dev libc6-dev-i386
g++-multilib libgsl0-dev libgsl0ldbl libxml2 libxml2-
dev texlive texlive-extra-utils texlive-latex-extra
dvipng bzr mercurial gdb valgrind gsl-bin libgs-dev
flex bison libfl-dev tcpdump sqlite sqlite3
libsqlite3-dev libgtk2.0-0 libgtk2.0-dev vtun lxc
uncrustify doxygen graphviz imagemagick python-sphinx
dia python-pygraphviz python-kiwi python-pygoocanvas
libgoocanvas-dev libboost-signals-dev libboost-
filesystem-dev openmpi-bin openmpi-common openmpi-doc
libopenmpi-dev
```

Gambar 2-5. Perintah untuk memasang dependensi NS-3

Setelah semua dependensi terpasang, unduh dan ekstrak modul ns-3. Pada Tugas Akhir ini proses ini menggunakan Tarball dan ekstrasi *file* modul dengan perintah seperti pada Gambar 2-6.

```
cd
mkdir workspace
cd workspace
wget https://www.nsnam.org/releases/ns-allinone-
3.22.tar.bz2
tar xjf ns-allinone-3.22.tar.bz2
```

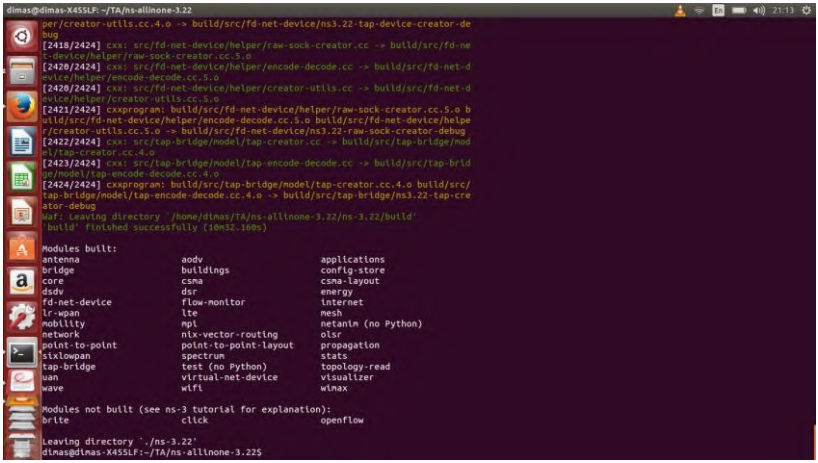
Gambar 2-6. Perintah untuk mengunduh modul NS-3

File modul NS-3 yang telah diunduh dan diekstrak kemudian dilakukan proses *build* dengan perintah build.py yang ada pada modul ns-3. Perintah untuk melakukan konfigurasi dengan build.py dapat dilihat pada Gambar 2-7.

```
./build.py --enable-examples --enable-tests
```

Gambar 2-7. Perintah untuk build NS-3 dengan build.py

Pada Tugas Akhir ini, dibutuhkan modul example dan test, maka pada saat proses build.py ditambahkan argumen untuk mengaktifkan modul tests dan example. Hasil setelah proses *build* modul ns-3 dengan menggunakan build.py dapat dilihat pada Gambar 2-8.



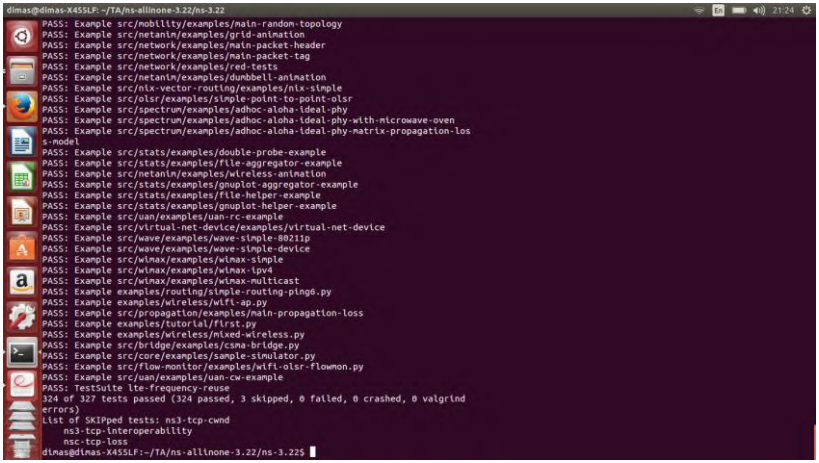
Gambar 2-8. Hasil output build.py

Proses *build* akan memakan waktu yang cukup lama. Pastikan proses *build* sukses untuk setiap modul kecuali modul brite, click, dan openflow. Kemudian lakukan tes untuk menguji fungsi-fungsi NS-3 dapat dijalankan dengan benar menggunakan perintah test.py seperti pada Gambar 2-9.

```
./test.py -c core
```

Gambar 2-9. Perintah untuk cek fungsi NS-3

Proses tes juga berlangsung cukup lama dan terdapat tiga fungsi yang dilewati yaitu, ns3-tcp-cwnd, ns3-tcp-interoperability, dan nsc-tcp-loss, tetapi hal tesebut tidak masalah. Hasil dari proses ini dapat dilihat pada Gambar 2-10.



Gambar 2-10. Hasil output test.py

Untuk percobaan running skrip pertama dapat dilakukan dengan skrip yang telah disediakan oleh NS-3 yaitu hello-simulator. Perintah untuk menjalankan suatu skrip dapat dilakukan dengan waf. Perintah untuk menjalankan contoh hello-simulator dapat dilakukan seperti pada Gambar 2-11.

```
./waf --run hello-simulator
```

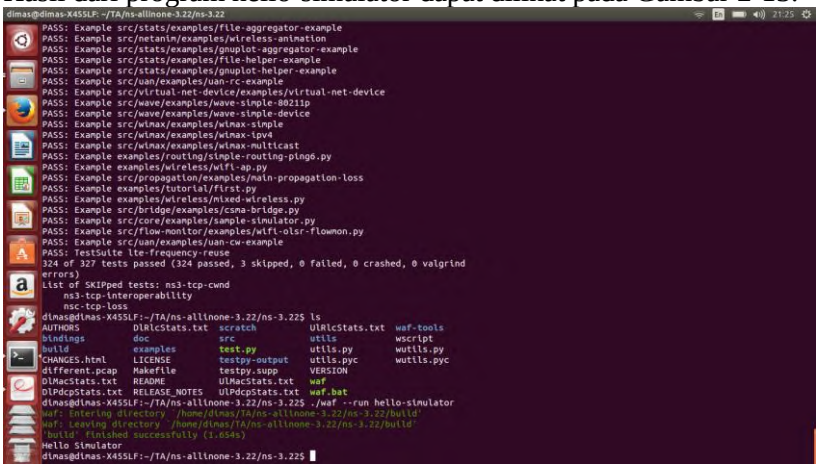
Gambar 2-11. Perintah untuk menjalankan program hello simulator

Pada penggunaan NS-3 nantinya, disarankan untuk menyalin skrip yang ingin dijalankan ke dalam folder scratch terlebih dahulu, agar mudah untuk *tracking* dan *editing* pada skrip tersebut. Setelah melakukan penyalinan, perintah waf akan sedikit berbeda, dapat dilihat pada Gambar 2-12.

```
./waf --run "scratch/hello-simulator"
```

Gambar 2-12. Perintah untuk menjalankan program hello simulator setelah disalin ke direktori scratch

Hasil dari program hello-simulator dapat dilihat pada Gambar 2-13.



Gambar 2-13. Hasil output program hello simulator

2.8.2 vanet-routing-compare.cc

Pada NS-3, terdapat skrip program yang ditujukan untuk mempermudah riset dalam bidang VANET, yaitu vanet-routing-compare.cc. Skrip ini dapat ditemukan pada direktori src/wave/examples. Skrip ini merupakan kombinasi dari beberapa modul *examples* milik NS-3 yang sangat membantu dalam pengembangan simulasi VANET.

Program vanet-routing-compare memiliki berbagai parameter simulasi yang dapat diubah sesuai dengan kebutuhan simulasi yang akan dilakukan. Parameter-parameter yang dimiliki oleh vanet-routing-compare dan dapat diubah sesuai dengan kebutuhan ditunjukan oleh Tabel 2-1 [7].

Tabel 2-1. Parameter vanet-routing-compare

Parameter	Keterangan	Tipe Data	Default
m_80211mode	Menentukan tipe koneksi yang digunakan. 1=802.11p 2=802.11b 3=WAVE-PHY	Integer	1
m_asciiTrace	Mengaktifkan ascii tracing. 0=tidak aktif 1=aktif	Integer	0
m_gpsAccuracyNs	Akurasi GPS, dalam satuan ns.	Double	40
m_rate	Rate pengiriman data.	String	512bps
m_nNodes	Jumlah *node* dalam simulasi.	Integer	156
m_mobility	Mobilitas yang digunakan pada skenario. 1=trace 2=RWP	Integer	1
m_nodePause	Menghentikan *node* untuk sementara.	Integer	0
m_nodeSpeed	Kecepatan *node*.	Integer	20
m_nSinks	Jumlah *routing sinks*.	Integer	10
m_pcap	Mengaktifkan pcap	Integer	0

Parameter	Keterangan	Tipe Data	Default
	tracing untuk setiap *node*. 0=tidak aktif 1=aktif		
m_protocol	Protokol yang digunakan untuk simulasi. 1=OLSR 2=ODV 3=DSDV 4=DSR	Integer	2
m_routingTables	Mengisi *routing table* setiap t=5. 0=tidak aktif 1=aktif	Integer	0
m_scenario	Memilih skenario yang akan digunakan. 1=synthetic 2=playback-trace	Integer	1
m_TotalSimTime	Total waktu simulasi.	Double	300.01
m_traceFile	NS-2 trace file untuk mobilitas skenario.	String	/src/wave/examples/low_ct-unterstrass-1day.filt.5.adj.mob
m_txp	Kekuatan transmisi.	Double	7.5
m_lossModel	1=Friis 2=ItuR1411Los 3=TwoRayGround 4=LogDistance	Integer	3
m_phyModeB	Wifi Phy mode	String	OfdmRate6MbpsBW10MHz
m_txMaxDelayMs	Tx maksimal delay, dalam satuan ms	Double	10

2.8.3 NS-3 Trace File

NS-3 Trace File merupakan hasil *output* dari simulasi skenario yang dijalankan denga NS-3. NS-3 memiliki dua jenis *trace file* yaitu, pcap *trace file* berekstensi.pcap yang menyimpan informasi *routing* dari setiap *node* dan ascii *trace file* berekstensi .tr yang menyimpan informasi jaringan secara keseluruhan. Ascii *trace file* adalah bentuk *tracing* yang *familiar* bagi pengguna NS-2. Pcap *trace file* dapat dianalisis dengan menggunakan perkakas bantu yang dapat menganalisa jaringan seperti wireshark, sedangkan ascii *trace file* menyimpan informasi dalam bentuk teks yang dapat dianalisa dengan pengolah data teks. Pada Tugas Akhir ini *trace file* yang digunakan adalah ascii *trace file*.

Ascii *trace file* berisi semua pengiriman dan penerimaan paket data. Informasi yang didapatkan akan diolah untuk menghitung PDR dan RO dari *routing protocol* yang digunakan dalam simulasi pada Tugas Akhir ini. Pengitungan metriks dilakukan dengan menggunakan skrip awk untuk mengolah data teks dari asci *trace file*. Gambar 2-14 merupakan contoh dari pengiriman *Hello Message* pada NS-3 ascii *trace file*.

```
t 0.0182087
/NodeList/1/DeviceList/0/$ns3::WifiNetDevice/Phy/Stat
e/Tx ns3::WifiMacHeader (DATA ToDS=0, FromDS=0,
MoreFrag=0, Retry=0, MoreData=0
Duration/ID=0usDA=ff:ff:ff:ff:ff:ff,
SA=00:00:00:00:00:02, BSSID=ff:ff:ff:ff:ff:ff,
FragNumber=0, SeqNumber=0) ns3::LlcSnapHeader (type
0x800) ns3::Ipv4Header (tos 0x0 DSCP Default ECN Not-
ECT ttl 64 id 0 protocol 17 offset (bytes) 0 flags
[none] length: 48 10.1.0.2 > 10.1.255.255)
ns3::UdpHeader (length: 28 698 > 698)
ns3::olsr::PacketHeader () ns3::olsr::MessageHeader
() ns3::WifiMacTrailer ()
```

Gambar 2-14. Contoh pengiriman control message pada asci trace file NS-3

Gambar 2-15 merupakan contoh dari penerimaan *Hello Message* pada NS-3 ascii *trace file*.

```
r 0.0183693
/NodeList/24/DeviceList/0/$ns3::WifiNetDevice/Phy/Sta
te/RxOk ns3::WifiMacHeader (DATA ToDS=0, FromDS=0,
MoreFrag=0, Retry=0, MoreData=0
Duration/ID=0usDA=ff:ff:ff:ff:ff:ff,
SA=00:00:00:00:00:02, BSSID=ff:ff:ff:ff:ff:ff,
FragNumber=0, SeqNumber=0) ns3::LlcSnapHeader (type
0x800) ns3::Ipv4Header (tos 0x0 DSCP Default ECN Not-
ECT ttl 64 id 0 protocol 17 offset (bytes) 0 flags
[none] length: 48 10.1.0.2 > 10.1.255.255)
ns3::UdpHeader (length: 28 698 > 698)
ns3::olsr::PacketHeader () ns3::olsr::MessageHeader
() ns3::WifiMacTrailer ()
```

Gambar 2-15. Contoh penerimaan control message pada asci trace file NS-3

Pembeda dari pengiriman dan penerimaan dari sebuah paket data adalah huruf pertama dari baris *trace*. Untuk pengiriman ditandai dengan "t" sedangkan untuk penerimaan ditandai dengan "r". Salah satu ciri yang menandakan bahwa pesan tersebut merupakan *hello message* adalah penerima pada saat pengiriman *hello message* adalah alamat *broadcast* pada jaringan tersebut. Sedangkan untuk paket data lain *Topology Control Message* (TC Message) ditandai dengan pengirim dan penerima sebagal alamat asal dan tujuan, serta terdapat ukuran dari paket data tersebut. Paket yang terekam dalam ascii *trace file* juga memiliki id yang membedakan antar pesan. Id dapat dilihat pada sub ke 29 dari asci *trace file*. Gambar 2-16 merupakan contoh dari pengiriman paket data pada NS-3 ascii *trace file*. Gambar 2-17 merupakan contoh penerimaan paket data pada NS-3 ascii *trace file*.

Ciri lain dari paket data yang digunakan dalam *routing protocol* OLSR terdapat pada sub yang menunjuka bahwa paket data tersebut merupakan paket data dari OLSR yaitu ns3::olsr::PacketHeader. Informasi ini berguna untuk menghitung *routing overhead* yang terjadi pada jaringan.

```
t 10.698
/NodeList/1/DeviceList/0/$ns3::WifiNetDevice/Phy/Stat
e/Tx ns3::WifiMacHeader (DATA ToDS=0, FromDS=0,
MoreFrag=0, Retry=0, MoreData=0
Duration/ID=120usDA=00:00:00:00:00:19,
SA=00:00:00:00:00:02, BSSID=ff:ff:ff:ff:ff:ff,
FragNumber=0, SeqNumber=104) ns3::LlcSnapHeader (type
0x800) ns3::Ipv4Header (tos 0x0 DSCP Default ECN Not-
ECT ttl 64 id 0 protocol 17 offset (bytes) 0 flags
[none] length: 92 10.1.0.2 > 10.1.0.1) ns3::UdpHeader
(length: 72 49153 > 9) Payload (size=64)
ns3::WifiMacTrailer ()
```

Gambar 2-16. Contoh pengiriman paket data pada asci trace file NS-3

```
r 10.7311
/NodeList/0/DeviceList/0/$ns3::WifiNetDevice/Phy/Stat
e/RxOk ns3::WifiMacHeader (DATA ToDS=0, FromDS=0,
MoreFrag=0, Retry=0, MoreData=0
Duration/ID=120usDA=00:00:00:00:00:01,
SA=00:00:00:00:00:16, BSSID=ff:ff:ff:ff:ff:ff,
FragNumber=0, SeqNumber=9) ns3::LlcSnapHeader (type
0x800) ns3::Ipv4Header (tos 0x0 DSCP Default ECN Not-
ECT ttl 60 id 0 protocol 17 offset (bytes) 0 flags
[none] length: 92 10.1.0.2 > 10.1.0.1) ns3::UdpHeader
(length: 72 49153 > 9) Payload (size=64)
ns3::WifiMacTrailer
```

Gambar 2-17. Contoh penerimaan paket data pada asci trace file NS-3

Pada NS-3 ascii *trace file* dapat diketahui *node* mana yang mengirim atau menerima paket data tersebut pada sub ke 3 dari tiap baris *trace*. Angka sesudah "NodeList" menujukan *node* yang bersangkutan, sedangkan angka setelah "DeviceList" menujukan *interface* keberapa yang digunakan untuk mengirim atau menerima paket data. Pada Tugas Akhir ini, Semua *node* hanya memiliki satu *interface*.

2.9 PyVis

PyVis merupakan animator untuk simulator NS dengan basis python. PyVis digunakan untuk menampilkan simulasi dalam bentuk grafis. Pada Tugas Akhir ini, PyVis digunakan untuk menampilkan simulasi sederhana seperti menguji kekuatan transmisi, melihat perubahan selang waktu TC Message, dan melihat isi *routing table* dari suatu node.

BAB III
PERANCANGAN

Pada bab ini akan dijelaskan secara khusus mengenai perancangan sistem yang dibuat pada Tugas Akhir ini. Perancangan terdiri dari deskripsi umum, perancangan protokol, perancangan skenario, perancangan simulasi, dan perancangan metrik analisis untuk hasil simulasi.

3.1 Deskripsi Umum

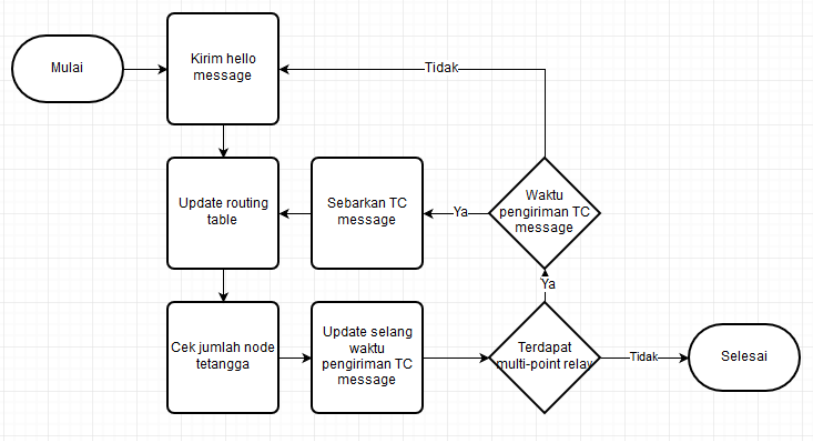
Pada Tugas Akhir ini, protokol yang akan digunakan adalah Optimized Link State Routing Protocol (OLSR) dan modifikasinya. Modifikasi yang dilakukan pada Tugas Akhir ini adalah dengan merubah selang waktu pengiriman *Topology Control Message* (TC Message) dari OLSR yang semula statis menjadi dinamis dan adaptif terhadap jumlah *node* tetangga (*1-hop nodes*) di sekitarnya.

Pada proses pengujian, terdapat dua jenis skenario yang digunakan sebagai pengukur yaitu, skenario peta grid dan skenario peta riil. Peta skenario dibuat dengan SUMO, OpenStreetMap, dan JOSM. Simulasi dari skenario menggunakan NS-3. Hasil *tracing* dari simulasi yang dilakukan di NS-3 dianalisis dengan menggunakan skrip AWK untuk *packet delivery* ratio (PDR) dan *routing overhead* (RO) dari protokol. Dari hasil analisis tersebut, dapat disimpulkan kualitas protokol yang diajukan dibandingkan dengan protokol OLSR sebelum dimodifikasi.

3.2 Perancangan Protokol

Pada Tugas Akhir ini protokol yang digunakan adalah OLSR dan OLSR dengan modifikasi pada selang waktu pengiriman TC Message. Selang waktu yang digunakan bersifat adaptif terhadap banyak *node* tetangga (*1-hop nodes*) yang ada disekitarnya. Pada buku Tugas Akhir ini protokol yang dimodifikasi akan disebut OLSR adaptif. Tujuan dari perubahan yang dilakukan pada protokol ini adalah untuk mengurangi *routing overhead* yang merupakan masalah utama dari protokol proaktif, dengan tetap mempertahankan

packet delivery ratio dari protokol tersebut. Gambaran umum alur dari protokol yang dapat dilihat pada Gambar 3-1.



Gambar 3-1. Gambaran umum alur routing protocol dengan modifikasi pada selang waktu TC Message

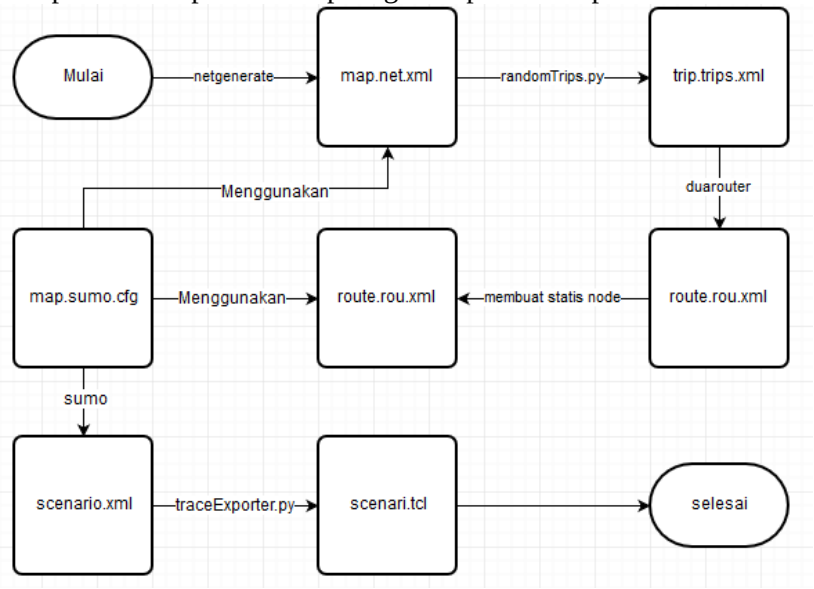
3.3 Perancangan Skenario

Perancangan skenario uji coba protokol dimlai dari perancangan mobilitas, yang meliputi peta jalan raya, pergerakan *node*, pembuatan rute lalu lintas, dan implementasi pergerakan. Pada Tugas Akhir ini, perancangan peta jalan raya dibagi menjadi dua jenis yaitu, peta grid dan peta riil. Peta grid adalah sebuah peta dengan jalan-jalan yang saling berpotongan membentuk petak yang menggambarkan kondisi riil secara sederhana. Peta grid digunakan sebagai tes awal pada protokol VANET karena peta grid seimbang dan stabil. Sedangkan peta riil adalah peta yang menggambarkan lingkungan jalan raya yang nyata. Pada Tugas Akhir ini bagian peta riil yang diambil adalah bagian peta Kota Surabaya.

3.3.1 Perancangan Peta Grid

Pembuatan peta grid diawali dengan menentukan panjang dan lebar dari peta grid yang digunakan. Setelah panjang dan lebar ditentukan, selanjutnya tentukan berapa banyak garis vertikal, horizontal, dan jumlah petak yang ingin digunakan. Pada Tugas

Akhir ini, peta grid yang digunakan berukuran 900 m x 900 m, dengan luas 0.81 km^2, jumlah garis vertikal dan horizontal sama yaitu sebanyak 7 (tujuh), dan jumlah petak yang dihasilkan adalah 6x6 petak. Alur pembuatan peta grid dapat dilihat pada Gambar 3-2.

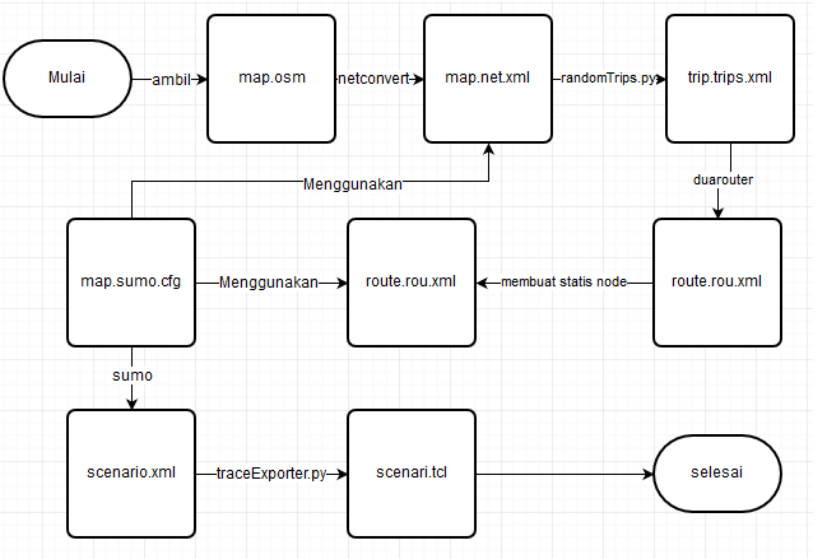


Gambar 3-2. Alur pembuatan peta grid

3.3.2 Perancangan Peta Riil

Pembuaatan peta riil diawali dengan menentukan daerah mana yang akan dijadikan sebagai peta rill. Daerah yang dipilih sebaiknya mendekati bentuk grid dengan ukuran yang tidak jauh berbeda dengan peta grid. Pembuatan peta riil diusahakan mendekati kondisi nyatanya, maka digunaka OpenStreetMap untuk melakukan *capture* peta. Agar simulasi berjalan dengan baik, hasil impor peta dari OpenStreetMap disempurnakan lagi dengan menggunakan kakas bantu JOSM. JOSM digunakan untuk penghapusan jalan yang tidak utuh ketika proses *capture* dan penambahan jalan yang hilang karena proses *capture*. Setelah proses edit selesai, peta riil dikonversi menjadi *file* SUMO berkstensi .net.xml dengan

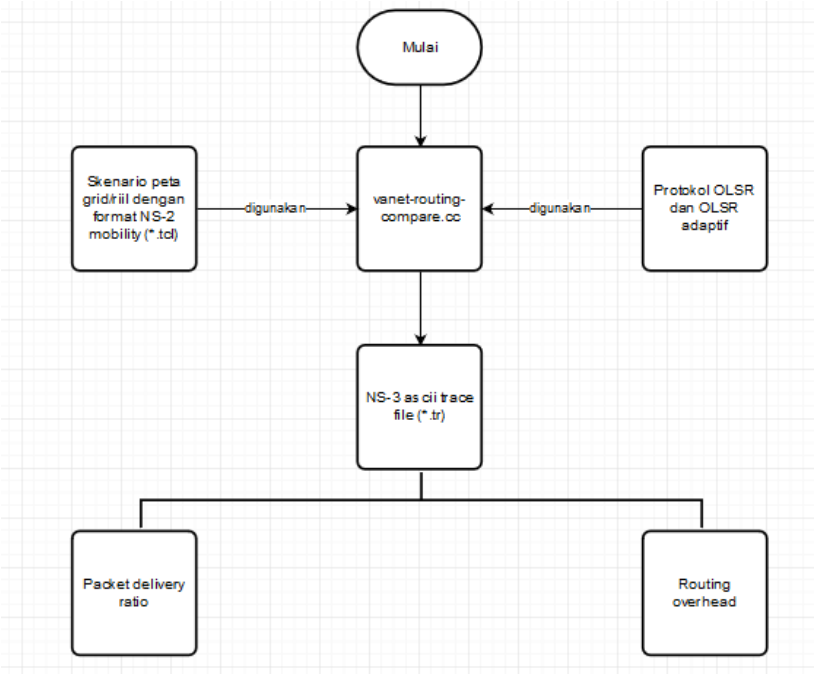
menggunakan kakas bantu netconver. Pada Tugas Akhir ini peta riil yang digunakan adalah peta Kota Surabaya bagian Timur. Alur pembuatan peta riil dapat dilihat pada Gambar 3-3.



Gambar 3-3. Alur pembuatan peta real

3.4 Perancangan Simulasi

Simulasi yang dilakukan pada Tugas Akhir ini menggunakan skenario mobilitas dan kode program vanet-routing-compare yang merupakan salah satu fitur yang disediakan oleh NS-3. Simulasi dilakukan dua kali untuk setiap skenario, satu dengan menggunakan OLSR dan yang lain menggunakan OLSR yang dimodifikasi. Sedangkan alur jalannya proses simulasi *routing protocol* yang bersifat adaptif dengan menggunakan *Optimized Link State Routing Protocol* yang telah dimodifikasi pada Tugas Akhir ini dapat dilihat pada Gambar 3-4.



Gambar 3-4. Alur simulasi

3.5 Perancangan Metrik Analisis

Metrik analisis digunakan sebagai sumber informasi apakah protokol yang diujikan mengalami kenaikan peforma dibandingkan dengan protokol pembandingnya. Metrik analisis diambil dari *trace file* hasil keluaran dari simulasi. Pada Tugas Akhir ini metric analisis yang akan digunakan adalah *packet delivery ratio*(PDR) dan *routing overhead*(RO) karena tujuan dari modifikasi yang dilakukan adalah mengurangi RO dari OLSR dengan tetap menjaga PDR nya.

3.5.1 Packet Delivery Ratio

Packet delivery ratio didapatkan dari perbandingan antara paket yang diterima dengan paket yang dikirim dari *node* asal menuju *node* tujuan. Semakin besar nilai PDR maka protokol semakin baik. Nilai PDR didapatkan dengan menggunakan

persamaan (3.1), yaitu jumlah paket yang diterima oleh *node* tujuan dibandingkan dengan jumlah paket yang dikirimkan oleh *node* asal.

$$\boldsymbol{PDR} = \frac{\boldsymbol{received}}{\boldsymbol{transmitted}} \times \mathbf{100}\% \quad \text{.....(3.1)}$$

3.5.2 Routing Overhead

Routing overhead adalah jumlah keseluruhan *routing packet* dalam jaringan yang digunakan dalam proses *routing*. RO menunjukkan kepadatan pengiriman paket data *routing* di dalam jaringan. Semakin sedikit nilai RO dari suatu protokol maka protokol tersebut semakin baik, karena protokol tersebut tidak memenuhi jaringan untuk proses *routing*. Nilai RO pada *routing protocol* OLSR dihitung dengan menggunakan persamaan (3,2), yaitu jumlah total Hello Message ditambah dengan jumlah total TC Message di dalam jaringan.

$$\boldsymbol{RO} = \sum \boldsymbol{Hello} + \sum \boldsymbol{TC} \quad \text{.....(3.2)}$$

BAB IV
IMPLEMENTASI

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

4.1 Lingkungan Pembangunan Perangkat Lunak

Pembangunan perangkat lunak dilaksanakan pada lingkungan pengembangan sebagai berikut:

4.1.1 Lingkungan Perangkat Lunak

Pengembangan sistem dilakukan pada lingkungan perangkat lunak sebagai berikut :

- Sistem Operasi berupa Linux 14.04
- SUMO versi 0.22.0 untuk mendesain skenario mobilitas VANET
- Network Simulator 3 versi 3.22 untuk simulasi skenario mobilitas VANET
- Firefox versi 44.0.1 untuk mengimpor peta riil dari OpenStreetMap
- JOSM versi 10145 untuk mengedit peta hasil impor dari OpenStreetMap

4.1.2 Lingkungan Perangkat Keras

Pengembangan sistem dilakukan pada lingkungan perangkat keras sebagai berikut :

- *Processor* Intel(R) Core(TM) i5 5200U CPU @ 2.20GHz
- RAM sebesar 4 GB DDR3
- Media Penyimpanan sebesar 125 GB
- Tipe sistem 64-bit

4.2 Implementasi Protokol

Pada Tugas Akhir ini *routing protocol* OLSR di modifikasi sesuai dengan yang telah dijabarkan pada bab sebelumnya. Berbeda dengan NS-2, NS-3 telah memiliki *routing protocol* OLSR yang telah terinstal sebelumnya. Kode sumber untuk *routing protocol* OLSR dapat ditemukan pada direkori ns-3.22/src/olsr/model. Pada Tugas Akhir ini, modifikasi hanya dilakukan pada skrip olsr-routing-protocol.cc. Skrip olsr-routing-protocol.cc memiliki beberapa atribut yang dapat diubah sesuai kebutuhan. Atribut yang umum untuk dimodifikasi antara lain:

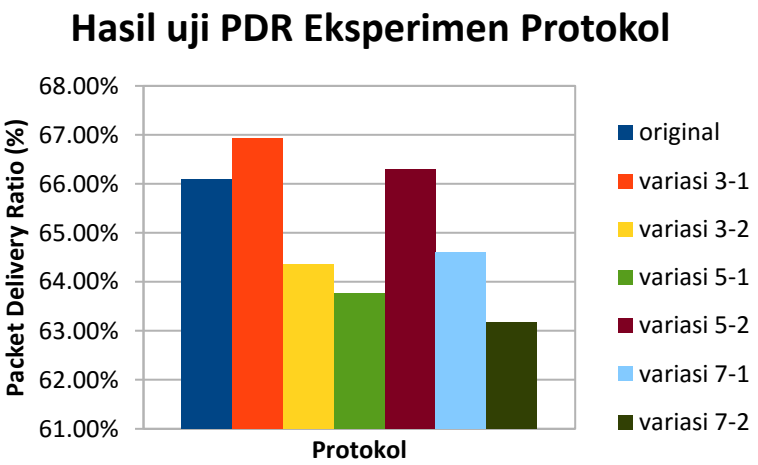
- HelloInterval : selang waktu pengiriman Hello Message
- TcInterval : selang waktu pengiriman TC Message.
- MidInterval : selang waktu pengiriman MID Message.
- HnaInterval : selang waktu pengiriman HNA Message.
- Willingness : ketersediaan *node* untuk membawa dan melanjutkan pesan.
- Parameter-parameter lain yang terdapat pada OLSR dapat diubah sesuai kebutuhan.

Pemilihan nilai dan banyaknya variasi perubahan *TC interval* ditentukan dengan menggunakan eksperimen. Protokol yang diujikan dalam eksperimen memiliki tiga jenis variasi perubahan *TC interval* yaitu 3 variasi, 5 variasi, dan 7 variasi. Masing-masing variasi meiliki dua jenis pertambahan selang waktu yaitu 1 detik dan 2 detik. Pembuatan variasi perubahan *TC interval* disesuaikan dengan jumlah maksimal dari *1-hop nodes* dari suatu node dalam uji coba, yaitu 30 *nodes*. Minimum *TC interval* yang digunakan adalah 3 detik, agar tidak lebih cepat dari pengiriman *hello message* (2 detik). Jika disamakan atau lebih cepat dari pengiriman *hello message*, informasi *TC message* akan sama dan menyebabkan kerugian pada *routing overhead* jaringan. Protokol yang diujikan dalam eksperimen ini dapat dilihat pada Tabel 4-1.

Tabel 4-1. Protokol yang digunakan dalam eksperimen

Protokol	**Variasi TC Interval**
Original	5 detik
3 Variasi Δt=1	*1-hop nodes*<10 = 4 detik *1-hop nodes*<20 = 5 detik *1-hop nodes*>=20 = 6 detik
3 Variasi Δt=2	*1-hop nodes*<10 = 3 detik *1-hop nodes*<20 = 5 detik *1-hop nodes*>=20 = 7 detik
5 Variasi Δt=1	*1-hop nodes*<6 = 3 detik *1-hop nodes*<12 = 4 detik *1-hop nodes*<18 = 5 detik *1-hop nodes*<24 = 6 detik *1-hop nodes*>=24 = 7 detik
5 Variasi Δt=2	*1-hop nodes*<6 = 3 detik *1-hop nodes*<12 = 5 detik *1-hop nodes*<18 = 7 detik *1-hop nodes*<24 = 9 detik *1-hop nodes*>=24 = 11 detik
7 Variasi Δt=1	*1-hop nodes*<4 = 3 detik *1-hop nodes*<8 = 4 detik *1-hop nodes*<12 = 5 detik *1-hop nodes*<16 = 6 detik *1-hop nodes*<20 = 7 detik *1-hop nodes*<24 = 8 detik *1-hop nodes*>=24 = 9 detik
7 Variasi Δt=2	*1-hop nodes*<4 = 3 detik *1-hop nodes*<8 = 5 detik *1-hop nodes*<12 = 7 detik *1-hop nodes*<16 = 9 detik *1-hop nodes*<20 = 11 detik *1-hop nodes*<24 = 13 detik *1-hop nodes*>=24 = 15 detik

Hasil dari eksperimen dapat dilihat pada Gambar 4-1.



Gambar 4-1. Hasil uji eksperimen dilihat dari nilai PDR

Berdasarkan hasil eksperimen, maka protokol dengan hasil PDR paling baik adalah protokol dengan 3 variasi *TC interval* dan $\Delta t = 1$ detik.

Modifikasi yang dilakukan pada Tugas Akhir ini berhubungan dengan TcInterval dan jumlah *node* tetangga (*1-hop nodes*) yang berada di sekitarnya. Semakin banyak *node* tetangga yang ada di sekitar suatu *node*, maka TcIntervalnya akan diperpanjang. Begi pula sebaliknya, jika *node* tetangga yang ada disekitar suatu *node* semakin sedikit, maka TcInterval akan dipercepat. Harapan dari modifikasi ini adalah mengurangi *routing overhead* yang ada dalam jaringan disebabkan oleh pegurangan selang waktu pengiriman *TC message* dan tetap menjaga nilai *packet delivery ratio* dari protokol OLSR. Modifikasi yang digunakan untuk mengetahui jumlah *node* tetangga (*1-hop nodes*) di sekitarnya dapat dilihat pada Gambar 4-2.

```
void
RoutingProtocol::MprComputation ()
{
  NS_LOG_FUNCTION (this);

  MprSet mprSet;

  NeighborSet N;
  int neighborsCount=0;
  int intervalm=0;
  for (NeighborSet::const_iterator neighbor =
m_state.GetNeighbors ().begin ();
       neighbor != m_state.GetNeighbors ().end ();
neighbor++)
    {
      if (neighbor->status ==
NeighborTuple::STATUS_SYM) // I think that we need
this check
        {
          N.push_back (*neighbor);
          neighborsCount++;
        }
    }
...
```

Gambar 4-2. Potongan kode yang dimodifikasi pada olsr-routing-protocol.cc untuk menghitung node tetangga

Setelah mengetahui jumlah *node* tetangga di sekitarnya, TcInterval dirubah sesuai dengan jumlah yang didapat. Modifikasi untuk mengubah TcInterval dapat dilihat pada Gambar 4-3 dan perhitunganya pada Gambar 4-4.

```
...
intervalm = modification(neighborsCount);
m_tcInterval = Seconds (intervalm);
neighborsCount=0;
...
```

Gambar 4-3. Potongan kode yang ditambahkan pada olsr-routing-protocol.cc untuk mengubah TcInterval secara dinamis

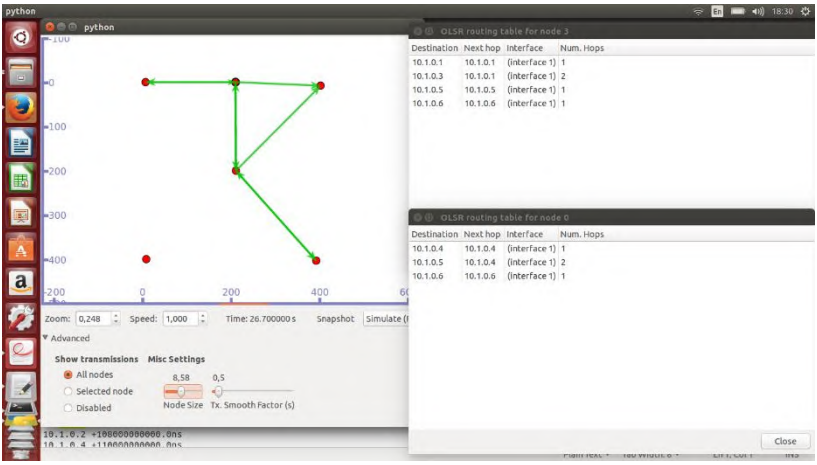
```
...
int modification (int x)
{
 int modIntv;
 if (tempx<10)
        {
         modIntv = 4;
        }
 else if (tempx<20)
        {
         modIntv = 5;
        }
 else if (tempx<30)
        {
         modIntv = 6;
        }
 else if (tempx>=30)
        {
         modIntv = 6;
        }
 return modIntv; };
}
...
```

Gambar 4-4. Potongan kode yang ditambahkan pada olsr-routing-protocol.cc untuk mengatur perubahan TcInterval

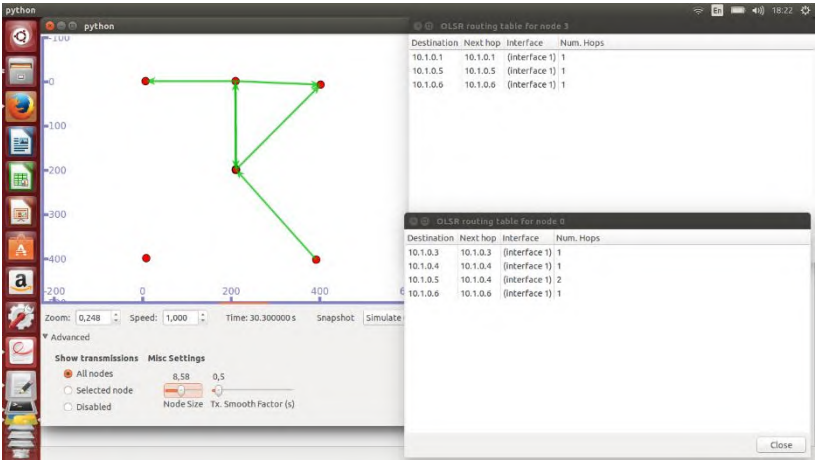
Dengan modifikasi yang telah dilakukan, dapat membuat TcInterval yang bervariasi bergantung pada jumlah *node* tetangga (*1-hop nodes*) di sekitarnya. Hal ini dibuktikan dengan percobaan skenario sederhana dengan dua MPR. Pada percobaan ini TcInterval di ubah menjadi 3 detik untuk 0,1,2 *node* tetangga, 5 detik untuk 3 *node* tetangga, dan 7 detik untuk *node* tetangga lebih dari 3. Gambar 4-5, 4-6, dan 4-7 menunjukan perbuahan jumlah *node* tetangga, sedangkan Gambar 4-8 menunjukan waktu pengiriman TC Message dalam simulasi ini.

Pada Gambar 4-5, t=26s, jumlah *node* tetangga untuk *node* 0 adalah 2 *nodes*. Sedangkan pada Gambar 4-6, t=30s, jumlah *node* tetangga dari *node* 0 adalah 3 *nodes*, sehingga seharusnya selang waktu TcInterval menjadi 5 detik, tetapi pada Gambar 4-7, dengan

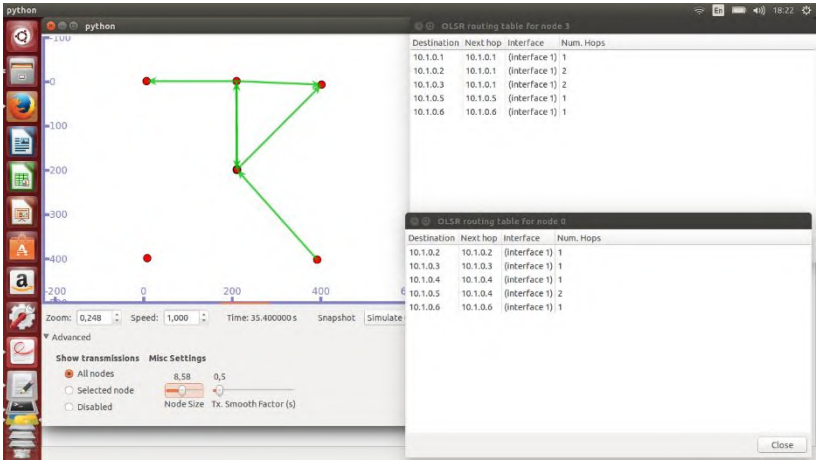
t=35s, jumlah *node* tetangga dari *node* 0 berubah menjadi 4 *nodes*, sehingga TcInterval berubah menjadi 7 detik.



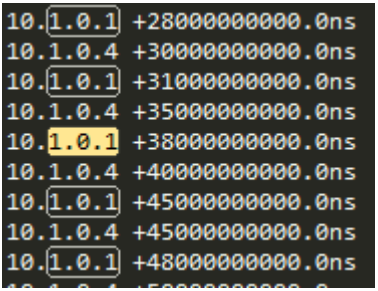
Gambar 4-5. Routing table skenario percobaan protokol pada detik ke-26



Gambar 4-6. Routing table skenario percobaan protokol pada detik ke-30



Gambar 4-7. Routing table skenario percobaan protokol pada detik ke-35



Gambar 4-8. Waktu pengiriman TC Message

4.3 Implementasi Skenario

Implementasi skenario mobilitas VANET dibagi menjadi dua yaitu, skenario grid dan skenario riil. Masing-masing skenario dibuat dan diuji sebanyak 10 kali dengan mobilitas berbeda. *Node* asal dan tujuan dibuat statis untuk melihat realibilitas dalam pengiriman data pada *multi-hop nodes*. *Node* asal adalah *node* 1

berada di pojok kanan atas dan *node* tujuan adalah *node* 0 berada di pojok kiri bawah.

4.3.1 Implentasi Skenario Grid

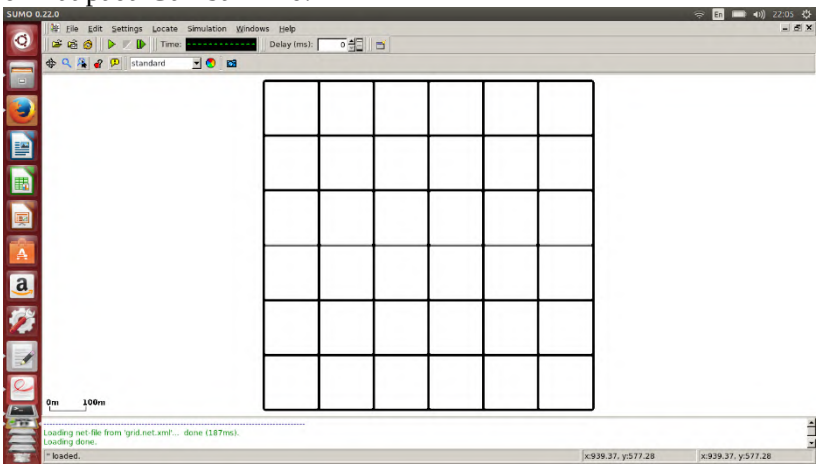
Implementasi skenario grid menggunakan fungsi netgenerate yang ada pada SUMO. Pada Tugas Akhir ini peta yang dibuat dengan luas 900 m x 900m dan jumlah petak 6x6 petak. Titik persimpangan antara jalan vertikal dan horizontal sebanyak 7 titik x 7 titik dan luasan per petak 150 m x 150 m. Untuk kecepatan kendaraan pada peta grid ditentuka 20 m/s.

Melalui terminal di Linux dengan direktori tempat file akan dibuat, dilakukan perintah seperti pada Gambar 4-9.

```
netgenerate --grid --grid.number=7 --grid.length=150
--default.speed=20 --output-file=map.net.xml
```

Gambar 4-9. Perintah untuk membuat peta grid dengan netgenerate pada SUMO

Gambar hasil peta yang dibuat dengan netgenerate bisa dilihat pada Gambar 4-10.



Gambar 4-10. Peta grid hasil netgenerate

Setelah peta terbentuk, dilakukan pembuatan *node* beserta asal dan tujuan dari tiap *node* menggunakan fungsi randomTrips.py seperti pada Gambar 4-10. Parameter penting perintah pada Gambar 4-11 yaitu -n untuk nama *file* peta, -e untuk jumlah *node*, dan departSpeed untuk kecepatan.

```
python ~/TA/sumo-0.22.0/tools/trip/randomTrips.py -n
map.net.xml -e 50 -l --trip-
attributes="departLane=\"best\" departSpeed=\"max\"
departPos=\"random_free\"" -o trip.trips.xml
```

Gambar 4-11. Perintah untuk membuat node dengan asal dan tujuanya dengan randomTrips.py pada SUMO

File output trip.trips.xml yang berisi asal dan tujuan dari semua *node* kemudian dibuatkan rute dengan menggunakan fungsi duarouter dari SUMO. Rute dibuat dengan algorima djikstra secara default. Contoh perintahnya dapat dilihat pada Gambar 4-12.

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

Gambar 4-12. Perintah untuk membuat rute pergerakan node dengan duarouter pada SUMO

Langkah selanjutnya adalah membuat *node* asal dan tujuan menjadi statis. Hal ini diperlukan untuk menguji apakah protokol dapat berjalan dengan baik dalam jaringan dengan *multi-hop node*. Pada Tugas Akhir ini, *node* yang dijadikan statis adalah *node* 1 (*node* asal) dan 0 (*node* tujuan). Bagian yang perlu dirubah adalah *edges* pada “<route>” dan menambahkan atribut “<stop>” untuk *node* 1 dan 0 di *file* .rou.xml. Contoh modifikasi dapat dilihat pada Gambar 4-13.

Setelah peta, *node*, dan rute terbuat, selanjutnya digabungkan dengan fungsi sumo atau sumo-gui untuk melihat proses simulasinya. Dibutuhkan *file* konfigurasi untuk menggunakan sumo atau sumo-gui. Contoh *file* konfigurasi dapat dilihat pada Gambar 4-14.

```
<vehicle id="0" depart="0.00" departSpeed="max">
        <route edges="0/0to1/0"/>
        <stop lane="0/0to1/0_0" endPos="6"
duration="200"/>
    </vehicle>
    <vehicle id="1" depart="0.00" departSpeed="max">
        <route edges="6/6to6/5"/>
        <stop lane="6/6to6/5_0" endPos="6"
duration="200"/>
    </vehicle>
```

Gambar 4-13. Contoh modifikasi pada file route.rou.xml untuk membuat node asal dan tujuan menjadi statis

```
<configuration
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://sumo.sf.net/xsd
/sumoConfiguration.xsd">

       <input>
              <net-file value="map.net.xml"/>
              <route-files value="route.rou.xml"/>
       </input>

       <time>
              <begin value="0"/>
              <end value="200"/>
       </time>

</configuration>
```

Gambar 4-14. Contoh isi file .sumo.cfg

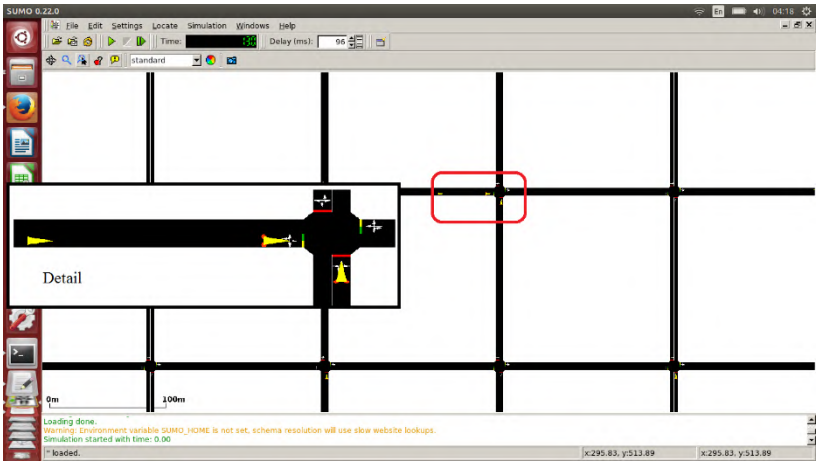
File sumo.cfg disimpan pada director yang sama dengan *file* .net.xml dan .rou.xml. Perintah menggunakan sumo atau sumo-gui untuk menghasilkan file skenario dengan ekstensi .xml dapat dilihat pada Gambar 4-15.

```
sumo-gui -c map.sumo.cfg --fcd-output scenario.xml
```

Gambar 4-15. Perintah untuk membuat file skenario berformat .xml pada SUMO

Contoh pergerakan skenario mobilitas dengan sumo-gui dapat dilihat pada Gambar 4-16.

File skenario berekstensi .xml kemudian dikonversi menjadi *file* mobilitas dalam format NS-2. Format NS-2 dapat digunakan untuk simlasi pada NS-3. Modul yang digunakan untuk melakukan konversi adalah traceExporter.py. Contoh perintahnya dapat dilihat pada Gambar 4-17.



Gambar 4-16. Contoh mobilitas pada peta grid dengan sumo-gui

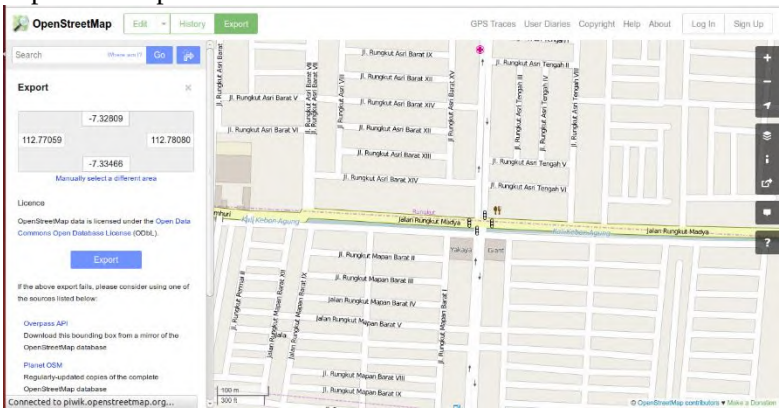
```
python ~/TA/sumo-0.22.0/tools/traceExporter.py --fcd-
input=scenario.xml --ns2mobility-output=scenario.tcl
```

Gambar 4-17. Perintah untuk mengkonversi file skenario xml SUMO menjadi format mobilitas NS-2

4.3.2 Implentasi Skenario Riil

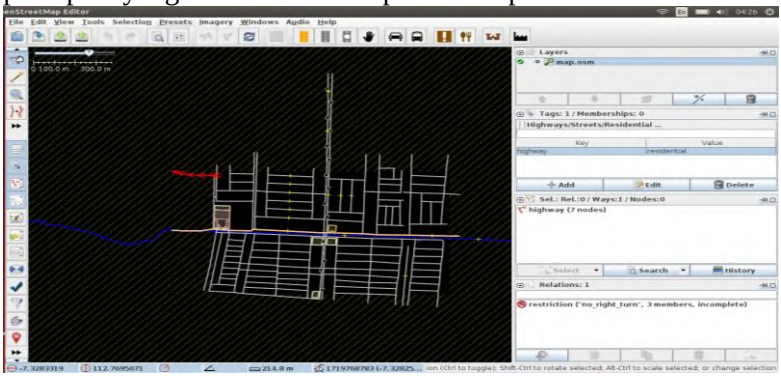
Skenario riil pada Tugas Akhir ini menggunakan peta daerah Kota Surabaya. Peta diambil dari OpenStreetMap dengan cara menandai wilayah yang ingin diambil, kemudian melakukan ekspor. Secara otomatis OpenStreetMap akan memberikan *file* dengan ekstensi .osm untuk diunduh. Pada Tugas Akhir ini, ukuran dari peta

riil yang digunakan sebesar ±1200 m x 900 m, dengan batas kecepatan sama dengan kondisi aslinya. Proses pengambilan peta dapat dilihat pada Gambar 4-18.



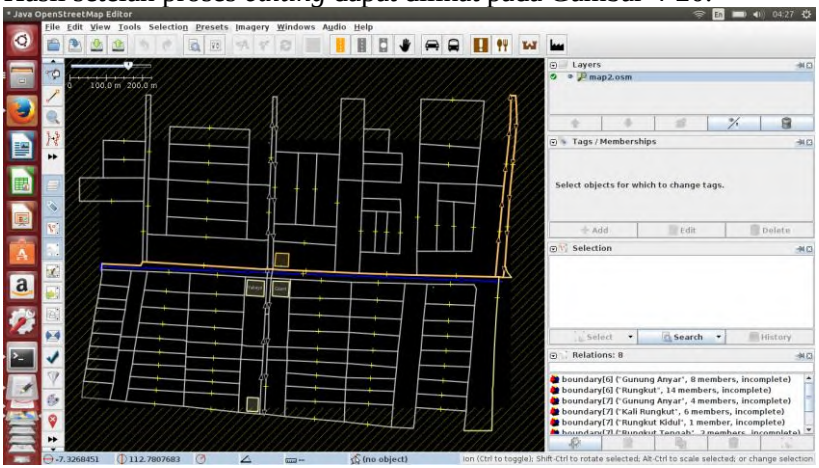
Gambar 4-18. Proses impor peta riil dari OpenStreetMap [9]

Peta yang telah diunduh kemudian diedit dengan menggunakan JOSM. Proses edit digunakan untuk membuat jalan tambahan yang menghubungkan jalan yang terputus dan menghapus jalan yang terlalu jauh dari luas yang diinginkan. Kecepatan pada peta riil telah disesuaikan dengan keadaan nyata. Proses *editting* pada peta yang telah diunduh dapat dilihat pada Gambar 4-19.



Gambar 4-19. Proses mengedit peta riil dari OpenStretMap dengan JOSM

Hasil setelah proses *editing* dapat dilihat pada Gambar 4-20.



Gambar 4-20. Peta riil setelah diedit dengan JOSM

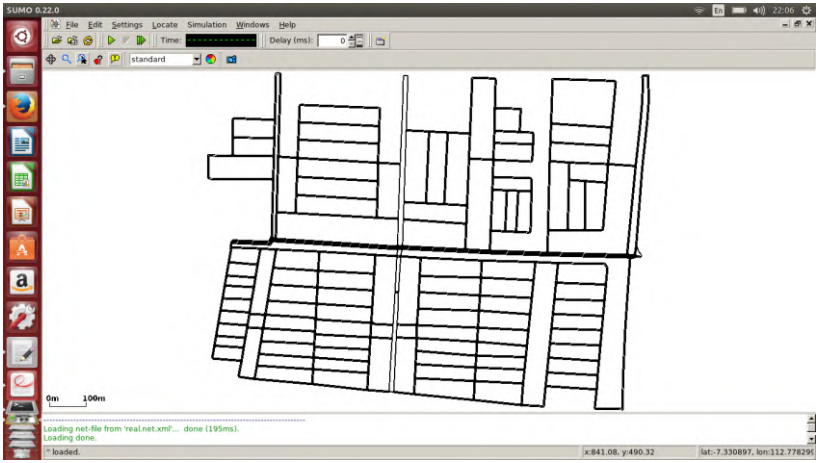
Setelah peta selasai melalui proses *editing*, *file* .osm dikonversi menjadi format SUMO dengan ekstensi .net.xml menggunakan fungsi netconvert. Perintahnya dapat dilihat pada Gambar 4-21.

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

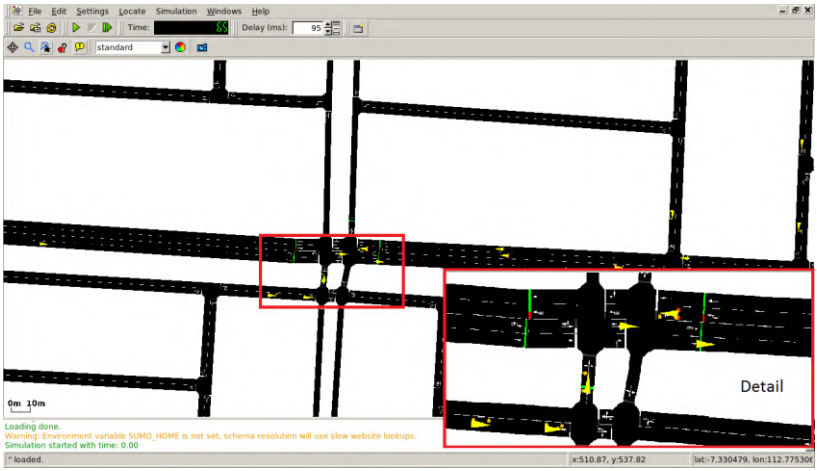
Gambar 4-21. Perintah untuk mengkonversi file peta .osm OpenStreetMap menjadi format xml SUMO

Hasil konversi *file* peta ke dalam format .net.xml dapat dilihat pada Gambar 4-22.

Setelah peta terbentuk, maka langkah selanjutnya sama dengan skenario grid sampai dikonversi menjadi *file* berformat NS-2, yaitu *file* dengan ekstensi .tcl. Contoh pergerakan mobilitas pada peta riil dengan sumo-gui dapat dilihat pada Gambar 4-23.



Gambar 4-22. Peta hasil konversi file .osm OpenStreetMap ke dalam format peta .net.xml SUMO



Gambar 4-23. Contoh mobilitas pada peta riil dengan sumo-gui

4.4 Implementasi Simulasi pada NS-3

Tugas Akhir ini menggunakan program vanet-routing-compare.cc yang telah disediakan oleh NS-3 sebagai sarana simulasi pada lingkungan VANET. Setelah skenario dibuat pada subbab diatas, maka selanjutnya adalah proses *running* skenario dengan menggunakan vanet-routing-compare pada NS-3.

Proses *running* skenario menggunakan parameter vanet-routing-compare sepert pada Tabel 2-1. Parameter utama yang dipakai pada Tugas Akhir ini adalah:

- traceFile : berfungsi untuk menunjukan file skenario dengan ekstensi .tcl yang akan digunakan untuk simlasi.
- asciiTrace : berfungsi untuk mengaktifkan asci *trace file* sehingga menghasilkan *trace file* dengan ekstensi .tr yang akan digunakan untuk analisis. Bertipe Boolean dengan nilai 1.
- routingTables : berfungsi untuk mencetak *ip address* tiap *node*
- nSinks : dibuat 1 agar *node* pengirim dan penerima ada 1 (*node* 1 dan 0)
- Txp : dibuat 5, untuk mengatur kekuatan transmisi dengan kekuatan 5 dBm (sekitar 290 m)
- TotalSimTime : berfungsi untuk menentukan lama simulasi berjalan. Nilainya ditentukan 200 detik.
- Node : dibuat 50, untuk 50 *node* dalam simulasi.
- trName : hasil output *trace file* dengan ekstensi .tr
- protocol : *routing protocol* yang dipilih untuk simulasi. Bernilai 1 untuk OLSR

Potongan kode penting terdapat pada Lampiran 3, Lampiran 4, dan Lampran 5.

Gambar 4-24 adalah contoh potongan kode untuk satu skenario.

```
else if (m_scenario == 61)
    {
      m_traceFile = "scratch/grid1.tcl";
      m_trName = "grid1";
      m_mobility = 1;
      m_nNodes = 50;
      m_TotalSimTime = 200;
      m_nSinks = 1;
      m_asciiTrace = 1;
      m_txp = 5;
      m_protocol = 1;
    }
```

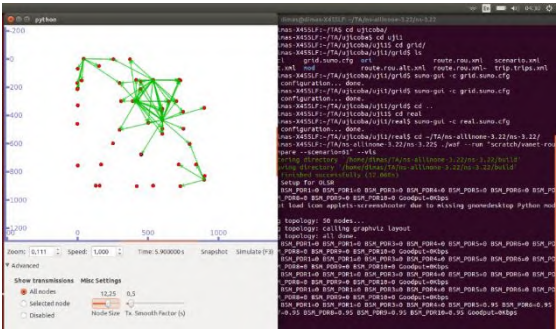
Gambar 4-24. Contoh kode untuk satu skenario

Perintah untuk menjalankan simulasi dapat dilihat pada Gambar 4-25. Parameter tambahan bisa ditambahkan, seperti --scenario untuk memilih skenario mana yang akan disimulasikan atau --vis untuk menampilkan simulasi secara grafikal.

```
./waf --run "scratch/vanet-routing-compare --
scenario=61" --vis
```

Gambar 4-25. Perintah untuk menjalankan satu skenario

Pada PyVis dapat diatur besar *node*, kecepatan simulasi, garis yang menunjukan pengiriman paket data, *routing table*, dan paket data yang terakhir dari sebuah *node*. Contoh cuplikan saat menjalankan simulasi dengan menggunakan PyVis dapat dilihat pada Gambar 4.26.



Gambar 4-26. Contoh simulasi yang dianimaskan dengan PyVis

4.5 Implementasi Metrik Analisis

Hasil dari simulasi skenario dengan NS-3 yang digunakan pada Tugas Akhir ini adalah asci *trace file*, yang berbasis teks. Analisa yang dilakukan ada dua, yaitu *packet delivery ratio* dan *routing overhead* dari *trace file* yang dihasilkan tiap skenario.

4.5.1 Implementasi Packet Delivery Ratio

Implementasi perhitungan PDR diawali dengan menghitung jumlah paket data dari *node* asal. Paket data yang digunakan berukuran 64 byte, sesuai dengan konfigurasi pada vanet-routing-compare. Gambar 4-27 merupakan *pseudocode* untuk perhitungan PDR.

```
ALGORITMA PDR
//Input : trace file .tr output simulasi skenario
//Output: jumlah paket terkirim, paket diterima, dan
//        PDR
sent ← 0
receive ← 0
receive_id ← 0
pdr ← 0

if ($1 == "t" dan $3 ==
"/NodeList/1/DeviceList/0/$ns3::WifiNetDevice/Phy/Sta
te/Tx" dan $9 == "Retry=0" dan $49 == "(size=64)")
sent + 1
if ($1 == "r" dan $3 ==
"/NodeList/0/DeviceList/0/$ns3::WifiNetDevice/Phy/Sta
te/Tx" dan $49 == "(size=64)" dan receive_id != &29)
receive + 1
receive_id ← $29

pdr ← (receive/sent)*100
print sent
print receive
print pdr
```

Gambar 4-27. Pseudocode untuk analisis PDR

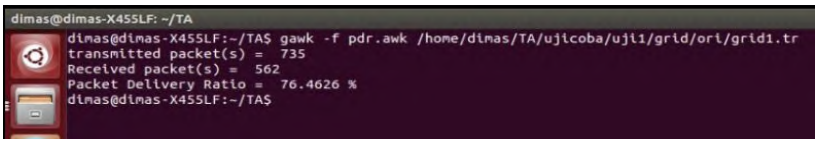
Pengiriman atau penerimaan paket data ditandai pada kolom 1 tiap baris. Nilai dari kolom 1 adalah “t” untuk pengiriman dan “r” untuk penerimaan paket data. Pemeriksaan apakah *node* tersebut *node* asal/tujuan dapat dilihat pada kolom 3 tiap baris. Angka setelah “/NodeList/*” merupakan id dari *node* yang mengirim/menerima paket data. Ukuran paket data dapat dilihat pada kolom 49, yaitu “(size=64)”.

Pengiriman paket data ditambahkan sebuah parameter lain yaitu kolom 9 yang berisi “Retry=0”. Hal ini untuk mencegah perhitungan paket data yang dikirim lebih dari satu kali untuk paket data yang sama. Sementara pada penerimaan paket data, ditambahkan parameter pada kolom 29 yaitu, id dari paket data. Paket data yang memiliki id yang sama tidak dihitung lagi. Setelah diketahui jumlah pengiriman dan penerimaan paket data pada *node* asal dan tujuan, selanjutnya adalah perkalian dengan 100%, sesuai dengan persamaan (3.1). Contoh perintah untuk analisis PDR dari sebuah *trace file* dapat dilihat pada Gambar 4-28.

```
gawk -f pdr.awk
/home/dimas/TA/ujicoba/uji1/grid/ori/grid1.tr
```

Gambar 4-28. Perintah untuk menganalisa PDR satu trace file

Contoh hasil dari perintah pada Gambar 4-27 dapat dilihat pada Gambar 4-29.



Gambar 4-29. Hasil analisa PDR dari satu trace file

4.5.2 Implementasi Routing Overhead

Implementasi RO diawali dengan perhitungan jumlah paket *routing* yang ada dalam jaringan. Hal ini ditandakan dengan “ns3::olsr::PacketHeader” pada tiap baris paket data yang dikirim,

yaitu baris pada trace file yang diawali oleh huruf “t”. Jika tidak ditemukan “ns3::olsr:PacketHeader” pada semua kolom di suatu baris, maka paket data tersebut tidak termasuk paket *routing*. Gambar 4-30 merupakan *pseudocode* untuk menghitung RO.

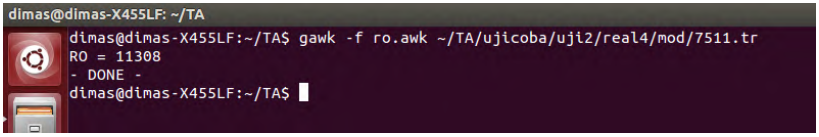
```
ALGORITMA RO
//Input : trace file .tr output simulasi skenario
//Output: Routing overhead

routing_packet ← 0

if ($1 == “t” dan terdapat “ns3::olsr::PacketHeader”)
routing_packet + 1

if (routing_packet > 0)
print routing_packet
```

Gambar 4-30. Pseudocode untuk analisis PDR

Contoh perintah untuk menganalisa RO dari sebuah *trace file* dapat dilihat pada Gambar 4-31, sedangkan *outputnya* dapat dilihat pada Gambar 4-32.

```
gawk -f ro.awk
/home/dimas/TA/ujicoba/uji1/grid/ori/grid1.tr
```

Gambar 4-31. Perintah untuk menganalisa RO satu trace file



Gambar 4-32. Hasil analisa RO dari satu trace file

BAB V
PENGUJIAN 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 yang meliputi hasil uji coba pada peta grid dan hasil uji coba pada peta riil.

5.1 Lingkungan Uji Coba

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

1. Perangkat keras
    a. *Processor* Intel(R) Core(TM) i5 5200U CPU @ 2.20GHz
    b. RAM sebesar 4 GB DDR3
    c. Media Penyimpanan sebesar 125 GB
    d. Tipe sistem 64-bit
2. Perangkat lunak
    a. Sistem Operasi berupa Linux 14.04
    b. Network Simulator 3 versi 3.22 untuk simulasi skenario mobilitas VANET
    c. GNU AWK versi 4.0.1 untuk analisa *trace file*

5.2 Skenario Uji Coba

Uji coba dilakukan dengan menjalankan simulasi dari skenario yang telah dibuat, baik grid maupun riil dengan menggunakan OLSR dan OLSR yang telah dimodifikasi dan perubahan pada jumlah kendaraan yang di uji coba. *Output* dari uji coba adalah sebuah NS-3 ascii *trace file* dengan ekstensi .tr untuk tiap simulasinya. Hasil uji coba didapatkan dengan mengolah *file* .tr menggunakan skrip AWK. Uji coba dilakukan tiga kali untuk peta

grid dan peta riil, dengan jumlah kendaraan 25, 50, dan 75 kendaraan. Masing-masing uji coba diuji sebanyak 10 kali menggunakan OLSR dan 10 kali menggunakan OLSR yang telah dimodifikasi. Parameter uji coba dapat dilihat pada Tabel 5-1.

Tabel 5-1. Parameter uji coba

No.	Parameter	Spesifikasi
1	Simulator	NS-3 versi 3.22
2	Routing protocol	OLSR OLSR modifikasi TcInterval *1-hop nodes*<10 = 4 detik *1-hop nodes*<20 = 5 detik *1-hop nodes*>=20 = 6 detik
3	Waktu simulasi	200
4	Jumlah *node*	25, 50, dan 75 kendaraan
5	Luas area	Grid : 900 m x 900 m Riill : ±1200 m x ±1000 m
6	Radius Transmisi	±290 m
7	Kecepatan Maksimal	Grid : 20 m/s Riil : 28 m/s
8	*Node* asal dan tujuan	*Node* 1 dan *node* 0 (statis)
9	Tipe data	Constant Bit Rate (CBR)
10	Kecepatan generasi paket data	1 paket per detik
11	Ukuran paket data	64 byte
12	Protocol MAC	IEEE 802.11p
13	Mode propagasi	Two-ray ground reflection model
14	Tipe mobilitas	NS-2
15	Model mobilitas	Peta grid dan peta riil
16	Tipe Kanal	Wireless channel

5.3 Hasil Uji Coba

Hasil uji coba pada skenario peta grid dan skenario peta riil dapat dilihat sebagai berikut:

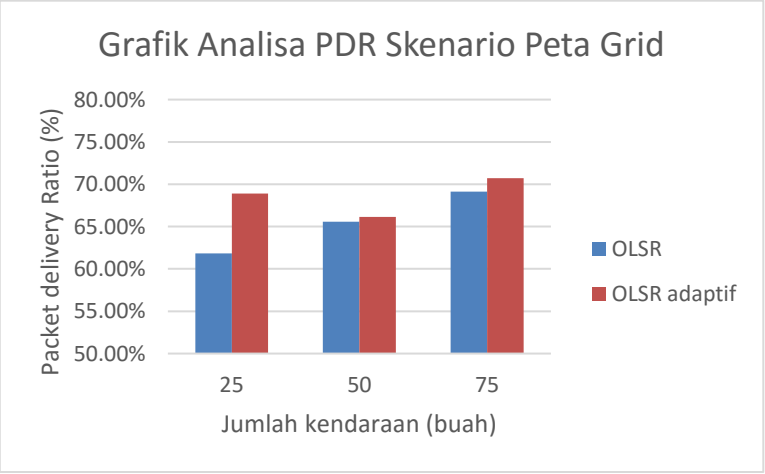
5.3.1 Hasil Uji Coba Grid

Hasil pengujian PDR pada uji coba skenario peta grid dapat dilihat pada Tabel 5-2.

Tabel 5-2. Hasil uji PDR skenario peta grid

Jumlah kendaraan	OLSR	OLSR adaptif
25 kendaraan	61.84%	68.89%
50 kendaraan	65.57%	66.13%
75 kendaraan	69.13%	70.71%

Dari data diatas dapat dibuat grafik yang menggambarkan hasil uji coba. Grafik PDR dapat dilihat pada Gambar 5-1.



Gambar 5-1. Grafik analisa PDR untuk skenario peta grid

Berdasarkan hasil uji coba skenario peta grid yang telah dilakukan, uji coba dengan menggunakan 25 kendaraan dan *routing protocol* OLSR memiliki rata-rata nilai PDR sebesar 61.84%. Uji coba dengan 25 kendaraan dan OLSR yang adaptif memiliki rata-rata nilai PDR sebesar 68.89%. Hasil uji coba memperlihatkan bahwa OLSR adaptif mengalami peningkatan peforma dilihat dari nilai PDR-nya dibandingkan dengan *routing protocol* OLSR untuk uji coba skenario peta grid dengan 25 kendaraan. Peningkatan rata-rata nilai PDR pada uji coba peta skenario grid dengan 25 kendaraan untuk OLSR adaptif adalah 7.05%.

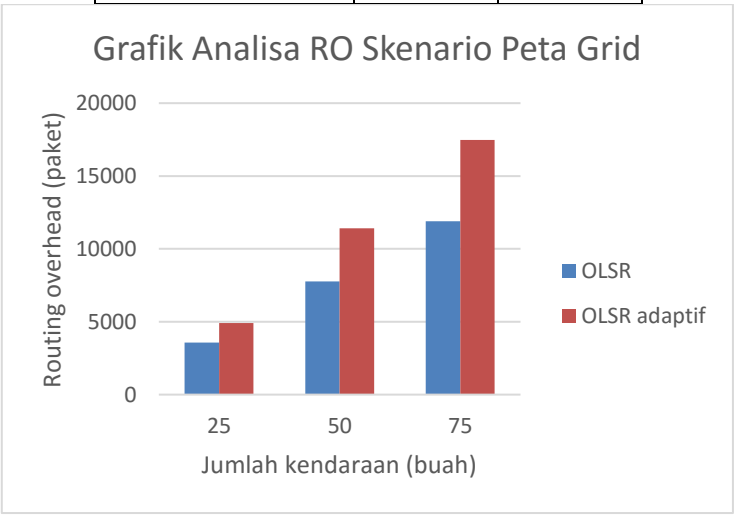
Uji coba dengan menggunakan 50 kendaraan dan *routing protocol* OLSR memiliki rata-rata nilai PDR sebesar 65.57%. Uji coba dengan 50 kendaraan dan *routing protocol* OLSR adaptif memiliki rata-rata nilai PDR sebesar 66.13%. Hasil uji coba memperlihatkan bahwa OLSR adaptif mengalami kenaikan peforma dilihat dari nilai PDR-nya dibandingkan dengan *routing protocol* OLSR pada uji coba skenario peta grid dengan 50 kendaraan. Peningkatan rata-rata nilai PDR pada uji coba peta skenario grid dengan 50 kendaraan untuk OLSR adaptif adalah 0.56%. Peningkatan nilai PDR pada uji coba skenario peta grid dengan 50 kendaraan untuk *routing protocol* OLSR adaptif mengalami penurunan dibandingkan dengan peningkatan nilai PDR pada uji coba skenario peta grid dengan 25 kendaraan.

Uji coba dengan menggunakan 75 kendaraan dan *routing protocol* OLSR memiliki rata-rata nilai PDR sebesar 69.13%. Uji coba dengan 75 kendaraan dan *routing protocol* OLSR adaptif memiliki rata-rata nilai PDR sebesar 70.71%. Hasil uji coba memperlihatkan bahwa OLSR adaptif mengalami kenaikan peforma dilihat dari nilai PDR-nya dibandingkan dengan *routing protocol* OLSR pada uji coba skenario peta grid dengan 75 kendaraan. Peningkatan rata-rata nilai PDR pada uji coba peta skenario grid dengan 75 kendaraan untuk OLSR adaptif adalah 1.57%. Peningkatan nilai PDR pada uji coba skenario peta grid dengan 75 kendaraan untuk *routing protocol* OLSR adaptif mengalami penurunan dibandingkan dengan peningkatan nilai PDR pada uji

coba skenario peta grid dengan 25 dan mengalami kenaikan dibandingkan dengan uji coba skenario peta grid dengan 50 kendaraan. Sedangkan hasil pengujian RO pada uji coba skenario peta grid dapat dilihat pada Tabel 5-3 dan digambarkan dengan grafik pada Gambar 5-2.

Tabel 5-3. Hasil uji RO skenario peta grid

Jumlah kendaraan	OLSR	OLSR adaptif
25 kendaraan	3565.4	4911.2
50 kendaraan	7757	11421.2
75 kendaraan	11886.8	17477.9



Gambar 5-2. Grafik analisa RO untuk skenario peta grid

Berdasarkan hasil uji coba skenario peta grid yang telah dilakukan, uji coba dengan menggunakan 25 kendaraan dan *routing protocol* OLSR memiliki rata-rata nilai RO sebesar 3565.4 paket. Uji coba dengan 25 kendaraan dan OLSR yang adaptif memiliki rata-rata nilai RO sebesar 4911.2 paket. Hasil uji coba

memperlihatkan bahwa OLSR adaptif mengalami penurunan peforma dilihat dari nilai RO-nya dibandingkan dengan *routing protocol* OLSR untuk uji coba skenario peta grid dengan 25 kendaraan. Rata-rata kenaikan nilai RO pada uji coba peta skenario grid dengan 25 kendaraan untuk OLSR adaptif adalah 1345.8 paket.

Uji coba dengan menggunakan 50 kendaraan dan *routing protocol* OLSR memiliki rata-rata nilai RO sebesar 7757 paket. Uji coba dengan 50 kendaraan dan OLSR yang adaptif memiliki rata-rata nilai RO sebesar 11421.2 paket. Hasil uji coba memperlihatkan bahwa OLSR adaptif mengalami penurunan peforma dilihat dari nilai RO-nya dibandingkan dengan *routing protocol* OLSR untuk uji coba skenario peta grid dengan 50 kendaraan. Rata-rata kenaikan nilai RO pada uji coba peta skenario grid dengan 50 kendaraan untuk OLSR adaptif adalah 3664.2 paket.

Uji coba dengan menggunakan 75 kendaraan dan *routing protocol* OLSR memiliki rata-rata nilai RO sebesar 11886.8. Uji coba dengan 75 kendaraan dan OLSR yang adaptif memiliki rata-rata nilai RO sebesar 17477.9 paket. Hasil uji coba memperlihatkan bahwa OLSR adaptif mengalami penurunan peforma dilihat dari nilai RO-nya dibandingkan dengan *routing protocol* OLSR untuk uji coba skenario peta grid dengan 75 kendaraan. Rata-rata kenaikan nilai RO pada uji coba peta skenario grid dengan 75 kendaraan untuk OLSR adaptif adalah 5591.1 paket.

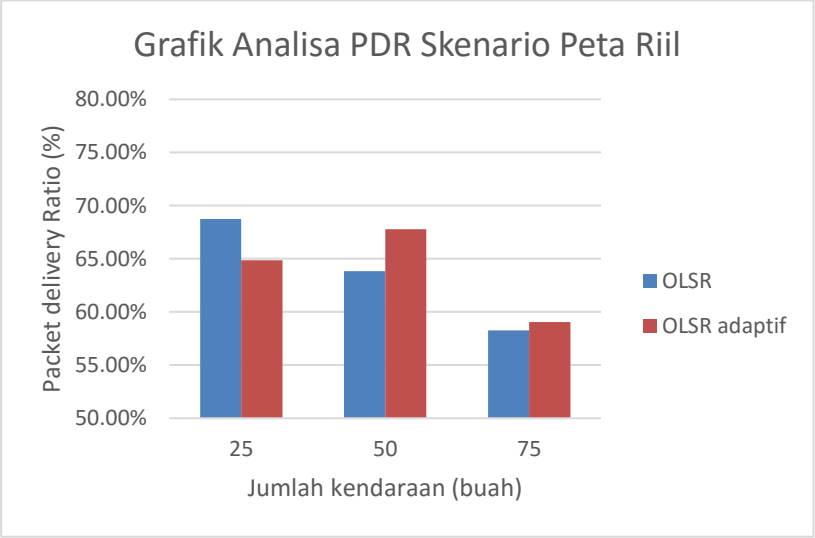
5.3.2 Hasil Uji Coba Riil

Hasil pengujian PDR pada uji coba skenario peta riil dapat dilihat pada Tabel 5-4.

Tabel 5-4. Hasil uji PDR skenario peta riil

Jumlah kendaraan	OLSR	OLSR adaptif
25 kendaraan	68.75%	64.87%
50 kendaraan	63.83%	67.79%
75 kendaraan	58.27%	59.06%

Dari data diatas dapat dibuat grafik yang menggambarkan hasil uji coba. Grafik PDR dapat dilihat pada Gambar 5-3.



Gambar 5-3. Grafik analisa PDR untuk skenario peta riil

Berdasarkan hasil uji coba skenario peta riil yang telah dilakukan, uji coba dengan menggunakan 25 kendaraan dan *routing protocol* OLSR memiliki rata-rata nilai PDR sebesar 68.75%. Uji coba dengan 25 kendaraan dan OLSR yang adaptif memiliki rata-rata nilai PDR sebesar 64.87%. Hasil uji coba memperlihatkan bahwa OLSR adaptif mengalami penurunan peforma dilihat dari nilai PDR-nya dibandingkan dengan *routing protocol* OLSR untuk uji coba skenario peta riil dengan 25 kendaraan. Penurunan rata-rata nilai PDR pada uji coba peta skenario riil dengan 25 kendaraan untuk OLSR adaptif adalah 3.88%.

Uji coba dengan menggunakan 50 kendaraan dan *routing protocol* OLSR memiliki rata-rata nilai PDR sebesar 63.83%. Uji coba dengan 50 kendaraan dan *routing protocol* OLSR adaptif memiliki rata-rata nilai PDR sebesar 67.79%. Hasil uji coba memperlihatkan bahwa OLSR adaptif mengalami kenaikan peforma

dilihat dari nilai PDR-nya dibandingkan dengan *routing protocol* OLSR pada uji coba skenario peta riil dengan 50 kendaraan. Peningkatan rata-rata nilai PDR pada uji coba peta skenario riil dengan 50 kendaraan untuk OLSR adaptif adalah 3.96%. Peningkatan nilai PDR pada uji coba skenario peta riil dengan 50 kendaraan untuk *routing protocol* OLSR adaptif mengalami kenaikan dibandingkan dengan peningkatan nilai PDR pada uji coba skenario peta riil dengan 25 kendaraan.

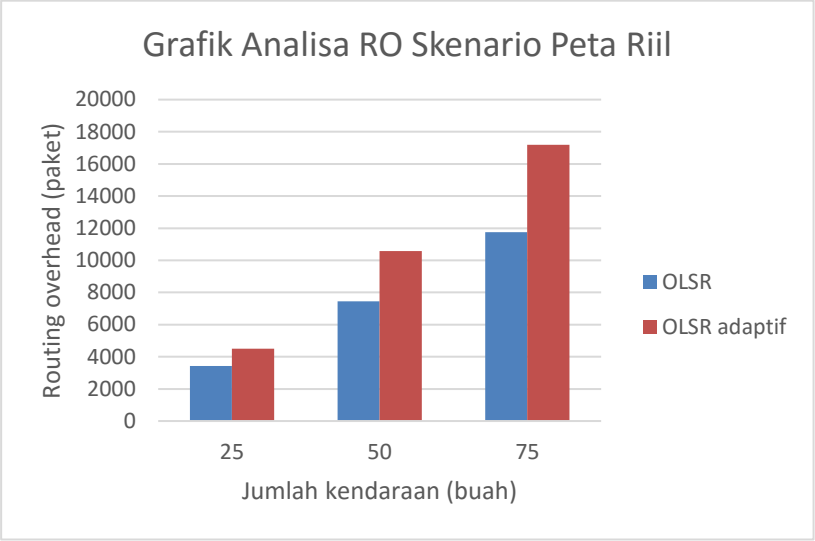
Uji coba dengan menggunakan 75 kendaraan dan *routing protocol* OLSR memiliki rata-rata nilai PDR sebesar 58,27%. Uji coba dengan 75 kendaraan dan *routing protocol* OLSR adaptif memiliki rata-rata nilai PDR sebesar 59,06%. Hasil uji coba memperlihatkan bahwa OLSR adaptif mengalami kenaikan peforma dilihat dari nilai PDR-nya dibandingkan dengan *routing protocol* OLSR pada uji coba skenario peta riil dengan 75 kendaraan. Peningkatan rata-rata nilai PDR pada uji coba peta skenario riil dengan 75 kendaraan untuk OLSR adaptif adalah 0.79%. Peningkatan nilai PDR pada uji coba skenario peta riil dengan 75 kendaraan untuk *routing protocol* OLSR adaptif mengalami penurunan dibandingkan dengan peningkatan nilai PDR pada uji coba skenario peta riil dengan 50 kendaraan dan mengalami kenaikan dibandingkan dengan peningkatan nilai PDR pada uji coba skenario peta riil dengan 25 kendaraan.

Sedangkan hasil pengujian RO pada uji coba skenario peta riil dapat dilihat pada Tabel 5-5.

Tabel 5-5. Hasil uji RO skenario peta riil

Jumlah kendaraan	OLSR	OLSR adaptif
25 kendaraan	3416.9	4507.8
50 kendaraan	7451.9	10579.9
75 kendaraan	11751.2	17194.2

Dari data diatas dapat dibuat grafik yang menggambarkan hasil uji coba. Grafik RO dapat dilihat pada Gambar 5-4.



Gambar 5-4. Grafik analisa RO untuk skenario peta riil

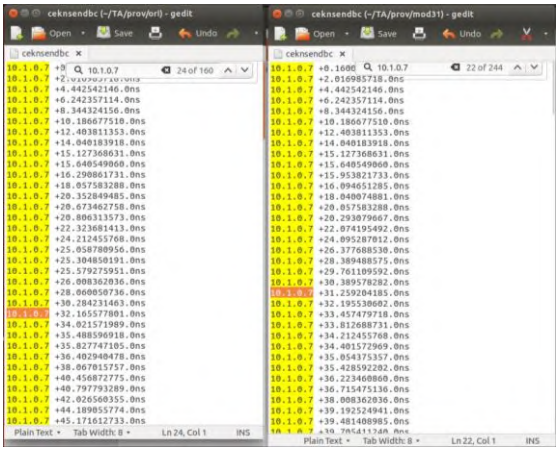
Berdasarkan hasil uji coba skenario peta riil yang telah dilakukan, uji coba dengan menggunakan 25 kendaraan dan *routing protocol* OLSR memiliki rata-rata nilai RO sebesar 3416.9 paket. Uji coba dengan 25 kendaraan dan OLSR yang adaptif memiliki rata-rata nilai RO sebesar 4507.8 paket. Hasil uji coba memperlihatkan bahwa OLSR adaptif mengalami penurunan peforma dilihat dari nilai RO-nya dibandingkan dengan *routing protocol* OLSR untuk uji coba skenario peta riil dengan 25 kendaraan. Rata-rata kenaikan nilai RO pada uji coba peta skenario riil dengan 25 kendaraan untuk OLSR adaptif adalah 1090.9 paket.

Uji coba dengan menggunakan 50 kendaraan dan *routing protocol* OLSR memiliki rata-rata nilai RO sebesar 7451.9 paket. Uji coba dengan 50 kendaraan dan OLSR yang adaptif memiliki rata-rata nilai RO sebesar 10759.9 paket. Hasil uji coba memperlihatkan bahwa OLSR adaptif mengalami penurunan peforma dilihat dari nilai RO-nya dibandingkan dengan *routing*

protocol OLSR untuk uji coba skenario peta riil dengan 50 kendaraan. Rata-rata kenaikan nilai RO pada uji coba peta skenario riil dengan 50 kendaraan untuk OLSR adaptif adalah 3128 paket.

Uji coba dengan menggunakan 75 kendaraan dan *routing protocol* OLSR memiliki rata-rata nilai RO sebesar 11751.2 paket. Uji coba dengan 75 kendaraan dan OLSR yang adaptif memiliki rata-rata nilai RO sebesar 17194.2 paket. Hasil uji coba memperlihatkan bahwa OLSR adaptif mengalami penurunan peforma dilihat dari nilai RO-nya dibandingkan dengan *routing protocol* OLSR untuk uji coba skenario peta riil dengan 75 kendaraan. Rata-rata kenaikan nilai RO pada uji coba peta skenario riil dengan 75 kendaraan untuk OLSR adaptif adalah 5443 paket.

Pada analisa RO untuk skenario peta grid dan peta riil, semakin banyak jumlah kendaraan akan menghasilkan nilai RO yang semakin besar. Uji coba RO pada uji coba mengalami penurunan peforma disebabkan oleh proses *re-broadcast* yang tidak bersamaan dikarenakan selang waktu pengiriman *TC Message* yang berbeda antar *node*. Sehingga proses *re*-broadcast pada OLSR adaptif lebih banyak dibandingkan dengan OLSR. Sebagai contoh dapat dilihat proses *broadcast* dari *node* 7 pada salah satu uji coba skenario peta grid dengan 50 kendaraan pada Gambar 5-5 untuk OLSR di sebelah kiri dan OLSR adaptif di sebelah kanan. Pada OLSR, *node* 7 hanya melakukan *broadcast* sebanyak 160 kali, sedangkan pada OLSR adaptif *broadcast* dilakukan sebanyak 244 kali. Hal ini menyebabkan kenaikan *routing overhead* pada jaringan. *Broadcast* pada kedua protokol terjadi pada saat pegiriman *hello message*, pengiriman *TC message*, dan *re-broadcast TC message*. Pada OLSR, *re-broadcast* hanya terjadi pada detik kelipatan TcInterval (*default*=5 detik). Sedangkan pada OLSR adaptif, *re-broadcast* mengikuti pengiriman *TC message* dari *node* yang memilih *node* 7 menjadi MPR.



Gambar 5-5. Perbedaan jumlah broadcast control message OLSR dan OLSR adaptif

[Halaman ini sengaja dikosongkan]

LAMPIRAN

Lampiran 1. Skrip awk untuk analisa PDR.

```
#!/usr/bin/gawk -f

BEGIN   {
       sent=0;
       recv=0;
       recv_id=0;
       pdr=0;
}

{
       if ( $1 == "t" && $3 ==
"/NodeList/1/DeviceList/0/$ns3::WifiNetDevice/Phy/Sta
te/Tx" && $9 == "Retry=0," && $49 == "(size=64)" ){
              sent++;
       }

       if ( $1 == "r" && $3 ==
"/NodeList/0/DeviceList/0/$ns3::WifiNetDevice/Phy/Sta
te/RxOk" && $49 == "(size=64)" && recv_id != $29 ){
              recv++;
              recv_id = $29;
       }
}

END{
       pdr = (recv / sent ) * 100;
       print "transmitted packet(s) = ", sent;
       print "Received packet(s) = ", recv;
       print "Packet Delivery Ratio = ", pdr,"%";
}
```

Lampiran 2. Skrip awk untuk analisa RO.

```
#!/usr/bin/gawk -f

BEGIN   {
numRoutingPackets=0;
numDataPackets=0;
```

```
regexOLSR="ns3::olsr::PacketHeader";
}

{

        if ($1 == "r" && match($0,regexOLSR))
        {
                numRoutingPackets++;
        }
        else
        {
                numDataPackets++;
        }


}
END     {
if (numDataPackets > 0)
{
        printf ("NRO = %f\n", numRoutingPackets /
numDataPackets);
}
else
{
       printf "NRO = NaN\n"
}
printf "- DONE -\n"
}
```

Lampiran 3. Potongan kode untuk membaca file mobilitas NS-2 pada vanet-routing-compare.

```
// Create Ns2MobilityHelper with the specified trace
log file as parameter
      Ns2MobilityHelper ns2 = Ns2MobilityHelper
(m_traceFile);
      ns2.Install (); // configure movements for each
node, while reading trace file
      // initially assume all nodes are not moving
      WaveBsmHelper::GetNodesMoving ().resize
(m_nNodes, 0);
```

Lampiran 4. Potongan kode untuk mengirimkan paket data pada *node* asal yang telah ditentukan

```
void
RoutingHelper::SetupRoutingMessages (NodeContainer &
c,

Ipv4InterfaceContainer & adhocTxInterfaces)
{
  // Setup routing transmissions
  OnOffHelper onoff1 ("ns3::UdpSocketFactory",Address
());
  onoff1.SetAttribute ("OnTime", StringValue
("ns3::ConstantRandomVariable[Constant=1.0]"));
  onoff1.SetAttribute ("OffTime", StringValue
("ns3::ConstantRandomVariable[Constant=0.0]"));

  Ptr<UniformRandomVariable> var =
CreateObject<UniformRandomVariable> ();
  int64_t stream = 2;
  var->SetStream (stream);
  for (uint32_t i = 0; i < m_nSinks; i++)
    {
      // protocol == 0 means no routing data, WAVE
BSM only
      // so do not set up sink
      if (m_protocol != 0)
        {
          Ptr<Socket> sink =
SetupRoutingPacketReceive
(adhocTxInterfaces.GetAddress (i), c.Get (i));
        }
AddressValue remoteAddress (InetSocketAddress
(adhocTxInterfaces.GetAddress (i), m_port));
      onoff1.SetAttribute ("Remote", remoteAddress);
ApplicationContainer temp = onoff1.Install (c.Get (i
+ m_nSinks));
      temp.Start (Seconds (var->GetValue (1.0,2.0)));
      temp.Stop (Seconds (m_TotalSimTime));
    }
}
```

Lampiran 5. Potongan kode untuk mengaktifkan ascii *trace file* pada vanet-routing-compare.

```
if (m_asciiTrace != 0)
    {
      AsciiTraceHelper ascii;
      Ptr<OutputStreamWrapper> osw =
ascii.CreateFileStream ( (m_trName + ".tr").c_str
());
      wifiPhy.EnableAsciiAll (osw);
      wavePhy.EnableAsciiAll (osw);
    }
```

Lampiran 6. Potongan kode untuk mengatur skenario yang disimulasikan dalam Tugas Akhir ini pada vanet-routing-compare.

```
if (m_scenario == 1)
    {
      // 40 nodes in RWP 300 m x 1500 m synthetic
highway, 10s
      m_traceFile = "";
      m_logFile = "";
      m_mobility = 2;
      if (m_nNodes == 156)
        {
          m_nNodes = 40;
        }
      if (m_TotalSimTime == 300.01)
        {
          m_TotalSimTime = 10.0;
        }
    }
  else if (m_scenario == 2)
    {
      // Realistic vehicular trace in 4.6 km x 3.0 km
suburban Zurich
      // "low density, 99 total vehicles"
      m_traceFile = "src/wave/examples/low99-ct-
unterstrass-1day.filt.7.adj.mob";
      m_logFile = "low99-ct-unterstrass-
1day.filt.7.adj.log";
```

```
      m_mobility = 1;
      m_nNodes = 99;
      m_TotalSimTime = 300.01;
      m_nodeSpeed = 0;
      m_nodePause = 0;
      m_CSVfileName = "low_vanet-routing-
compare.csv";
      m_CSVfileName = "low_vanet-routing-
compare2.csv";
    }
  else if (m_scenario == 3)
    {
      m_traceFile = "scratch/scenario.tcl";
      m_trName = "grid1";
      m_routingTables = 1;
      m_mobility = 1;
      m_nNodes = 100;
      m_TotalSimTime = 200;
      m_nSinks = 1;
      m_asciiTrace = 1;
      m_txp = 7;
    }
  else if (m_scenario == 32)
    {
      m_traceFile = "scratch/testatis.tcl";
      m_trName = "statisgrid1";
      m_routingTables = 1;
      m_mobility = 1;
      m_nNodes = 50;
      m_TotalSimTime = 200;
      m_nSinks = 1;
      m_asciiTrace = 1;
      m_txp = 7;
    }
  else if (m_scenario == 31)
    {
      m_traceFile = "scratch/realscenario1.tcl";
      m_trName = "real1";
      m_routingTables = 1;
      m_mobility = 1;
      m_nNodes = 50;
      m_TotalSimTime = 200;
      m_nSinks = 1;
      m_asciiTrace = 1;
      m_txp = 7;
    }
```

```
else if (m_scenario == 4)
    {
      m_traceFile = "scratch/11nsce.tcl";
      m_trName = "11n";
      m_routingTables = 1;
      m_mobility = 1;
      m_nNodes = 11;
      m_TotalSimTime = 260;
      m_nSinks = 1;
      m_asciiTrace = 1;
      m_txp = 7;
    }
  else if (m_scenario == 41)
    {
      m_traceFile = "scratch/6node.tcl";
      m_trName = "6n";
      m_routingTables = 1;
      m_mobility = 1;
      m_nNodes = 6;
      m_TotalSimTime = 150;
      m_nSinks = 1;
      m_asciiTrace = 1;
      m_txp = 3.4;
      m_protocol = 1;
    }
  else if (m_scenario == 42)
    {
      m_traceFile = "scratch/2n.tcl";
      m_trName = "2n";
      m_routingTables = 1;
      m_mobility = 1;
      m_nNodes = 2;
      m_TotalSimTime = 150;
      m_nSinks = 1;
      m_asciiTrace = 1;
      m_txp = 7;
      m_protocol = 1;
    }
  else if (m_scenario == 61)
    {
      m_traceFile = "scratch/grid1.tcl";
      m_trName = "grid1";
      m_routingTables = 1;
      m_mobility = 1;
      m_nNodes = 50;
      m_TotalSimTime = 200;
```

```
      m_nSinks = 1;
      m_asciiTrace = 1;
      m_txp = 5;
      m_protocol = 1;
    }
  else if (m_scenario == 62)
    {
      m_traceFile = "scratch/grid2.tcl";
      m_trName = "grid2";
      m_routingTables = 1;
      m_mobility = 1;
      m_nNodes = 50;
      m_TotalSimTime = 200;
      m_nSinks = 1;
      m_asciiTrace = 1;
      m_txp = 5;
      m_protocol = 1;
    }
  else if (m_scenario == 63)
    {
      m_traceFile = "scratch/grid3.tcl";
      m_trName = "grid3";
      m_routingTables = 1;
      m_mobility = 1;
      m_nNodes = 50;
      m_TotalSimTime = 200;
      m_nSinks = 1;
      m_asciiTrace = 1;
      m_txp = 5;
      m_protocol = 1;
    }
else if (m_scenario == 64)
    {
      m_traceFile = "scratch/grid4.tcl";
      m_trName = "grid4";
      m_routingTables = 1;
      m_mobility = 1;
      m_nNodes = 50;
      m_TotalSimTime = 200;
      m_nSinks = 1;
      m_asciiTrace = 1;
```

```
      m_txp = 5;
      m_protocol = 1;
    }
  else if (m_scenario == 65)
    {
      m_traceFile = "scratch/grid5.tcl";
      m_trName = "grid5";
      m_routingTables = 1;
      m_mobility = 1;
      m_nNodes = 50;
      m_TotalSimTime = 200;
      m_nSinks = 1;
      m_asciiTrace = 1;
      m_txp = 5;
      m_protocol = 1;
    }
  else if (m_scenario == 66)
    {
      m_traceFile = "scratch/grid6.tcl";
      m_trName = "grid6";
      m_routingTables = 1;
      m_mobility = 1;
      m_nNodes = 50;
      m_TotalSimTime = 200;
      m_nSinks = 1;
      m_asciiTrace = 1;
      m_txp = 5;
      m_protocol = 1;
    }
  else if (m_scenario == 67)
    {
      m_traceFile = "scratch/grid7.tcl";
      m_trName = "grid7";
      m_routingTables = 1;
      m_mobility = 1;
      m_nNodes = 50;
      m_TotalSimTime = 200;
      m_nSinks = 1;
      m_asciiTrace = 1;
      m_txp = 5;
      m_protocol = 1;
    }
  else if (m_scenario == 68)
    {
      m_traceFile = "scratch/grid8.tcl";
      m_trName = "grid8";
```

```
      m_routingTables = 1;
      m_mobility = 1;
      m_nNodes = 50;
      m_TotalSimTime = 200;
      m_nSinks = 1;
      m_asciiTrace = 1;
      m_txp = 5;
      m_protocol = 1;
    }
  else if (m_scenario == 69)
    {
      m_traceFile = "scratch/grid9.tcl";
      m_trName = "grid9";
      m_routingTables = 1;
      m_mobility = 1;
      m_nNodes = 50;
      m_TotalSimTime = 200;
      m_nSinks = 1;
      m_asciiTrace = 1;
      m_txp = 5;
      m_protocol = 1;
    }
  else if (m_scenario == 610)
    {
      m_traceFile = "scratch/grid10.tcl";
      m_trName = "grid10";
      m_routingTables = 1;
      m_mobility = 1;
      m_nNodes = 50;
      m_TotalSimTime = 200;
      m_nSinks = 1;
      m_asciiTrace = 1;
      m_txp = 5;
      m_protocol = 1;
    }
else if (m_scenario == 251)
    {
      m_traceFile = "scratch/25r1.tcl";
      m_trName = "251";
      m_routingTables = 1;
      m_mobility = 1;
      m_nNodes = 25;
      m_TotalSimTime = 200;
```

```
      m_nSinks = 1;
      m_asciiTrace = 1;
      m_txp = 5;
      m_protocol = 1;
    }
  else if (m_scenario == 252)
    {
      m_traceFile = "scratch/25r2.tcl";
      m_trName = "252";
      m_routingTables = 1;
      m_mobility = 1;
      m_nNodes = 25;
      m_TotalSimTime = 200;
      m_nSinks = 1;
      m_asciiTrace = 1;
      m_txp = 5;
      m_protocol = 1;
    }
  else if (m_scenario == 253)
    {
      m_traceFile = "scratch/25r3.tcl";
      m_trName = "253";
      m_routingTables = 1;
      m_mobility = 1;
      m_nNodes = 25;
      m_TotalSimTime = 200;
      m_nSinks = 1;
      m_asciiTrace = 1;
      m_txp = 5;
      m_protocol = 1;
    }
else if (m_scenario == 254)
    {
      m_traceFile = "scratch/25r4.tcl";
      m_trName = "254";
      m_routingTables = 1;
      m_mobility = 1;
      m_nNodes = 25;
      m_TotalSimTime = 200;
      m_nSinks = 1;
      m_asciiTrace = 1;
      m_txp = 5;
```

```
      m_protocol = 1;
    }
  else if (m_scenario == 255)
    {
      m_traceFile = "scratch/25r5.tcl";
      m_trName = "255";
      m_routingTables = 1;
      m_mobility = 1;
      m_nNodes = 25;
      m_TotalSimTime = 200;
      m_nSinks = 1;
      m_asciiTrace = 1;
      m_txp = 5;
      m_protocol = 1;
    }
  else if (m_scenario == 256)
    {
      m_traceFile = "scratch/25r6.tcl";
      m_trName = "256";
      m_routingTables = 1;
      m_mobility = 1;
      m_nNodes = 25;
      m_TotalSimTime = 200;
      m_nSinks = 1;
      m_asciiTrace = 1;
      m_txp = 5;
      m_protocol = 1;
    }
  else if (m_scenario == 257)
    {
      m_traceFile = "scratch/25r7.tcl";
      m_trName = "257";
      m_routingTables = 1;
      m_mobility = 1;
      m_nNodes = 25;
      m_TotalSimTime = 200;
      m_nSinks = 1;
      m_asciiTrace = 1;
      m_txp = 5;
      m_protocol = 1;
    }
else if (m_scenario == 258)
```

```
    {
      m_traceFile = "scratch/25r8.tcl";
      m_trName = "258";
      m_routingTables = 1;
      m_mobility = 1;
      m_nNodes = 25;
      m_TotalSimTime = 200;
      m_nSinks = 1;
      m_asciiTrace = 1;
      m_txp = 5;
      m_protocol = 1;
    }
  else if (m_scenario == 259)
    {
      m_traceFile = "scratch/25r9.tcl";
      m_trName = "259";
      m_routingTables = 1;
      m_mobility = 1;
      m_nNodes = 25;
      m_TotalSimTime = 200;
      m_nSinks = 1;
      m_asciiTrace = 1;
      m_txp = 5;
      m_protocol = 1;
    }
  else if (m_scenario == 2510)
    {
      m_traceFile = "scratch/25r10.tcl";
      m_trName = "2510";
      m_routingTables = 1;
      m_mobility = 1;
      m_nNodes = 25;
      m_TotalSimTime = 200;
      m_nSinks = 1;
      m_asciiTrace = 1;
      m_txp = 5;
      m_protocol = 1;
    }
}
```

BAB VI
KESIMPULAN DAN SARAN

6.1. Kesimpulan

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

1. *Routing protocol* OLSR adaptif mengalami kenaikan nilai rata-rata *packet delivery ratio* sebesar 3.06% dibandingkan dengan *routing protocol* OLSR pada skenario peta grid. Sedangkan pada skenario peta riil, *routing protocol* OLSR adaptif mengalami kenaikan nilai rata-rata *packet delivery ratio* sebesar 0.29% dibandingkan dengan *routing protocol* OLSR.
2. *Routing protocol* OLSR adaptif mengalami kenaikan nilai rata-rata *routing overhead* sebesar 3533.7 paket dibandingkan dengan *routing protocol* OLSR pada skenario peta grid. Sedangkan pada skenario peta riil, *routing protocol* OLSR adaptif mengalami penurunan nilai rata-rata *routing overhead* sebesar 3220.63 paket dibandingkan dengan *routing protocol* OLSR.
3. Selang waktu TC Messages yang diubah menjadi adaptif terhadap jumlah *node* tetangga di sekitarnya (*1-hop nodes*) memiliki pengaruh terhadap *routing overhead*. Nilai RO bertambah dikarenakan proses *re-broadcast TC Message* yang tidak bersamaan.

6.2. Saran

Saran yang diberikan untuk pengembangan sistem ini adalah sebagai berikut:

1. Perubahan selang waktu TC message dapat dibuat lebih adaptif dengan tambahan modifikasi algoritma, misal algoritma fuzzy.

2. Sinkronisasi update selang waktu pengiriman TC message. Semua *node* melakukan perubahan pada TcInterval secara bersamaan.
3. Analisis delay dan *collision* dapat meningkatkan hasil dari analisa uji coba pada Tugas Akhir ini.
4. *Control message* yang bersifat adaptif dapat diterapkan pada *routing protocol* proaktif yang lain.

DAFTAR PUSTAKA

[1] patusa.cyber, "Echo Magazine Volume X, Issue #26," 12 December 2012. [Online]. Available: http://ezine.echo.or.id/issue26/008.txt. [Accessed 10 December 2015].

[2] J. Hoebeke, I. Moerman, B. Dhoedt and P. Demeester, "An Overview of Mobile Ad Hoc Networks: Applications and Challanges," *Ghent University Academic Publications*, vol. III, no. 3, pp. 60-66, September 2004.

[3] G. Chandrasekaran, "VANETs: The Networking Platform for Future Vehicular Applications," *Department of Computer Science Rutgers University*, pp. 45-51, May 2007.

[4] The Linux Information Project, 1 November 2005. [Online]. Available: http://www.linfo.org/link_state_routing.html.

[5] J. Doyle., Routing TCP/IP Volume I (CCIE Professional Development), 1st ed., L. McGuire, Ed., Indianapolis: Cisco Press, 1998.

[6] P. Jacquet, P. Mühlethaler, T. Clausen, A. Laouiti, A. Qayyum and L. Viennot, "Optimized Link State Routing Protocol for Ad Hoc Network," *Multi Topic Conference, IEEE INMIC* , pp. 62-68, 2001.

[7] T. Clausen and P. Jacquet, RFC 3626 - Optimized Link State Routing Protocol, IETF, 2003.

[8] nsnam, "nsnam official page," nsnam, July 2008. [Online]. Available: https://www.nsnam.org/. [Accessed 1 March 2016].

[9] OpenStreetMap Community, "OpenStreetMap The Free Wiki World Map," OpenStreetMap Community, 1 July 2004. [Online]. Available: https://www.openstreetmap.org/. [Accessed 1 April 2016].

[10] N. Akhter, A. Masood2 and I. Laone, "Performance Improvement of Optimized Link State Routing (OLSR)

Protocol," *Optimization, Reliabilty, and Information Technology (ICROIT)*, pp. 182-187, February 2014.

[11] Y. C. Huang, S. Bhatti and D. Parker, "Tuning OLSR," *The 17th Annual IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC'06)*, pp. 1-5, September 2006.

[12] GNU, "GNU awk's User Guide," GNU, [Online]. Available: https://www.gnu.org/software/gawk/manual/gawk.html. [Accessed 1 April 2016].

[13] SUMO, "Sumo official documentation," DLR Institute of Transportation System, [Online]. Available: http://sumo.dlr.de/wiki/Main_Page. [Accessed 28 March 2016].

[14] D. S. Immanuel Scholz, "JOSM official page," JOSM, 2000. [Online]. Available: https://josm.openstreetmap.de/. [Accessed 20 April 2016].

BIODATA PENULIS

Dimas Aulia Rahman adalah nama penulis pada buku Tugas Akhir ini. Penulis lahir dari orang tua H. Syamsul Komar, S.H. M.Hum dan Drg. Hj. Eny Inayati M.Kes. sebagai anak ketiga dari tiga bersaudara. Penulis lahir pada tanggal 18 Oktober 1993 di Surabaya, Jawa Timur. Penulis menempuh pendidikan dimulai dari SDN Kalirungkut I/264 (lulus tahun 2006), melanjutkan ke SMPN 35 Surabaya (lulus tahun 2009) dan SMAN 16 Surabaya (lulus tahun 2012). Pada saat ini, penulis merupakan seorang mahasiswa yang sedang menempuh studi di Jurusan Teknik Informatika Institut Teknologi Sepuluh Nopember angkatan 2012. Memiliki hobi antara lain bermain game dan menonton film. Selama kuliah di teknik informatika ITS, penulis mengambil rumpun mata kuliah Komputasi Berbasis Jaringan (KBJ). Komunikasi dengan penulis dapat melalui email : **dimas.aulia12@gmail.com.**