Institut
Teknologi
Sepuluh Nopember

TUGAS AKHIR – KI141502

STUDI KINERJA 802.11p PADA PROTOKOL *AD HOC ON-DEMAND DISTANCE VECTOR* (AODV) DI LINGKUNGAN *VEHICULAR AD HOC NETWORK* (VANET) MENGGUNAKAN *NETWORK SIMULATOR 2* (NS-2)

ILMAL ALIFRIANSYAH RAHARDJO
NRP 5110100077

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 2017

TUGAS AKHIR – KI141502

STUDI KINERJA 802.11p PADA PROTOKOL *AD HOC ON-DEMAND DISTANCE VECTOR* (AODV) DI LINGKUNGAN *VEHICULAR AD HOC NETWORK* (VANET) MENGGUNAKAN *NETWORK SIMULATOR 2* (NS-2)

ILMAL ALIFRIANSYAH RAHARDJO
NRP 5110100077

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 2017

(Halaman ini sengaja dikosongkan)

UNDERGRADUATE THESES – KI14102

THE PERFORMANCE OF 802.11P ON AD HOC ON-DEMAND DISTANCE VECTOR (AODV) ROUTING PROTOCOL IN VANET USING NETWORK SIMULATOR 2 (NS-2)

ILMAL ALIFRIANSYAH RAHARDJO
NRP 5110100077

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

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

DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY
SEPULUH NOPEMBER INSTITUTE OF TECHNOLOGY
SURABAYA
2017

(Halaman ini sengaja dikosongkan)

LEMBAR PENGESAHAN

STUDI KINERJA 802.11P PADA PROTOKOL AD HOC ON-DEMAND DISTANCE VECTOR (AODV) DI LINGKUNGAN VEHICULAR AD HOC NETWORK (VANET) MENGGUNAKAN NETWORK SIMULATOR 2 (NS-2)

TUGAS AKHIR
Diajukan Untuk Memenuhi Salah Satu Syarat Memperoleh
Gelar Sarjana Komputer
pada
Bidang Studi Komputasi Berbasis Jaringan
Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Oleh:
ILMAL ALIFRIANSYAH RAHARDJO
NRP: 5110 100 077
Disetujui oleh Pembimbing Tugas Akhir:

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

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

SURABAYA
DESEMBER 2016

(Halaman ini sengaja dikosongkan)

STUDI KINERJA 802.11P PADA PROTOKOL AD HOC ON-DEMAND DISTANCE VECTOR (AODV) DI LINGKUNGAN VEHICULAR AD HOC NETWORK (VANET) MENGGUNAKAN NETWORK SIMULATOR 2 (NS-2)

Nama	: Ilmal Alifriansyah Rahardjo
NRP	: 5110100077
Jurusan	: Teknik Informatika – FTIf
Universitas	: Institut Teknologi Sepuluh Nopember
Dosen Pembimbing I	: Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.
Dosen Pembimbing II	: Ir. F.X. Arunanto, M.Sc.

ABSTRAK

Dalam dunia jaringan nirkabel sudah tidak asing dengan salah satu konsep yang memungkinkan komunikasi antar kendaraan (*inter-vehicle*) dan komunikasi antara kendaraan dengan infrastruktur dis sekitar jalan (*vehicle-to-roadside*). Konsep ini juga merupakan subset dari dari *Mobile Ad-Hoc Network* (MANET), yaitu *Vehicular Ad-hoc Network* (VANET). Pada VANET kendaraan yang ada bertindak sebagai *node* pada suatu jaringan. VANET terdiri dari banyak *node* yang juga berfungsi sebagai *router*. Berbeda dengan MANET, tingkat mobilitas pada VANET lebih tinggi.

VANET termasuk ke dalam jaringan komunikasi nirkabel dimana komunikasi terjadi melalui *link* nirkabel yang dipasang di setiap *node*. Tiap *node* pada VANET berlaku baik sebagai partisipan ataupun *router* pada jaringan, baik bagi *node* utama atau *neighborhood node* yang berkomunikasi di dalam radius transmisi dari VANET tersebut. Tujuan utama dari adanya VANET ini adalah nantinya dapat digunakan untuk menciptakan dan menyediakan aplikasi-aplikasi transportasi untuk mendukung

keamanan maupun kenyamanan bagi pengendara. Hal ini tentunya memerlukan implementasi protokol *routing* yang sesuai dengan karakteristik dari VANET dimana kendaraan-kendaraan yang notabene memiliki dinamika pergerakan *node* yang tinggi dalam suatu jaringan.

Adapun *routing* protokol pada jaringan nirkabel ini sendiri dibagi menjadi tiga, yaitu proaktif, reaktif dan *hybird*. Telah diketahui bahwa *routing* protokol reaktif memiliki kinerja yang yang lebih baik daripada proaktif pada suatu jaringan dengan mobilitas tinggi. Salah satu protokol *routing* reaktif yang telah teruji dalam VANET adalah *routing* protokol *Ad hoc On-Demand Distance Vector* (AODV). Dan pada Tugas Akhir ini dilakukan analisa performa dari *routing* protokol *Ad hoc On-Demand Distance Vector* (AODV) tersebut dengan parameter 802.11p dalam lingkungan VANET.

Dari percobaan Tugas Akhir ini dihasilkan suatu performa bahwa *routing* protokol AODV dengan parameter 802.11p pada skenario riil mengalami peningkatan nilai rata-rata pada *packet deliery ratio* dan *routing overhead* dan mengalami penurunan nilai rata-rata pada *delay* seiring dengan bertambahnya kepadatan kendaraan dibandingkan dalam skenario *grid*.

Kata kunci: VANET, AODV, 802.11p, *Network Simulator*, NS-2.

PERFORMANCE OF 802.11P ON AD HOC ON-DEMAND DISTANCE VECTOR (AODV) ROUTING PROTOCOL IN VANET USING NETWORK SIMULATOR 2 (NS-2)

Name	: Ilmal Alifriansyah Rahardjo
NRP	: 5110100077
Department	: Teknik Informatika - FTIf
University	: Sepuluh Nopember Institute of Technology
Supervisor I	: Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.
Supervisor II	: Ir. F.X. Arunanto, M.Sc.

ABSTRACT

In the world of wireless networks are no stranger to one of them that allows communication between a vehicle (inter-vehicle) and communication between a vehicle to infrastructure in about the way (vehicle-to-roadside) .This concept is also is a subset of of mobile ad-hoc network (MANET), which is vehicular ad-hoc network (VANET) .In VANET vehicles act as node in a network .VANET consisting of many node which also functions as router .Different from MANET, the mobility in VANET is higher than in MANET

Vanet part of the wireless communications networks where the communication occurs wireless through the links installed in each node .Every nodes on VANET applies well as participants or router on the network, good for nodes primary or neighborhood nodes that are communicate in the transmission of the VANET. The main purpose of the VANET is will be used to provide some applications for transportation to support security and comfortability for road users .This obviously need of the protocol routing according to characteristic of VANET where vehices that having the dynamics of movement nodes high in a network .

However, routing protocol on wireless networks are divided into three, namely proactive, reactive and hybird.Have been known that routing reactive protocol have a better than proactive at a network with high mobility.One protocol routing reactive who has useful in vanet protocol is routing ad hoc on-demand distance vector (AODV).And in the late analysis protocol performance of routing ad hoc on-demand distance vector (AODV) the parameters 802.11p in the environment VANET.

Experiment duty from the end of this produced a performance that routing protocol AODV with parameters 802.11p on real scenario increased average value on packet deliery ratio and routing overhead and experienced a decline in the value of the average delay along with increasing density of vehicles compared in scenario grid.

Keywords: VANET, AODV, 802.11p, Network Simulator, NS-2.

.

KATA PENGANTAR

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

Segala puji bagi Allah SWT yang telah melimpahkan rahmat dan anugerah-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul **"*Studi Kinerja 802.11p pada protokol Ad hoc On-Demand Distance Vector (AODV) di lingkungan Vehicular Ad Hoc Network (VANET) menggunakan Network Simulator 2 (NS-2)*"** dengan tepat waktu.

Harapan dari penulis semoga apa yang tertulis di dalam buku Tugas Akhir ini dapat bermanfaat bagi pengembangan ilmu pengetahuan saat ini, serta dapat memberikan kontribusi yang nyata bagi kampus Teknik Informatika, ITS, dan bangsa Indonesia.

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

1. Allah SWT atas limpahan rahmat dan hidayah-Nya sehingga akhirnya penulis dapat menyelesaikan Tugas Akhir ini dengan baik dan indah.
2. Keluarga penulis, Ayah dan Ibu yang telah berada di surga yang semasa hidupnya selalu mendoakan, selalu memberi semangat dan melakukan yang terbaik kepada penulis, sehingga penulis selalu termotivasi setiap saat untuk dapat menyelesaikan Tugas Akhir ini. Serta tak lupa berterima kasih kepada kedua kakak yang membuat penulis selama menjalani masa kuliah termotivasi untuk menjadi lebih baik dan akan menjadi seorang yang dapat membimbing mereka dijalan yang benar. Amin Ya Robbal Alamin.
3. Keluarga penulis selama orang tua tiada, Om Tot, Tante Atiek, Topeng, Mbak Rani, Titi, Mas Harto, Mbak Rizka dan semua keluarga besar, penulis sangat berterima kasih sekali atas bantuan segalanya baik doa, semangat, materil

maupun non-materil semuanya. Penulis sangat berterima kasih sekali.

4. Bapak Dr.Eng. Radityo Anggoro, S.Kom., M.Sc. selaku dosen pembimbing yang dengan segala kebaikannya dan kesabarannya beliau tidak pernah lelah memberikan dukungan, bimbingan, nasehat, perhatian, kepercayaan serta semua yang telah diberikan kepada penulis. Penulis sangat berterima kasih sekali kepada beliau. Doa penulis selalu untuk beliau sekeluarga.
5. Bapak Ahmad Saikhu, S.Si., M.T. selaku dosen wali awal dan Ibu Nanik Suciati, S.Kom., M.Kom., Dr.Eng. selaku dosen wali akhir yang selama 6,5 tahun ini dengan sabar membimbing dan memberikan nasehat kepada penulis. Terima kasih Bapak dan Ibu.
6. Bapak Dr.Eng. Darlis Herumurti, S.Kom, M.Kom. selaku Ketua Jurusan Teknik Informatika – ITS, Bapak Dr.Eng. Radityo Anggoro, S.Kom., M.Sc. selaku koordinator TA sekaligus dosen pembimbing, dan segenap Bapak/Ibu dosen Teknik Informatika yang telah memberikan ilmunya kepada penulis.
7. Segenap staf Tata Usaha dan karyawan yang telah memberikan segala bantuan dan kemudahan kepada penulis selama menjalani kuliah di Teknik Informatika ITS.
8. Keluarga besar administrator laboratorium GCL, Alief, Rasyid, Naufal, Valen, Adhan "Tulang Ayam", Ubal, Mas Mahfud, Mas Beny, Mas Dading, Mas Alfa dan semua admin GCL terdahulu, terima kasih selalu memberikan rasa kekeluargaan yang diberikan kepada penulis selama menjadi administrator GCL.
9. Keluarga besar administrator dan keluarga laboratorium ALPRO, Chol, Demsy, Ridho, Lutfy, Admiral, Mbah, Cimenk, Pentol dan semua warga ALPRO, terima kasih selalu memberikan rasa kekeluargaan yang diberikan kepada penulis selama mengerjakan Tugas Akhir ini di laboraturium ALPRO.

10. Teman-teman seperjuangan bidang minat AJK, Guruh, Azi, Ucup dan Tora yang berjuang bersama-sama menuju Wisuda 115 ITS, terima kasih atas kebersamaannya dan dukungan selama ini.
11. Seluruh teman Teknik Informatika ITS angkatan 2010, terima kasih atas rasa kekeluargaan yang telah kalian berikan dan teima kasih telah lulus terlebih dahulu daripada penulis. Itu membuat motivasi tersendiri kepada penulis.
12. Andi, Nanda, Meerza, Reza, Danan, Ira, Dedek sebagai teman mulai dari TK hingga sekarang. Okky, Syaiful, Nando sebagai sahabat SD-SMP-SMA. Alief, Nanda, Bowo, Rasyid, Pras, sahabat semasa kuliah. Chol yang selalu memberi ilmu agama yang sangat bermanfaat kepada penulis yang selalu menemani disaat penulis mengerjakan Tugas Akhir ini.
13. Adina sebagai sesorang yang membuat penulis selalu semangat dalam mengerjakan dan terima kasih atas pelajaran dan pengalaman hidup selama ini yang saling kita bagi bersama. Terima kasih.
14. Mbak Sieh sebagai penolong pertama ketika terdengar suara bergemuruh di dalam perut ini. Terima kasih segala masakan yang telah kau buat, Mbak.
15. Dan semuanya yang belum disebutkan satu-persatu, terima kasih banyak.

Penulis mengharapkan adanya saran dan kritik yang membangun dari pembaca, sehingga memperlancar Tugas Akhir ini agar dapat menjadi manfaat bagi masyarakat.

Surabaya, Desember 2016

Ilmal Alifriansyah Rahardjo

(Halaman ini sengaja dikosongkan)

DAFTAR ISI

(Halaman ini sengaja dikosongkan)

DAFTAR TABEL

(Halaman ini sengaja dikosongkan)

DAFTAR GAMBAR

(Halaman ini sengaja dikosongkan)

LAMPIRAN

(Halaman ini sengaja dikosongkan)

BAB I
PENDAHULUAN

Pada bab ini akan dijelaskan beberapa hal dasar yang berhubungan dalam pembuatan Tugas Akhir antara lain latar belakang, perumusan masalah, tujuan, dan manfaat pembuatan Tugas Akhir serta metodologi dan sistematika pembuatan buku ini. Dari uraian dibawah ini, diharapkan dapat memudahkan mendapatkan gambaran Tugas Akhir secara umum dengan baik.

1.1 Latar Belakang

Dewasa ini perkembangan dunia internet semakin berkembang dengan pesat, mulai digunakan untuk *browsing*, *chatting* dan lain sebagainya. Kegiatan tersebut merupakan termasuk suatu aplikasi yang berhubungan dengan proses pengiriman paket data dalam jaringan internet. Ini tidak lepas dari teknologi nirkabel yang terus berkembang lebih maju tiap saat. Berdasarkan hasil survei yang dilakukan oleh Asosiasi Penyelenggara Jasa Internet Indonesia (APJII) dengan Pusat Kajian Komunikasi Universitas Indonesia (PUSKAKOM UI). Hasil survei menunjukkan bahwa jumlah pengguna internet di Indonesia untuk saat ini telah mengalami peningkatan sebesar 34,9% dari jumlah seluruh penduduk Indonesia. Jika pada tahun sebelumnya ada sekitar 71,9 juta pengguna internet di Indonesia, maka pada semester pertama 2015 ini jumlah tersebut mencapai sekitar 88,1 juta pengguna. Dengan semakin banyak pengguna internet ini semakin bertambah pula kebutuhan yang diperlukan untuk berkomunikasi antar pengguna dan pasti akan semakin kompleks. Tidak dipungkiri, hampir semua kalangan, dari anak-anak hingga dewasa sudah menggunakan perangkat bergerak, mulai dari *smartphone*, *notebook*, *tablet* hingga PC (*Personal Computer*) dan perangkat itu semua terkoneksi dengan internet. Itulah mengapa pada saat ini teknologi internet dan nirkabel khususnya ini sering dikatakan menjadi indikator kemajuan

peradaban manusia karena dengan teknlogi ini pengguna memungkinkan berkomunikasi secara langsung dengan pengguna lain dalam kondisi yang tidak menetap. Namun saat ini sebagian besar koneksi antara perangkat nirkabel masih mengunakan layanan penyedia infrastruktur yang tetap. Dengan demikian diperlukan sebuah model baru untuk menyediakan layanan jaringan tanpa infrastruktur.

Salah satu model baru tersebut adalah *Vehicular Ad-hoc Network* (VANET) [1]. VANET merupakan konsep subset dari *Mobile Ad-Hoc Network* (MANET) dimana kendaraan bertindak sebagai *node* pada jaringan. VANET tergolong ke dalam jaringan komunikasi nirkabel dimana komunikasi terjadi melalui link nirkabel yang dipasang di setiap *node*. Tiap *node* pada VANET berlaku baik sebagai partisipan ataupun router pada jaringan, baik bagi *node* utama atau intermediate *node* yang berkomunikasi di dalam radius transmisinya. Mobilitas *node* yang tinggi merupakan karakteristik dasar VANET yang menyebabkan perubahan yang cepat pada topologi jaringan. Hal ini tentunya memerlukan implementasi protokol *routing* yang sesuai dengan karakteristiknya di dalam jaringan [2].

Implementasi pada lingkungan VANET dapat dilakukan dengan menggunakan simulasi sehingga penelitian ini dapat dilakukan untuk mempelajari sistem dengan baik. Simulasi dilakukan dengan menggunakan Network Simulator 2 (NS-2). Implementasi ini akan dilakukan analisa performa 802.11p (WAVE) pada protokol *routing reactive* yaitu *Ad hoc On-Demand Distance Vector* (AODV).

Dalam Tugas Akhir ini penulis mengimplementasikan *routing* protokol AODV dengan parameter 802.11p dalam lingkungan VANET pada *network simulator* NS-2. Penulis melakukan studi kinerja performa pada *routing* protokol AODV dengan melakukan simulasi pada skenario yang memiliki jumlah *node* yang bervariasi. Kemudian dari hasil simulasi tersebut menghasilkan suatu nilai performa yang ditampilkan pada skenario *grid* dan skenario riil.

.

1.2 Rumusan Masalah

Rumusan masalah yang diangkat pada Tugas Akhir ini adalah sebagai berikut:

1. Bagaimana performa *routing* protokol AODV dengan parameter 802.11p dalam lingkungan VANET pada skenario *grid* dan skenario riil?
2. Bagaimana cara mengimplementasikan *routing* protokol AODV pada NS-2?

1.3 Batasan Masalah

Implementasi dari metode pada perangkat lunak yang dibangun pada Tugas Akhir ini memiliki beberapa batasan, yaitu sebagai berikut:

1. Area simulasi *grid* dibuat dengan aplikasi SUMO.
2. Area simulasi riil dibuat dengan aplikasi OpenStreetMap.
3. Area yang digunakan berukuran 1200 m x 1200 m pada skenario *grid* dan 1000 m x 800 m pada skenario riil.
4. Jumlah *node* yang digunakan dalam masing-masing skenario, yaitu 50, 100, 150 dan 200.

1.4 Tujuan

Tujuan dari pembuatan Tugas Akhir ini adalah untuk mengetahui performa dari *routing* protokol AODV dengan parameter 802.11p dalam lingkungan VANET pada skenario *grid* dan skenatio riil.

1.5 Manfaat

Adapun manfaat yang bisa diambil dari pengerjaan Tugas Akhir ini adalah:

1. Menjadi dasar dalam menciptakan aplikasi-aplikasi untuk transportasi guna mendukung keamanan dan kenyamanan bagi pengendara dalam berkendara di jalan.
2. Mendapatkan informasi yang *reliable* tentang kondisi ataupun keadaan jalan pada saat pengendara melewati jalan tersebut.
3. Menjadi acuan untuk penelitian kedepan terhadap *routing* protokol AODV dalam VANET.

1.6 Metodologi

Adapun langkah-langkah yang ditempuh dalam pengerjaan Tugas Akhir ini adalah sebagai berikut :

1. Penyusunan Proposal Tugas Akhir

Penyusunan proposal ini merupakan tahap awal dalam pengerjaan Tugas Akhir. Proposal ini berisi gambaran secara umum percobaan yang akan diimplementasikan nantinya. Proposal ini juga menjelaskan rencana implementasi percobaan *routing protocol* AODV pada VANET menggunakan NS-2.

2. Studi Literatur

Pada tahap ini dilakukan pencarian informasi dan studi literatur yang digunakan untuk pemrosesan data dan desain perangkat lunak yang akan dibuat. Informasi didapatkan dari buku dan literatur lain yang berhubungan dengan algoritma yang digunakan dalam pengerjaan Tugas Akhir. Informasi yang dicari dan dipahami antara lain protokol AODV, VANET, SUMO dan NS-2.

3. Implementasi

Implementasi merupakan tahap pembangunan perangkat lunak yang mengimplementasikan algoritma-algoritma yang sudah diajukan. Sesuai dengan rancangan

yang diajukan pada proposal, pembangunan perangkat lunak diimplementasikan sesuai dengan konsep yang telah didapatkan saat studi literatur.

4. **Pengujian dan Evaluasi**

   Pada tahap ini dilakukan uji coba pada aplikasi yang telah dibuat. Pengujian dan evaluasi akan dilakukan dengan melihat kesesuaian dengan perencanaan. Uji coba ini dilakukan untuk membuktikan bahwa perangkat lunak yang dibangun telah bekerja sesuai dengan tujuan dan menjadi solusi dari permasalahan.

5. **Penyusunan Buku Tugas Akhir**

   Pada tahap ini dilakukan penyusunan laporan yang menjelaskan dasar teori dan metode yang diterapkan dalam Tugas Akhir. Buku Tugas Akhir merupakan dokumentasi perangkat lunak yang mencakup teori, penerapan, serta kesimpulan dari perangkat lunak yang dibangun.

1.7 Sistematika Penulisan

Buku Tugas Akhir ini disusun dengan sistematika penulisan sebagai berikut:

1. **Bab I. Pendahuluan**

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

2. **Bab II. Tinjauan Pustaka**

   Bab ini berisi tentang teori yang digunakan dan diimplementasikan pada Tugas Akhir. Teori-teori ini mencakup konsep dasar, protokol yang digunakan, dan permasalahan yang berhubungan dengan protokol tersebut. Kajian teori utama yang dijelaskan pada bab ini

antara lain adalah teori pengenalan protokol AODV, Network Simulator 2 dan model transmisi 802.11p.

3. **Bab III. Desain dan Perancangan**

   Bab ini merupakan gambaran perangkat lunak secara umum. Pada bab ini dijelaskan tahapan-tahapan implementasi dengan menggunakan diagram alir, penjelasan variabel, serta desain antarmuka yang akan dibuat.

4. **Bab IV. Implementasi**

   Bab ini menjelaskan tentang pembangunan aplikasi, penjelasan fungsi-fungsi yang digunakan, dan kode Matlab yang diaplikasikan agar perangkat lunak berjalan sesuai dengan rencana yang diajukan.

5. **Bab V. Uji Coba**

   Bab ini berisi hasil pemrosesan data dengan menggunakan perangkat lunak yang telah dibangun. Pada bab ini juga disertakan analisis dari hasil perangkat lunak.

6. **Bab VI. Penutup**

   Bab ini merupakan penjelasan dari hasil akhir yang dapat ditarik dari keseluruhan proses dan percobaan Tugas Akhir. Pada bab ini juga merupakan penjelasan atas permasalahan yang ada sebelum percobaan ini dilakukan. Pada bab ini juga dituliskan saran-saran yang berisi hal-hal yang masih dapat diperbaiki dan dikembangkan.

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 dibuat dan berguna sebagai penunjang dalam pengembangan perangkat lunak.

2.1 *Ad hoc On-Demand Distance Vector* (AODV)

Ad hoc On-Demand Distance Vector (AODV) adalah protokol *routing* yang termasuk dalam klasifikasi reaktif protokol *routing*, yang hanya me-*request* sebuah rute saat dibutuhkan. AODV yang standar ini dikembangkankan oleh C. E. Perkins, E.M. Belding-Royer dan S. Das pada RFC 3561. Ciri utama dari AODV adalah menjaga *timer-based state* pada setiap *node* sesuai dengan penggunaan tabel *routing*. Tabel *routing* akan kadaluarsa jika jarang digunakan. AODV memiliki *route discovery* dan *route maintenance*. *Route Discovery* berupa *Route Request* (RREQ) dan *Route Reply* (RREP). Sedangkan *Route Maintenance* berupa Data, *Route Update* dan *Route Error* (RRER).

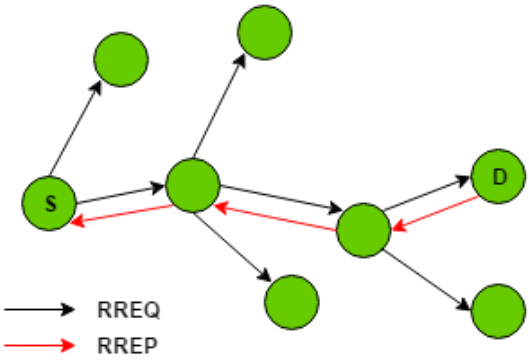
AODV memerlukan setiap *node* untuk menjaga tabel *routing* yang berisi *field* :

- *Destination IP Address* : berisi alamat IP dari *node* tujuan yang digunakan untuk menentukan rute.
- *Destination Sequence Number* : *destination sequence number* bekerjasama untuk menentukan rute.
- *Next Hop* : 'Loncatan' (*hop*) berikutnya, bisa berupa tujuan atau *node* tengah, *field* ini dirancang untuk meneruskan paket ke *node* tujuan.
- *Hop Count* : Jumlah *hop* dari alamat IP sumber sampai ke alamat IP tujuan.
- *Lifetime* : Waktu dalam milidetik yang digunakan untuk *node* menerima RREP.

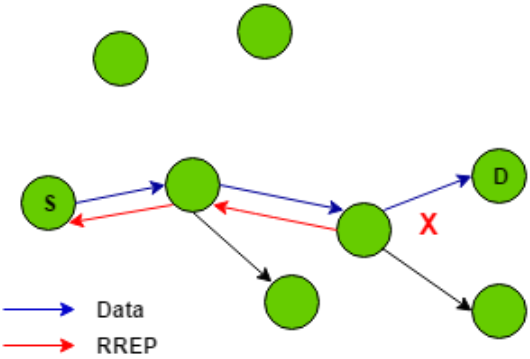
- *Routing Flags* : Status sebuah rute; jika *up* (valid), jika *down* (tidak valid) atau sedang diperbaiki.

AODV mengadopsi mekanisme yang sangat berbeda untuk menjaga informasi *routing*. AODV menggunakan tabel *routing* dengan satu *entry* untuk setiap tujuan. Tanpa menggunakan *routing* sumber, AODV mempercayakan pada tabel *routing* untuk menyebarkan *Route Reply* (RREP) kembali ke sumber dan secara sekuensial akan mengarahkan paket data menuju ketujuan. AODV juga menggunakan *sequence number* untuk menjaga setiap tujuan agar didapat informasi *routing* yang terbaru dan untuk menghindari *routing loops*. Semua paket yang diarahkan membawa *sequence number* ini.

Penemuan jalur (*Path discovery*) atau *Route Discovery* di-inisiasi dengan menyebarkan *Route Reply* (RREP). Ketika RREP menjelajahi *node*, maka selanjutnya akan secara otomatis menmbuat suatu *path* baru. Jika sebuah *node* menerima RREP, maka *node* tersebut akan mengirimkan RREP lagi ke *node* atau *destination sequence number* seperti terlihat pada Gambar 2.1 dibawah ini.



Gambar 2.1. Mekanisme Penemuan Rute (*Route Discover*).



Gambar 2.2. Mekanisme Data (*Route Update*) dan *Route Error*.

Pada proses ini, *node* pertama kali akan mengecek *destination sequence number* pada tabel *routing* pada *node* yang menerima paket tersebut, apakah nilainya lebih besar daripada *Route Request* (RREQ), jika lebih besar, maka *node* akan mengirim RREP. Ketika RREP berjalan kembali ke *source* melalui jalur yang telah diatur, maka paket tersebut akan menngatur kembali jalur rute ke *source node* dan memperbarui waktu *timeout* [3].

Dan juga terlihat pada Gambar 2.2 bahwa jika sebuah *link* ke *hop* berikutnya tidak dapat dideteksi dengan metode penemuan rute, maka *link* tersebut akan diasumsikan putus dan *Route Error* (RERR) akan disebarkan ke *node* tetangganya. Dengan demikian sebuah *node* bisa menghentikan pengiriman data melalui rute ini atau meminta rute baru dengan menyebarkan RREQ kembali.

2.2 Network Simulator 2 (NS-2)

Network Simulator 2 (NS-2) adalah suatu interpreter yang *object-oriented*, dan *discrete event-driven* yang dikembangkan oleh University of California Berkeley dan USC ISI sebagai bagian dari projek *Virtual Internet Testbed* (VINT) [4]. *Network Simulator* ini menjadi salah satu alat simulasi yang sangat berguna untuk menunjukkan simulasi jaringan melibatkan *Local Area Network*

(LAN), *Wide Area Network* (WAN), dan dengan berkembangnya zaman berkembang pula fungsi dari alat simulasi ini, yaitu selama beberapa tahun belakangan ini alat simulasi ini melibatkan pada jaringan nirkabel (*wireless*) serta pada jaringan *ad hoc*. Adapun beberapa keuntungan menggunakan *Network Simulator* sebagai alat simulasi pembantu analisi dalam riset, antara lain adalah *Network Simulator* ini dilengkapi dengan *tools* validasi yang digunakan untuk menguji kebenaran pemodelan suatu sistem. Secara *default*, semua pemodelan akan dapat melewati proses validasi ini, seperti media, protokol dan komponen jaringan yang lengkap dengan perilaku trafiknya sudah disediakan pada *library Network Simulator*.

Versi terbaru dari NS-2 adalah ns-2.35 yang dirilis pada tahun 2011. Pada NS-2 ini digunakan dua bahasa pemrograman untuk sebuah simulasi, yang pertama adalah C++, dimana dalam NS-2 ini C++ bertugas untuk mengimplementasikan bagian-bagian jaringan yang akan disimulasikan. Dan kedua adalah Otcl, sedangkan OTcl digunakan untuk menulis skenario simulasi jaringan.NS-2 dapat digunakan di berbagai macam sistem operasi GNU?Linux, FeeBSD, OS X, Solaris dan Windows (dijalankan dengan menggunakan Cygwin) karena memang pada dasarnya NS-2 ini bersifat *open-source*.

2.2.1 Instalasi *Network Simulator 2* (NS-2)

Pada bidang yang terbentuk dari pola data, SVM akan membentuk *hyperplane* yang memisahkan 2 *class*.

Sebelum melakukan instalasi NS-2 hal paling awal pastinya unduh *source file* NS-2. Sementara itu, dilakuan pembaharuan pada komponen pada sistem operasi terlihat pada Gambar 2.3 dan meng-*install* paket yang dubutuhkan NS-2 yang ditunjukkan pada Gambar 2.4 berikut.

```
$ sudo apt-get update
$ sudo apt-get dist-upgrade
$ sudo apt-get update
```

Gambar 2.3. Perintah untuk memperbarui komponen pada sitem operasi.

```
$ sudo apt-get install build-essential autoconf
  automake
$ sudo apt-get install tcl8.5-dev tk8.5-dev
$ sudo apt-get install perl xgraph libxt-dev
  libx11-dev libxmu-dev
```

Gambar 2.4. Perintah untuk meng-install paket yang dibutuhkan oleh NS-2.

Setelah semua paket ter-*install*, selanjutnya ekstrak *source file* NS-2 tadi yang telah diunduh sebelumnya. Lakukan navigasi menuju direktori NS-2 dan meng-*install* nya seperti Gambar 2.5 dibawah. Setelah NS-2 sudah ter-*install* dalam sistem, langkah selanjutnya adalah untuk mengatur *environment variables* pada NS-2 yaitu dengan cara mengubah sedikit *file* ".bashrc" dengan menambahkan *script* yang terlampir pada Gambar 2.6. Dalam *script* tersebut sudah diberi keterangan bahwa untuk mengubah *path* NS-2 pada masing-masing sistem yang digunakan. Setelah sudah diatur sedemikian rupa maka hal terakhir yang perlu dilakukan adalah memeriksa kembali NS-2 yang telah ter-*install* bahwa semua berjalan dengan lancar, yaitu dengan cara validasi seperti terlihat pada Gambar 2.7.

```
$ ~/directory_where_you_extract_the_file/ns-
  allinone-2.35
$ ./install
```

Gambar 2.5. Perintah untuk meng-*install* NS-2.

```
# LD_LIBRARY_PATH
  OTCL_LIB=/path_to/ns-allinone-2.35/otcl-1.14/
  NS2_LIB=/path_to/ns-allinone-2.35/lib/
  USR_Local_LIB=/usr/local/lib/
  export
  LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$OTCL_LIB:$N
  2_LIB:$USR_Local_LIB

# TCL_LIBRARY
  TCL_LIB=/path_to/ns-allinone-
  2.35/tcl8.5.10/library/
  USR_LIB=/usr/lib/
  export
  TCL_LIBRARY=$TCL_LIBRARY:$TCL_LIB:$USR_LIB

# PATH
  XGRAPH=/path_to/ns-allinone-2.35/xgraph
  12.2/:/path_to/ns-allinone
  2.35/bin/:/path_to/ns-allinone
  2.35/tcl8.5.10/unix/:/path_to/ns-allinone
  2.35/tk8.5.10/unix/
  NS=/path_to/ns-allinone-2.35/ns-2.35/
  NAM=/path_to/ns-allinone-2.35/nam-1.15/
  export PATH=$PATH:$XGRAPH:$NS:$NAM
```

Gambar 2.6. Menambahkan *script* dan mengubah sedikit keterangan pada *file* ".bashrc".

```
$ ~/directory_where_you_extract_the_file/ns-
  allinone-2.35/ns-2.35
$ ./validate
```

Gambar 2.7. Memeriksa bahwa NS-2 sudah berjalan dengan lancar.

2.2.2 Penggunaan Skrip *Object-oriented Tool Command Language* (OTcl)

OTcl merupakan ekstensi *object-oriented* dari bahasa pemrograman Tcl. Bahasa OTcl digunakan sebagai bahasa *scripting* pada NS-2 untuk mengatur lingkungan dan skenario simulasi. OTcl menangani interaksi langsung antara pengguna dengan simulator serta menangani interaksi antara objek-objek OTcl lainnya. Variabel-variabel pada domain OTcl dipetakan pada objek C++ yang biasa dikenal sebagai sebuah *handle*. Secara konseptual, sebuah *handle* (misal, n sebagai *handle* untuk *node*) hanyalah sebuah kalimat atau karakter biasa dan tidak memiliki fungsional apapun dalam domain OTcl. Fungsionalitas *handle* tersebut (misal, penerimaan paket) didefinisikan pada objek C++ yang dipetakan. Dalam OTcl, sebuah *handle* berfungsi sebagai substansi untuk menangani interaksi simulator dengan pengguna, maupun interaksi dengan objek OTcl lainnya. Hal ini memungkinkan pembuatan skenario simulasi tanpa perlu menggunakan bahasa C++ secara langsung karena setiap *class* yang ada pada OTcl memiliki binding pada C++. Gambar 2.8 menunjukkan contoh potongan kode OTcl pada NS-2 untuk melakukan lingkungan simulasi.

```
set val(chan)           Channel/WirelessChannel    ;#
channel type
set val(prop)           Propagation/TwoRayGround   ;#
radio-propagation model
set val(netif)          Phy/WirelessPhyExt         ;#
network interface type
set val(mac)            Mac/802_11Ext              ;#
MAC type
set val(ifq)               Queue/DropTail/PriQueue
;# interface queue type
set val(ll)             LL                         ;#
link layer type
set val(ant)            Antenna/OmniAntenna        ;#
antenna model
set val(ifqlen)         100                        ;#
max packet in ifq
```

```
set val(rp)             AODV                        ;#
routing protocol
set val(x)              1200
set val(y)              1200
set val(seed)           0.0
set val(tr)             coba.tr
set val(nn)             50
set val(cp)             "cbr-coba.txt"
;# connection pattern file
set val(sc)
"mobility_map_grid_lima_puluh.tcl"  ;# node movement
file.

set val(stop)           360.0

set ns_                 [new Simulator]

# open traces
set tracefd  [open coba.tr w]
$ns_ trace-all $tracefd

set topo       [new Topography]
$topo load_flatgrid $val(x) $val(y)

set god_ [create-god $val(nn)]
set chan_1_ [new $val(chan)]

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

Gambar 2.8. Potongan kode pengaturan lingkungan simulasi VANET.

2.2.3 NS-2 *Trace File*

Trace file merupakan *file* hasil simulasi dari sebuah skenario pada NS-2. Isi dari sebuah *trace file* adalah catatan dari setiap paket yang dikirim dan diterima oleh setiap *node* dalam

simulasi. Setiap jenis paket pada jaringan memiliki pola penulisan tersendiri sehingga dapat dibedakan satu sama lain dan membantu memudahkan analisa terhadap hasil simulasi. Pada Tugas Akhir ini penulis menggunakan *routing protocol* AODV sehingga jenis paket yag digunakan adalah HELLO, RREQ, RREP, RRER dari paket data.

Contoh jenis paket yang beredar di dalam simulasi ditunjukkan pada Gambar 2.9. Terlihat pada Gambar 2.9 hingga Gambar 2.11 menunjukkan contoh paket *routing protocol* AODV dari NS-2. Paket *routing protocol* selalu ditandai dengan tulisan "RTR" pada kolom keempat. Kolom ketujuh menunjukkan informasi nama *routing protocol*. Kolom kedelapan menunjukkan ukuran dari paket *routing protocol*. Kolom terakhir menunjukkan jenis paket *routing protocol*.

Gambar 2.12 menunjukkan paket data dari agen CBR (*Constant Bit Rate*). Paket data selalu ditandai dengan tulisan "AGT" pada kolom keempat. Kolom ketujuh menunjukkan informasi nama agen. Kolom kedelapan menunjukkan ukuran adari paket data.

Pola penting lainnya adalah paket yang dikirimkan selalu bertuliskan "s" dan paket yang diterima selalu bertuliskan "r" pada kolom pertama. Kolom kedua adalah waktu (dalam detik) ketika *event* tersebut terjadi. Dengan mengetahui pola yang terdapat pada *trace file*, analisa hasil simulasi dapat dilakukan.

```
s 2.556838879 _1_ RTR  --- 0 AODV 48 [0 0 0 0] ------
------ [1:255 -1:255 30 0] [0x2 1 1 [2 0] [1 4]]
(REQUEST)
```

Gambar 2.9. Contoh pola paket RREQ pada *Trace File* di NS-2.

```
s 2.570491531 _2_ RTR  --- 0 AODV 44 [0 0 0 0] -
------ [2:255 1:255 30 20] [0x4 1 [2 4] 10.000000]
(REPLY)
```

Gambar 2.10. Contoh pola paket RREP pada *Trace File* di NS-2.

```
s 11.686031372 _8_ RTR  --- 0 AODV 32 [0 0 0 0] -
------ [8:255 -1:255 1 0] [0x8 1 [2 0] 0.000000]
(ERROR)
```

Gambar 2.11. Contoh pola paket RRER pada *Trace File* di NS-2.

```
r 214.999509927 _2_ AGT  --- 409 cbr 532 [78 2 25
0] ------- [1:0 2:0 18 2] [409] 13 0
```

Gambar 2.12. Contoh pola paket data pada *Trace File* di NS-2.

2.3 *Simulation of Urban MObility* (SUMO)

Simulation of Urban MObility (SUMO) merupakan paket simulasi lalu lintas yang bersifat *open-source* dimana dimulai pengembangannya pada tahun 2001 [6]. Dan semenjak itu SUMO telah berubah menjadi sebuah simulasi lalu lintas mewah dengan kelengkapan fitur dan permodelannya termasuk kemampuan jalannya jaringan untuk membaca format yang berbeda, permintaan dengan skala besar hingga pengaturan *routing* dan semua yang berhubungan dengan penelitian pergerakan dalam lalu lintas yang berfokus pada daerah penduduk padat (*urban*). Publikasi referensi tentang SUMO ini ditulis oleh Daniel Krajzewicz pada thaun 2012.

Dalam SUMO ini terdiri beberapa macam *tools* yang digunakan dalam membangun simulasi lalu lintas pada tahap yang berbeda. Beriut penjelasan fungsi dari masing-masing *tools* yang digunakan dalam pembuatan Tugas Ahir ini:

- netconvert
  netconvert merupakan program CLI yang berfungsi untuk melakukan konversi dari peta seperti OpenSteetMap menjadi format mentah SUMO. Pada Tugas Akhir ini penulis menggunakan netconvert untuk mengkonversi peta dari OpenStreetMap [7].
- netgenerate
  netgenerate adalah *tool*yang berfungsi untuk membuat peta berbentuk seperi *grid, spider* dan bahkan *random network*. Sebelum melakukan proses netgenerate, pengguna dapat membuat traffic light pada peta dan menentukan kecepatan maksimum jalan. Hasil dari netgenerate ini berupa *file* dengan ekstensi (.net.xml). Pada Tugas Akhir ini netgenerate digunakan untuk membuat peta skenario *grid*.
- randomTrips.py
  randomTrips.py merupakan *tool* yang berfungsi membuat rute random yang akan dilalui oleh kendaraan dalam simulasi.
- route2trips.py
  route2trips.py merupakan *tool* yang berfungsi membuat detail perjalanan setiap kendaraan berdasarkan output dari randomTrips.py.
- duarouter
  duarouter merupakan *tool* yang berfungsi melakukan perhitungan rute berdasarkan definisi yang diberikan dan memperbaiki kerusakan rute.
- sumo
  sumo merupakan program yang melakukan simulasi lalu lintas berdasarkan data-data yang didapatkan dari netgenerate (skenario *grid*) atau netconvert dan randomTrips.py. Hasil simulasi dapat di-*export* ke sebuah *file* untuk nantinya dikonversi menjadi format lain.

- sumo-gui
  sumo-gui merupakan GUI untu melihat simulasi yang dilakukan oleh SUMO secara grafis.
- traceExporter.py
  traceExporter.py adalah *tool* yang berfungsi untuk mengkonversi output dari sumo menjadi format yang dapat digunakan pada *simulator* lain.

Pada Tugas Akhir ini penulis menggunaan traceExporter.py untu mengkonversi data menjadi format (.tcl) yang nantinya dapat digunaan dalam NS-2.

2.4 *OpenStreetMap* (OSM)

OpenStreetMap (OSM) merupakan sebuah proyek gabungan untu membuat contoh peta dunia yang dapat dengan bebas diubah oleh siapapun [8]. Dua buah faktor pendukung dalam pembuatan dan perkembangan OSM adalah kurangnya ketersediaan dari informasi peta mengenai sebagian besar daerah di dunia dan munculnya alat navigasi portabel yang terjangkau. OSM merupakan contoh utama dalam informasi geografis yang diberikan secara bebas. Hal yang paling penting adalah peta OSM dapat disimpan di dalam internet, dan siapapun dapat mengakses peta tersebut kapanpun, secara gratis.

OSM digambarkan sebagai "Wikipedianya Peta", yaitu tidak terlepas dari tersedianya mekanisme dimana relawan atau siapapun da[at berkontribusi langsung mengubah atau memperbarui data geogrrafis untuk membuat peta yang lebih akurat, terperinci dan *up-to-date*. Hal ini terjadi arena lebih dari dua juta yang terdaftar editor OSM yang terus-menerus membuat perubahan peta di seluruh dunia. Hampir setiap menit ada perbaruan sehingga peta seringkali lebih rinci daripada daripada peta komersial bahkan ada wilayah yang lebih lengkap daripada Google Maps.

Pada Tugas Akhir ini penulis menggunakan data yang tersedia pada OpenStreetMap untuk membuat skenario lalu lintas peta Surabaya.

2.5 *Java OpenStreetMap Editor* (JOSM)

Java OpenStreetMap Editor (JOSM) merupakan sebuah aplikasi dekstop untuk menyunting data pada *OpenStreetMap*. JOSM tidak membutuhkan koneksi Internet untuk menyunting. Aplikasi JOSM dapat diunduh pada alamat web berikut (*https://josm.openstreetmap.de/*). Penulis menggunakan aplikasi ini untuk menyunting dan erapikan potongan peta yang idunduh dari *OpenStreetMap*.

2.6 AWK

AWK merupakan sebuah adalah sebuah *command* pada Linux yang didesain untuk *text processing* dan berfungsi sebagai alat penyaringan (*filtering tools*) yang fungsinya hampir sama dengan perintah *grep*. Perintah AWK juga biasanya dipakai untuk mengolah dan analisis *file log* yang isinya sangat panjang. Perintah AWK mendukung fitur *regex* (*regular expressions*) karena fungsinya yang mirip perintah *grep*. AWK paling sering digunakan untuk mengolah, menganalisa dan menyunting sebuah *file log*. Pada Tugas Akhir ini penulis menggunakan AWK untuk memproses data yang dihasilkan dari simulasi pada NS-2 dan mendapatkan analisis mengenai *packet delivery ratio*, *end-to-end delay*, *routing overhead* dan lainnya.

2.7 802.11p - *Wireless Access Vehicular Environments* (WAVE)

Untuk memanfaatkan potensi yang ada pada sistem komunikasi antar kendaraan, saat ini IEEE sedang mengembangkan suatu perubahan standar IEEE 802.11p atau yang biasa disebut dengan *Wireless Access Vehicular Environments*

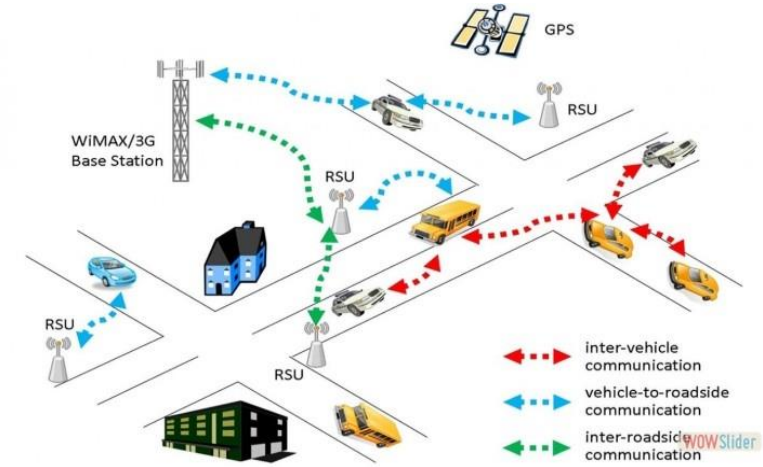
(WAVE) [13]. WAVE merupakan penyempurnaan standar IEEE 802.11 yang diperlukan untuk mendukung pengaplikasian ITS (*Intelligent Transportation Systems*). WAVE juga merupakan pengembangan sistem IEEE 802.11a dengan memperkenalkan *physical layer* dan MAC *layer* yang dapat meningkatkan sistem operasi dan aplikasi keselamatan dengan memberikan tingkat *latency* rendah. WAVE sendiri beroperasi pada band 5.9 GHz dengan menggunakan sistem *multiplexing* OFDM (*Orthogonal Frequency Division Multiplexing*) dan dapat mencapai kecepatan pentransmisian data antara 6 – 27 Mbps. WAVE terdiri dari tujuh *channel* pada frekuensi 10 MHz yang dari satu *control channel* dan enam *service channel* pada band 5.9 GHz. *Service channel* digunakan untuk *public safety* dan *private service*, sedangkan *control channel* digunakan sebagai referensi *channel* untuk membangun link komunikasi antara RSU (*Road – Side Unit*) dan OBU (*On – Board Unit*). *Control channel* digunakan oleh OBU dan RSU untuk *broadcast application service*, *warning message* dan *safety* status *message*. Dari penjelasan di atas dapat diketahui bahwa aplikasi utama dari IEEE 802.11p adalah untuk sistem komunikasi antar kendaraan dengan sistem komunikasi yang digunakan adalah DSRC (*Dedicated Short Range Communication*). Pada Tugas Akhir ini penulis menguji kinerja dan performa dari WAVE ini pada *routing protocol* AODV menggunakan NS-2.

2.8 *Vehicular Ad hoc Network* (VANET)

Vehicular Ad hoc Network (VANET) adalah sebuah jaringan terorganisir yang dibentuk dengan menghubungkan kendaraan dan RSU (*Roadside Unit*) disebut *Vehicular Ad Hoc Network* (VANET), dan RSU lebih lanjut terhubung ke jaringan *backbone* berkecepatan tinggi melalui koneksi jaringan. Kepentingan peningkatan baru-baru ini telah diajukan pada aplikasi melalui V2V (*Vehicle to Vehicle*) dan V2I (*Vehicle to Infrastructure*) komunikasi, dimana bertujuan untuk meningkatkan keselamatan

mengemudi dan manajemen lalu lintas sementara menyediakan pengemudi dan penumpang dengan akses internet.

Dalam VANETs, RSUs dapat memberikan bantuan dalam menemukan fasilitas seperti restoran dan pompa bensin, dan mem-*broadcast* pesan yang terkait seperti (maksimum kurva kecepatan) pemberitahuan untuk memberikan pengendara informasi. Sebagai contoh, sebuah kendaraan dapat berkomunikasi dengan lampu lalu lintas cahaya melalui V2I komunikasi, dan lampu lalu lintas dapat menunjukkan ke kendaraan ketika keadaan lampu ke kuning atau merah. Ini dapat berfungsi sebagai tanda pemberitahuan kepada pengemudi dan akan sangat membantu para pengendara ketika mereka sedang berkendara selama kondisi cuaca musim dingin atau di daerah asing. Hal ini dapat mengurangi terjadinya kecelakaan.



Gambar 2.13. Ilustrasi VANET.
Source :
http://adrianlatorre.com/projects/pfc/img/vanet_full.jpg

Melalui komunikasi V2V, pengendara bisa mendapatkan informasi yang lebih baik dan mengambil tindakan awal untuk

menanggapi situasi yang abnormal. Untuk mencapai hal ini, suatu OBU secara teratur menyiarkan pesan yang terkait dengan informasi dari posisi pengendara, waktu saat ini, arah mengemudi, kecepatan, status rem, sudut kemudi, lampu sen, percepatan/perlambatan, kondisi lalu lintas. Dari masalah yang disebabkan oleh berbagai macam karakteristik tersebut, periset mengajukan barbagai isu riset dalam bidang *routing*, diseminasi data, data *sharing* dan sekuritas. Protokol yang digunakan untuk VANET saat ini adalah protokol-protokol yang didesain untuk jaringan *Mobile Ad hoc Network* (MANET). Protokol-protokol tersebut tidak dapat menyelesaikan permasalahan dari karakteristik unik VANET sehingga tidak ccok untuk komunikasi V2V dalam VANET ini. Ilustrasi VANET dapat dilihat pada Gambar 2.13.

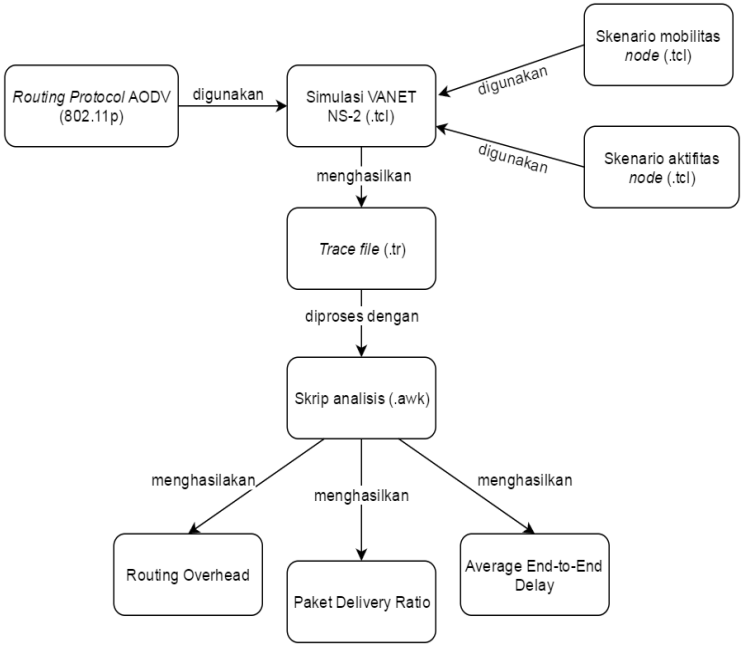
Dalam Tugas Akhir ini penulis akan mengimplementasikan 802.11p pada *routing protocol* AODV dan menguji kinerja dalam lingkungan VANET.

BAB III
DESAIN DAN PERANCANGAN

Perancangan merupakan bagian penting dari pembuatan perangkat lunak yang isinya berupa perencanaan-perencanaan secara teknis aplikasi yang dibuat. Sehingga bab ini secara khusus akan menjelaskan perancangan sistem yang dibuat dalam Tugas Akhir ini. Berawal dari deskripsi umum aplikasi hingga perancangan proses, alur dan implementasinya.

3.1 Deskripsi Umum

Pada Tugas Akhir ini akan dilakukan implementasi dan analisis dari 802.11p pada *routing protocol* AODV pada NS-2. Diagram rancangan simulasi dapat dilihat pada Gambar 3.1. Dimana terdapat 2 (dua) jenis skenario yang digunakan sebagai perbandingan pengukuran lalu lintas kota Surabaya, yaitu skenario *grid* dan skenario riil. Pada skenario *grid* peta dibuat dengan bantuan aplikasi SUMO. Sedangkan dalam skenario riil peta yang diambil langsung keadaan saat pengambilan menggunakan OpenStreetMap dengan bantuan *editor* JOSM. Setelah *file* peta sudah berbentuk, dilakukan simulasi lalu lintas dengan SUMO. Hasil simulasi SUMO digunakan untuk mencari kinerja dari 802.11p pada simulasi protokol AODV pada NS-2. Kemudian hasil simulasi dari NS-2 dianalisis dengan menggunakan skrip AWK untuk menghitung metrik analisis berupa *routing overhead*, *packet delivery* dan *average end-to-end delay*. Perhitungan metrik analisis bertujuan untuk mengukur performa dari protokol AODV untuk melihat performa dari 802.11p.



Gambar 3.1. Diagram Alur Rancangan Simulasi.

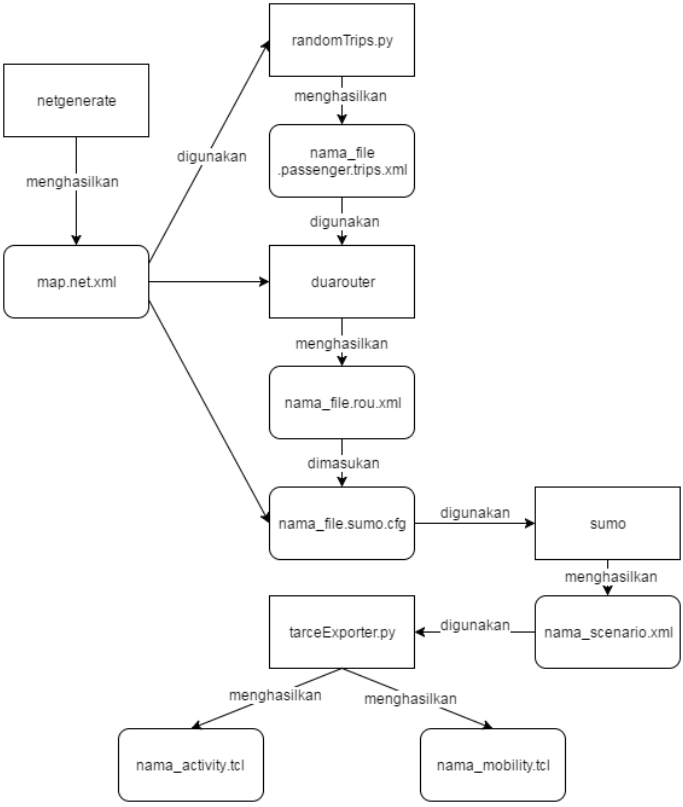
3.2 Perancangan Skenario *Grid*

Dalam pembuatan peta *grid* diawali dengan menntukan panjang jalan dan jumlah *vertex*. Secara *default*, peta *grid* akan berbentuk seperti persegi empat dengan dilamnya terdapat beberapa *vertex* tergantung jumlah yang akan di-*input*. Alur pembuatan peta *grid* dapat dilihat pada Gambar 3.2.

Panjang jalan dan jumalh *vertex* yang udah ditentutkan akan dimasukkan sebagai argumen untuk netgenerate. Secara opsional, melalui argumen netgenerate dapat ditentukan pula batas maksimal kecepatan untuk setiap jalan pada map. Kemudian hasil dari netgenerate ini digunakan sebagai argumen untuk randomTrips.py. Program randomTrips.py ini berfungsi untuk mendefinsikan rute

perjalanan dari seluruh *node*. Hasil yang diproses dari randomTrips.py ini selanjutnya diproses oleh duarouter untuk memperbaiki masalah konektivitas rute (jika ada). Setelah itu dilakukan simulasi lalu lintas dengan SUMO.

Hasil dari simulasi tersebut di-*export* ke dalam format (.tcl) agar dapat diproses oleh NS-2 nantinya. Dalam porses meng-*export* ini semua dilakukan oleh traceExporter.py yang ada pada SUMO. Hasilnya berupa *file* yang berisi mobilitas dari setiap *node* (mobility.tcl) dan informasi *lifetime* dari setiap *node* (activity.tcl).



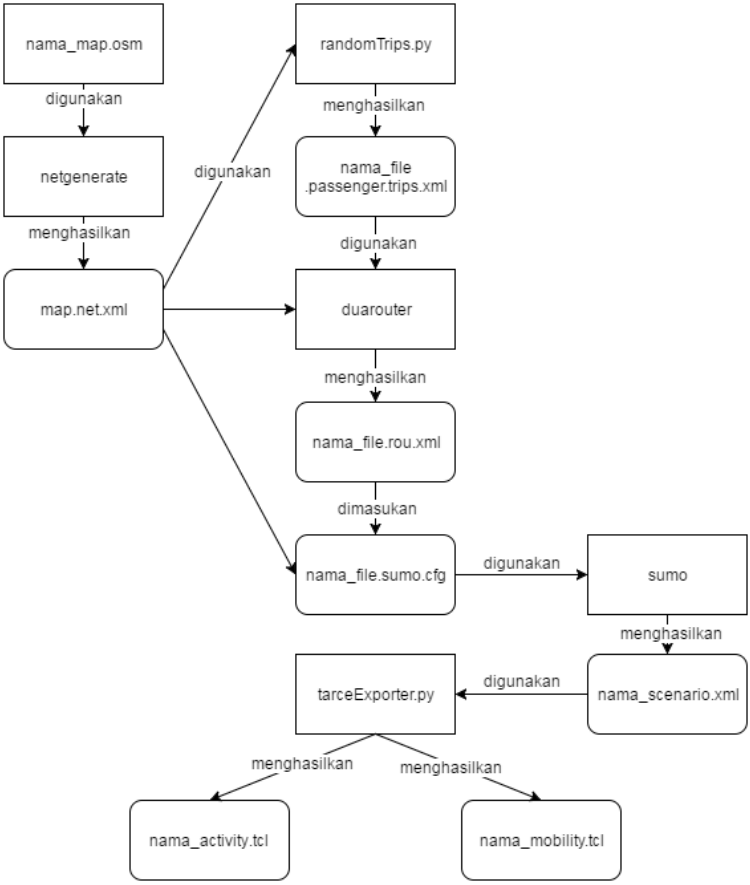
Gambar 3.2. Alur Pembuatan Skenario *Grid*.

3.3 Perancangan Skenario Riil

Perancangan skenario riil diawali dengan pemilihan daerah yang akan digunakan sebagai model untuk simulasi. Daerah yang dipilih dalam map bebas. Setelah mendapatkan daerah yang diinginkan, unduh daerah tersebut dengan menggunakan OpenStreetMap, dengan menggunakan fungsi *export* pada OpenStreetMap. Setelah diunduh, lakukan pebaikan dan pengaturan agar terlihat semirip mungkin dengan dunia nyata dengan bantuan JOSM. Dengan JOSM ini dapat menghapus, menambahkan, mengubah berbagai variabelyang ada, seperti tipe jalan, lampu lalu lintas, bangunan dan lainnya. Lakukan supaya peta yang diunduh menjadi suatu daerah yang tertutup (tidak ada jalan yang putus atau buntu).

Setelah potongan peta dirapikan, selanjutnya buat sebuah *file type* yang mendefinisikan spesifikasi batasan lalu lintas pada peta yang diunduh tadi, seperti batas kecepatan pada suatu jalan tertentu dan lain-lain. Kemudian peta yang telah diunduh dikonversi dengan batuan netconvert dari SUMO berdasarkan *file type* yang telah dibuat. Hasil dari konversi tersebut kemudian digunakan untuk membuat *file* rute pergerakan kendaraan melalui randomTrips.py dan duarouter dari SUMO. Selanjutnya hasil dari konversi tersebut berupa *file* peta dan *file* yang berisi rute yang nantinya akan digunakan untuk simulasi pada SUMO.

Dan pada proses konversi terakhir, hasil dari simulasinya di-*export* ke dalam format yang dapat diproses oleh NS-2 melalui traceExporter.py. Hasil akhir yang diharapkan adalah berupa *file* yang berisi akitifitas atau informasi *lifetime* dari setiap *node* (nama_activity.tcl) dan *file* yang berisi mobilitas dari setiap *node* (nama_mobility.tcl). Untuk lebih jelas dalam melakukan konversi ini dapat dilihat pada Gambar 3.3 dibawah ini yang merupakan alur pembuatan skenario riil.



Gambar 3.3. Alur Pembuatan Skenario Riil.

3.4 Perancangan Metrik Analisis

Berikut ini adalah perancangan beberapa metrik yang akan dijadikan analisis dalam Tugas Akhir ini :

3.4.1 *Routing Overhead* (RO)

Routing overhead merupakan jumlah paket *routing control* yang ditransmisikan selama simulasi terjadi [9]. Paket kontrol yang dihitung adalah jumlah *Route Request* (RREQ), *Route Reply* (RREP) dan *Route Error* (RRER). Rumus dari *routing overhead* dapat dilihat pada Persamaan 3.1.

$$RO = RREQ_{sent} + RREP_{sent} + RRER_{sent} \quad (3.1)$$

3.4.2 *Average End-to-End Delay*

Average end-to-end delay merupakan waktu rata-rata dari setiap paket ketika sampai di tujuan. Semua paket, termasuk *delay* yang dikarenakan oleh paket *routing*, ini juga akan diperhitungkan dalam memperoleh nilai akhir. Paket yang akan dimasukkan ke dalam perhitungan hanya paket yang berhasil sampai tujuan. *Average end-to-end delay* dihitung menggunakan Persamaan 3.2, di mana *i* adalah nomor paket yang berhasil sampai di tujuan $t_{received}[i]$ adalah waktu ketika paket *i* dikirim, sedangkan $t_{sent}[i]$ adalah waktu ketika paket *i* diterima dan *pktCounter* adalah jumlah paket yang berhasil sampai di tujuan.

$$Delay = \frac{\sum_{i=0}^{n} t_{received}[i] - t_{sent}[i]}{pktCounter} \quad (3.2)$$

3.4.3 *Packet Delivery Ratio* (PDR)

Packet deliery ratio merupakan perbandingan dari jumlah paket data yang dikirim dengan paket data yang diterima. *Packet*

delivery ratio dihitung menggunakan Persamaan 3.3, di mana *received* adalah jumlah paket data yang diterima dan *sent* adalah jumlah paket data yang dikirim [10].

$$PDR = \frac{Data_{\ received}}{Data_{sent}} \tag{3.3}$$

(Halaman ini sengaja dikosongkan)

BAB IV
IMPLEMENTASI

Bab ini membahas implementasi perancangan perangkat lunak dari aplikasi ini yang merupakan penerapan data, kebutuhan, dan alur sistem yang mengacu pada desain dan perancangan yang telah dibahas sebelumnya.

4.1 Lingkungan Implementasi

Pada lingkungan implementasi ini terdapat dua lingkungan, yaitu pada skenario *grid* dan pada skenario riil. Dan masing-masing dari lingkungan skenario ini akan dijelaskan bagaimana cara pembuatan dan implementasinya pada bab ini.

4.1.1 Skenario *Grid*

Pada skenario *grid* ini dijelaskan bagaimana skenario yang digunakan untuk membuat suatu rute yang berbentuk seperti jala-jala atau *grid* dengan bantuan *tool* netgenerate yang telah disediakan oleh SUMO [11]. Skenario *grid* ini dibuat dengan panjang jalan 200 meter dan luas peta pada skenario *grid* ini adalah 1200 meter x 1200 m. Jumlah ujung persimpangan antara jalan vertikal dan horizontal sebanyak 6 ujung x 6 ujung. Kecepatan kendaraan yang diperbolehkan untuk melintasi jalan pada skenario ini diatur dengan kecepatan maksimal 15 m/s.

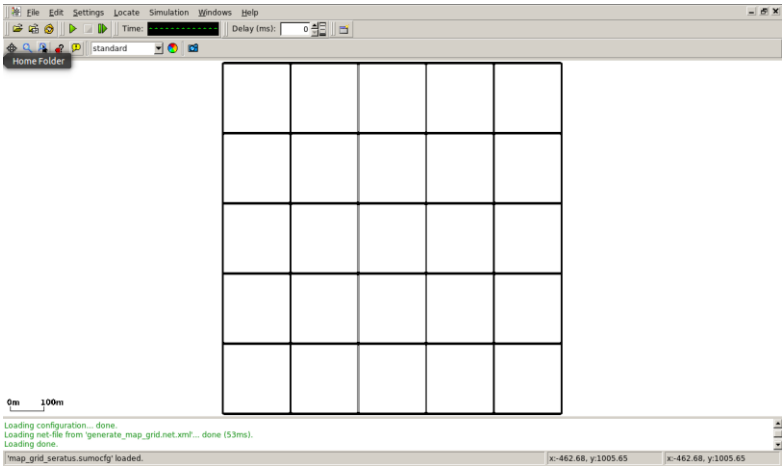
Untuk membuat suatu peta *grid* dengan spesifikasi yang dijelaskan diatas, dapat digunakan perintah seperti pada Gambar 4.1 dibawah ini:

```
$ netgenerate --grid --grid.number=6
   --grid.length=200
   --default.speed=15 --tls.guess=1
   --output-file=(nama_file_map_grid).net.xml
```

Gambar 4.1. Perintah untuk Membuat Peta *Grid*.

Pada *sript* diatas terdapat keterangan "–tls.guess=1" ini adalah *script* untuk membuat suatu *traffic light* pada suatu simulasi. Script ini bernilai default *false* atau 0, maka dari itu diisi dengan nilai 1 atau *true* agar simulasi yang dibuat ada *traffic light* yang berjalan secara otomatis dan dikendalikan penuh oleh simulator. Pada proses ini menghasilkan suatu *file* berekstensi (.net.xml) yang nantinya *file* ini akan menjadi *file* patokan yang akan digunakan untuk beberapa kali percobaan dalam simulasi ini. Adapun hasil dari *script* diatas menghasilkan suatu peta *grid* yang dapat dilihat pada Gambar 4.2 berikut.

Selanjutnya, setelah peta terbentuk dilakukan pembuatan posisi dan jumlah *node* untuk setiap kendaraan secara acak melalui modul randomTrips.py yang dimiliki SUMO. Pada proses ini terdapat beberapa variabel pendukung untuk simulasi yang akan dibuat seperti dapat dilihat pada Gambar 4.3. Pada modul ini terdapat variabel pendukung untuk membantu membuat simulasi sesuai apa yang diinginkan. Yaitu misal pada variabel -e diisi dengan jumlah kendaraan yang diinginkan pada simulasi dan juga pada opsi --intermediate diisi dengan nilai yang besar agar setiap kendaraan memiliki banyak rute alternatif sehingga setiap kendaraan dapat dipastikan aktif hingga simulasi mobilitas berakhir, tapi dapat diisi dengan nilai random karena pada nilai *default*-nya nilai dari opsi ini sudah besar.



Gambar 4.2. Hasil peta *grid* dari Perintah *netgenerate*.

pada variabel –seed diisi dengan nilai *random* pula karena opsi ini menghasilkan angka acak dari jumlah *node* yang digunakan dalam simulasi. Lalu ada adisional parameter untuk menghasilkan kendaraan secara acak pada posisi awal muncul dalam simulasi, yaitu dengan variabel –trips-attributes. Serta yang paling penting dari keterangan tersebut dapat dilihat masing-masing perintah dan tugas yang akan dilakukan untuk membuat posisi dan jumlah *node* pada skenario *grid*. Pada proses ini menghasilkan suatu *file* berekstensi (.passenger.trips.xml) dimana *file* ini nantinya akan akan digunakan untuk proses selanjutnya.

Setelah posisi dan jumlah dari *node* didefinisikan, selanjutnya dilakukan pembuatan rute yang akan digunakan oleh kendaraan menggunakan perintah pada Gambar 4.4 berikut. Pada perintah ini menggunakan modul duarouter yang terdapat pada SUMO. Inti tujuan dari modul ini adalah untuk membangun sebuah rute untuk kendaraan sesuai definisi yang diinginkan, menghitung rute selama simulasi berlangsung dan memperbaiki masalah konektivitas pada rute, maka dari itu terdapat variabel –ignore-errors dan --repair.

Untuk menghasilkan suatu *file* yang diinginkan maka dari itu dalam perintah ini dimasukkan dua *file* sebelumnya yang dihasilkan oleh perintah netgenerate dan randomTrips.py untuk selanjutnya diproses dan menghasilkan sebuah *file* berekstensi (.rou.xml), *file* ini nantinya akan digunakan dalam proses pembuatan *file* SUMO dengan ekstensi *file* (.sumocfg).

```
$ python $SUMO_HOME/tools/randomTrips.py -n \
  (nama_file).net.xml -e $num_nodes \
  --seed=$RANDOM --fringe-factor 5.0 \
  --intermediate=$RANDOM \
  --trip-attributes='departLane="best"
     departPos="random_free"' -o \
  (nama_file).passenger.trips.xml;
```

Gambar 4.3. Perintah untuk Membuat Posisi dan Jumlah *Node* dalam Skenario.

```
$ SUMO_HOME/bin/duarouter \
  -n $SAVEDIR/(nama_file).net.xml \
  -t $SAVEDIR/(nama_file).passenger.trips.xml \
  -o $SAVEDIR/(nama_file).rou.xml \
  --ignore-errors --repair;
```

Gambar 4.4. Perintah untuk Membuat Rute *Node*.

Selanjutnya dilakukan pembuatan *file* dengan ekstensi (.sumocfg) dimana *file* ini nanti yang akan digunakan sebagai argumen pada perintah SUMO. Adapun fungsi dari dibuatnya *file* .sumocfg ini digunakan untuk mendefinisikan lokasi *file* (.net.xml) dan *file* (passenger.trips.xml) serta durasi yang dibutuhkan dalam simulasi. Dan isi dari perintah *file* .sumocfg ini dapat dilihat pada Gambar 4.5.

```
<?xml version="1.0" encoding="UTF-8"?>

<configuration>

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

    <time>
            <begin value="0"/>
            <end value="360"/>
    </time>

</configuration>
```

Gambar 4.5. Isi dari Perintah *file* (.sumocfg).

Selain itu untuk melihat visualisasi simulasi lalu lintas pada *file* (.sumocfg) ini menggunakan perintah, yaitu sumo-gui. sumo-gui ini merupakan program GUI yang disediakan oleh SUMO dan dapat dilihat pada Gambar 4.6. Sedangkan untuk melihat cuplikan visualisasi pergerakan kendaraan yang dihasilkan oleh perintah sumo-gui sebelumnya, dapat dilihat pada Gambar 4.7. Kedua gambar tersebut dapat dilihat dibawah ini. Setelah selesai dibuat, *file* (.sumocfg) ini harus disimpan dalam direktori yang sama dengan *file* (.net.xml) dan *file* (passenger.trips.xml).

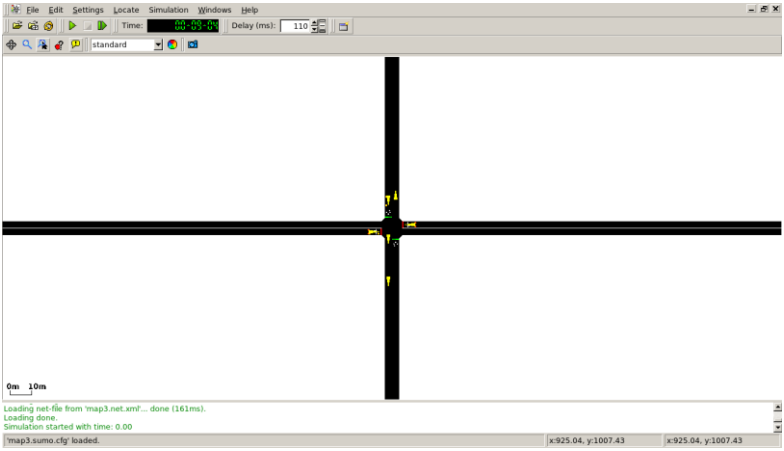
Setelah semua perintah sebelumnya sudah dilakukan, maka selanjutnya yang dilakukan adalah perintah untuk melakukan simulasi tanpa menggunakan GUI untuk menghasilkan *file* dengan ekstensi (.xml). Perintah tersebut dapat dilihat pada Gambar 4.8 berikut.

Dimana nanti dalam proses ini menghasilkan suatu *file* yang akan diekspor untuk dijalankan di NS-2, karena dalam NS-2 tidak

bisa menjalankan *file* berekstensi (.xml). Proses ini sangat penting karena dalam proses ini menghasilkan *file* akhir untuk simulasi yang dilakukan. *File* akhir dari proses simulasi dari SUMO ini dapat dilihat pada Gambar 4.9.

```
$ sumo-gui (nama_file_sumocfg).sumocfg
```

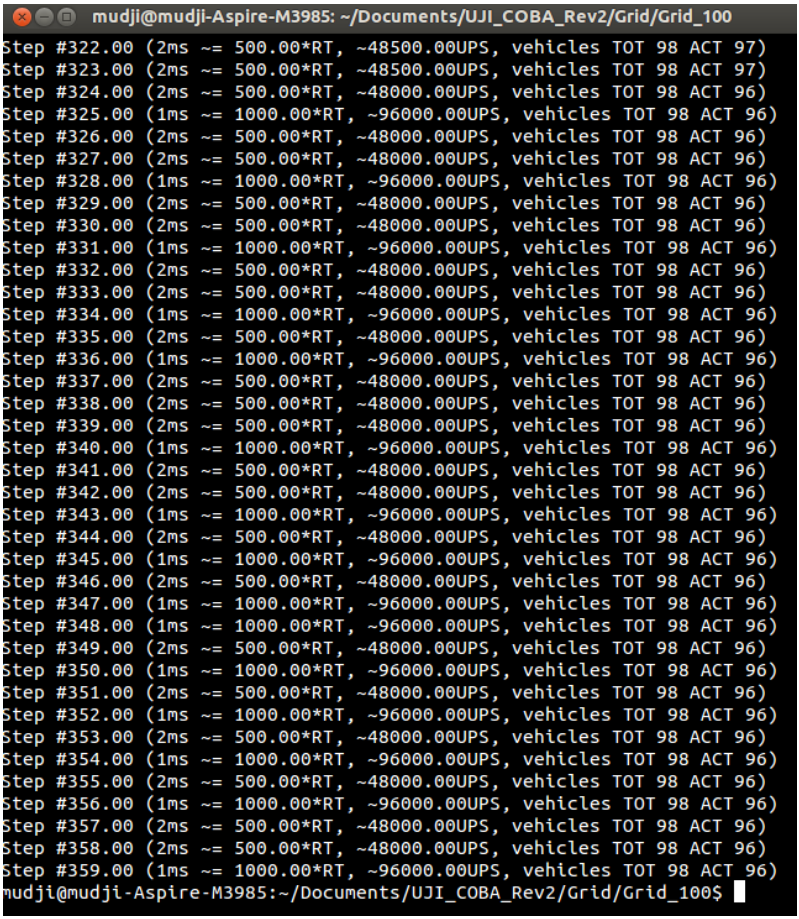
Gambar 4.6. Perintah untuk Melihat Visualisasi Simulasi pada *file* .sumocfg.



Gambar 4.7. Cuplikan Visualisasi Pergerakan Kendaraan dengan SUMO-GUI.

```
$ sumo -c (nama_file_sumocfg).sumocfg \
  --fcd-output (nama_file_hasil_simulasi).xml
```

Gambar 4.8. Perintah untuk Melakukan Simulasi Lalu Lintas dengan SUMO.



Gambar 4.9. Hasil Akhir Simulasi dari Perintah SUMO.

Kemudian agar *file* yang dihasilkan tersebut dapat digunakan pada NS-2, maka hasil dari proses sumo diatas dikonversi kedalam format yang dapat dipahami oleh NS-2 yaitu sebuah *file* (.tcl). Maka dari itu diperlukan sebuah modul dari SUMO, yaitu traceExporter.py. Pada modul ini inputnya adalah *file* hasil dari

proses sumo sebelumnya, yaitu *file* (.xml). dan menghasilkan suatu *file* (.tcl) yang nantinya *file* ini akan digunakan dalam skrip NS untuk menjalankan seluruh simulasi yang digunakan.

Perintah untuk melakukan konversi tersebut dapat dilihat pada Gambar 4.10 dibawah ini.

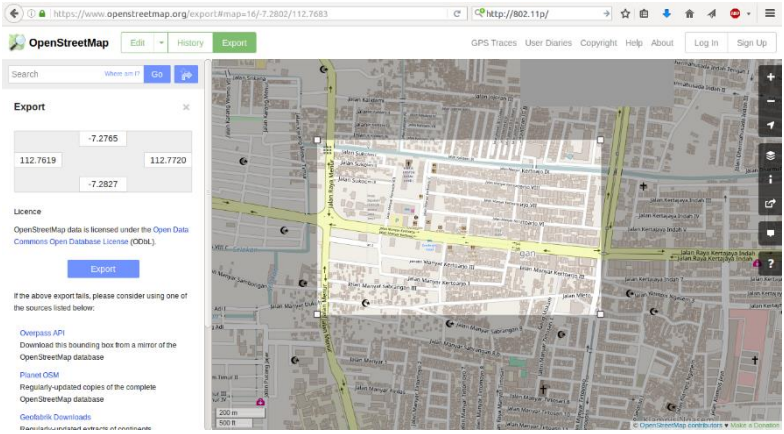
```
$ python $SUMO_HOME/tools/traceExporter.py
   --fcd-input(nama_file_hasil_simulasi).xml \
   --ns2mobility-output
   (nama_file_mobility_grid).tcl \
   --ns2activity-output
   (nama_file_activity_grid).tcl;
```

Gambar 4.10. Perintah untuk Mengkonversi Hasil dari Proses SUMO ke dalam NS-2.

Setelah semua proses selesai, selanjutnya dijalankan program NS yang berisikan skrip NS dan dengan parameter pada *file* mobility_grid.tcl) yang telah dihasilkan pada proses sebelumnya. Seperti terlihat pada Gambar 4.23.

4.1.2 Skenario Riil

Pembuatan peta pada skenario riil ini berbeda dengan cara pembuatan peta pada skenario *grid*. Pada skenario riil ini pembuatan suatu rute dihasilkan langsung dari sebuah peta asli. Pembuatannya menggunakan OpenSteetMap. Peta yang digunakan untuk membuat skenario riil ini adalah peta Surabaya, lebih tepatnya daerah sekitar Manyar. Untuk mengambil peta yang akan digunakan dalam skenario ini, bisa dengan cara membuat area seleksi sesuai dengan wilayah yang diinginkan atau bisa juga secara menyeluruh, sesuai dengan kebutuhan. Setelah area seleksi yang diinginkan sudah terpilih, kemudian dilakukan *Export* pada OpenStreetMap. Dengan *Export* ini area yang diseleksi secara otomatis akan tersimpan dalam bentuk *file* (.osm), seperti terlihat pada Gambar 4.11 di bawah ini.



Gambar 4.11. Proses Pengambilan Peta dari OpenStreetMap.

Kemudian, peta yang telah diekspor dari OpenStreetMap selanjutnya disunting dengan menggunakan program JOSM. Perintah untuk menggunakan JOSM, seperti terlihat pada Gambar 4.12 di bawah. Dengan proses penyuntingan ini diharapkan jalan-jalan yang ada dalam peta bisa diperbarui dengan menghapus atau menambahkan jalan, sehingga tidak ada jalan buntu ataupun jalan terputus dan kepadatan kendaraan yang ada pada rute tetap stabil sehingga tidak ada daerah dalam peta yang jarang dikunjungi oleh kendaraan.

```
$ java –jar $SAVE_PATH/josm-tested.jar
```

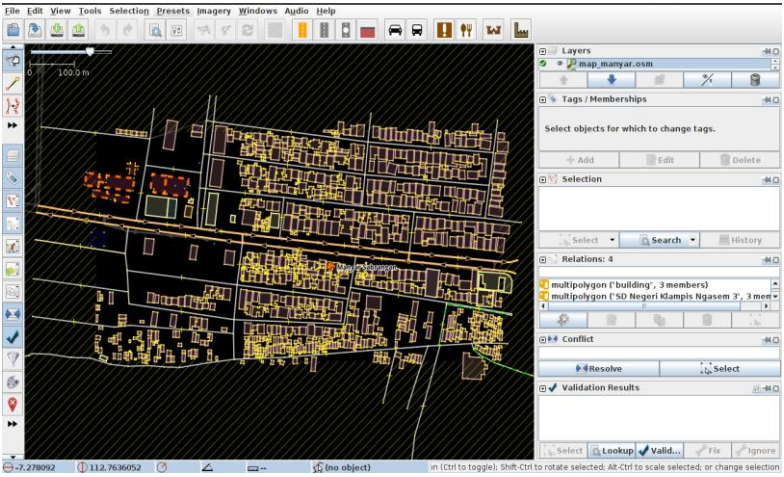
Gambar 4.12. Perintah untuk Melakukan Penyuntingan pada *file* (.osm).

Berikut merupakan tampilan proses dari penyuntingan peta hasil *export* dari OpenStreetMap yang tersimpan otomatis dengan format

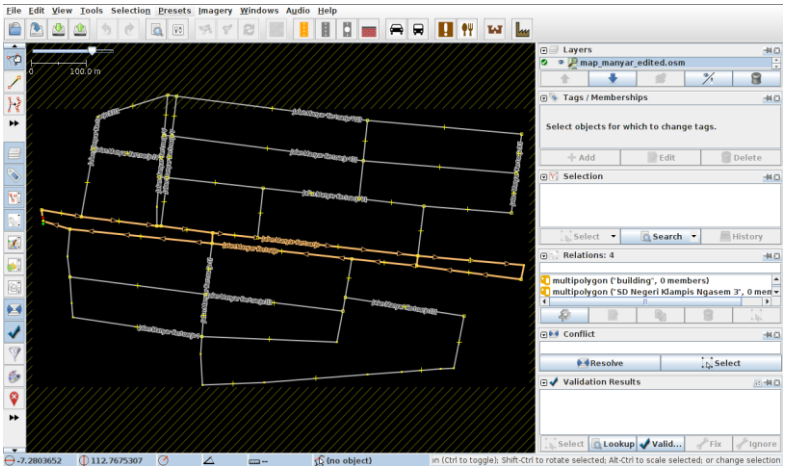
(.osm) menggunakan JOSM yang dapat dilihat pada Gambar 4.13 dan Gambar 4.14.

Setelah proses penyuntingan peta selesai, selanjutnya peta tersebut dikonversi ke dalam format (.net.xml) menggunakan *tool* netconvert yang disediakan oleh SUMO. *Tool* netconvert ini bertugas untuk mengimpor suatu jalan pada suatu jaringan digital dari sumber yang berbeda dan menghasilkan suatu jalan baru pada suatu jaringan yang dapat digunakan untuk *tool*s lain dari SUMO. Jadi, secara garis besar, *tool* ini berfungsi untuk mengkonversi suatu *file* agar dapat dijalankan dan diproses oleh *tools* lainnya.

Adapun parameter pada skenario riil ini dibuat dengan luas yang tidak berbeda jauh dengan senario *grid* agar hasil yang diinginkan maksimal, yaitu luas peta pada skenario riil ini adalah 1200 meter x 800 meter. Perintah konversi dapat dilihat pada Gambar 4.15.



Gambar 4.13. Proses Penyuntingan Peta dengan Menggunakan JOSM.



Gambar 4.14. Hasil Peta setelah Penyuntingan dengan Menggunakan JOSM.

```
$ netconvert --try-join-tls --osm-files
   (nama_file).osm --output-file
   (nama_file).net.xml --remove-edges.isolated
   --type-files specification.typ.xml
```

Gambar 4.15. Perintah Konversi dari *file* (.osm) ke (.net.xml).

Setelah proses penyuntingan pada peta tersebut sudah selesai, tahap selanjutnya sama dengan proses pembuatan posisi dari *node* pada skenario *grid*, yaitu mulai dari menggunakan randomTrips.py untuk pembuatan *file* .net.xml hingga mengonversi *file* untuk dikonversi ke dalam format *file* .tcl sehingga dapat diproses oleh NS-2, seperti terlihat pada Gambar 4.3 hingga Gambar 4.9 dan pada akhirnya nanti diporses oleh NS seperti pada Gambar 4.23.

4.2 Implementasi Metrik Analisis

Hasil dari simulasi skenario dalam NS-2 adalah sebuah *trace file*. *Trace file* yang berkekstensi .tr digunakan sebagai bahan analisis untuk pengukuran performa dari protokol yang disimulasikan. Dalam Tugas Akhir ini, terdapat 3 (empat) metrik yang akan menjadi parameter analisis, yaitu *Routing Overhead* (RO), *Average End-to-End Delay* dan *Packet Delivery Ratio* (PDR). Implementasi dari tiap metrik tersebut menggunakan suatu *text-processing* AWK dan dijelaskan seperti berikut.

4.2.1 Implementasi *Routing Overhead*

Pada implementasi *Routing Overhead* ini hasil yang didapat dengan cara menyaring baris pada *trace file* yang mengandung *REQUEST*, *REPLY* dan *ERROR*. Setiap ditemukan tiga *string* tersebut, dilakukan *increment* untuk menghitung jumlah paket *routing* yang beredar dalam jaringan dan dilakukan perhitungan RO dengan Persamaan 3.1. Kode implementasi dari perhitungan *routing overhead* dapat dilihat pada Lampiran A.4.

Contoh perintah untuk memanggil skrip AWK untuk menganalisis *trace file* dan contoh hasilnya setelah menggunakan skrip AWK dapat dilihat pada Gambar 4.16 dan Gambar 4.17.

```
awk –f ro.awk (nama_file_trace).tr
```

Gambar 4.16. Perintah untuk Menjalankan Skrip AWK Perhitungan *Routing Overhead*.

```
Overhead : 3143
```

Gambar 4.17. Contoh Hasil dari Proses Skrip AWK pada *Routing Overhead*.

4.2.2 Implementasi Average End-To-End Delay

Implementasi Dalam pembacaan baris *trace file* untuk perhitungan *average end-to-end delay* terdapat 5 (lima) kolom yang haus diperhatikan, yaitu kolom pertama yang berisi penanda *event* pengiriman atau penerimaan, kolom kedua yang berisi waktu terjadinya *event*, kolom keempat yang berisi informasi *layer* komunikasi paket, kolom keenam yang berisi ID paket dan kolom ketujuh yang berisi tipe paket.

Dari setiap paket yang ada terdapat *delay* yang dihitung dengan cara mengurangi waktu penerimaan dengan waktu pengiriman berdasarkan ID paket. Hasil pengurangan waktu dari masing-masing paket dijumlahkan dan dibagi dengan jumlah paket CBR yang ID-nya terlibat dalam perhitungan dalam mencari nilai *delay* tersebut seperti pada Persamaan 3.2. Kode implementasi dari perhitungan *average end-to-end delay* dapat dilihat pada lampiran A.3.

Contoh perintah untuk memanggil skrip AWK untuk menganalisis *trace file* dan contoh hasilnya setelah menggunakan skrip AWK dapat dilihat pada Gambar 4.18 dan Gambar 4.19.

```
awk –f delay.awk (nama_file_trace).tr
```

Gambar 4.18. Perintah Untuk Menjalankan Skrip AWK Perhitungan Average End-To-End Delay.

```
Delay : 0.7338
```

Gambar 4.19. Contoh Hasil dari Proses Skrip AWK pada *Average End-to-End Delay*.

4.2.3 Implementasi *Packet Delivery Ratio*

Implementasi nilai dari *Packet Delivery Ratio* didapatkan dari perhitungan setipa baris yang terjadi *event* pengiriman dan

penerimaan paket data yang dikirim melalui agen pada *trace file*. Pada lampiran A.2 ditunjukkan cara menyaring data yang dibutuhkan untuk perhitungan PDR. Pada skrip tersebut dilakukan penyaringan pada setiap baris yang terdapat *string* AGT karena *event* tersebut berhubungan dengan paket data. Paket data yang dikirim dan diterima dibedakan dari kolom pertama pada baris yang telah disaring. Setelah pembacaan setiap baris dalam *trace file* selesai dilakukan, selanjutnya dilakukan perhitungan PDR dengan persamaan 3.3.

Contoh perintah untuk memanggil skrip AWK untuk menganalisis *trace file* dan contoh keluarannya dapat dilihat pada Gambar 4.20 dan Gambar 4.21.

```
awk –f delay.awk (nama_file_trace).tr
```

Gambar 4.20. Perintah Untuk Menjalankan Skrip AWK Perhitungan PDR.

```
Sent  : 721
Recv  : 499
Ratio : 0.692
```

Gambar 4.21. Contoh Hasil dari Proses Skrip AWK pada *Packet Delivery Ratio*.

4.3 Implementasi Simulasi pada NS-2

Sebelum melakukan simulasi pada NS-2 ini dibutuhkan suatu koneksi CBR dengan menggunakan *Agent* UDP. Dimana koneksi UDP ini merupakan konfigurasi antara *node* kesatu dan kedua. Interval pengiriman paket data dilakukan setiap satu detik dengan besar paket data 512 byte dan maksimal pengiriman paket 10.000. Seperti terlihat pada Gambar 4.22 dibawah ini.

```
# nodes: 50, max conn: 10, send rate: 0.5, seed: 1
#
# 1 connecting to 2 at time 2.5568388786897245
#
set udp_(0) [new Agent/UDP]
$ns_ attach-agent $node_(1) $udp_(0)
set null_(0) [new Agent/Null]
$ns_ attach-agent $node_(2) $null_(0)
set cbr_(0) [new Application/Traffic/CBR]
$cbr_(0) set packetSize_ 512
$cbr_(0) set interval_ 0.5
$cbr_(0) set random_ 1
$cbr_(0) set maxpkts_ 10000
$cbr_(0) attach-agent $udp_(0)
$ns_ connect $udp_(0) $null_(0)

$ns_ at 2.5568388786897245 "$cbr_(0) start"
#
```

Gambar 4.22. *File* cbr-coba.txt.

Untuk melakukan simulasi VANET pada lingkungan NS-2 dibutuhkan sebuah *file* OTcl yang berisikan deskripsi tentang lingkungan simulasi yang digunakan untuk simulasi. *File* tersebut berisikan pengaturan untuk setiap *node* dan beberapa *event* yang perlu untuk diatur agar berjalan pada waktu tertentu. Contoh potongan skrip pengaturan *node*.

Pada Gambar 4.23. dibawah ini ditunjukkan skrip konfigurasi awal parameter-parameter yang diberikan untuk menjalankan simulasi VANET pada lingkungan NS-2.

```
set val(chan)        Channel/WirelessChannel;
set val(prop)        Propagation/TwoRayGround;
set val(netif)       Phy/WirelessPhyExt;
set val(mac)         Mac/802_11Ext;
set val(ifq)         Queue/DropTail/PriQueue;
set val(ll)          LL;
set val(ant)         Antenna/OmniAntenna;
set val(ifqlen)      100;
```

```
set val(rp)          AODV;
set val(x)           1200;
set val(y)           1200;
set val(seed)        0.0;
set val(tr)          coba.tr;
set val(nn)          50;
set val(cp)          "cbr-coba.txt";
set val(sc)          "mobility_map_grid_lima_puluh.tcl";
set val(stop)        360.0

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

Gambar 4.23. Potongan Skrip Pengaturan *Trace File* dan *Node* pada lingkungan NS-2.

Penjelasan dari pengaturan *node* dpat dilihat pada Tabel 4.1. pengaturan lainnya yang dilakukan pada *file* tersebut antara lain lokasi penyimpanan *trace file*, lokasi *file* mobilitas *node*, *node* tujuan dan *node* sumber serta konfigurasi *event* pengiriman paket data. Kode implementasi skenario yang digunakan Tugas Akhir ini dapat dilihat pada lampiran A.1.

Tabel 4.1. Penjelasan dari Parameter Pengaturan *Node* pada *file* NS.

Parameter	Value	Penjelasan
llType	LL	Menggunakan link-layer standar.

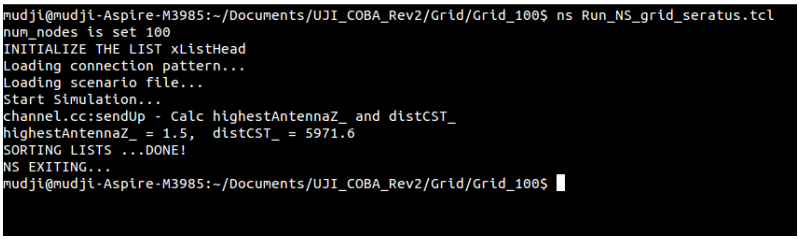
macType	Mac/802_11Ext	Menggunakan tipe MAC 802.11 karena komunikasi data bersifat wireless.
ifqType	Queue/DropTail/ PriQueue	Menggunakan *priority queue* sebagai antrian paket dan paket yang dihapus saat antrian penuh adalah paket yang paling baru.
ifqLen	100	Jumlah maksimal paket pada antrian.
antType	Antenna/OmniA ntenna	Jenis antena yang digunakan adalah *omni antenna*.
propType	Propagation/Two RayGround	Tipe propagasi sinyal wireless adalah *two-ray-ground*.
phyType	Phy/WirelessPhy	Komunikasi menggunakan media nirkabel.
topoInstance	$topo	Topologi yang digunakan saat menjalankan skenario.
agentTrace	ON	Mengaktifkan pencatatan (*record*). aktifitas dari agen *routing protocol*.
routerTrace	ON	Mengaktifkan pencatatan (*record*) pada aktifitas *routing protocol*.
macTrace	OFF	Menonaktifkan pencatatan (*record*)

		MAC *layer* pada *trace file*.
movementTrace	OFF	Menonaktifkan pencatatan (*record*) pergerakan *node*.
channel	Hannel/Wireless Channel	Kanal komunikasi yang digunakan.
nn	50	Jumlah *node* yang digunakan dalam simulasi.
sc	cbr-coba.txt	Koneksi CBR dengan menggunakan *Agent* UDP
cp	mobility_map_grid_lima_puluh.tcl	*File* yang berisikan pergerakan *node*.
stop	360.0	Lama waktu yang dibutuhkan dalam simulasi.

Skenario simulasi dijalankan dengan perintah pada Gambar 4.24 dan proses perintah menjalankan NS dapat dilihat pada Gambar 4.25.

```
ns (nama_file_ns).tcl
```

Gambar 4.24. Perintah Untuk Menjalankan Skenario NS-2.



Gambar 4.25. Proses dari Perintah Menjalankan *file* NS.

Setelah proses simulasi selesai selanjutnya akan keluar hasil berupa *trace file* lagi yang nantinya *file* ini akan digunakan untuk analisis. Adapun isi dari *trace file* ini adalah catatan seluruh *event* yang dari setiap paket yang tersebar di dalam lingkungan simulasi. Potongan hasil dari *trace file* ini dapat dilihat pada Lampiran A.

(Halaman ini sengaja dikosongkan)

BAB V
UJI COBA DAN ANALISIS HASIL

Bab ini membahas uji coba dan evaluasi hasil simulasi dari skenario NS-2 yang telah dibuat. Sistem ini diuji dari segi fungsionalitas dan dari contoh kasus. Pengujian akan dilakukan sesuai dengan perancangan pada Bab 3 dan implementasi yang semuanya telah dijelaskan pada Bab 4.

5.1 Lingkungan Uji Coba

Uji coba ini dilakukan dengan menggunakan sebuah komputer. Berikut ini adalah spesifikasi perangkat yang digunakan, dapat dilihat pada Tabel 5.1.

Tabel 5.1. Spesifikasi Perangkat Keras yang Digunakan.

Komponen	**Spesifikasi**
CPU	Intel (R) Core ™ i3-2130 CPU @3.40GH x 4
Sistem Operasi	Linux Ubuntu 12.04 LTS 32-bit
Memori	3.8 GB
Harddisk	500 GB

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

- SUMO versi 0.26.0 untuk pembuatan skenario mobilitas VANET.
- JOSM versi 11223 untuk penyuntingan pada OpenSteetMap
- ns23.5 untuk simulasi skenario VANET.

Parameter lingkungan dalam uji coba ini yang digunakan pada NS-2 dapat dilihat pada Tabel 5.2.

Pengujian ini dilakukan dengan menjalankan skenario melalui NS-2 dan nantinya hasil uji coba dari skenario tersebut berupa *trace file* yang nantinya akan dianalisis dengan srkip AWK.

Tabel 5.2. Keterangan Parameter dan Spesifikasi yang Digunakan pada Simulasi NS-2

No	Parameter	Spesifikasi
1	*Network simulator*	NS-2.35
2	*Routing protocol*	AODV
3	Area simulasi	1200 m x 1200 m (*grid*) 1200 m x 800 m (riil)
4	Waktu	360 detik
5	Radius transmisi	50 m
6	Agen	*Constant Bit Rate*
7	*Source*/*Detination*	Stationary
8	Jumlah kendaraan	50, 100, 150, 200
9	Kecepatan Statis	15 m/s
10	*Packet Rate*	2 kB/s
11	*Packet Interval*	1 paket/detik
12	Ukuran Paket	512 Bytes
13	Protokol MAC	IEEE 802.11p
14	Model Propagasi	*Two-ray Ground*

5.2 Hasil Uji Coba

Hasil uji coba dari skenario *grid* dan skenario riil dari masing-masing analisis metrik, yaitu *Packet Delivery Ratio* (PDR), *Average End-to-End Delay* dan *Routing Overhead* (RO) dapat dilihat sebagai berikut:

5.2.1 Hasil Uji Coba *Packet Delivery Ratio* (PDR)

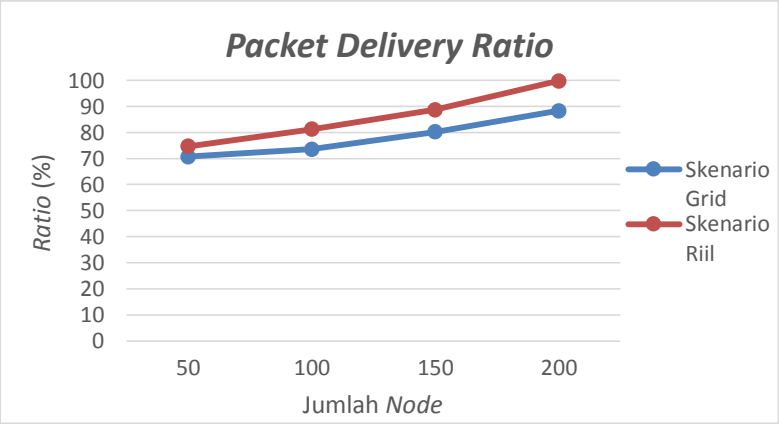
Adapun hasil uji coba dari *Packet Delivery Ratio* (PDR) untuk skenario *grid* dan skenario riil masing-masing dilakukan sebanyak 10 kali dengan skenario mobilitas *random* pada peta *grid*

dengan luas area, jumlah *node* pad masing-masing skenario seperti disebutkan pada Tabel 5.2. Adapun jumlah maksimal kendaraan yang digunakan dalam skenario ini adalah 200. Kecepatan maksimum dari setiap *node* adalah 15/ms. Hasil dari tiap perhitungan *node* pada *Packet Delivery Ratio* (PDR) ditabulasikan dan dirata-rata menjadi seperti pada Tabel 5.3.

Tabel 5.3. Hasil Perhitungan Rata-rata PDR pada Skenario *Grid* dan Skenario Riil.

Jumlah *Node*	Skenario *Grid*	Skenario Riil	Perbedaan (%)
50	70,7	74,7	4
100	73,5	81,2	7,7
150	80,2	88,7	8,5
200	88,3	99,7	11,4

Dari data diatas dapat ditarik menjadi suatu grafik yang merepresentasikan hasil perhitungan PDR dari skenario *grid* dan skenario riil yang ditunjukkan pada Gambar 5.1.



Gambar 5.1. Grafik PDR pada Skenario *Grid* dan Skenario Riil.

Berdasarkan data tersebut, dapat dilihat bahwa pada skenario riil nilai rata-rata PDR lebih tinggi dibandingkan dengan skenario *grid*. Pada jumlah *node* 50, nilai PDR pada skenario riil memiliki 4% lebih baik dibandingkan pada skenario *grid*. Pada jumlah *node* 100 dan 150, pada skenario riil nilai PDR naik ke angka 7,7% hingga 8,5% lebih baik dibandingkan dengan skenario *grid*. Dan pada jumlah *node* tertinggi, yaitu 200, pada skenario riil nilai PDR naik lagi ke angka 11,4% lebih baik dibandingkan dengan skenario *grid*.

Banyaknya jumlah *node* yang pada skenario *grid* dan skenario riil sama-sama berpengaruh terhadap nilai PDR. Terbukti dari jumlah kepadatan rendah, 50 *node*, hingga kepadatan paling tinggi, 200 *node*, nilai PDR dari kedua skenario sama-sama meningkat. Namun pada skenario riil nilai PDR sedikit lebih meningkat dibandingkan dengan skenario *grid*. Hal ini disebabkan perbedaan jumlah paket RREQ dari AODV yang tersebar dalam jaringan. Semakin banyak paket RREQ yang tersebar, maka semakin banyak pula kemungkinan rute yang akan terjadi. Karena pada skenario riil rute yang ada lebih bervariasi dalam pencarian rute karena berbentuk asimetris dibandingkan dengan skenario *grid* yang lebih simetris dan pencarian rute yang lebih statis.

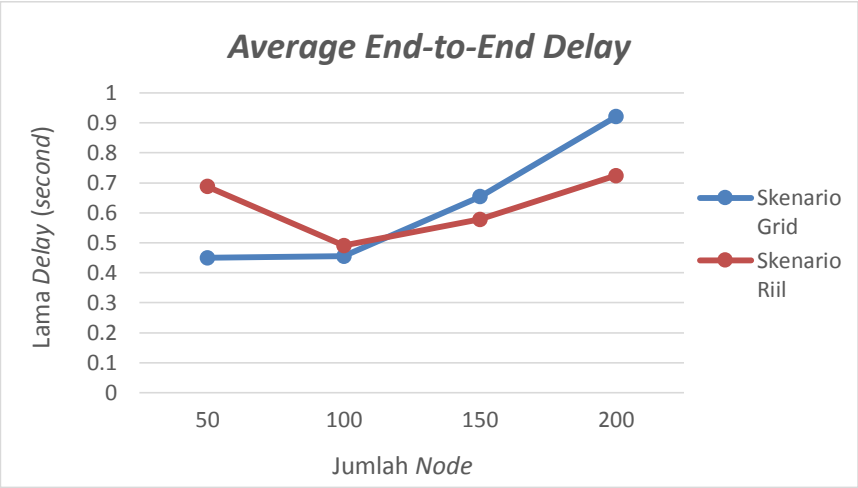
5.2.2 Hasil Uji Coba *Average End-to-End Delay*

Adapun hasil uji coba dari *Average End-to-End Delay* untuk skenario *grid* dan skenario riil masing-masing dilakukan sebanyak 10 kali dengan skenario mobilitas *random* pada peta *grid* dengan luas area, jumlah *node* pad masing-masing skenario seperti disebutkan pada Tabel 5.2. Adapun jumlah maksimal kendaraan yang digunakan dalam skenario ini adalah 200. Kecepatan maksimum dari setiap *node* adalah 15/ms. Hasil dari tiap perhitungan *node* pada *Average End-to-End Delay* ditabulasikan dan dirata-rata menjadi seperti pada Tabel 5.4.

Tabel 5.4. Hasil Perhitungan Rata-rata *Delay* pada Skenario *Grid* dan Skenario Riil.

Jumlah *Node*	Skenario *Grid* (detik)	Skenario Riil (detik)	Perbedaan (detik)
50	0,450	0,687	0,237
100	0,455	0,491	0,036
150	0,654	0,578	0,076
200	0,921	0,724	0,197

Dari data diatas dapat ditarik menjadi suatu grafik yang merepresentasikan hasil perhitungan *Delay* dari skenario *grid* dan skenario riil yang ditunjukkan pada Gambar 5.2.



Gambar 5.2. Grafik *Delay* pada Skenario *Grid* dan Skenario Riil.

Berdasarkan data tersebut, dapat dilihat bahwa pada skenario riil nilai rata-rata *Delay* pada kepadatan awal dimana jumlah *node* masih 50, nilai *Delay* jauh lebih tinggi hingga memperoleh selisih 0,237 detik dibandingkan dengan skenario

grid. Namun pada kondisi dengan jumlah *node* 100, nilai *Delay* turun drastis hingga selisih dengan skenario *grid* berkurang menjadi hanya 0,036 detik saja. Namun seiring bertambahnya kepadatan kendaraan dalam skenario riil, yaitu pada jumlah *node* 150 hingga 200, nilai *Delay* ikut naik menjadi 0,076 hingga 0,197 detik, namun kali ini lebih rendah dibandingkan dengan skenario *grid*. Pada kepadatan rendah, AODV yang ada pada skenario riil memerlukan waktu yang lebih lama untuk melakukan proses *route discovery*. Ini karena rute dari skenario riil yang lebih bervariasi dari skenario *grid* yang selalu sama di setiap 200 m, berbeda dengan skenario riil yang setiap jalan mempunyai panjang yang berbeda-beda.

Adapun tren *delay* yang semakin meningkat pada skenario *grid* ketika jumlah *node* semakin banyak ini dikarenakan selain AODV terlalu lama dalam melakukan proses *route discovery*, juga karena terjadi kemacetan jalan di setiap perempatan yang tedapat lalu lintas. Karena AODV melakukan *broadcast*, *buffer* paket dari setiap *node* di sekitar area perempatan menjadi sangat panjang. Berbeda dengan skenario riil yang lebih sedikit memiliki perempatan dan karena jarak antar lampu lalu lintas yang satu dengan yang lain yag bervariasi dan termasuk lebih jauh dibandingkan dengan pada skenario *grid*.

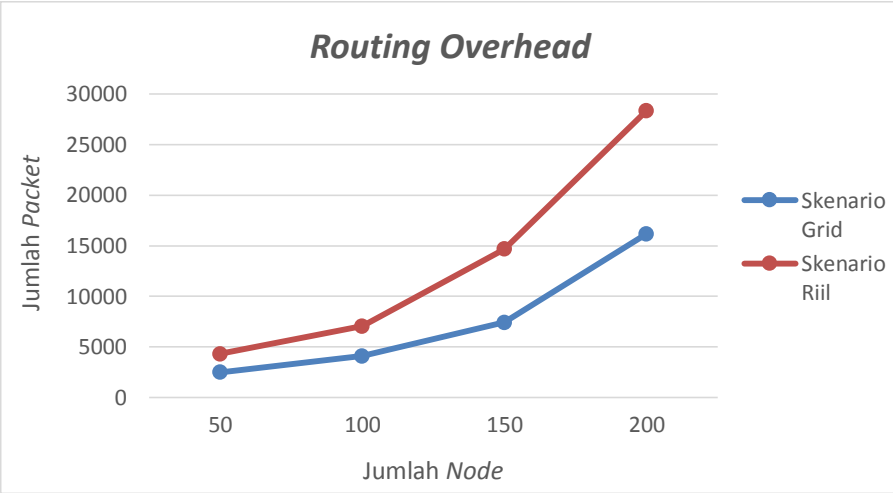
5.2.3 Hasil Uji Coba *Routing Overhead* (RO)

Adapun hasil uji coba dari *Routing Overhead* (RO) untuk skenario *grid* dan skenario riil masing-masing dilakukan sebanyak 10 kali dengan skenario mobilitas *random* pada peta *grid* dengan luas area, jumlah *node* pad masing-masing skenario seperti disebutkan pada Tabel 5.2. Adapun jumlah maksimal kendaraan yang digunakan dalam skenario ini adalah 200. Kecepatan maksimum dari setiap *node* adalah 15/ms. Hasil dari tiap perhitungan *node* pada *Routing Overhead* (RO) ditabulasikan dan dirata-rata menjadi seperti pada Tabel 5.5.

Tabel 5.5. Hasil Perhitungan Rata-rata RO pada Skenario *Grid* dan Skenario Riil.

Jumlah *Node*	Skenario *Grid* (paket)	Skenario Riil (paket)	Perbedaan (paket)
50	2493	4314	1821
100	4100	7062	2962
150	7416	14688	7272
200	16179	28368	12189

Dari data diatas dapat ditarik menjadi suatu grafik yang merepresentasikan hasil perhitungan RO dari skenario *grid* dan skenario riil yang ditunjukkan pada Gambar 5.3.



Gambar 5.3. Grafik *Routing Overhead* pada Skenario *Grid* dan Skenario Riil.

Berdasarkan data tersebut, dapat dilihat bahwa skenario riil memiliki nilai rata-rata RO yang lebih tinggi dibandingkan dengan Skenario *Grid*. Terlihat pada kepadatan awal, dengan

jumlah *node* 50, skenario riil memiliki nilai RO lebih banyak 1821 paket dibandingkan dengan Skenario Grid. Kemudian pada jumlah *node* 100 hingga 200, masing-masing dari skenario sama-sama meningkat nilai dari RO. Karena memang pada skenario *grid* dan skenario riil ini berbeda dalam rute perjalanan pada saat AODV melakukan *broadcast*, sehingga secara otomatis banyak jumlah paket *routing* yang tersebar dalam jaringan.

5.3 Analisis Hasil Uji Coba

Setelah melakukan uji coba diatas, dapat dihasilkan sebuah analisis. Uji coba yang dilakukan pada metrik *Packet Delivery Ratio* (PDR), *Average End-to-End Delay* dan *Routing Overhead* (RO) diatas menyebabkan beberapa perbedaan diantara model skenario yang digunakan yaitu Skenario *Grid* dan skenario riil. Beberapa faktor yang menyebabkan terjadinya perbedaan ini adalah lebih kepada bentuk jalan, luas daerah yang menyebabkan rute yang dilewati *node* cukup berbeda.

BAB VI
PENUTUP

Bab ini membahas kesimpulan yang dapat diambil dari hasil pembuatan perangkat lunak dan hasil uji cobanya. Selain itu, terdapat saran untuk pengembangan perangkat lunak lebih lanjut.

6.1 Kesimpulan

Dari hasil uji coba yang telah dilakukan, dapat diambil kesimpulan sebagai berikut:

1. Simulasi yang dijalankan pada Skenario *Grid* dengan penambahan pada jumlah *node* memiliki performa sebagai berikut:
   - *Packet Delivery Ratio* (PDR) dari AODV meningkat cukup stabil dari angka 70,7% dengan kepadatan minimal dan naik ke angka 88,3% dengan kepadatan maksimal. Nilai ini meningkat karena seiring bertambahnya kepadatan kendaraan dalam jaringan.
   - *Average End-to-End Delay* dari AODV sedikit tidak stabil pada awal karena kepadatan bermula masih rendah, namun seiring bertambahnya kepadatan yang semakin tinggi, nilai *Delay* pun langsung naik cukup signifikan.
   - *Routing Overhead* yang dihasilkan memiliki nilai yang cukup stabil dengan nilai yang meningkat secara signifikan dari 2493 paket hingga naik ke angka 16179 paket seiring dengan bertambahnya kepadatan kendaraan dari rendah ke kepadatan tinggi.

2. Simulasi yang dijalankan pada Skenario Riil dengan penambahan pada jumlah *node* memiliki performa sebagai berikut:

- *Packet Delivery Ratio* (PDR) dari AODV meningkat cukup stabil dari angka 74,7% dengan kepadatan minimal dan naik ke angka 99,7% dengan kepadatan maksimal. Nilai ini meningkat karena seiring bertambahnya kepadatan kendaraan dalam jaringan.
- *Average End-to-End Delay* dari AODV sedikit tidak stabil pada awal karena kepadatan masih rendah, pada kondisi ini nilai *delay* tinggi, namun dengan bertambahnya kepadatan, nilai *delay* pun menurun. Pada akhirnya nilai *delay* naik kembali seiring bertambahnya *node* hingga kepadatan maksimal.
- *Routing Overhead* yang dihasilkan memiliki nilai yang cukup stabil dengan nilai yang meningkat secara signifikan dari 4314 paket hingga naik ke angka 28368 paket seiring dengan bertambahnya kepadatan kendaraan dari rendah ke kepadatan tinggi.

3. Berdasarkan hasil uji coba perbedaan nilai dari skenario *grid* dengan skenario riil didapatkan hasil bahwa pada skenario riil protokol AODV memiliki nilai lebih baik daripada dalam skeanrio *grid*. Seperti dilihat pada nilai rasio perbedaan berikut:
   - Nilai PDR naik dari angka 4% pada kepadatan minimum ke angka 11,4% pada kepadatan maksimum.
   - Nilai Delay turun dari 127% pada kepadatan minimum hingga 65,5% pada kepadatan maksimum.
   - Nilai RO turun dari 57,7% pada kepadatan minimum ke angka 57% pada kepadatan maksimum.

   Nilai rasio ini semua tergantung seiring bertambahnya jumlah kepadatan kendaraan pada skenario.

4. Faktor-faktor yang dapat mempengaruhi nilai PDR, *Delay* dan RO yang dihasilkan dari simulasi dengan skenario *grid* dan skenario riil adalah:

- Bentuk rute dari masing-masing skenario yang berbeda.
- Pergerakan *node* yang dibuat secara acak.
- Parameter pada masing-masing skenario.

6.2 Saran

Saran yang dapat diberikan dari hasil uji coba dan evaluasi dari Tugas Akhir ini untuk pengembangan simulasi kedepan, antara lain:

1. Studi lebih lanjut mengenai pengaruh tipe *routing* protokol lain (proaktif atau *hybird*) dengan kepadatan kendaraan yang lebih variatif pada skenario *grid* dan skenario riil.
2. Dilakukan penambahan model transmisi selain *TwoRay-Ground*.
3. Dilakukan pengurangan dan penambahan pada kecepatan *node* maksimal.
4. Diperlukan penyempurnaan sistem lampu lalu lintas agar lebih realistis. Lampu lalu lintas yang diimplementasikan pada Tugas Akhir ini ditangani sepenuhnya oleh *simulator*, bukan berdasarkan desain penulis.

(Halaman ini sengaja dikosongkan)

DAFTAR PUSTAKA

[1] A. M. C dan T. C, "A CLUSTER BASED ENHANCEMENT TO AODV FOR INTER-VEHICULAR COMMUNICATION IN VANET," *Int. J. Grid Comput. Appl. IJGCA*, vol. 3, hal. 41–50, Sep 2012.

[2] "A Noble Routing Protocol for Vehicular ad hoc Networks (VANETs) with Less Routing Overheads," *Int. J. Future Gener. Commun. Netw.*, vol. 7, hal. 23–34, 2014.

[3] B. Ramakrishnan, "Performance Analysis of AODV Routing Protocol In Vehicular Ad-hoc Network Service Discovery Architecture," *ARPN J. Syst. Softw.*, vol. 2, hal. 65–72, Feb 2012.

[4] G. Gautam dan B. Sen, "Design and Simulation of Wireless Sensor Network in NS2," *Int. J. Comput. Appl. 0975 – 8887*, vol. 113, hal. 14–16, Mar 2015.

[6] "DLR - Institute of Transportation Systems - SUMO – Simulation of Urban MObility." [Daring]. Tersedia pada: http://www.dlr.de/ts/en/desktopdefault.aspx/tabid-9883/16931_read-41000/. [Diakses: 18-Jan-2017].

[7] "NETCONVERT - Sumo." [Daring]. Tersedia pada: http://sumo.dlr.de/wiki/NETCONVERT. [Diakses: 01-Jan-2017].

[8] P. TS, "SUMO, Open Street Maps and NS2 - A Real Traffic Simulation.".

[9] K. Thenmozhi dan D. E. Ramaraj, "Routing Overhead in MANET - Minimization with Multipath Local Route Discovery Routing Protocol," *Int. J. Eng. Trends Technol. IJETT*, vol. 13, hal. 144–147, Jul 2014.

[10] P. Rohal, R. Dahiya, dan P. Dahiya, "Study and Analysis of Throughput, Delay and Packet Delivery Ratio in MANET for Topology Based Routing Protocols (AODV, DSR and DSDV)," *Int. J. Adv. Res. Eng. Technol.*, vol. 1, no. II, hal. 54–58, Mar 2013.

[11] "NETGENERATE - Sumo." [Daring]. Tersedia pada: http://sumo.dlr.de/wiki/NETGENERATE. [Diakses: 01-Jan-2017].
[12] "OpenStreetMap," *OpenStreetMap*. [Daring]. Tersedia pada: https://www.openstreetmap.org/. [Diakses: 30-Des-2016].
[13] Jagadevi dan D. S. Terdal, "Performance Evaluation of IEEE 802.11p MAC Protocol for VANETs," *Int. J. Sci. Res. IJSR*, vol. 4, no. 9, hal. 711–715, Sep 2015.

LAMPIRAN

Dalam lampiran ini disertakan potongan kode, percobaan dengan node diluar parameter yang digunakan dan parameter 802.11a yang digunakan dalam pengerjaan Tugas Akhir ini.

A.1 Kode Skenario NS-2

```
set val(chan) Channel/WirelessChannel ;# channel type
set val(prop) Propagation/TwoRayGround ;# radio-propagation
model
set val(netif) Phy/WirelessPhyExt ;# network interface type
set val(mac) Mac/802_11Ext ;# MAC type
set val(ifq) Queue/DropTail/PriQueue ;# interface queue
type
set val(ll) LL  ;# link layer type
set val(ant) Antenna/OmniAntenna  ;# antenna model
set val(ifqlen) 100 ;# max packet in ifq
set val(rp) AODV ;# routing protocol
set val(x) 1000
set val(y) 1000
set val(seed) 0.0
set val(tr) coba.tr
set val(nn) 50
set val(cp) "cbr-coba.txt" ;# connection pattern file
set val(sc) "mobility_map_grid_lima_puluh.tcl"  ;# node
movement file.
set val(stop)           360.0

# Initialize ns
set ns_                 [new Simulator]

# open traces
set tracefd  [open coba.tr w]
```

```
$ns_ trace-all $tracefd

set topo       [new Topography]
$topo load_flatgrid $val(x) $val(y)

set god_ [create-god $val(nn)]

set chan_1_ [new $val(chan)]

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


# 802.11p default parameters

Phy/WirelessPhyExt set CSThresh_
3.9810717055349694e-13  ;# -94 dBm wireless interface
sensitivity
Phy/WirelessPhyExt set Pt_                          0.0005
                  ;# equals 20dBm when considering antenna
gains of 1.0
Phy/WirelessPhyExt set freq_                        5.9e+9
Phy/WirelessPhyExt set noise_floor_                 1.26e-13
            ;# -99 dBm for 10MHz bandwidth
Phy/WirelessPhyExt set L_                           1.0
            ;# default radio circuit gain/loss
```

```
Phy/WirelessPhyExt set PowerMonitorThresh_
3.981071705534985e-18   ;# -174 dBm power monitor
sensitivity (=level of gaussian noise)
Phy/WirelessPhyExt set HeaderDuration_          0.000040
            ;# 40 us
Phy/WirelessPhyExt set BasicModulationScheme_   0
Phy/WirelessPhyExt set PreambleCaptureSwitch_   1
Phy/WirelessPhyExt set DataCaptureSwitch_       1
Phy/WirelessPhyExt set SINR_PreambleCapture_    3.1623;
            ;# 5 dB
Phy/WirelessPhyExt set SINR_DataCapture_        10.0;
            ;# 10 dB
Phy/WirelessPhyExt set trace_dist_              1e6
            ;# PHY trace until distance of 1 Mio. km
("infinity")
Phy/WirelessPhyExt set PHY_DBG_                 0

Mac/802_11Ext set CWMin_                        15
Mac/802_11Ext set CWMax_                        1023
Mac/802_11Ext set SlotTime_                     0.000013
Mac/802_11Ext set SIFS_                         0.000032
Mac/802_11Ext set ShortRetryLimit_              7
Mac/802_11Ext set LongRetryLimit_               4
Mac/802_11Ext set HeaderDuration_               0.000040
Mac/802_11Ext set SymbolDuration_               0.000008
Mac/802_11Ext set BasicModulationScheme_        0
Mac/802_11Ext set use_802_11a_flag_             true
Mac/802_11Ext set RTSThreshold_                 2346
Mac/802_11Ext set MAC_DBG                       0

###

        for {set i 0} {$i < $val(nn) } {incr i} {
               set node_($i) [$ns_ node]
               $node_($i) random-motion 0 ;# disable random
motion
        }

puts "Loading connection pattern..."
```

```
source $val(cp)

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

for {set i 0} {$i < $val(nn) } {incr i} {
    $ns_ initial_node_pos $node_($i) 20
}

for {set i 0} {$i < $val(nn) } {incr i} {
    $ns_ at $val(stop).000000001 "$node_($i) reset";
}

$ns_ at $val(stop).000000001 "puts \"NS EXITING...\"; $ns_
halt"

proc stop {} {
    global ns_ tracefd namtrace
    $ns_ flush-trace
    close $tracefd
#    close $namtrace
}

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

A.2 Kode cbr-coba.txt

```
#
# nodes: 50, max conn: 10, send rate: 0.5, seed:
1
#
#
# 1 connecting to 2 at time 2.5568388786897245
#
set udp_(0) [new Agent/UDP]
$ns_ attach-agent $node_(1) $udp_(0)
set null_(0) [new Agent/Null]
$ns_ attach-agent $node_(2) $null_(0)
set cbr_(0) [new Application/Traffic/CBR]
$cbr_(0) set packetSize_ 512
$cbr_(0) set interval_ 0.5
$cbr_(0) set random_ 1
$cbr_(0) set maxpkts_ 10000
$cbr_(0) attach-agent $udp_(0)
$ns_ connect $udp_(0) $null_(0)
$ns_ at 2.5568388786897245 "$cbr_(0) start"
#
```

A.3 Kode *awk* Perhitungan *Packet Delivery Ratio*

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

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

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

END {
      printf "Sent:  %d\nRecv:  %d\nRatio:
%.4f\n", sendLine, recvLine, (recvLine /
sendLine);
}
```

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

```
BEGIN {
      highest_packet_id = 0;
      }

{
      # Using old trace file format
      action = $1;
      time = $2;
      layer = $4;
      packet_id = $6;

      # Check for the latest packet id
      if ( packet_id > highest_packet_id ) {
      highest_packet_id = packet_id;
      }

      # If start_time of packet_id is empty,
then get read time
      if ( start_time[packet_id] == 0 ) {
      start_time[packet_id] = time;
      }

      if ( $7 == "cbr" ) {
            # If packet is not dropped
            if ( action != "D" ) {
                  # If packet is received
                  if ( action == "r" ) {
                  # Get received time
                  end_time[packet_id] = time;
                  }
            }
            # If packet is dropped
```

```
            else {
                  # There is no "end time"
                  end_time[packet_id] = -1;
                  }
            }
      }

      END {
            sigma_duration = 0;
            count = 0;
            for ( packet_id = 0; packet_id <=
highest_packet_id; packet_id++ )
            {
                  type = packet_type[packet_id];
                  start = start_time[packet_id];
                  end = end_time[packet_id];
                  packet_duration = end - start;

                  if ( start < end ) {
                        sigma_duration +=
packet_duration;
                        count++;
                        }
                  }

            if ( count == 0 ) {
            printf("no_packet_counted\n");
      }
      else {
            printf("Average Delay : %.4f\n",
sigma_duration / count);
```

A.5 Kode *awk* Perhitungan *Routing Overhead*

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

$0 ~/^s.* \(REQUEST\)/ {
            sendLine ++;
            }

$0 ~/^s.* \(REPLY\)/ {
            sendLine ++;
            }

$0 ~/^s.* \(ERROR\)/ {
            sendLine ++;
            }

END {
      printf "Overhead :  %d\n", (sendLine +
recvLine + errLine);
}
```

A.6 Potongan Kode *Trace File*

```
M 0.00000 0 (398.35, 825.51, 0.00), (398.35, 825.51),
0.00
M 1.00000 0 (398.35, 825.51, 0.00), (398.35, 823.84),
1.67
M 1.00000 1 (601.65, 13.41, 0.00), (601.65, 13.41),
0.00
M 2.00000 0 (398.35, 823.84, 0.00), (398.35, 820.20),
3.63
M 2.00000 1 (601.65, 13.41, 0.00), (601.65, 15.25),
1.83
M 2.00000 2 (988.38, 1.65, 0.00), (988.38, 1.65), 0.00
s 2.556838879 _1_ AGT  --- 0 cbr 512 [0 0 0 0] -------
[1:0 2:0 32 0] [0] 0 0
r 2.556838879 _1_ RTR  --- 0 cbr 512 [0 0 0 0] -------
[1:0 2:0 32 0] [0] 0 0
s 2.556838879 _1_ RTR  --- 0 AODV 48 [0 0 0 0] -------
[1:255 -1:255 30 0] [0x2 1 1 [2 0] [1 4]] (REQUEST)
r 2.557144942 _9_ RTR  --- 0 AODV 48 [0 ffffffff 1 0]
------- [1:255 -1:255 30 0] [0x2 1 1 [2 0] [1 4]]
(REQUEST)
r 2.557144974 _46_ RTR  --- 0 AODV 48 [0 ffffffff 1 0]
------- [1:255 -1:255 30 0] [0x2 1 1 [2 0] [1 4]]
(REQUEST)
r 2.557145260 _40_ RTR  --- 0 AODV 48 [0 ffffffff 1 0]
------- [1:255 -1:255 30 0] [0x2 1 1 [2 0] [1 4]]
(REQUEST)
r 2.557145549 _17_ RTR  --- 0 AODV 48 [0 ffffffff 1 0]
------- [1:255 -1:255 30 0] [0x2 1 1 [2 0] [1 4]]
(REQUEST)
s 2.557286989 _9_ RTR  --- 0 AODV 48 [0 ffffffff 1 0]
------- [9:255 -1:255 29 0] [0x2 2 1 [2 0] [1 4]]
(REQUEST)
r 2.557593027 _46_ RTR  --- 0 AODV 48 [0 ffffffff 9 0]
------- [9:255 -1:255 29 0] [0x2 2 1 [2 0] [1 4]]
(REQUEST)
r 2.557593052 _1_ RTR  --- 0 AODV 48 [0 ffffffff 9 0]
------- [9:255 -1:255 29 0] [0x2 2 1 [2 0] [1 4]]
(REQUEST)
```

```
r 2.557593321 _40_ RTR  --- 0 AODV 48 [0 ffffffff 9 0]
------- [9:255 -1:255 29 0] [0x2 2 1 [2 0] [1 4]]
(REQUEST)
r 2.557593618 _17_ RTR  --- 0 AODV 48 [0 ffffffff 9 0]
------- [9:255 -1:255 29 0] [0x2 2 1 [2 0] [1 4]]
(REQUEST)
s 2.558026201 _46_ RTR  --- 0 AODV 48 [0 ffffffff 1 0]
------- [46:255 -1:255 29 0] [0x2 2 1 [2 0] [1 4]]
(REQUEST)
r 2.558332240 _9_ RTR  --- 0 AODV 48 [0 ffffffff 2e 0]
------- [46:255 -1:255 29 0] [0x2 2 1 [2 0] [1 4]]
(REQUEST)
r 2.558332297 _1_ RTR  --- 0 AODV 48 [0 ffffffff 2e 0]
------- [46:255 -1:255 29 0] [0x2 2 1 [2 0] [1 4]]
(REQUEST)
r 2.558332494 _40_ RTR  --- 0 AODV 48 [0 ffffffff 2e
0] ------- [46:255 -1:255 29 0] [0x2 2 1 [2 0] [1 4]]
(REQUEST)
r 2.558332792 _17_ RTR  --- 0 AODV 48 [0 ffffffff 2e
0] ------- [46:255 -1:255 29 0] [0x2 2 1 [2 0] [1 4]]
(REQUEST)
s 2.561556994 _40_ RTR  --- 0 AODV 48 [0 ffffffff 1 0]
------- [40:255 -1:255 29 0] [0x2 2 1 [2 0] [1 4]]
(REQUEST)
r 2.561863287 _46_ RTR  --- 0 AODV 48 [0 ffffffff 28
0] ------- [40:255 -1:255 29 0] [0x2 2 1 [2 0] [1 4]]
(REQUEST)
r 2.561863301 _17_ RTR  --- 0 AODV 48 [0 ffffffff 28
0] ------- [40:255 -1:255 29 0] [0x2 2 1 [2 0] [1 4]]
(REQUEST)
r 2.561863326 _9_ RTR  --- 0 AODV 48 [0 ffffffff 28 0]
------- [40:255 -1:255 29 0] [0x2 2 1 [2 0] [1 4]]
(REQUEST)
r 2.561863375 _1_ RTR  --- 0 AODV 48 [0 ffffffff 28 0]
------- [40:255 -1:255 29 0] [0x2 2 1 [2 0] [1 4]]
(REQUEST)
s 2.565137103 _17_ RTR  --- 0 AODV 48 [0 ffffffff 1 0]
------- [17:255 -1:255 29 0] [0x2 2 1 [2 0] [1 4]]
(REQUEST)
r 2.565443410 _40_ RTR  --- 0 AODV 48 [0 ffffffff 11
0] ------- [17:255 -1:255 29 0] [0x2 2 1 [2 0] [1 4]]
(REQUEST)
```

```
r 2.565443694 _46_ RTR  --- 0 AODV 48 [0 ffffffff 11
0] ------- [17:255 -1:255 29 0] [0x2 2 1 [2 0] [1 4]]
(REQUEST)
r 2.565443732 _9_ RTR  --- 0 AODV 48 [0 ffffffff 11 0]
------- [17:255 -1:255 29 0] [0x2 2 1 [2 0] [1 4]]
(REQUEST)
r 2.565443770 _39_ RTR  --- 0 AODV 48 [0 ffffffff 11
0] ------- [17:255 -1:255 29 0] [0x2 2 1 [2 0] [1 4]]
(REQUEST)
r 2.565443773 _1_ RTR  --- 0 AODV 48 [0 ffffffff 11 0]
------- [17:255 -1:255 29 0] [0x2 2 1 [2 0] [1 4]]
(REQUEST)
s 2.568534561 _39_ RTR  --- 0 AODV 48 [0 ffffffff 11
0] ------- [39:255 -1:255 28 0] [0x2 3 1 [2 0] [1 4]]
(REQUEST)
r 2.568840912 _14_ RTR  --- 0 AODV 48 [0 ffffffff 27
0] ------- [39:255 -1:255 28 0] [0x2 3 1 [2 0] [1 4]]
(REQUEST)
r 2.568841099 _35_ RTR  --- 0 AODV 48 [0 ffffffff 27
0] ------- [39:255 -1:255 28 0] [0x2 3 1 [2 0] [1 4]]
(REQUEST)
r 2.568841120 _48_ RTR  --- 0 AODV 48 [0 ffffffff 27
0] ------- [39:255 -1:255 28 0] [0x2 3 1 [2 0] [1 4]]
(REQUEST)
r 2.568841227 _17_ RTR  --- 0 AODV 48 [0 ffffffff 27
0] ------- [39:255 -1:255 28 0] [0x2 3 1 [2 0] [1 4]]
(REQUEST)
s 2.569269044 _35_ RTR  --- 0 AODV 48 [0 ffffffff 27
0] ------- [35:255 -1:255 27 0] [0x2 4 1 [2 0] [1 4]]
(REQUEST)
r 2.569575156 _48_ RTR  --- 0 AODV 48 [0 ffffffff 23
0] ------- [35:255 -1:255 27 0] [0x2 4 1 [2 0] [1 4]]
(REQUEST)
r 2.569575583 _39_ RTR  --- 0 AODV 48 [0 ffffffff 23
0] ------- [35:255 -1:255 27 0] [0x2 4 1 [2 0] [1 4]]
(REQUEST)
s 2.570482367 _14_ RTR  --- 0 AODV 48 [0 ffffffff 27
0] ------- [14:255 -1:255 27 0] [0x2 4 1 [2 0] [1 4]]
(REQUEST)
r 2.570788718 _39_ RTR  --- 0 AODV 48 [0 ffffffff e 0]
------- [14:255 -1:255 27 0] [0x2 4 1 [2 0] [1 4]]
(REQUEST)
```

```
r 2.570788975 _43_ RTR  --- 0 AODV 48 [0 ffffffff e 0]
------- [14:255 -1:255 27 0] [0x2 4 1 [2 0] [1 4]]
(REQUEST)
s 2.575025542 _43_ RTR  --- 0 AODV 48 [0 ffffffff e 0]
------- [43:255 -1:255 26 0] [0x2 5 1 [2 0] [1 4]]
(REQUEST)
s 2.575241796 _48_ RTR  --- 0 AODV 48 [0 ffffffff 27
0] ------- [48:255 -1:255 27 0] [0x2 4 1 [2 0] [1 4]]
(REQUEST)
r 2.575332150 _14_ RTR  --- 0 AODV 48 [0 ffffffff 2b
0] ------- [43:255 -1:255 26 0] [0x2 5 1 [2 0] [1 4]]
(REQUEST)
r 2.575646333 _35_ RTR  --- 0 AODV 48 [0 ffffffff 30
0] ------- [48:255 -1:255 27 0] [0x2 4 1 [2 0] [1 4]]
(REQUEST)
r 2.575646781 _39_ RTR  --- 0 AODV 48 [0 ffffffff 30
0] ------- [48:255 -1:255 27 0] [0x2 4 1 [2 0] [1 4]]
(REQUEST)
s 2.807008667 _1_ AGT  --- 1 cbr 512 [0 0 0 0] -------
[1:0 2:0 32 0] [1] 0 0
     r 2.807008667 _1_ RTR  --- 1 cbr 512 [0 0 0 0] --
----- [1:0 2:0 32 0] [1] 0 0
```

(Halaman ini sengaja dikosongkan)

A.7 Potongan Kode *File* mobility_map.tcl

```
$node_(0) set X_ 398.35
$node_(0) set Y_ 825.51
$node_(0) set Z_ 0
$ns_ at 0.0 "$node_(0) setdest 398.35 825.51
0.00"
$ns_ at 1.0 "$node_(0) setdest 398.35 823.84
1.67"
$node_(1) set X_ 601.65
$node_(1) set Y_ 13.41
$node_(1) set Z_ 0
$ns_ at 1.0 "$node_(1) setdest 601.65 13.41
0.00"
$ns_ at 2.0 "$node_(0) setdest 398.35 820.2
3.63"
$ns_ at 2.0 "$node_(1) setdest 601.65 15.25
1.83"
$node_(2) set X_ 988.38
$node_(2) set Y_ 1.65
$node_(2) set Z_ 0
$ns_ at 2.0 "$node_(2) setdest 988.38 1.65 0.00"
$ns_ at 3.0 "$node_(0) setdest 398.35 814.94
5.27"
$ns_ at 3.0 "$node_(1) setdest 601.65 19.36
4.11"
$ns_ at 3.0 "$node_(2) setdest 986.49 1.65 1.89"
$node_(3) set X_ 601.65
$node_(3) set Y_ 546.7
$node_(3) set Z_ 0
$ns_ at 3.0 "$node_(3) setdest 601.65 546.7
0.00"
$ns_ at 4.0 "$node_(0) setdest 398.35 807.07
7.86"
$ns_ at 4.0 "$node_(1) setdest 601.65 25.19
5.84"
```

```
$ns_ at 4.0 "$node_(2) setdest 982.6 1.65 3.89"
$ns_ at 4.0 "$node_(3) setdest 601.65 548.71
2.01"
$node_(4) set X_ 312.63
$node_(4) set Y_ 998.35
$node_(4) set Z_ 0
$ns_ at 4.0 "$node_(4) setdest 312.63 998.35
0.00"
$ns_ at 5.0 "$node_(0) setdest 399.01 801.52
5.66"
$ns_ at 5.0 "$node_(1) setdest 601.65 33.55
8.36"
$ns_ at 5.0 "$node_(2) setdest 977.25 1.65 5.35"
$ns_ at 5.0 "$node_(3) setdest 601.65 552.9
4.19"
$ns_ at 5.0 "$node_(4) setdest 314.34 998.35
1.71"
.
.
.
$ns_ at 359.0 "$node_(28) setdest 198.35 820.75
0.00"
$ns_ at 359.0 "$node_(29) setdest 201.65 394.25
0.00"
$ns_ at 359.0 "$node_(3) setdest 605.75 401.65
0.00"
$ns_ at 359.0 "$node_(30) setdest 798.35 405.75
0.00"
$ns_ at 359.0 "$node_(31) setdest 198.35 205.75
0.00"
$ns_ at 359.0 "$node_(32) setdest 201.65 594.25
0.00"
$ns_ at 359.0 "$node_(33) setdest 594.25 798.35
0.00"
$ns_ at 359.0 "$node_(34) setdest 867.3 401.65
14.51"
$ns_ at 359.0 "$node_(35) setdest 598.35 813.25
0.00"
```

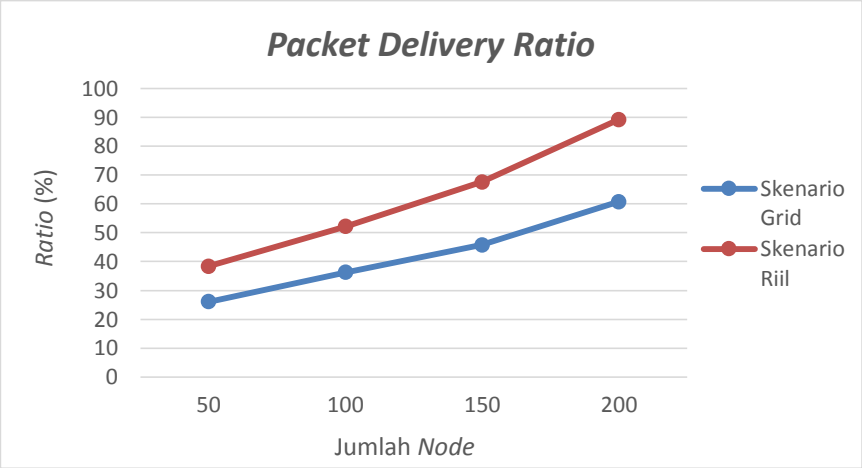
```
$ns_ at 359.0 "$node_(36) setdest 398.35 919.04
14.27"
$ns_ at 359.0 "$node_(37) setdest 401.65 386.75
0.00"
$ns_ at 359.0 "$node_(38) setdest 405.75 201.65
0.00"
$ns_ at 359.0 "$node_(39) setdest 401.65 394.25
0.00"
$ns_ at 359.0 "$node_(4) setdest 1.65 386.75
0.00"
$ns_ at 359.0 "$node_(40) setdest 198.35 813.25
0.00"
$ns_ at 359.0 "$node_(41) setdest 405.75 601.65
0.00"
$ns_ at 359.0 "$node_(42) setdest 594.25 998.35
0.00"
$ns_ at 359.0 "$node_(43) setdest 205.75 1001.65
0.00"
$ns_ at 359.0 "$node_(44) setdest 1.65 831.53
14.93"
$ns_ at 359.0 "$node_(45) setdest 401.65 594.25
0.00"
$ns_ at 359.0 "$node_(46) setdest 794.25 998.35
0.00"
$ns_ at 359.0 "$node_(47) setdest 1.65 394.25
0.00"
$ns_ at 359.0 "$node_(5) setdest 201.65 579.25
0.00"
$ns_ at 359.0 "$node_(6) setdest 654.96 601.65
13.79"
$ns_ at 359.0 "$node_(7) setdest 198.35 805.75
0.00"
$ns_ at 359.0 "$node_(8) setdest 198.35 605.75
0.00"
$ns_ at 359.0 "$node_(9) setdest 632.2 801.65
14.51"
$node_(48) set X_ 198.35
$node_(48) set Y_ 207.12
$node_(48) set Z_ 0
```

```
$node_(49) set X_ 801.60
$node_(49) set Y_ 591.19
$node_(49) set Z_ 0
```

A.8 Hasil Uji Coba Skenario Grid dan Skenario Riil untuk parameter 802.11a.

Tabel Hasil Perhitungan Rata-rata PDR pada Skenario *Grid* dan Skenario Riil untuk parameter 802.11a.

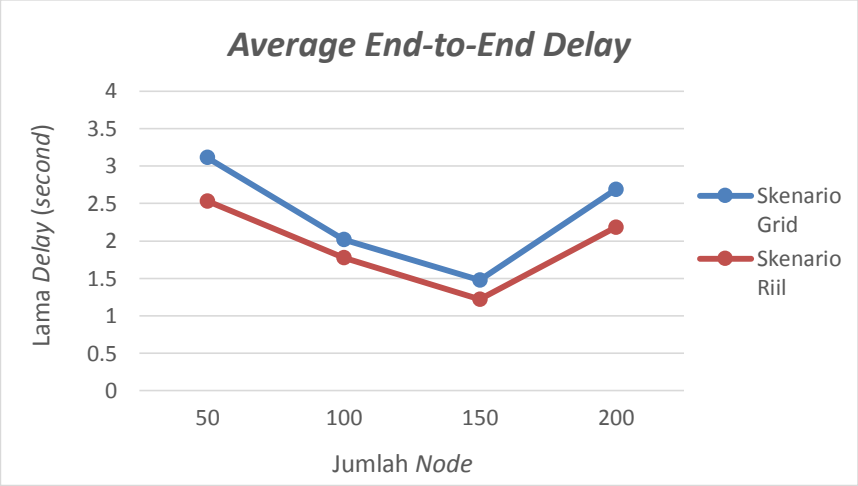
Jumlah *Node*	Skenario *Grid* (%)	Skenario Riil (%)	Perbedaan (%)
50	26,1	38,4	12,3
100	36,2	52,1	15,9
150	45,8	67,6	21,8
200	60,7	89,2	28,5



Gambar Grafik PDR pada Skenario *Grid* dan Skenario Riil untuk parameter 802.11a.

Tabel Hasil Perhitungan Rata-rata *Delay* pada Skenario *Grid* dan Skenario Riil untuk parameter 802.11a.

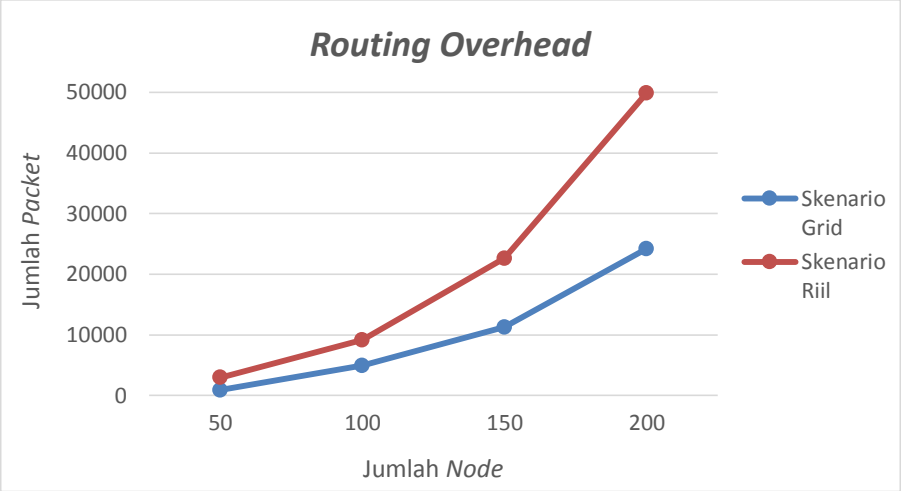
Jumlah *Node*	Skenario *Grid* (detik)	Skenario Riil (detik)	Perbedaan (detik)
50	3,108	2,530	0,57
100	2,015	1,774	0,241
150	1,476	1,217	0,259
200	2,687	2,180	0,507



Gambar. Grafik *Delay* pada Skenario *Grid* dan Skenario Riil untuk parameter 802.11a.

Tabel 8.1. Hasil Perhitungan Rata-rata RO pada Skenario *Grid* dan Skenario Riil untuk parameter 802.11a.

Jumlah *Node*	Skenario *Grid* (paket)	Skenario Riil (paket)	Perbedaan (paket)
50	885	2975	
100	4942	9153	
150	11276	22617	
200	24212	49871	

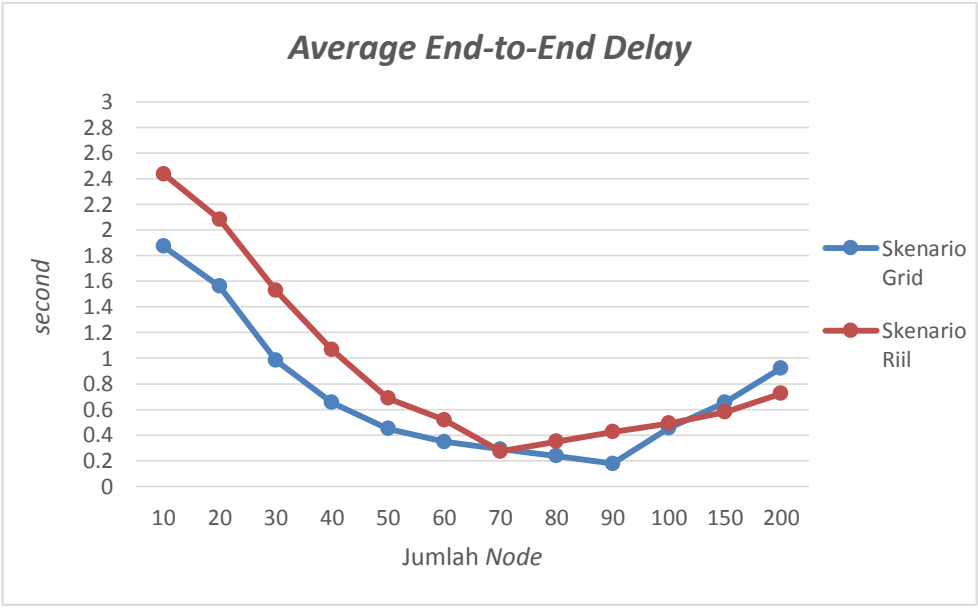


Gambar 8.1. Grafik *Routing Overhead* pada Skenario *Grid* danSkenario Riil untuk parameter 802.11a.

(Halaman ini sengaja dikosongkan)

Tabel Hasil Perhitungan Rata-rata *Delay* pada Skenario *Grid* dan Skenario Riil dengan penambahan node 10, 20, 30, 40, 60, 70, 80, 90.

Jumlah *Node*	Skenario *Grid* (detik)	Skenario Riil (detik)	Perbedaan (detik)
10	1,872	2,435	0,563
20	1,560	2,082	0,522
30	0,992	1,527	0,535
40	0,654	1,066	0,412
50	0,450	0,687	0,237
60	0,348	0,519	0,171
70	0,291	0,273	0,018
80	0,238	0,351	0,113
90	0,139	0,425	0,286
100	0,455	0,491	0,036
150	0,654	0,578	0,076
200	0,921	0,724	0,197



(Halaman ini sengaja dikosongkan)

BIODATA PENULIS

Ilmal Alifriansyah Rahardjo, biasa dipanggil Ilmal atau Iil atau Ma'il, lahir di Gresik pada tanggal 28 April 1992, merupakan anak bungsu dari tiga bersaudara. Penulis telah menempuh pendidikan mulai dari SD Negeri Latsari 2 Tuban (1998-2004), SMP Negeri 3 Tuban (2004-2007), SMA Negeri 2 Tuban (2007-2010), dan pada tahun 2010 penulis meneruskan pendidikannya di Teknik Informatika ITS. Selama menempuh kuliah, Penulis aktif dalam anggota Departemen Dalam Negeri dan Instructure Commite dalam organisasi Himpunan Mahasiswa Teknik Computer - Informatika (HMTC) ITS. Dalam perkuliahan, penulis mengambil bidang minat Komputasi Berbasis Jaringan (KBJ). Hobi penulis adalah semua yang berhubungan dengan olahraga kaki, sepak bola, futsal, takraw, lari dll. Penulis dapat dihubungi melalui email di *ilmal.alifriansyah@gmail.com*.