Institut
Teknologi
Sepuluh Nopember

TUGAS AKHIR - KI141502

IMPLEMENTASI *ROUTING PROTOCOL* AODV DENGAN PREDIKSI PERGERAKAN KENDARAAN DALAM VANET

REYHAN ARIEF
NRP 5112100175

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

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

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

UNDERGRADUATE THESIS - KI141502

IMPLEMENTATION OF AODV ROUTING PROTOCOL WITH VEHICLE MOVEMENT PREDICTION IN VANET

REYHAN ARIEF
NRP 5112100175

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

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

DEPARTMENT OF INFORMATICS
Faculty of Information Technology
Institut Teknologi Sepuluh Nopember
Surabaya, 2016

IMPLEMENTASI *ROUTING PROTOCOL* AODV DENGAN PREDIKSI PERGERAKAN KENDARAAN DALAM VANET

TUGAS AKHIR

Diajukan Guna Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Arsitektur dan Jaringan Komputer
Program Studi S1 Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember

Oleh :
Reyhan Arief
NRP: 5112100175

Disetujui oleh Dosen Pembimbing Tugas Akhir :

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

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

SURABAYA
Juni 2016

IMPLEMENTASI *ROUTING PROTOCOL* AODV DENGAN PREDIKSI PERGERAKAN KENDARAAN DALAM VANET

Nama	: REYHAN ARIEF
NRP	: 5112100175
Jurusan	: Teknik Informatika FTIf
Pembimbing I	: Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.
Pembimbing II	: Ir. F.X. Arunanto, M.Sc.

Abstrak

AODV merupakan salah satu *routing protocol* yang populer karena proses *route discovery*-nya yang efisien. Namun jika diterapkan pada lingkungan VANET yang memiliki topologi dinamis dan berubah dengan cepat, AODV kurang mampu menjaga kestabilan dalam komunikasi antar kendaraan. Hal ini tentu disebabkan oleh AODV yang pada dasarnya diciptakan untuk lingkungan MANET. Agar dapat bekerja lebih baik di lingkungan VANET, maka diperlukan modifikasi pada *routing protocol* AODV.

Modifikasi yang akan dilakukan adalah pada proses *route request*, yaitu dengan memperhitungkan faktor pergerakan kendaraan dan kualitas komunikasi antar kendaraan. Kemudian dilakukan prediksi terhadap faktor-faktor tersebut untuk beberapa detik yang akan datang. Dari hasil perhitungan tersebut akan diputuskan node mana saja yang dipilih sebagai node yang akan melanjutkan paket *route request*.

Dari uji coba yang dilakukan, AODV-PNT mengalami peningkatan pada nilai rata-rata *packet delivery ratio*, penurunan rata-rata *delay* dan *routing overhead* seiring dengan bertambahnya kepadatan kendaraan dibandingkan dengan AODV.

Kata-Kunci: VANET, AODV, NS-2, AODV-PNT.

IMPLEMENTATION OF AODV ROUTING PROTOCOL WITH VEHICLE MOVEMENT PREDICTION IN VANET

Name	: REYHAN ARIEF
NRP	: 5112100175
Major	: Informatics FTIf
Supervisor I	: Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.
Supervisor II	: Ir. F.X. Arunanto, M.Sc.

Abstract

AODV is one of the popular routing protocol because of its efficiency in route discovery process. However if applied in VANET environment which topology is dynamic and changing rapidly, AODV is less able to maintain its stability in vehicle-to-vehicle communication. This is certainly due to AODV which is basically created for MANET environment. In order to work better in VANET environment, it's necessary to modify AODV routing protocol.

The part that will be modified is route request process, namely by taking the movement information and link quality between vehicles into account. Then do a prediction based on those factors for possible future values. From the calculation, it will be decided which node is selected as the node that will relay the route request packet.

The simulation results show that AODV-PNT is able to achieve better routing performance in packet delivery ratio, average end-to-end delay, and routing overhead with increasing vehicle density compared to AODV.

***Keywords**: VANET, AODV, NS-2, AODV-PNT.*

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB 1

PENDAHULUAN

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

1.1 Latar Belakang

Perkembangan teknologi di bidang komunikasi data sangat pesat sehingga memungkinkan adanya perangkat nirkabel yang bersifat *ad hoc*. Dengan adanya teknologi tersebut, komunikasi data antar perangkat bisa dilakukan meskipun dalam posisi bergerak. Perangkat tersebut dapat diaplikasikan pada kendaraan sehingga bisa berbagi informasi antar kendaraan berupa peristiwa-peristiwa yang terjadi di jalanan, seperti kemacetan lalu lintas, kecelakaan, dan lain-lain. Jika banyak kendaraan yang memiliki perangkat tersebut, maka *Vehicular Ad Hoc Network* (VANET) dapat terwujud. Oleh karena itu, semakin banyak riset yang dilakukan di bidang VANET.

Ada beberapa jenis *routing protocol* yang dapat diterapkan dalam lingkungan VANET, salah satunya AODV. AODV berasal dari *routing protocol mobile ad hoc network* (MANET) yang bersifat reaktif sehingga tidak perlu menyimpan informasi keseluruhan topologi jaringan. Namun *routing protocol* MANET kurang cocok diterapkan pada lingkungan VANET karena topologi yang berubah dengan cepat dan adanya perbedaan karakteristik. Karakteristik dari VANET adalah mobilitas yang dibatasi oleh jalan dan bangunan, adanya kecepatan dan akselerasi dari node, dan perbedaan lokasi geografis [1]. Agar dapat bekerja lebih baik di lingkungan VANET, maka diperlukan modifikasi pada *routing protocol* AODV. Salah satu protokol modifikasi dengan basis AODV adalah AODV-PNT [2]. Optimasi yang dilakukan oleh AODV-PNT adalah dengan melakukan perhitungan bobot dan prediksi tren node sebelum mengirim atau

meneruskan paket route request sehingga dapat memilih *node* yang mana yang akan membawa paket tersebut.

Dalam Tugas Akhir ini, penulis mengimplementasikan routing protocol AODV-PNT pada network simulator NS-2. Penulis akan melakukan studi kinerja dari AODV-PNT dengan melakukan simulasi pada skenario yang memiliki jumlah node yang bervariasi. Kemudian hasil simulasi tersebut akan dibandingkan dengan kinerja AODV.

1.2 Rumusan Masalah

Rumusan masalah yang diangkat dalam tugas akhir ini adalah sebagai berikut:

1. Bagaimana cara mengimplementasikan protokol AODV-PNT dalam NS-2?
2. Bagaimana perbedaan kinerja antara AODV dan AODV-PNT pada skenario grid dan skenario nyata?

1.3 Batasan Masalah

Permasalahan yang dibahas pada Tugas Akhir ini memiliki beberapa batasan, yaitu:

1. Proses pengujian menggunakan *network simulator* NS-2.
2. Area simulasi dibuat dengan SUMO dan berukuran 1000 m x 1000 m untuk skenario *grid* dan 1200 m x 1000 m untuk skenario riil.
3. Skenario dibedakan berdasarkan jumlah node, yaitu 50, 100, 150, 200, dan 400.

1.4 Tujuan

Berikut merupakan tujuan dari Tugas Akhir ini adalah melakukan analisis perbandingan performa antara AODV-PNT dengan AODV

1.5 Manfaat

Manfaat dari pengerjaan Tugas Akhir ini adalah:

1. Mengetahui perbandingan performa protokol AODV-PNT dan AODV.
2. Meneliti pengaruh prediksi tren *node* dalam AODV.
3. Menjadi acuan untuk penelitian lebih lanjut pada protokol AODV.

1.6 Metodologi

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

1. Penyusunan proposal Tugas Akhir
   Tahap awal untuk memulai pengerjaan Tugas Akhir adalah penyusunan proposal Tugas Akhir. Pada proposal tersebut dijelaskan secara garis besar tentang alur pembuatan sistem.
2. Studi literatur
   Pada tahap ini dilakukan studi literatur mengenai alat dan metode yang digunakan. Literatur yang dipelajari dan digunakan meliputi buku referensi, artikel, jurnal dan dokumentasi dari internet.
3. Implementasi protokol
   Tahap ini meliputi perancangan sistem berdasarkan studi literatur dan pembelajaran konsep teknologi dari perangkat lunak yang ada. Tahap ini merupakan tahap yang paling penting dimana bentuk awal aplikasi yang akan diimplementasikan didefinisikan. Pada tahapan ini dibuat *prototype* sistem, yang merupakan rancangan dasar dari sistem yang akan dibuat. Serta dilakukan desain suatu sistem dan desain proses-proses yang ada.
4. Uji coba dan evaluasi
   Pada tahapan ini dilakukan uji coba terhadap aplikasi yang telah dibuat. Pengujian dan evaluasi akan dilakukan dengan

melihat kesesuaian dengan perencanaan. Tahap ini dimaksudkan juga untuk mengevaluasi jalannya sistem, mencari masalah yang mungkin timbul dan mengadakan perbaikan jika terdapat kesalahan.

5. Penyusunan buku Tugas Akhir
   Pada tahapan ini disusun buku sebagai dokumentasi dari pelaksanaan tugas akhir.

1.7 Sistematika Penulisan

Buku Tugas Akhir ini disusun degnan sistematika penulisan sebagai berikut:

BAB I. PENDAHULUAN

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

BAB II. TINJAUAN PUSTAKA

Bab ini berisi penjelasan secara detail mengenai dasar-dasar penunjang untuk mendukung pembuatan tugas akhir ini.

BAB III. PERANCANGAN

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

BAB IV. IMPLEMENTASI

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

BAB V. UJI COBA DAN EVALUASI

Bab ini menjelaskan tahap pengujian sistem dan pengujian performa dalam skenario mobilitas *vehicular* yang dibuat.

BAB VI. PENUTUP

Bab ini merupakan bab terakhir yang menyampaikan kesim-

pulan dari hasil uji coba yang dilakukan terhadap rumusan masalah yang ada dan saran untuk pengembangan lebih lanjut.

BAB 2

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*, alat, serta definisi yang digunakan dalam pembuatan Tugas Akhir.

2.1 *Vehicular Ad hoc Network* (VANET)

Vehicular Ad Hoc Network (VANET) [3] merupakan bentuk dari jaringan nirkabel yang bersifat *ad hoc* dengan fungsi menyediakan sarana komunikasi antar kendaraan dan *Roadside Unit* (RSU). Tujuan utama dari jaringan VANET adalah sebagai berikut:

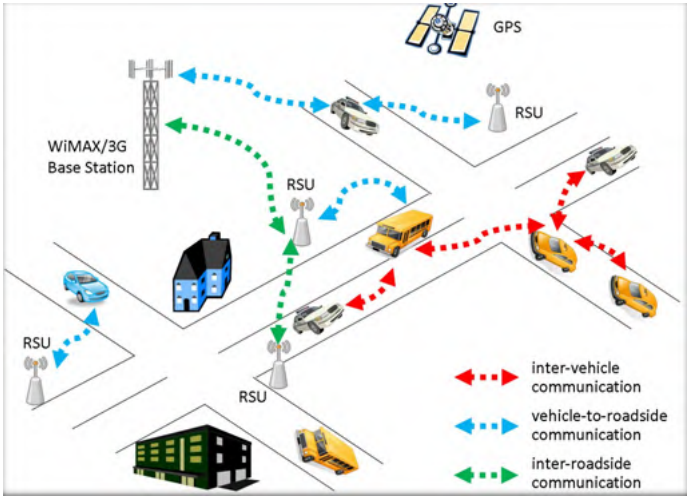
- Untuk membangun konektivitas yang dapat berlangsung di mana saja (*ubiquitous connectivity*), khususnya ketika pengguna sedang berada di jalan.
- Komunikasi *vehicle-to-vehicle* (V2V) yang efisien sehingga memungkinkan adanya *Intelligent Transportation System*.

Selain kedua hal tersebut, VANET juga bisa dimanfaatkan untuk meningkatkan keselamatan pengemudi dalam berkendara dan informasi lalu lintas (berupa informasi kemacetan, kecelakaan, dan lain-lain).

Jika dibandingkan dengan jaringan komunikasi lainnya, VANET memiliki karakteristik yang unik. Dalam VANET, mobilitas kendaraan dibatasi oleh jalan dan bangunan. Kecepatan kendaraan dimulai dari nol hingga batas kecepatan dari kendaraan atau aturan lalu lintas yang ada di jalan tersebut. Jaringan VANET dipengaruhi oleh ukuran dan kecepatan kendaraan, lokasi geografis antar kendaraan, dan frekuensi komunikasi yang biasanya jarang terjadi karena kanal komunikasi yang tidak reliable. Dari masalah yang ditimbulkan oleh berbagai karakteristik tersebut, periset mengajukan berbagai isu riset di bidang *routing*, diseminasi data, data sharing, dan isu 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 permasalah dari karakteristik unik VANET sehingga tidak cocok untuk komunikasi *vehicle-to-vehicle* (V2V) dalam VANET [1].

Ilustrasi VANET dapat dilihat pada Gambar 2.1.



Gambar 2.1: Ilustrasi VANET [4]

Dalam Tugas Akhir ini, penulis akan mengimplementasikan *routing protocol* AODV-PNT dan menguji performa protokol tersebut pada lingkungan VANET.

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

Ad hoc On-Demand Distance Vector merupakan sebuah *routing protocol* yang bersifat reaktif, hal ini berarti protokol akan mengolah dan memproses rute hanya ketika melakukan *request* saja. Berdasarkan RFC 3561 [5], terdapat dua proses utama dalam AODV, yaitu *route discovery* dan *route maintenance*. *Route discovery* dila-

kukan dengan menginisiasi paket *route request* (RREQ) dan diikuti sebuah *route reply* (RREP) yang akan diberikan oleh *node* tujuan ketika sudah menerima paket RREQ. Selain paket RREQ dan RREP, terdapat paket *route error* (RERR) yang membantu dalam proses *route maintenance*.

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

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

Tahapan yang dilakukan pada proses *route discovery* dalam AODV dimulai dengan node pengirim membuat paket *route request* (RREQ) dan melakukan *broadcast*. *Neighbor node* yang menerima paket RREQ akan memeriksa *routing table* masing-masing. Jika *destination sequence number* yang terdapat pada paket RREQ sama atau lebih kecil dari yang ada pada routing table dan rute menuju *node* tujuan belum ditemukan, maka paket tersebut tidak akan dilanjutkan (*drop*). Jika *destination sequence number* pada RREQ lebih besar dibandingkan yang terdapat pada *routing table*, maka *entry* pada *routing table* akan diperbarui dan paket tersebut akan di-*broadcast* ke *neighbor node* sekaligus membuat *reverse path* menuju *node* pengirim. Jika rute menuju *node* tujuan sudah ada di dalam *routing table* dan memiliki *routing flags* "*up*", maka *node* akan mengirimkan paket RREP ke *node* pengirim.

Tujuan dari *route maintenance* adalah untuk mengontrol status dari rute-rute yang tercatat pada *routing table*. Jika ada koneksi an-

tar *node* yang terputus pada suatu rute, maka *node* yang mendeteksi kejadian tersebut akan mengirimkan paket RERR kepada semua *node* yang bekomunikasi menggunakan rute tersebut. Seluruh *node* yang menerima paket RERR akan menandai jalur tersebut menjadi "*down*" dan meneruskan paket tersebut hingga sampai pada *node* pengirim. Jika *node* pengirim masih memerlukan komunikasi data dengan *node* penerima, maka proses *route discovery* antara *node* pengirim dan tujuan akan dilakukan kembali.

2.3 *Ad hoc On-Demand Distance Vector Predicting Node Trend* (AODV-PNT)

AODV-PNT [2] merupakan modifikasi dari *routing protocol* AODV. Perubahan yang dilakukan dalam AODV-PNT adalah menambahkan mekanisme bobot yang selanjutnya disebut sebagai *Total Weight of the Route* (TWR) dan memprediksikan nilai TWR yang akan datang (*future* TWR). Kedua hal tersebut akan menjadi dasar dalam pemilihan *relay node* (*neighbor node* yang "layak" untuk meneruskan paket RREQ). TWR dihitung sebelum *node* sumber mengirimkan paket RREQ dan sebelum *relay node* meneruskan paket RREQ.

Formula TWR dirancang dengan mempertimbangkan beberapa faktor berikut:

- Kecepatan dan akselerasi
  Semakin besar perbedaan kecepatan dan akselerasi antar dua kendaraan, maka semakin besar pula kemungkinan dua kendaraan tersebut saling berjauhan. Asumsi tersebut berimplikasi pada TWR dalam bentuk pelebaran nilai TWR. Alasannya adalah jika kedua kendaraan saling berjauhan, maka semakin besar kemungkinan terjadinya kerusakan koneksi antar kendaraan tersebut. Jika kedua kendaraan tersebut berada di dalam radius komunikasi dan bergerak dengan kecepatan dan akselerasi yang relatif sama antara satu dengan yang lainnya, maka bisa diasumsikan kedua kendaraan tersebut akan bera-

da di radius komunikasi dalam durasi yang lebih lama.

- Arah kendaraan
  Jika ada dua kendaraan yang bergerak dengan arah yang relatif sama, maka kedua kendaraan tersebut akan berada di radius komunikasi dalam durasi yang lebih lama.

- Kualitas hubungan antar kendaraan
  Kualitas hubungan yang dimaksud adalah apakah *node* yang akan mengirimkan paket dan *node* penerimanya masih berada di dalam radius komunikasi. Dalam lingkungan VANET, terdapat banyak *neighbor node*, bangunan, dan halangan lainnya yang dapat mempengaruhi kualitas hubungan.

  Kualitas hubungan didefinisikan dalam indeks stabilitas (*stability index*) [6]. Formula dari indeks stabilitas dapat dilihat pada persamaan 2.1.

$$s_{ij} = 1 - \frac{min\left(\sqrt{(i_x - j_x)^2 + (i_y - j_y)^2}; r\right)}{r} \tag{2.1}$$

  dimana,
  i_x, i_j : Koordinat dari kendaraan i
  j_x, j_y : Koordinat dari kendaraan j
  r : Jarak transmisi maksimum

  Nilai indeks stabilitas yang paling baik adalah 1. Indeks stabilitas dengan nilai 1 dihasilkan ketika kendaraan i dan j memiliki vektor pergerakan yang sama. Nilai indeks stabilitas yang paling buruk adalah 0. Indeks stabilitas dengan nilai 0 didapatkan ketika kendaraan i dan j memiliki jarak yang lebih jauh daripada jarak transmisi maksimum. Indeks stabilitas dengan nilai 0 mengimplikasikan kendaraan i dan j berada

di luar radius komunikasi dari masing-masing kendaraan. Kemudian nilai kualitas hubungan Q [2] dihitung dengan rumus pada persamaan 2.2.

$$Q = \frac{1}{s_{ij}} \tag{2.2}$$

Berdasarkan faktor-faktor di atas, rumus TWR [2] diekspresikan dengan persamaan 2.3.

$$TWR = f_s \times |S_n - S_d| + f_a \times |A_n - A_d| + f_d \times |\Theta_n - \Theta_d| + f_q \times Q \tag{2.3}$$

di mana,
S_n, A_n, Θ_n : Kecepatan, akselerasi, dan arah *next-hop node*
S_d, A_d, Θ_d : Kecepatan, akselerasi, dan arah *node* tujuan
f_s : Faktor pengali kecepatan
f_a : Faktor pengali akselerasi
f_d : Faktor pengali arah
f_q : Faktor pengali kualitas hubungan
Q : Kualitas hubungan antara *node* sumber dengan *next-hop node*

Nilai TWR ditentukan oleh perbedaan kecepatan, akselerasi, dan arah dari *next-hop node* dan *node* tujuan, serta kualitas hubungan antara *node* sumber dengan *next-hop node*. Formula TWR mengimplikasikan *next-hop node* yang baik adalah *node* yang memiliki nilai TWR yang rendah karena memiliki kecepatan, akselerasi, dan arah yang relatif sama dengan *node* tujuan, serta memiliki kualitas hubungan yang relatif bagus dengan *node* sumber [2].

Dengan mempertimbangkan karakteristik topologi VANET yang sering berubah dengan cepat, maka setelah melakukan perhitungan TWR akan dilakukan prediksi nilai TWR untuk 3 detik berikutnya.

- Prediksi kecepatan dan akselerasi kendaraan
  Karena interval dari waktu sekarang dengan waktu berikutnya sangat singkat, maka akselerasi diasumsikan bernilai konstan. Kecepatan pada waktu berikutnya dapat dihitung de-

ngan rumus akselerasi.

- Prediksi arah kendaraan
  Jika ada persimpangan jalan dan kendaraan ingin berbelok, biasanya kendaraan akan melakukan pengereman dan memberikan lampu sein sebagai tanda akan berbelok. Kendaraan yang memenuhi kondisi tersebut akan menghitung vektor arah dengan bantuan layanan GPS.

- Prediksi kualitas hubungan antar kendaraan
  Indeks stabilitas dihitung dengan memanfaatkan koordinat dari kendaraan. Posisi koordinat yang akan datang dari kendaraan tersebut dapat diperoleh dengan memanfaatkan informasi kecepatan dan akselerasi yang telah dihitung sebelumnya.

Hasil prediksi nilai dari faktor-faktor di atas digunakan untuk menghitung nilai TWR yang akan datang (*future* TWR). Setelah mendapatkan hasil TWR dan TWR untuk 3 detik yang akan datang, dilakukan proses pemilihan *next-hop node* (*relay node*) dengan aturan pada Tabel 2.1.

Tabel 2.1: Aturan Pemilihan *Relay Node* [2]

TWR	*State*	*Future* TWR	Keputusan
Optimal	Tidak stabil	Lebih baik	Pilih *node* tersebut
Optimal	Stabil	Lebih buruk	Pilih *node* tersebut
Suboptimal	Tidak stabil	Lebih baik	Pilih *node* tersebut
Suboptimal	Stabil	Lebih buruk	Pilih *node* tersebut
Kondisi lainnya			Abaikan *node*

Berikut adalah penjelasan dari aturan pemilihan *relay node*:

- TWR sekarang
  Jika *node* sumber sudah menghitung TWR dari seluruh *node* yang ada di sekitarnya, nilai-nilai TWR tersebut akan dievaluasi. TWR dikatakan “optimal” apabila TWR tersebut adalah

nilai yang paling rendah, selain itu TWR dianggap sebagai "suboptimal".

- *State*
  Sebelum menentukan *state*, *threshold* W akan didefinisikan terlebih dahulu. *State* merupakan perbandingan antara *threshold* W dengan nilai absolut dari selisih TWR dan TWR yang akan datang (*future* TWR). Selanjutnya nilai absolut ini disebut sebagai ΔTWR. Jika ΔTWR lebih kecil dari *threshold* W, maka *next-hop node* tersebut dianggap stabil. Jika ΔTWR lebih besar dari *threshold* W, maka *next-hop node* tersebut dianggap tidak stabil.

- TWR yang akan datang (*future* TWR)
  TWR yang akan datang dikatakan "lebih baik" apabila nilainya lebih rendah dari TWR sekarang. TWR yang akan datang dikatakan "lebih buruk" apabila nilainya lebih besar dari TWR sekarang.

Dalam proses pemilihan *relay node*, *node* sumber mendapatkan informasi pergerakan dari *neighbor node*. Kemudian dilakukan evaluasi apakah nilai TWR dari *neighbor node* tersebut optimal, apakah *node* tersebut stabil, dan apakah *future* TWR lebih baik daripada TWR sekarang. Setelah keputusan pemilihan node dilakukan, *node* sumber melakukan *multicast* ke seluruh *relay node*.

Dalam AODV, setiap *node* melakukan *broadcast* HELLO *message* untuk *route maintenance* secara periodik. Dalam AODV-PNT dilakukan modifikasi pada struktur HELLO *message* agar memungkinkan *node* pengirim melakukan perhitungan TWR. Struktur HELLO *message* dalam AODV-PNT dapat dilihat pada Tabel 2.2

Tabel 2.2: Struktur Paket HELLO AODV-PNT

Dst IP Address	*Dst Sequence Number*	*Hop Count*
Lifetime	***Speed***	***Acceleration***

Setelah HELLO *message* diterima oleh *node* pengirim, dilakukan perhitungan TWR dan menghasilkan sebuah *relay node set*. Kemudian *node* pengirim melakukan *multicast* RREQ ke *relay node set*. Jika *relay node* mengetahui rute menuju *node* tujuan, *node* tersebut akan mengirimkan RREP dan *reverse path*-nya. Jika *relay node* tidak mengetahui rute menuju *node* tujuan, maka *relay node* tersebut akan membuat *relay node set* yang baru dan melakukan *multicast* RREQ ke *relay node set* tersebut. Proses ini akan terus berlanjut hingga RREQ sampai di *node* tujuan atau akan berhenti jika *node* tujuan tidak ditemukan.

2.4 *Simulation of Urban Mobility* (SUMO)

Simulation of Urban Mobility atau SUMO merupakan sebuah program *simulator* lalu lintas. SUMO dikembangakan sejak tahun 2000 dengan tujuan untuk mengakomodasi penelitian-penelitian yang melibatkan pergerakan kendaraan di jalan raya, terutama dalam daerah dengan penduduk yang padat (*urban*). Publikasi referensi terbaru tentang SUMO ditulis oleh Krajzewicz et al. [7] pada tahun 2012.

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

- netgenerate
  netgenerate merupakan *tool* yang berfungsi untuk membuat peta berbentuk seperti *grid*, *spider*, dan bahkan *random network*. Sebelum proses netgenerate, pengguna dapat menentukan kecepatan maksimum jalan dan membuat *traffic light* pada peta. Hasil dari netgenerate ini berupa *file* dengan ekstensi .net.xml. Pada Tugas Akhir ini netgenerate digunakan untuk membuat peta skenario *grid*.

- netconvert
  netconvert merupakan program CLI yang berfungsi untuk melakukan konversi dari peta seperti OpenStreetMap menjadi format *native* SUMO. Pada Tugas Akhir ini penulis menggunakan netconvert untuk mengkonversi peta dari OpenStreetMap.
- randomTrips.py
  Tool dalam SUMO untuk membuat rute acak yang akan dilalui oleh kendaraan dalam simulasi.
- route2trips.py
  Membuat detail perjalanan setiap kendaraan berdasarkan output dari randomTrips.py.
- duarouter
  Tool dalam SUMO untuk melakukan perhitungan rute berdasarkan definisi yang diberikan dan memperbaiki kerusakan rute.
- sumo
  Program yang melakukan simulasi lalu lintas berdasarkan data-data yang didapatkan dari netgenerate (skenario *grid*) atau netconvert dan randomTrips.py. Hasil simulasi dapat di-*export* ke sebuah file untuk nantinya dikonversi menjadi format lain.
- sumo-gui
  GUI untuk melihat simulasi yang dilakukan oleh SUMO secara grafis.
- traceExporter.py
  Tool yang bertujuan untuk mengkonversi output dari sumo menjadi format yang dapat digunakan pada *simulator* lain. Pada Tugas Akhir ini penulis menggunakan traceExporter.py untuk mengkonversi data menjadi format .tcl yang dapat digunakan pada NS-2.

2.5 OpenStreetMap

OpenStreetMap (OSM) [8] merupakan sebuah proyek kolaboratif untuk membuat sebuah peta dunia yang dapat dengan bebas diubah oleh siapapun. 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 dianggap sebagai contoh utama dalam informasi geografis yang diberikan secara sukarela.

Pengembangan OSM diinspirasi oleh kesuksesan Wikipedia pengaruh dari peta *proprietary* di UK dan tempat lain. Sejak diluncurkan hingga sekarang OSM telah berkembang hingga memiliki lebih dari dua juta pengguna yang teregistrasiyang dapat mengambil data menggunakan survey manual, piranti GPS, *aerial photography* dan sumber bebas lainnya. Data yang terdapat pada OSM berada dalam lisensi *Open Database License* sehingga data dari OSM dapat dengan bebas digunakan oleh semua orang.

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

2.6 JOSM

JOSM (*Java OpenStreetMap Editor*) adalah sebuah alat untk menyunting data yang didapatkan dari OpenStreetMap. Aplikasi JOSM dapat diunduh pada alamat https://josm.openstreetmap.de/. Penulis menggunakan aplikasi ini untuk menyunting dan merapikan potongan peta yang diunduh dari OpenStreetMap.

2.7 AWK

AWK merupakan sebuah *Domain Specific Language* (DSL) yang didesain untuk *text processing* dan biasanya digunakan sebagai alat

ekstraksi data dan *reporting*. AWK bersifat *data-driven* yang berisikan kumpulan perintah yang akan dijalankan pada data tekstual baik secara langsung pada file atau digunakan sebagai bagian dari *pipeline*. 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 lain-lain.

2.8 *Network Simulator 2* (NS-2)

NS-2 [9] merupakan sebuah *discrete event simulator* yang didesain untuk membantu penelitian pada bidang jaringan komputer. Versi terbaru dari NS-2 adalah ns-2.35 yang dirilis pada tanggal 4 November 2011. Dalam membuat sebuah simulasi, NS-2 menggunakan dua buah bahasa pemrograman, yaitu C++ dan OTcl. Bahasa C++ digunakan untuk mengimplementasi bagian-bagian jaringan yang akan disimulasikan. Sedangkan OTcl digunakan untuk menulis skenario simulasi jaringan. NS-2 mendukung sistem operasi GNU/Linux, FreeBSD, OS X dan Solaris. NS-2 juga dapat dijalankan pada sistem operasi Windows dengan menggunakan Cygwin.

2.8.1 Instalasi NS-2

Sebelum melakukan instalasi NS-2, hal pertama yang harus dilakukan adalah meng-*install* dependensi yang dibutuhkan. Salah satu dependensi dari NS-2 adalah GCC versi 4.4. Gambar 2.2 menunjukkan dependensi NS-2 beserta GCC 4.4 dan cara meng-*install*-nya pada distribusi Debian dan turunannya.

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

Gambar 2.2: Perintah untuk meng-*install* dependensi NS-2 pada distribusi Debian

Setelah semua dependensi ter-*install*, selanjutnya unduh *source code* NS-2. Jika proses unduh sudah selesai, lakukan ekstraksi *file* tarball seperti yang ditunjukkan pada Gambar 2.3.

```
wget \
http://jaist.dl.sourceforge.net/project/nsnam/allinone/ns-
 ↪  allinone-2.35/ns-allinone-2.35.tar.gz
tar -xvf ns-allinone-2.35.tar.gz
```

Gambar 2.3: Perintah untuk mengunduh dan mengekstrak NS-2

Lakukan navigasi menuju folder "linkstate" yang terdapat pada *ns-allinone-2.35/ns-2.35/linkstate*. Kemudian buka *file* yang bernama "ls.h" dan pergi ke baris 137 pada kode tersebut. Setelah itu ubah kata "erase" menjadi "this->erase". *Screenshot* dari perubahan yang dilakukan pada *file* tersebut dapat dilihat pada Gambar 2.4.

Dalam Tugas Akhir ini penulis akan menggunakan *aggregate initialization* sehingga harus menambahkan beberapa opsi CFLAGS dalam Makefile NS. Opsi CFLAGS yang ditambahkan adalah "-std=c++0x -Wno-literal-suffix". Perubahan yang dilakukan dapat dilihat pada Gambar 2.4 dan 2.5. Perubahan terakhir yang harus dilakukan adalah mengubah setiap baris yang menggunakan fungsi "hash" menjadi "::hash" pada *source code* "mdart_adp.cc" di dalam direktori *ns-allinone-2.35/ns-2.35/mdart*. Perubahan ini dilakukan agar fungsi *built-in* std::hash dapat dibedakan dengan fungsi hash yang digunakan pada *source code* "mdart_adp.cc" ketika proses *compile*.

Setelah semua tahap di atas selesai, jalankan skrip instalasi NS-2 dengan memasukkan perintah *ns-allinone-2.35/./install* pada terminal dan tunggu hingga proses instalasi selesai.

2.8.2 Penggunaan Skrip OTcl

OTcl [10] merupakan ekstensi *object oriented* dari bahasa pemrograman Tcl. Bahasa OTcl digunakan sebagai bahasa *scripting* pa-

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

Gambar 2.4: Perubahan yang Dilakukan pada Skrip ls.h

```
CFLAGS    += $(CCOPT) $(DEFINE) -std=c++0x -Wno-literal-suffix
```

Gambar 2.5: Perubahan yang Dilakukan pada Skrip Makefile.in

da NS-2 untuk mengatur lingkungan dan skenario simulasi. Setiap *class* yang terdapat pada OTcl memiliki *binding* pada kode C++. Hal ini memungkinkan pembuatan skenario simulasi tanpa perlu menggunakan bahasa C++ secara langsung. Gambar 2.6 memunjukkan contoh potongan kode OTcl pada NS-2 untuk melakukan pengaturan lingkungan simulasi.

Baris 1 hingga baris 9 digunakan untuk mengatur lingkungan simulasi. Kemudian pada baris 11, objek *simulator* yang akan menjalankan simulasi VANET diinstansiasi. Baris 12 menginstansiasi koneksi *wireless* yang akan digunakan oleh setiap *node*. Pada baris 13, semua variabel pengaturan pada *node* dimasukkan. Ketika simulasi berjalan, seluruh *node* akan menggunakan pengaturan yang sama.

2.8.3 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. Se-

```
set val(chan) Channel/WirelessChannel ;# channel type
set val(prop) Propagation/TwoRayGround ;# radio-propagation model
set val(netif) Phy/WirelessPhy ;# network interface type
set val(mac) Mac/802_11 ;# 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) 50 ;# max packet in ifq

set ns_ [new Simulator]
set topo       [new Topography]
$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
  ↪  [new $val(chan)] -agentTrace ON -routerTrace ON -macTrace OFF
  ↪  -movementTrace OFF -topoInstance $topo
```

Gambar 2.6: Potongan kode pengaturan lingkungan simulasi VANET

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

Contoh jenis paket yang beredar di dalam simulasi ditunjukkan pada Gambar 2.7. Gambar 2.6a hingga 2.6d menunjukkan contoh paket *routing control* AODV dari NS-2. Paket *routing control* selalu ditandai dengan tulisan "RTR" pada kolom keempat. Kolom ketujuh menunjukkan informasi nama *routing protocol*. Kolom kedelapan menunjukkan ukuran dari paket *routing control*. Kolom terakhir menunjukkan jenis paket *routing control*.

Gambar 2.6e 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 dari paket data.

Pola penting lainnya adalah paket yang dikirim 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*, analisis hasil simulasi dapat dilakukan.

```
r 0.000916039 _69_ RTR  --- 0 AODV 44 [0 ffffffff 17 800] -------
 ↪   [23:255 -1:255 1 0] [0x1 1 [23 2] 4.000000] (HELLO)
```

(a) Contoh paket HELLO

```
s 50.000000000 _148_ RTR  --- 0 AODV 48 [0 0 0 0] ------- [148:255
 ↪  -1:255 30 0] [0x2 1 1 [149 0] [148 72]] (REQUEST)
```

(b) Contoh paket RREQ

```
s 50.021614475 _149_ RTR  --- 0 AODV 44 [0 0 0 0] ------- [149:255
 ↪   148:255 30 3] [0x4 1 [149 68] 10.000000] (REPLY)
```

(c) Contoh paket RREP

```
s 62.000000000 _125_ RTR  --- 0 AODV 32 [0 0 0 0] ------- [125:255
 ↪  -1:255 1 0] [0x8 1 [148 0] 0.000000] (ERROR)
```

(d) Contoh paket RRER

```
r 62.042373176 _149_ AGT  --- 12 cbr 532 [13a 95 3f 800] -------
 ↪   [148:0 149:0 25 149] [12] 6 0
```

(e) Contoh paket data

Gambar 2.7: Contoh Pola Paket *Trace File* NS-2

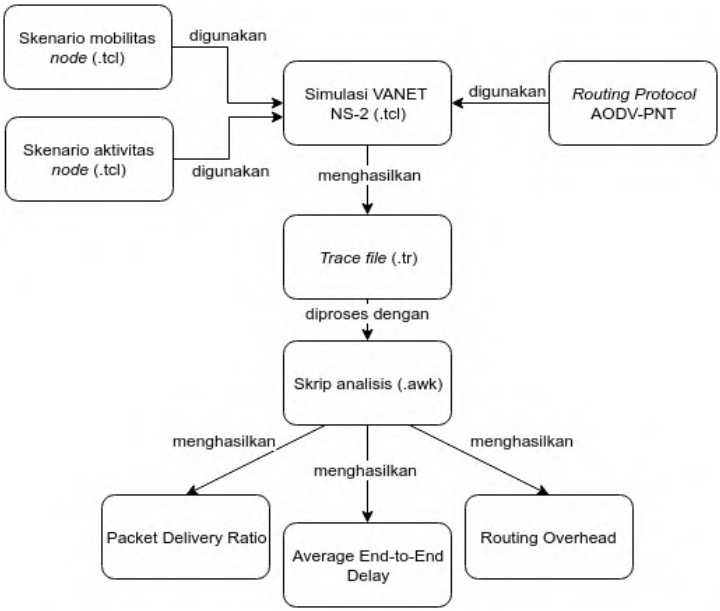
BAB 3

PERANCANGAN

Perancangan merupakan bagian yang sangat penting dari implementasi sistem. Bab ini secara khusus akan menjelaskan rancangan sistem yang dibuat pada Tugas Akhir. Bagian yang akan dijelaskan pada bab ini berawal dari deskripsi umum hingga perancangan skenario, alur dan implementasinya.

3.1 Deskripsi Umum

Pada Tugas Akhir ini akan dilakukan implementasi dan analisis dari *routing protocol* AODV-PNT pada NS-2. Diagram rancangan simulasi dapat dilihat pada Gambar 3.1.



Gambar 3.1: Diagram Alur Rancangan Simulasi

Dalam Tugas Akhir ini, terdapat 2 jenis skenario yang digunak-

an sebagai perbandingan pengukuran, yaitu peta berbentuk *grid* dan peta riil dari lingkungan lalu lintas kota Surabaya. Skenario *grid* dibuat dengan bantuan aplikasi SUMO. Skenario riil yang didasarkan pada peta lalu lintas kota Surabaya diambil dari OpenStreetMap dan dirapikan menggunakan aplikasi JOSM. Setelah *file* peta sudah terbentuk, dilakukan simulasi lalu lintas dengan SUMO. Hasil simulasi SUMO digunakan untuk simulasi protokol AODV-PNT pada NS-2. Kemudian hasil simulasi NS-2 dianalisis dengan menggunakan skrip AWK untuk menghitung metrik analisis berupa *packet delivery ratio*, *average end-to-end delay*, dan *routing overhead*. Perhitungan metrik analisis bertujuan untuk mengukur performa dari protokol AODV dan AODV-PNT.

3.2 Perancangan Skenario *Grid*

Pembuatan peta *grid* diawali dengan menentukan panjang jalan dan jumlah *vertex*. Secara *default*, peta *grid* akan berbentuk segi empat. Alur pembuatan peta grid dapat dilihat pada Gambar 3.2.

Panjang jalan dan jumlah *vertex* yang sudah ditentukan akan dimasukan sebagai argumen untuk netgenerate. Secara opsional, batas kecepatan untuk setiap jalan juga dapat ditentukan melalui argumen netgenerate. Kemudian hasil dari netgenerate digunakan sebagai argumen untuk randomTrips.py. Program randomTrips.py berfungsi untuk mendefinisikan rute perjalanan dari seluruh *node*. Keluaran dari randomTrips.py 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 yang bisa diproses oleh NS-2 melalui traceExporter.py. 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*

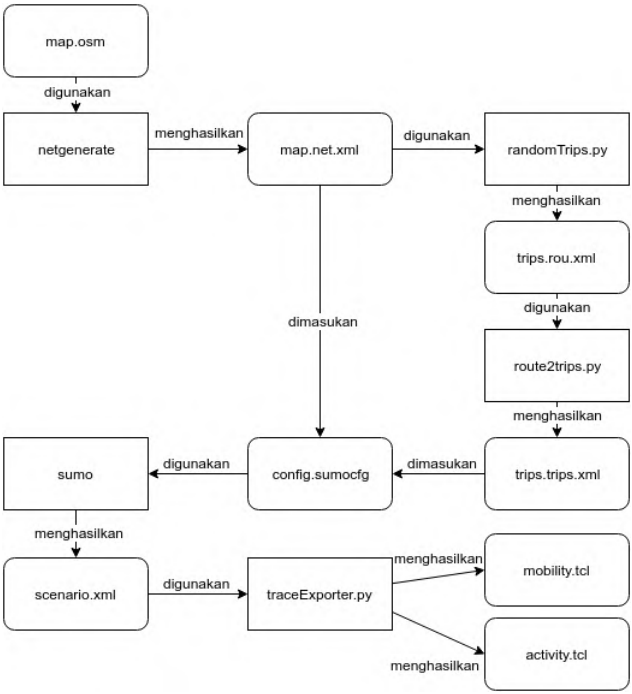
3.3 Perancangan Skenario Riil

Perancangan skenario riil diawali dengan pemilihan daerah yang akan digunakan sebagai model untuk simulasi. Setelah mendapatkan daerah yang diinginkan, unduh daerah tersebut di situs OpenStreetMap melalui fitur *export*. Kemudian potongan peta tersebut dirapikan menggunakan program JOSM. Agar bisa semirip mungkin dengan dunia nyata, lakukan pengaturan pada interval lampu lalu lintas. Kemudian hapus jalan yang terputus dari potongan peta tersebut sehingga menjadi daerah yang tertutup.

Setelah potongan peta dirapikan, buat sebuah *type file* yang mendefinisikan spesifikasi batasan simulasi lalu lintas, seperti batas kecepatan pada kelas jalan tertentu, dan lain-lain. Kemudian peta dikonversi dengan netconvert berdasarkan *type file* yang telah dibuat. Hasil konversi tersebut digunakan untuk membuat *file* rute pergerak-

an kendaraan melalui randomTrips.py dan route2trips.py. Kemudian *file* peta yang telah dikonversi dan *file* yang berisi rute digunakan untuk simulasi SUMO.

Hasil dari simulasi tersebut di-*export* ke dalam format yang bisa diproses oleh NS-2 melalui traceExporter.py. Hasilnya berupa *file* yang berisi mobilitas dari setiap *node* (mobility.tcl) dan informasi *lifetime* dari setiap *node* (activity.tcl). Alur pembuatan peta riil dapat dilihat pada Gambar 3.3.



Gambar 3.3: Alur Pembuatan Skenario Riil

3.4 Perancangan Implementasi AODV-PNT

Sebelum *relay node set* terbentuk, protokol AODV-PNT menghitung TWR dan *future* TWR dari masing-masing *neighbor node*. Perhitungan TWR dilakukan dengan memanfaatkan informasi kecepatan, akselerasi, dan posisi dari *neighbor node*. Sedangkan perhitungan *future* TWR menggunakan rumus fisika gerak lurus untuk memprediksi informasi yang akan datang.

AODV-PNT mengasumsikan *node* sumber dapat mengetahui posisi dari *neighbor node* dan *node* tujuan melalui layanan GPS, tetapi implementasi AODV pada NS-2 tidak menggunakan *location service* sehingga tidak memungkinkan untuk mendapatkan posisi dari setiap node. Untuk itu diperlukan modifikasi agar dapat mengetahui informasi posisi dari *neighbor node*. Modifikasi dilakukan dengan menambahkan *field* koordinat x dan y pada struktur paket HELLO. Perubahan struktur paket HELLO dapat dilihat pada Tabel 3.1.

Tabel 3.1: Modifikasi Struktur Paket HELLO AODV-PNT

Dst IP Address	*Dst Sequence Number*	*Hop Count*		
Lifetime	*Speed*	*Acceleration*	**x**	**y**

Dalam Tugas Akhir ini, simulasi dilakukan dengan *node* sumber dan *node* tujuan berada dalam posisi diam (*stationary node*) sehingga perlu dilakukan penyesuaian terhadap formula TWR. Perhitungan TWR terdiri atas empat faktor, yaitu kecepatan, akselerasi, arah, dan kualitas hubungan.

Faktor kecepatan, akselerasi dan arah didapat dari selisih nilai yang dimiliki oleh *next-hop node* terhadap *node* tujuan. *Node* tujuan yang berada dalam posisi diam menyebabkan faktor arah menjadi tidak relevan. Alasannya adalah *node* tujuan tidak memiliki arah. Agar tidak terjadi pengurangan bobot pada TWR, maka faktor arah diubah menjadi jarak antara *next-hop node* dengan *node* tujuan. Koordinat dari *node* tujuan dapat diketahui karena *node* tersebut adalah

stationary node. Berikut adalah perubahan formula TWR:

$$TWR = f_s \times |S_n - S_d| + f_a \times |A_n - A_d| + f_d \times dist\,(n,d) + f_q \times Q \tag{3.1}$$

dimana,
$dist\,(n,d)$: Jarak antara *next-hop node* dengan *node* tujuan

Selain formula TWR, formula indeks stabilitas juga perlu diubah karena jika hasil fungsi minimum antara jarak dan radius transmisi maksimum menghasilkan nilai dari radius transmisi maksimum, akan terjadi pembagian dengan nol (*division by zero*) pada persamaan 2.2. Berikut adalah perubahan formula indeks stabilitas:

$$s_{ij} = 1 - \frac{min\left(\sqrt{(i_x - j_x)^2 + (i_y - j_y)^2}; r\right)}{r+1} \tag{3.2}$$

Perhitungan *future* TWR membutuhkan prediksi nilai kecepatan dan posisi yang akan datang. Berikut adalah beberapa rumus fisika gerak lurus yang akan digunakan dalam menghitung nilai prediksi:

- Perhitungan akselerasi

$$a = \frac{\Delta v}{\Delta t} \tag{3.3}$$

dimana,
Δv : Selisih kecepatan sekarang dan kecepatan sebelumnya
Δt : Selisih waktu

- Perhitungan kecepatan yang akan datang

$$v' = v + a \times t \tag{3.4}$$

dimana,
v : Kecepatan sekarang

a : Akselerasi
t : Waktu (dalam detik)

- Perhitungan posisi yang akan datang

$$x' = x + v_0 t + \frac{1}{2} a t^2 \tag{3.5}$$

$$y' = y + v_0 t + \frac{1}{2} a t^2 \tag{3.6}$$

dimana,
x, y : Koordinat sekarang
v_0 : Kecepatan sekarang
a : Akselerasi
t : Waktu (dalam detik)

Nilai waktu yang digunakan pada rumus-rumus di atas adalah 3 detik. Nilai kecepatan dan akselerasi didapatkan dari paket HELLO yang dikirimkan oleh *neighbor node* secara periodik.

3.5 Perancangan Metrik Analisis

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

3.5.1 *Packet Delivery Ratio* (PDR)

Packet delivery ratio merupakan perbandingan dari jumlah paket data yang dikirim dengan paket data yang diterima. *Packet delivery ratio* dihitung menggunakan persamaan 3.7, di mana $received$ adalah jumlah paket data yang diterima dan $sent$ adalah jumlah paket data yang dikirim.

$$PDR = \frac{Data_{received}}{Data_{sent}} \tag{3.7}$$

3.5.2 *Average End-to-End Delay*

Average end-to-end delay adalah waktu rata-rata dari setiap paket ketika sampai di tujuan. Semua paket, termasuk *delay* yang diakibatkan oleh paket *routing*, juga akan diperhitungkan. Paket yang akan dimasukan ke dalam perhitungan hanya paket yang berhasil sampai di tujuan. *Average end-to-end delay* dihitung menggunakan persamaan 3.8, di mana i adalah nomor paket yang berhasil sampai di tujuan, $t_{received}\,[i]$ adalah waktu ketika paket i dikirim, $t_{sent}\,[i]$ adalah waktu ketika paket i diterima, dan $pktCounter$ adalah jumlah paket yang berhasil sampai ditujuan.

$$Delay = \frac{\sum_{i=0}^{n} t_{received}\,[i] - t_{sent}\,[i]}{pktCounter} \tag{3.8}$$

3.5.3 *Routing Overhead* (RO)

Routing overhead merupakan jumlah paket *routing control* yang ditransmisikan selama simulasi terjadi. Paket kontrol yang dihitung adalah jumlah *Route Request* (RREQ), *Route Reply* (RREP) dan *Route Error* (RRER). Rumus dari *routing overhead* dapat dilihat pada persamaan 3.9.

$$RO = RREQ_{sent} + RREP_{sent} + RRER_{sent} \tag{3.9}$$

BAB 4

IMPLEMENTASI

Bab ini memberikan bahasan mengenai implementasi dari perancangan sistem yang dijelaskan pada bab sebelumnya.

4.1 Implementasi Skenario *Grid*

Skenario *grid* dibuat melalui *tool* netgenerate dari SUMO. Skenario *grid* dibuat dengan panjang jalan 200 m dan luas peta 1000 m x 1000 m. Jumlah titik persimpangan antara jalan vertikal dan horizal sebanyak 6 titik x 6 titik. Kecepatan kendaraan yang diperbolehkan diatur sebesar 15 m/s.

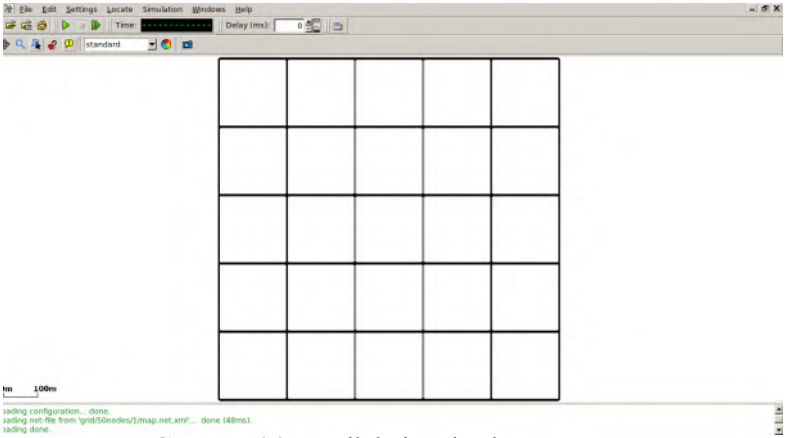
Untuk membuat peta *grid* dengan spesifikasi tersebut, digunakan perintah seperti Gambar 4.1.

```
netgenerate --grid --grid.number=6 --grid.length=200
 ↪  --default.speed=15 --tls.guess=1
 ↪  --output-file=map.net.xml
```

Gambar 4.1: Perintah untuk Membuat Peta *Grid*

Gambar peta yang dibuat dengan netgenerate dapat dilihat pada Gambar 4.2.

Setelah peta terbentuk, dilakukan pembuatan titik asal dan titik tujuan untuk setiap kendaraan secara acak melalui modul randomTrips.py seperti pada Gambar 4.3. Opsi -e diisi dengan jumlah kendaraan yang diinginkan dan --intermediate diisi dengan nilai yang besar agar setiap kendaraan memiliki banyak rute alternatif sehingga setiap kendaraan dapat dipastikan aktif hingga simulasi mobilitas berakhir.



Gambar 4.2: Hasil dari Perintah netgenerate

```
python $SUMO_HOME/tools/randomTrips.py --seed=$RANDOM
 ↪  --fringe-factor 5.0 -e num_nodes --intermediate=$RANDOM
 ↪  -n map.net.xml -r map.passenger.rou.xml --vehicle-class
 ↪   passenger --vclass passenger --trip-attributes
 ↪   'departLane="best"'
```

Gambar 4.3: Perintah untuk Membuat Titik Asal dan Titik Tujuan Kendaraan

Setelah titik asal dan titik akhir didefinisikan, dilakukan pembuatan rute yang akan digunakan oleh kendaraan menggunakan perintah pada Gambar 4.4.

```
duarouter -n $SAVEDIR/map.net.xml -t
 ↪   $SAVEDIR/map.passenger.trips.xml -o
 ↪   $SAVEDIR/route.rou.xml --ignore-errors --repair;
```

Gambar 4.4: Perintah untuk Membuat Rute Kendaraan

Selanjutnya dilakukan pembuatan *file* .sumocfg yang akan digunakan sebagai argumen perintah sumo. Gambar 4.5 menunjukkan

isi dari file .sumocfg.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <input>
        <net-file value="map.net.xml"/>
        <route-files value="route.rou.xml"/>
    </input>
    <time>
        <begin value="0"/>
        <end value="360"/>
    </time>
</configuration>
```

Gambar 4.5: Isi dari *file* .sumocfg

File .sumocfg disimpan pada direktori yang sama dengan .net.xml dan .trips.xml. File .sumocfg digunakan untuk mendefinisikan lokasi *file* .net.xml dan .trips.xml serta durasi simulasi. Untuk melihat visualisasi simulasi lalu lintas, *file* sumocfg dapat dibuka melalui sumo-gui. Cuplikan pergerakan kendaraan dapat dilihat pada Gambar 4.6.

Kemudian lakukan simulasi lalu lintas dengan perintah pada Gambar 4.7.

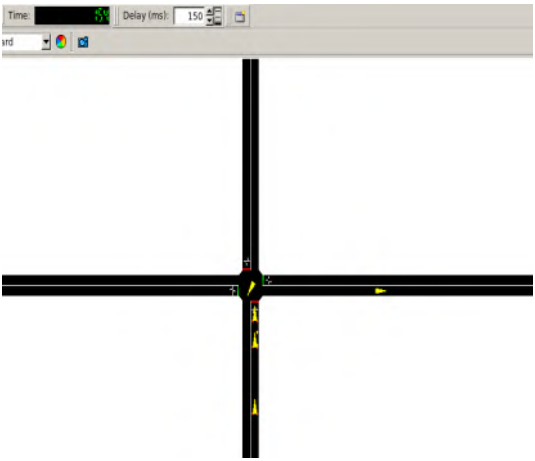
```
sumo -c file.sumocfg --fcd-output simulation-result.xml
```

Gambar 4.7: Perintah untuk Melakukan Simulasi Lalu Lintas

Agar dapat digunakan di NS-2, keluaran dari perintah sumo harus dikonversi ke format yang dapat dipahami oleh NS-2 melalui perintah pada Gambar 4.8.

```
python $SUMO_HOME/tools/traceExporter.py --fcd-input
 ↪  simulation-result.xml --ns2mobility-output mobility.tcl
 ↪  --ns2activity-output activity.tcl;
```

Gambar 4.8: Perintah untuk Mengonversi Keluaran dari sumo



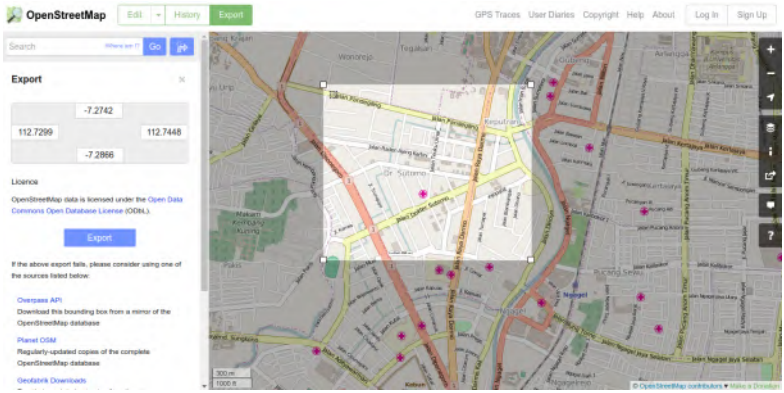
Gambar 4.6: Cuplikan Visualisasi Pergerakan Kendaraan melalui sumo-gui

4.2 Implementasi Skenario Riil

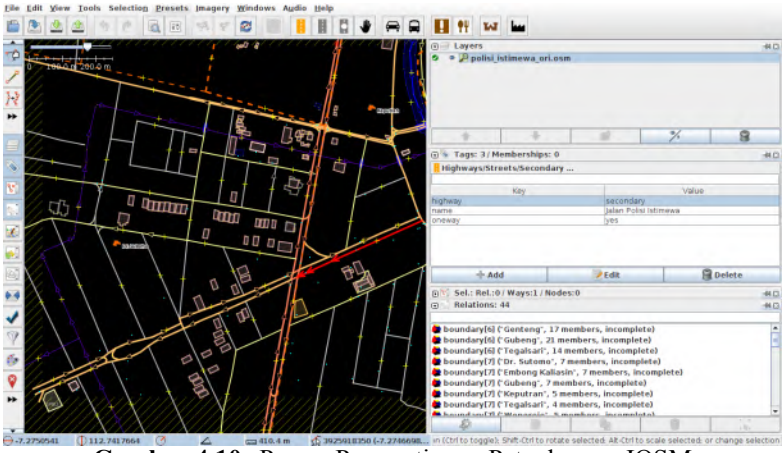
Skenario riil menggunakan bagian peta wilayah kota Surabaya yang diambil dari OpenStreetMap. Peta diambil dengan cara membuat area seleksi wilayah kemudian diekspor dalam bentuk *file* .osm melalui *browser* seperti pada Gambar 4.9.

Peta yang telah diekspor dari OpenStreetMap kemudian disunting melalui program JOSM. Tujuan dari penyuntingan adalah untuk menghapus jalan yang tidak digunakan. Beberapa jalan baru juga ditambahkan agar tidak ada jalan yang buntu dan kepadatan jalan tetap stabil sehingga tidak ada daerah pada peta yang jarang dikunjungi oleh kendaraan. *Screenshot* dari proses penyuntingan dapat dilihat pada Gambar 4.10 dan 4.11.

Setelah proses penyuntingan peta selesai, peta tersebut dikonversi ke dalam format .net.xml menggunakan *tool* netconvert. Perintah konversi dapat dilihat pada Gambar 4.12.



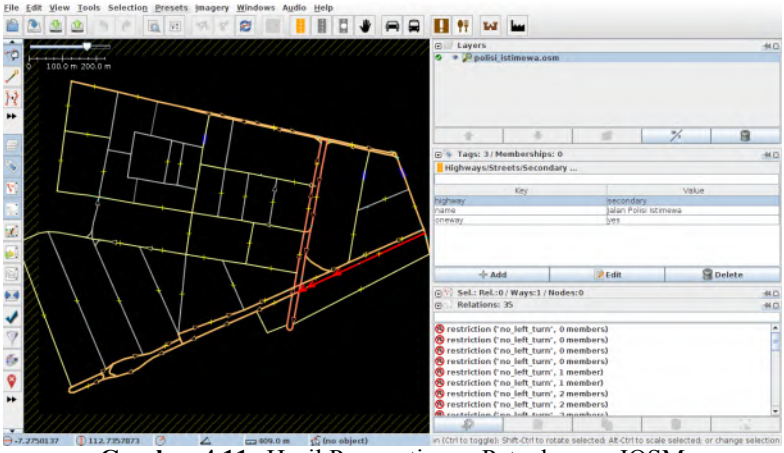
Gambar 4.9: Proses Pengambilan Peta dari OpenStreetMap



Gambar 4.10: Proses Penyuntingan Peta dengan JOSM

```
netconvert --try-join-tls --osm-files map.osm --output-file
 ↪  map.net.xml --remove-edges.isolated --type-files
 ↪  specification.typ.xml
```

Gambar 4.12: Perintah Konversi *file* .osm ke .net.xml



Gambar 4.11: Hasil Penyuntingan Peta dengan JOSM

Setelah peta terbentuk, langkah selanjutnya sama seperti tahapan membuat skenario *grid* mulai dari pembuatan titik asal dan titik tujuan dengan randomTrips.py sampai konversi ke dalam format yang dapat digunakan di NS-2.

4.3 Implementasi Protokol AODV-PNT

Protokol AODV-PNT merupakan modifikasi dari protokol AODV. Perubahan yang dilakukan pada implementasi protokol AODV di NS-2 antara lain adalah sebagai berikut:

- Pengaktifan pesan HELLO
- Struktur paket HELLO
- Penambahan atribut pada kelas AODV
- Struktur *neighbor cache*
- Penanganan paket HELLO
- Penanganan paket RREQ

Kode implementasi dari protokol AODV pada NS-2 versi 2.35 berada pada direktori ns2/aodv. Daftar kode sumber yang akan dimodifikasi pada direktori tersebut adalah sebagai berikut:

- aodv_packet.h untuk mengubah struktur paket HELLO dan RREQ
- aodv_rtable.h untuk mengubah struktur *neighbor cache*
- aodv.h untuk mengaktifkan pesan HELLO dan penambahan atribut pada kelas AODV
- aodv.cc untuk modifikasi penanganan HELLO dan RREQ

Pada bagian ini, penulis akan menjelaskan langkah-langkah dalam mengimplementasikan protokol AODV-PNT degan menggunakan protokol AODV sebagai dasarnya.

4.3.1 Pengaktifan Pesan HELLO

Secara *default*, implementasi AODV di NS-2 menonaktifkan pesan HELLO. Untuk mengaktifkannya, berikan *comment* pada baris 58 (*link layer detection*) dan 66 (*use LL metric*) dalam skrip aodv.h seperti Gambar 4.13.

```
// #define AODV_LINK_LAYER_DETECTION
// #define AODV_USE_LL_METRIC
```

Gambar 4.13: *Comment* pada Baris Untuk Mengaktifkan Pesan HELLO

4.3.2 Modifikasi Struktur Paket HELLO

Pada implementasi *routing protocol* AODV di NS-2, paket HELLO menggunakan *struct* yang sama dengan RREP karena pada dasarnya paket HELLO merupakan sebuah RREP dengan TTL bernilai 1 [5]. Dalam AODV-PNT, terdapat tiga *field* yang perlu ditambahkan ke dalam struktur paket HELLO, yaitu kecepatan, akselerasi, dan koordinat x dan y. Untuk itu dilakukan penambahan atribut rp_speed, rp_accel, rp_x, dan rp_y pada *struct* hdr_aodv_reply di skrip aodv_packet.h seperti pada Gambar 4.14.

```
116 struct hdr_aodv_reply {
117         u_int8_t        rp_type; // Packet Type
118         u_int8_t        reserved[2];
119         u_int8_t        rp_hop_count; // Hop Count
120         nsaddr_t        rp_dst; // Destination IP Address
121         u_int32_t       rp_dst_seqno; // Destination Sequence Number
122         nsaddr_t        rp_src; // Source IP Address
123         double            rp_lifetime; // Lifetime
124
125         double          rp_timestamp; // when corresponding REQ sent;
126                         // used to compute route discovery latency
127
128         // AODV-PNT hello message header
129         double          rp_speed;
130         double          rp_accel;
131         double          rp_x;
132         double          rp_y;
```

Gambar 4.14: Penambahan Atribut Paket HELLO

4.3.3 Modifikasi *Class* AODV

Implementasi AODV-PNT sangat bergantung pada informasi yang didapatkan dari objek MobileNode. Salah satunya adalah position_update_time_. Atribut tersebut berfungsi untuk mencatat pada detik berapa terjadinya perubahan posisi pada suatu *node*. Informasi tersebut digunakan sebagai bagian dari perhitungan akselerasi *node*. Informasi lainnya yang dibutuhkan untuk perhitungan akselerasi adalah kecepatan. Untuk itu dilakukan penambahan atribut lastUpdateTime, lastSpeed, dan lastAccel dalam skrip aodv.h seperti pada Gambar 4.15. Atribut lastAccel akan digunakan untuk menyimpan hasil perhitungan akselerasi.

4.3.4 Modifikasi *Neighbor Cache Entry*

Kelas *neighbor cache* digunakan oleh kelas AODV untuk menyimpan *list* dari *neighbor node*. Informasi dalam *neighbor cache* akan diperbarui setiap kali menerima paket HELLO. Sebelumnya *list* tersebut tidak menyimpan informasi kecepatan, akselerasi, dan

```
282         //AODV Contructor Attributes
283         nsaddr_t        index; // IP Address of this node
284         u_int32_t       seqno; // Sequence Number
285         int             bid;   // Broadcast ID
286
287         // AODV-PNT
288         double          lastUpdateTime; // store last update time
289         double          lastAccel;      // store last acceleration
290         double          lastSpeed;      // store last speed
```

Gambar 4.15: Penambahan Atribut *Class* AODV

koordinat *neighbor node* sehingga ditambahkan atribut nb_speed, nb_accel, nb_x dan nb_y dalam skrip aodv_rtable.h. Kode implementasi dapat dilihat pada Gambar 4.16.

```
46  class AODV_Neighbor {
47          friend class AODV;
48          friend class aodv_rt_entry;
49   public:
50          AODV_Neighbor(u_int32_t a) { nb_addr = a; }
51
52   protected:
53          LIST_ENTRY(AODV_Neighbor) nb_link;
54          nsaddr_t  nb_addr;
55          double    nb_expire; // ALLOWED_HELLO_LOSS * HELLO_INTERVAL
56
57          // AODV-PNT variable to store speed, accel, pos attributes
58          double    nb_speed;
59          double    nb_accel;
60          double    nb_x;
61          double    nb_y;
62  };
```

Gambar 4.16: Penambahan Atribut AODV_Neighbor dalam aodv_rtable.h

4.3.5 Modifikasi Proses Pengiriman HELLO

Proses pengiriman paket HELLO terdiri atas pembuatan paket, pengisian atribut paket, dan penjadwalan *event* pengiriman dalam NS-2. Dalam AODV-PNT, paket HELLO digunakan untuk mengirim informasi kecepatan, akselerasi, dan koordinat dari *node* pengirim. Dengan struktur paket HELLO yang sudah dimodifikasi, pengisian nilai dari atribut-atribut paket HELLO yang baru dapat dilakukan di fungsi sendHello pada skrip aodv.cc.

Informasi kecepatan dan koordinat dari *node* yang akan mengirim paket HELLO didapatkan dari kelas MobileNode. Pertama, cari MobileNode dengan alamat yang sama dengan pengirim. Kemudian, ambil nilai kecepatan dengan fungsi speed(), waktu *update* terakhir dengan fungsi getUpdateTime(), X() untuk mendapatkan koordinat x, dan Y() untuk mendapatkan koordinat Y(). Selanjutnya, hitung nilai akselerasi dengan pembagian selisih kecepatan dan selisih waktu. Kemudian, lakukan *assignment* ke masing-masing atribut paket HELLO. Hasil modifikasi dapat dilihat pada Gambar 4.17.

4.3.6 Modifikasi Proses Penerimaan HELLO

Ketika paket HELLO diterima, *node* akan melakukan pengecekan terhadap *neighbor cache*. Jika alamat pengirim paket HELLO tersebut tidak ada dalam *list*, maka tambahkan alamat tersebut ke dalam *list* dan simpan informasi kecepatan, akselerasi, dan koordinatnya. Jika alamat pengirim paket HELLO sudah ada dalam *list*, maka perbarui informasi kecepatan, akselerasi, koordinat, dan batas waktu penyimpanan informasi *node* tersebut di dalam *cache*. Hasil modifikasi dapat dilihat pada Gambar 4.18.

```
1670  void AODV::sendHello() {
1671    Packet *p = Packet::alloc();
1672    struct hdr_cmn *ch = HDR_CMN(p);
1673    struct hdr_ip *ih = HDR_IP(p);
1674    struct hdr_aodv_reply *rh = HDR_AODV_REPLY(p);
1675    // AODV-PNT
1676    MobileNode *iNode;
1677    iNode = (MobileNode *) (Node::get_node_by_address(index));
1678    double iSpeed = ((MobileNode *) iNode)->speed();
1679    double now = ((MobileNode *) iNode)->getUpdateTime();
1680    rh->rp_x = iNode->X(); // x coordinate
1681    rh->rp_y = iNode->Y(); // y coordinate
1682
1683    if (now - lastUpdateTime == 0) { // if not updated, just in case
1684     rh->rp_accel = lastAccel;
1685    } else {
1686     // acceleration = delta speed / delta time
1687     rh->rp_accel = (iSpeed - lastSpeed) / (now - lastUpdateTime);
1688     lastAccel    = rh->rp_accel;
1689     lastSpeed    = iSpeed;
1690    }
1691    rh->rp_speed = iSpeed; // speed
1692    lastUpdateTime = now; // Update its latest update time
1693
1694    // Fill other HELLO message attributes then send
```

Gambar 4.17: Modifikasi Fungsi sendHello pada aodv.cc

4.3.7 Implementasi Perhitungan TWR, *Future TWR*, dan Pengolahan *Relay Node Set*

Tahap perhitungan TWR hingga pengolahan *relay node set* akan dilakukan oleh *node* yang akan mengirimkan atau meneruskan paket RREQ. Sebelum menghitung TWR, nilai *threshold* W, faktor pengali dari kecepatan, akselerasi, jarak, dan kualitas hubungan ditentukan terlebih dahulu. Setelah nilai faktor pengali ditentukan, dapatkan informasi kecepatan, akselerasi, dan koordinat dari *node* pengirim.

Perhitungan TWR membutuhkan informasi kecepatan, akselerasi, dan koordinat dari *neighbor node* dan *node* tujuan. Kumpulan informasi tersebut bisa didapatkan dari *neighbor cache* dengan cara

```
1721  void AODV::recvHello(Packet *p) {
1722    struct hdr_aodv_reply *rp = HDR_AODV_REPLY(p);
1723    AODV_Neighbor *nb;
1724    nb = nb_lookup(rp->rp_dst); // Find neighbor with that addr
1725
1726    if(nb == 0) { // If that addr not found
1727      nb_insert(rp->rp_dst);
1728      nb = nb_lookup(rp->rp_dst); // Get this neighbor once again
1729      // AODV-PNT Extract speed and accel info from hello message
1730      nb->nb_speed = rp->rp_speed;
1731      nb->nb_accel = rp->rp_accel;
1732      nb->nb_x     = rp->rp_x;
1733      nb->nb_y     = rp->rp_y;
1734    } else {
1735      // AODV-PNT Extract speed and accel info from hello message
1736      nb->nb_speed = rp->rp_speed;
1737      nb->nb_accel = rp->rp_accel;
1738      nb->nb_x     = rp->rp_x;
1739      nb->nb_y     = rp->rp_y;
1740      nb->nb_expire = CURRENT_TIME +
1741                (1.5 * ALLOWED_HELLO_LOSS * HELLO_INTERVAL);
1742    }
1743    Packet::free(p);
1744  }
```

Gambar 4.18: Modifikasi Fungsi recvHello pada aodv.cc

mengarahkan *pointer* ke alamat *neighbor cache* dan lakukan perulangan hingga akhir *list* seperti pada Gambar 4.19.

Untuk *node* tujuan, kecepatan dan akselerasi bernilai nol karena *node* tersebut diam (*stationary node*). Koordinat *node* tujuan akan dituliskan langsung di kode sumber.

Jika informasi *node* pengirim, *neighbor node*, dan *node* tujuan sudah didapatkan, tahap selanjutnya adalah menghitung nilai TWR. Perhitungan TWR dimulai dengan menghitung jarak antara *next-hop node* dengan *node* tujuan. Kemudian menghitung kualitas hubungan antara *node* pengirim dan *next-hop node* dengan formula indeks stabilitas yang dinormalisasi. Selanjutnya kecepatan dan akselerasi dari *next-hop node* masing-masing dikurangkan dengan kecepatan dan akselerasi dari *node* tujuan. Tetapi karena *node* tujuan

```
// Traverse neighbor list
AODV_Neighbor *nb = nbhead.lh_first;

for(; nb; nb = nb->nb_link.le_next) {
    // Get speed, accel, coordinate
    // Calculate TWR
    // Calculate Future TWR
    // Create list of calculated TWR
}
```

Gambar 4.19: Cara Perulangan dalam *Neighbor Cache Entry*

diam, kecepatan dan akselerasi dari *next-hop node* masing-masing dikurangi nol. Kemudian kecepatan, akselerasi, jarak, dan kualitas hubungan dikalikan dengan faktor pengali dan dijumlahkan sehingga menghasilkan nilai TWR. Implementasi dapat dilihat pada Gambar 4.20.

```
// TWR Calculation
// Calculate distance between next-hop and dst
double nb_distance;
nb_distance = sqrt(pow((nb->nb_x - xDst), 2) +
                   pow((nb->nb_y - yDst), 2));
// distance between this node and neighbor
double radius = std::min(sqrt(pow((nb->nb_x - posX), 2)
    + pow((nb->nb_y - posY), 2)),(double) maxTxRange);
// link quality between this node and neighbor
double quality = 1.0 / (1.0 - (radius /
          ((double) maxTxRange + 1.0)));
// Value times factor multiplier
double modSpeed     = fSpeed * nb->nb_speed;
double modAccel     = fAccel * nb->nb_accel;
double modDistance  = fDistance * nb_distance;
double modQuality   = fQuality * quality;
// TWR = f s × |S n − S d | + f a × |A n − A d | + f d × dist + f q × Q
double TWR = modSpeed + modAccel + modDistance + modQuality;
```

Gambar 4.20: Perhitungan TWR

Setelah melakukan perhitungan TWR, dilakukan perhitungan TWR yang akan datang (*future* TWR) dengan memanfaatkan infor-

masi yang sudah didapatkan sebelumnya dan prediksi nilai dengan menggunakan beberapa rumus fisika gerak lurus yang sudah dirancang. Kecepatan *next-hop node* yang akan datang menggunakan persamaan 3.4, koordinat x dan y masing-masing dihitung dengan persamaan 3.5 dan 3.6. Implementasi dapat dilihat pada Gambar 4.21.

```
// Future speed v' = v + a × t
double nb_speedFuture = nb->nb_speed + (nb->nb_accel *
                        timeModifier);
// Future neighbor position; formula: x' = x + v0 t + 0.5at^2
double nb_xFuture = nb->nb_x + (nb->nb_speed * timeModifier)
    + (0.5 * nb->nb_accel * timeModifier * timeModifier);
double nb_yFuture = nb->nb_y + (nb->nb_speed * timeModifier)
    + (0.5 * nb->nb_accel * timeModifier * timeModifier);
// Future this_node position
double iXFuture = posX + (iSpeed * timeModifier)
    + (0.5 * iAccel * timeModifier * timeModifier);
double iYFuture = posY + (iSpeed * timeModifier)
    + (0.5 * iAccel * timeModifier * timeModifier);
// Calc future distance between next-hop and dst
// Calc distance between this node and neighbor using future values
// Calc link quality
// Calc future TWR
```

Gambar 4.21: Perhitungan *Future* TWR

Setelah TWR dan *future* TWR didapatkan, kedua nilai tersebut disimpan ke dalam sebuah vektor yang akan digunakan untuk mengklasifikasikan nilai TWR berdasarkan aturan pemilihan *relay node* pada Tabel 2.1. Kemudian setiap *node* yang masuk ke dalam kriteria *relay* akan dimasukkan ke dalam *array* bertipe data nsaddr_t. *Array* tersebut hanya menyimpan alamat-alamat dari *relay node set*.

Ketiga tahapan ini merupakan bagian dari modifikasi fungsi sendRequest dan recvRequest. Implementasi dari sendRequest dan recvRequest masing-masing dapat dilihat pada lampiran A.2 dan lampiran A.3.

4.3.8 Modifikasi Proses Pengiriman RREQ

Sebelum *node* sumber membuat paket RREQ, dilakukan perhitungan TWR, *future* TWR, dan pengolahan *relay node set* seperti yang sudah dijelaskan sebelumnya. Modifikasi dilakukan di fungsi sendRequest() pada skrip aodv.cc seperti pada Gambar 4.22.

```
Input: rreq
.
.
calculate TWR of next-hop nodes;
calculate future TWR of next-hop nodes;
classify TWR, stability, and future TWR;
list ← relay node set;
.
.
rq ← new RREQ packet;
fill up RREQ fields;
rq_list_length ← list.size();
rq_relay_node_set ← list;
broadcast rq;
```

Gambar 4.22: *Pseudocode* Modifikasi Pemrosesan RREQ yang akan Dikirim

Tahap akhir dari sendRequest adalah membuat paket RREQ dan melakukan *multicast* ke *relay node set*. Namun dalam implementasi AODV, paket RREQ dikirim dengan cara *broadcast*. Untuk itu perlu dilakukan modifikasi dalam menangani *broadcast* paket RREQ agar perilakunya bisa mirip dengan *multicast*.

Agar perilaku *broadcast* RREQ bisa mirip dengan *multicast*, struktur paket RREQ diubah agar dapat menyimpan *array* dan panjang *array* dari *relay node set*. Perubahan dilakukan dengan menambahkan *array* rq_eligible_nodes untuk menyimpan *relay node set* dan nodes_list_len untuk menyimpan panjang *array* seperti pada Gambar 4.23. Kemudian ketika *neighbor node* menerima *broadcast* paket RREQ, dilakukan pengecekan apakah *node* tersebut ada di dalam *array relay node set*. Jika *node* tersebut ada di dalam *array*

relay node set, maka lakukan proses selanjutnya. Jika *node* tersebut tidak ada di dalam *array relay node set*, maka *drop* paket RREQ tersebut. Penjelasan lebih lanjut mengenai modifikasi penerimaan paket RREQ akan dijelaskan pada subbab selanjutnya. Modifikasi dari sendRequest dapat dilihat pada lampiran A.2.

```
struct hdr_aodv_request {
        // RREQ packet fields
        .
        .
        // AODV-PNT addition
        // array of relay node set
        nsaddr_t        *rq_eligible_nodes = NULL;
        u_int32_t       nodes_list_len; // array length
        .
        .
```

Gambar 4.23: Modifikasi Paket RREQ pada aodv_packet.h

4.3.9 Modifikasi Proses Penerimaan RREQ

Seperti yang sudah dijelaskan pada subbab sebelumnya, proses penerimaan RREQ juga perlu dimodifikasi agar *broadcast* paket RREQ bisa memiliki perilaku yang mirip dengan *multicast*.

Ketika paket RREQ diterima oleh suatu *node*, normalnya akan dilakukan pengecekan kondisi seperti apakah *node* tersebut *node* sumber, apakah *node* sudah pernah menerima paket RREQ tersebut, apakah *node* tersebut adalah *node* tujuan, dan lain-lain. Agar perilaku dari penerimaan paket RREQ yang di-*broadcast* mirip dengan *multicast*, maka perlu ditambahkan kondisi baru di mana *node* yang tidak seharusnya menerima paket RREQ harus melakukan *drop* paket. Apabila *node* tersebut berhak menerima paket RREQ dan bukan merupakan *node* tujuan, maka lakukan proses perhitungan TWR hingga pembuatan *relay node set* yang baru dan teruskan paket tersebut.

Pseudocode perubahan yang dilakukan di fungsi recvRequest

```
Input: rreq
.
.
index ← address of this node;
if index is not in rreq_relay_node_set then
    drop packet;
else
    if rreq_dst is not index then
        calculate TWR of next-hop nodes;
        calculate future TWR of next-hop nodes;
        classify TWR, stability, and future TWR;
        new_list ← new relay node set;
        rreq_relay_node_set ← new_list;
    end
end
.
.
```

Gambar 4.24: *Pseudocode* Modifikasi Pemrosesan RREQ yang Diterima

pada skrip aodv.cc dapat dilihat pada Gambar 4.24. Keseluruhan modifikasi fungsi dapat dilihat pada lampiran A.3.

4.4 Implementasi Metrik Analisis

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

4.4.1 Implementasi *Packet Delivery Ratio*

Packet delivery ratio didapatkan dengan cara menghitung setiap baris terjadinya *event* pengiriman dan penerimaan paket data yang dikirim melalui agen pada *trace file*. Pada lampiran A.4 ditunjukkan cara menyaring data yang dibutuhkan untuk perhitungan

PDR. Pada skrip tersebut dilakukan penyaringan pada setiap baris yang mengandung *string* AGT karena *event* tersebut berhubungan dengan paket data. Paket yang dikirim dan diterima dibedakan dari kolom pertama pada baris yang telah disaring. Setelah pembacaan setiap baris selesai dilakukan, selanjutnya dilakukan perhitungan PDR dengan persamaan 3.7.

Contoh perintah untuk memanggil skrip AWK untuk menganalisis *trace file* dan contoh keluarannya dapat dilihat pada Gambar 4.25 dan 4.26.

```
awk -f pdr.awk tracefile.tr
```

Gambar 4.25: Perintah Untuk Menjalankan Skrip AWK Perhitungan PDR

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

Gambar 4.26: Keluaran dari Skrip pdr.awk

4.4.2 Implementasi *Average End-to-End Delay*

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

Delay dari setiap paket dihitung dengan cara mengurangi waktu penerimaan dengan waktu pengiriman berdasarkan ID paket. Hasil pengurangan waktu dari masing-masing paket dijumlahkan dan dibagi dengan jumlah paket CBR yang ID-nya terlibat dalam perhitungan pengurangan waktu.

Kode implementasi dari perhitungan *average end-to-end delay* dapat dilihat pada lampiran A.5.

Contoh perintah untuk memanggil skrip AWK untuk menganalisis *trace file* dan contoh keluarannya dapat dilihat pada Gambar 4.27 dan 4.28.

```
awk -f delay.awk tracefile.tr
```

Gambar 4.27: Perintah Untuk Menjalankan Skrip AWK Perhitungan *Average End-to-End Delay*

```
Average Delay: 0.0874
```

Gambar 4.28: Keluaran dari Skrip pdr.awk

4.4.3 Implementasi *Routing Overhead*

Routing overhead didapatkan dengan cara menyaring setiap baris yang mengandung *string* REQUEST, REPLY, dan ERROR. Setiap ditemukannya *string* tersebut, dilakukan *increment* untuk menghitung jumlah paket *routing* yang tersebar di jaringan. Kode implementasi dari perhitungan *routing overhead* dapat dilihat pada lampiran A.6.

Contoh perintah untuk memanggil skrip AWK untuk menganalisis *trace file* dan contoh keluarannya dapat dilihat pada Gambar 4.29 dan 4.30.

```
awk -f ro.awk tracefile.tr
```

Gambar 4.29: Perintah Untuk Menjalankan Skrip AWK Perhitungan *Routing Overhead*

```
Overhead: 2668
```

Gambar 4.30: Keluaran dari Skrip ro.awk

4.5 Implementasi Simulasi pada NS-2

Untuk melakukan simulasi VANET pada NS-2, dibutuhkan sebuah *file* OTcl yang berisi deskripsi dari lingkungan simulasi. *File* tersebut berisikan pengaturan untuk setiap *node* dan beberapa *event* yang perlu diatur agar berjalan pada waktu tertentu. Contoh potongan skrip pengaturan *node* dapat dilihat pada Gambar 4.31.

```
1  set val(chan) Channel/WirelessChannel ;# channel type
2  set val(prop) Propagation/TwoRayGround ;# radio-propagation model
3  set val(netif) Phy/WirelessPhy ;# network interface type
4  set val(mac) Mac/802_11 ;# MAC type
5  set val(ifq) Queue/DropTail/PriQueue ;# interface queue type
6  set val(ll) LL ;# link layer type
7  set val(ant) Antenna/OmniAntenna ;# antenna model
8  set val(ifqlen) 50 ;# max packet in ifq
9  set val(rp) AODV ;# routing protocol
10 set val(nn) 200
11 set val(cbrsize) 512     ;# 512 Bytes
12 set val(cbrrate) 2KB
13 set val(cbrinterval) 1   ;# 1 packet per second
14 set val(stop) 460.0
15
16 set ns_ [new Simulator]
17 set topo       [new Topography]
18 $topo load_flatgrid 1075.51 1075.51
19
20 $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
    ↪  [new $val(chan)] -agentTrace ON -routerTrace ON -macTrace OFF
    ↪  -movementTrace OFF -topoInstance $topo
```

Gambar 4.31: Potongan Skrip Pengaturan *Node*

Penjelasan dari pengaturan *node* dapat dilihat pada Tabel 4.1. Pengaturan lainnya yang dilakukan pada *file* tersebut antara lain lo-

kasi penyimpanan *trace file*, lokasi *file* mobilitas *node*, konfigurasi *node* sumber dan *node* tujuan, dan konfigurasi *event* pengiriman paket data. Kode implementasi skenario yang digunakan pada Tugas Akhir ini dapat dilihat pada lampiran A.1.

Tabel 4.1: Penjelasan dari Parameter Pengaturan *Node*

Parameter	**Value**	**Penjelasan**
llType	LL	Menggunakan *link layer* standar
mactType	Mac/802_11	Menggunakan tipe MAC 802.11 karena komunikasi data bersifat wireless
ifqType	Queue/Drop Tail/PriQueue	Menggunakan *priority queue* sebagai antrian paket dan paket yang dihapus saat antrian penuh adalah paket yang paing baru
ifqLen	50	Jumlah maksimal paket pada antrian
antType	Antenna /Omni Antenna	Jenis antena yang digunakan adalah *omni antenna*
propType	Propagati on/TwoRay Ground	Tipe propagasi sinyal wireless adalah *two-ray ground*
phyType	Phy/Wireless Phy	Komunikasi menggunakan media nirkabel
topoInstance	$topo	Topologi yang digunakan daat menjalankan skenario
agentTrace	ON	Aktifkan pencatatan aktifitas dari agen *routing protocol*

routerTrace	ON	Aktifkan pencatatan pada aktifitas *routing protocol*
macTrace	OFF	Matikan pencatatan MAC *layer* pada *trace file*
movementTrace	OFF	Matikan pencatatan pergerakan *node*
channel	Channel /Wireless Channel	Kanal komunikasi yang digunakan

Skenario simulasi dijalankan dengan perintah pada Gambar 4.32. Setelah simulasi selesai akan ada keluaran berupa *trace file* hasil simulasi yang akan digunakan untuk analisis. Isi dari *tracefile* adalah catatan seluruh *event* yang dari setiap paket yang tersebar di dalam lingkungan simulasi.

```
ns scenario.tcl
```

Gambar 4.32: Perintah Untuk Menjalankan Skenario NS-2

BAB 5

UJI COBA DAN EVALUASI

Pada bab ini akan dibahas mengenai uji coba dan evaluasi hasil simulasi dari skenario NS-2 yang telah dibuat.

5.1 Lingkungan Uji Coba

Spesifikasi perangkat keras yang digunakan pada Tugas Akhir ini dapat dilihat pada Tabel 5.1.

Tabel 5.1: Spesifikasi Perangkat Keras yang Digunakan

Komponen	**Spesifikasi**
CPU	Intel® Core™ i7-4700MQ @ 2.40GHz × 8
Sistem Operasi	Linux Ubuntu 14.04 LTS 64-bit
Memori	8GB PC3-12800 DDR3L SDRAM 1600 MHz
Penyimpanan	1 TB

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

- SUMO versi 0.25.0 untuk pembuatan skenario mobilitas VANET.
- JOSM versi 10301 untuk penyuntingan peta OpenStreetMap.
- ns2.35 untuk simulasi skenario VANET.

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

Pengujian dilakukan dengan menjalankan skenario melalui NS-2. Hasil dari skenario yang sudah dijalankan berupa *trace file* yang akan dianalisis dengan skrip AWK.

Tabel 5.2: Parameter Simulasi NS-2

No.	Parameter	Spesifikasi
1	*Network simulator*	NS-2.35
2	*Routing protocol*	AODV-PNT dan AODV
3	Waktu simulasi	360 detik
4	Area simulasi	1000 m x 1000 m (*grid*) 1200 m x 1000 m (riil)
5	Jumlah kendaraan	50, 100, 150, 200, 400
6	Radius transmisi	250 m
7	Kecepatan maksimum (*grid*)	15 m/s
8	Agen	*Constant Bit Rate* (CBR)
9	*Source / Destination*	*Stationary*
10	Ukuran paket	512 Bytes
11	*Packet Rate*	2 kB/s
12	*Packet Interval*	1 paket/detik
13	Protokol MAC	IEEE 802.11p
14	Model Propagasi	*Two-ray ground*
15	Parameter AODV-PNT	f_s = 15, f_a = 10, f_d = 20, f_q = 50, W = 100

5.2 Hasil Uji Coba

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

5.2.1 Hasil Uji Coba Skenario *Grid*

Uji coba skenario *grid* dilakukan sebanyak 20 kali dengan skenario mobilitas *random* pada peta *grid* dengan luas area 1000 m x 1000 m dan jumlah *node* sebanyak 50, 100, 150, 200, dan 400 di mana 400 *node* merupakan jumlah kendaraan maksimum yang dapat digunakan dalam skenario ini. Kecepatan maksimum dari setiap *node* adalah 15 m/s.

Hasil analisis dengan metrik *packet delivery ratio* (PDR), *average end-to-end delay*, dan *routing overhead* (RO) dapat dilihat pada Tabel 5.3, 5.4, dan 5.5.

Tabel 5.3: Hasil Perhitungan Rata-rata PDR Skenario *Grid*

Jumlah *Node*	AODV-PNT	AODV	Perbedaan
50	0.678	0.539	0.139
100	0.746	0.527	0.219
150	0.768	0.537	0.231
200	0.761	0.541	0.22
400	0.744	0.626	0.118

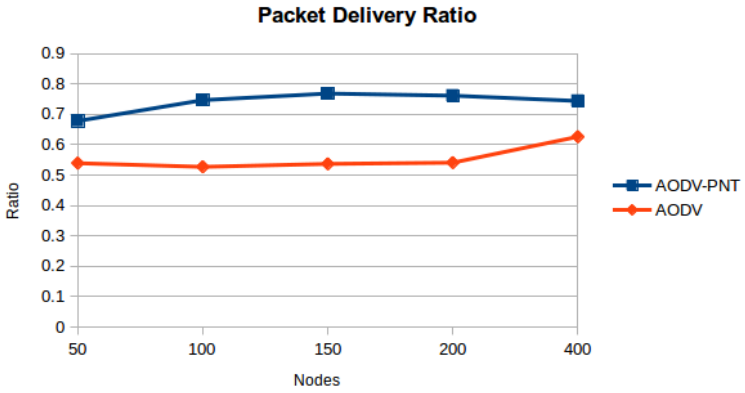
Tabel 5.4: Hasil Perhitungan Rata-rata *Delay* Skenario *Grid*

Jumlah *Node*	AODV-PNT	AODV	Perbedaan
50	0.760	0.124	0.636
100	0.163	0.155	0.008
150	0.106	0.226	0.12
200	0.158	0.373	0.215
400	2.055	3.076	1.021

Tabel 5.5: Hasil Perhitungan Rata-rata RO Skenario *Grid*

Jumlah *Node*	AODV-PNT	AODV	Perbedaan
50	422	964	542
100	783	2092	1309
150	1211	3330	2119
200	1746	4827	3081
400	5022	8792	3770

Dari data di atas, dibuat grafik yang merepresentasikan hasil perhitungan PDR, *delay*, dan RO yang ditunjukkan pada Gambar 5.1, 5.2, dan 5.3.



Gambar 5.1: Grafik PDR Skenario *Grid*

Berdasarkan data tersebut, dapat dilihat bahwa AODV-PNT memiliki rata-rata PDR yang lebih tinggi dibandingkan dengan AODV. Pada jumlah *node* 50, AODV-PNT memiliki PDR 13,9% lebih baik dibandingkan dengan AODV. Pada jumlah *node* 100, 150 dan 200, AODV-PNT memiliki PDR 21% hingga 23% lebih baik dibandingkan dengan AODV. Pada jumlah *node* 400, AODV-PNT memiliki PDR 11,8% lebih baik dibandingkan dengan AODV. Banyaknya jumlah *node* kurang berpengaruh terhadap PDR dari AODV sedangkan PDR dari AODV-PNT cenderung meningkat. Hal ini disebabkan perbedaan jumlah paket RREQ yang tersebar di jaringan. Semakin banyak paket RREQ yang tersebar, maka semakin banyak juga kemungkinan rute yang akan terjadi. Dari hasil pelacakan rute dengan skrip pada lampiran A.7, AODV sering mendapatkan rute yang sering mengalami *link failure* karena *node* penerima sudah tidak berada di area komunikasi. Pada AODV-PNT, rute yang didapatkan cenderung stabil. Pada tabel 5.6 dan 5.7 dapat dilihat perbedaan lama bertahannya rute antara AODV (PDR 0,5) dan AODV-PNT (PDR 0,7) pada salah satu skenario uji coba.

Tabel 5.6: Durasi Bertahannya Rute pada AODV

No.	Durasi (detik)	Jumlah Paket Diterima
1	0.09880	1
2	3.03561	4
3	17.03669	17
4	2.03143	3
5	3.05296	4
6	11.05003	11
7	0.08853	1
8	1.03561	2
9	1.03647	2
10	1.63125	2
11	18.03591	18
12	3.02521	4
13	12.02941	12

Tabel 5.7: Durasi Bertahannya Rute pada AODV-PNT

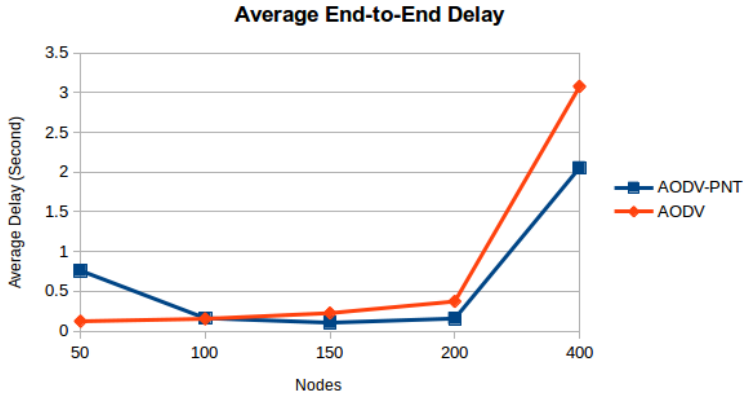
No.	Durasi (detik)	Jumlah Paket Diterima
1	31.03613	31
2	5.03821	5
3	18.05626	18
4	4.04317	4
5	4.04855	4
6	19.03211	19
7	27.03792	27

Dari tabel 5.6 dan 5.7 dapat dilihat rute yang terbentuk pada AODV durasi bertahannya cenderung sebentar sedangkan rute yang terbentuk pada AODV-PNT durasi bertahannya cenderung lebih lama. AODV sering mengalami putus *link* dengan *node* penerima sehing-

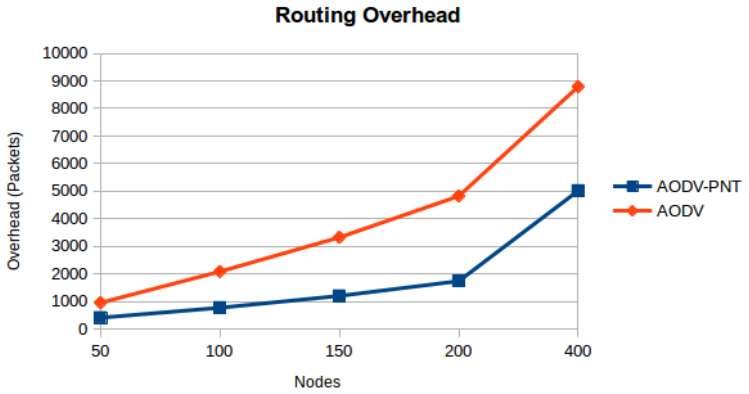
ga rute harus di-*request* lagi. Rute yang mengalami *packet drop* karena putusnya *link* pada AODV sebanyak 56 dan AODV-PNT sebanyak 28.

Pada kepadatan 50 *node*, *average delay* dari AODV-PNT lebih tinggi 0.637 detik dibandingkan dengan AODV. Pada kepadatan kendaraan yang rendah, AODV-PNT memerlukan waktu yang lama untuk melakukan proses *route discovery*. Pada beberapa skenario dengan kepadatan 50 *node*, AODV-PNT mengalami *delay* pada beberapa paket dengan durasi 5 hingga 11 detik sedangkan paket lainnya memiliki *delay* beragam 0.0X hingga 0.5 detik. Pada kepadatan 400 *node*, AODV dan AODV-PNT mengalami peningkatan *delay* yang disebabkan oleh *buffer* paket yang terlalu banyak.

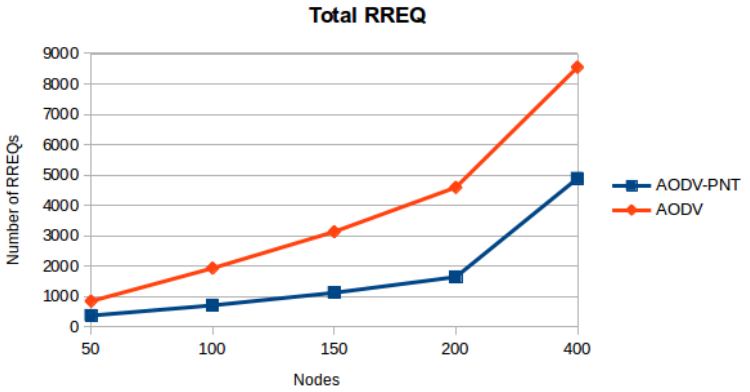
Routing overhead pada AODV-PNT lebih sedikit daripada AODV. Hal ini dikarenakan AODV-PNT hanya memilih *next-hop node* yang stabil dan melakukan *multicast* RREQ terhadap kumpulan *next-hop node* (*relay node set*) sedangkan AODV melakukan *broadcast* RREQ sehingga lebih banyak jumlah paket *routing* yang beredar di jaringan. Perbedaan jumlah RREQ dapat dilihat pada grafik 5.4.



Gambar 5.2: Grafik *Average End-to-End Delay* Skenario *Grid*



Gambar 5.3: Grafik RO Skenario *Grid*



Gambar 5.4: Grafik Jumlah RREQ Skenario *Grid*

5.2.2 Hasil Uji Coba Skenario Riil

Uji coba skenario riil dilakukan sebanyak 20 kali dengan skenario mobilitas *random* pada peta kota Surabaya dengan luas area 1200 m x 1000 m dan jumlah *node* sebanyak 50, 100, 150, 200,

dan 400 di mana 400 *node* merupakan jumlah kendaraan maksimum yang dapat digunakan dalam skenario ini. Kecepatan *node* ditentukan oleh *simulator* berdasarkan batasan aturan jalan.

Hasil analisis dengan metrik *packet delivery ratio* (PDR), *average end-to-end delay*, dan *routing overhead* (RO) dapat dilihat pada Tabel 5.8, 5.9, dan 5.10.

Tabel 5.8: Hasil Perhitungan Rata-rata PDR Skenario Riil

Jumlah *Node*	**AODV-PNT**	**AODV**	**Perbedaan**
50	0.498	0.413	0.085
100	0.550	0.450	0.1
150	0.622	0.548	0.074
200	0.659	0.572	0.087
400	0.781	0.669	0.112

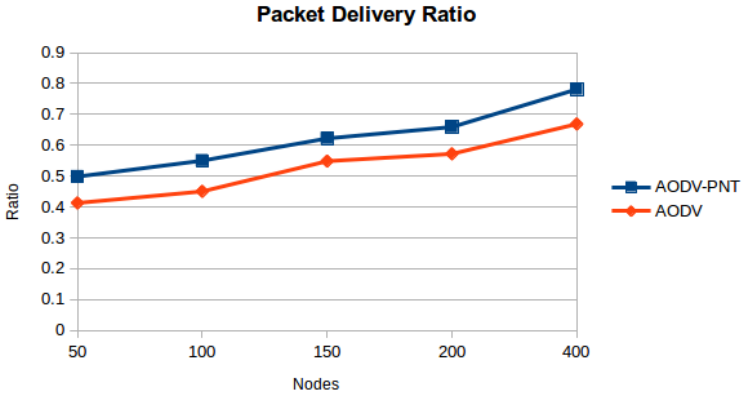
Tabel 5.9: Hasil Perhitungan Rata-rata *Delay* Skenario Riil

Jumlah *Node*	**AODV-PNT**	**AODV**	**Perbedaan**
50	0.657	0.124	0.533
100	0.690	0.216	0.474
150	0.476	0.396	0.08
200	0.295	0.972	0.677
400	1.685	2.413	0.728

Tabel 5.10: Hasil Perhitungan Rata-rata RO Skenario Riil

Jumlah *Node*	**AODV-PNT**	**AODV**	**Perbedaan**
50	798	1270	472
100	1734	2676	942
150	2270	3777	1507
200	3029	4743	1714
400	4125	6800	2675

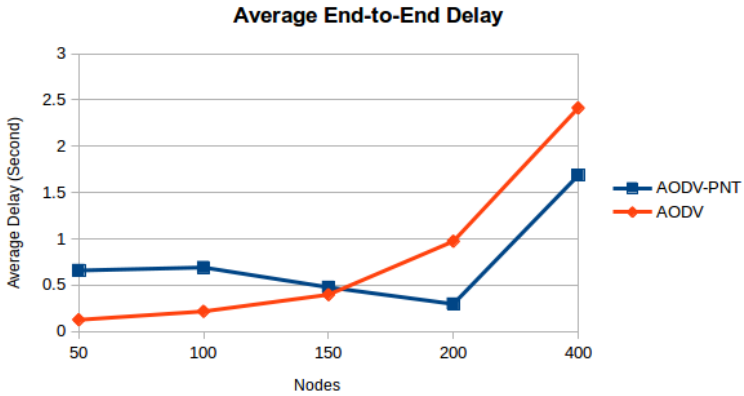
Dari data di atas, dibuat grafik yang merepresentasikan hasil perhitungan PDR, *delay*, dan RO yang ditunjukkan pada Gambar 5.5, 5.6, dan 5.7.



Gambar 5.5: Grafik PDR Skenario Riil

Berdasarkan data tersebut, dapat dilihat bahwa AODV-PNT memiliki rata-rata PDR yang lebih tinggi dibandingkan dengan AODV. Namun, perbedaan PDR antara AODV-PNT dan AODV kurang signifikan. Pada jumlah *node* 50, AODV-PNT memiliki PDR 8,6% lebih baik dibandingkan dengan AODV. Pada jumlah *node* 100, AODV-PNT memiliki PDR 10% lebih baik dibandingkan dengan AODV. Pada jumlah *node* 150, AODV-PNT memiliki PDR 7,4% lebih baik dibandingkan dengan AODV. Pada jumlah *node* 200, AODV-PNT memiliki PDR 8,7% lebih baik dibandingkan dengan AODV. Pada jumlah *node* 400, AODV-PNT memliki PDR 11,2% lebih baik dibandingkan dengan AODV.

Pada kepadatan 50 *node*, *average delay* dari AODV-PNT lebih tinggi 0.533 detik dibandingkan dengan AODV. Pada kepadatan kendaraan yang rendah, terdapat kasus di mana proses *route discovery* terlalu lama. Pada beberapa skenario dengan kepadatan 50 *node*, AODV-PNT mengalami *delay* pada beberapa paket dengan



Gambar 5.6: Grafik *Average End-to-End Delay* Skenario Riil

durasi hingga 14 detik sedangkan paket lainnya memiliki *delay* beragam antara 0.0X hingga 1 detik.

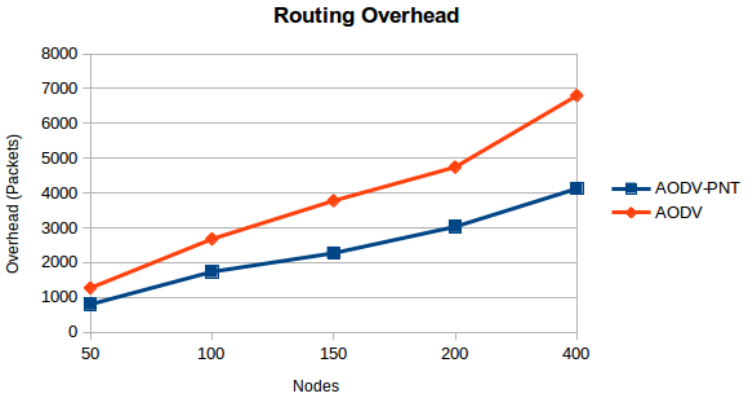
Tren *delay* yang meningkat pada AODV dalam kepadatan kendaraan 200 *node* dikarenakan AODV terlalu lama dalam melakukan proses *route discovery* karena terjadi kemacetan jalan di setiap perempatan yang terdapat lampu lalu lintas. Karena AODV melakukan *broadcast*, *buffer* paket dari setiap *node* di sekitar area perempatan menjadi sangat panjang. Pada AODV-PNT, peningkatan kepadatan kendaraan mengurangi nilai rata-rata *delay* karena paket *routing* yang dibuat lebih sedikit sehingga *buffer* paket tidak terlalu banyak namun pada kepadatan maksimum (400 *node*) AODV-PNT mengalami peningkatan *delay* dikarenakan *buffer* paket yang terlalu banyak. Tabel 5.11 menunjukkan proses terjadinya salah satu paket dengan *delay* yang lama, yaitu 11 detik.

Tabel 5.11: *Timeline* dari Paket ID 1 dengan Delay 11 Detik

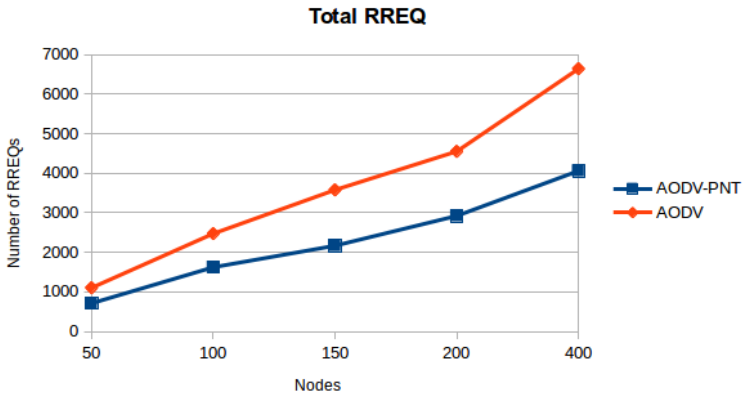
Waktu	***Event***
200.0000	Sumber *broadcast* RREQ
200.0020	*Node* 138 menerima RREQ dari Sumber

200.0063	*Node* 138 *broadcast* RREQ
200.0123	*Node* 133 menerima RREQ dari 138
200.0154	Tujuan menerima RREQ dari 133
200.0154	Tujuan mengirim RREP ke 133
202.0109	Tujuan menerima RREQ dari 138
202.0109	Tujuan mengirim RREP ke 138
206.1083	138 menerima RREP dari Tujuan
211.5014	138 mengirim RREP ke Sumber
211.5911	Sumber menerima RREP dari 138
211.6011	Sumber mengirim CBR ke 138

Routing overhead pada AODV-PNT lebih sedikit daripada AODV. Hal ini dikarenakan AODV-PNT hanya memilih *next-hop node* yang stabil dan melakukan *multicast* RREQ terhadap kumpulan *next-hop node* tersebut (*relay node set*) sedangkan AODV melakukan *broadcast* RREQ sehingga lebih banyak jumlah paket *routing* yang beredar di jaringan. Perbedaan jumlah RREQ dapat dilihat pada grafik 5.8.



Gambar 5.7: Grafik RO Skenario Riil



Gambar 5.8: Grafik Jumlah RREQ Skenario Riil

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/WirelessPhy ;# network interface type
set val(mac) Mac/802_11 ;# 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) 50 ;# max packet in ifq
set val(rp) AODV ;# routing protocol
set val(nn) 200
set val(cbrsize) 512         ;# 512 Bytes
set val(cbrrate) 2KB         ;# 2kB = 2 kiloBytes
set val(cbrinterval) 1     ;# 1 packet per second
set val(stop) 460
set val(mobilityfile) "mobility.tcl"
set val(activityfile) "activity.tcl"

# Initialize ns
set ns_ [new Simulator]
set tracefd [open trace.tr w]
$ns_ trace-all $tracefd

# Set up topography object
set topo       [new Topography]
$topo load_flatgrid 1100 1100

create-god [expr $val(nn) + 2]
set channel_ [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
 ↪   $channel_ -agentTrace ON -routerTrace ON -macTrace OFF
 ↪   -movementTrace OFF -topoInstance $topo

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

puts "Loading mobility file..."
set where [file dirname [info script]]
source [file join $where $val(mobilityfile)]
source [file join $where $val(activityfile)]
```

```
40
41  # Provide initial (X,Y, for now Z=0) co-ordinates for node_(100) and
        ↪  node_(101)
42  # Static nodes
43  set sender [$ns_ node]
44  $sender set X_ 149.92
45  $sender set Y_ 900.52
46  $sender set Z_ 0.0
47
48  set receiver [$ns_ node]
49  $receiver set X_ 901.01
50  $receiver set Y_ 747.36
51  $receiver set Z_ 0.0
52
53  #Setup a UDP connection
54  set udp [new Agent/UDP]
55  $ns_ attach-agent $sender $udp
56  set null [new Agent/Null]
57  $ns_ attach-agent $receiver $null
58  $ns_ connect $udp $null
59
60  #Setup a CBR over UDP connection
61  set cbr [new Application/Traffic/CBR]
62  $cbr attach-agent $udp
63  $cbr set type_ CBR
64  $cbr set packet_size_ $val(cbrsize)
65  $cbr set rate_ $val(cbrrate)
66  $cbr set interval_ $val(cbrinterval)
67
68
69  #Schedule events for the CBR and FTP agents
70  $ns_ at 250.0 "$cbr start"
71  $ns_ at 400.0 "$cbr stop"
72
73  # Tell nodes when the simulation ends
74  for {set i 0} {$i < $val(nn) } {incr i} {
75      $ns_ at [expr $val(stop) +1.0] "$node_($i) reset";
76  }
77
78  # Tell static node when simulation ends
79  $ns_ at [expr $val(stop) +1.0] "$sender reset"
80  $ns_ at [expr $val(stop) +1.0] "$receiver reset"
81
82  # What to do when scenario finished
83  $ns_ at [expr $val(stop) +1.0] "finish"
84  $ns_ at [expr $val(stop) +2.0] "puts \"Exiting NS2...\"; $ns_ halt"
85
```

```
86  proc finish {} {
87    global ns_ tracefd
88    $ns_ flush-trace
89    close $tracefd
90  }
91
92  puts "Starting simulation..."
93  $ns_ run
```

A.2 Kode Implementasi sendRequest AODV-PNT

```
1670 void AODV::sendHello() {
1671   Packet *p = Packet::alloc();
1672   struct hdr_cmn *ch = HDR_CMN(p);
1673   struct hdr_ip *ih = HDR_IP(p);
1674   struct hdr_aodv_reply *rh = HDR_AODV_REPLY(p);
1675   // AODV-PNT
1676   MobileNode *iNode;
1677   iNode = (MobileNode *) (Node::get_node_by_address(index));
1678   double iSpeed = ((MobileNode *) iNode)->speed();
1679   double now = ((MobileNode *) iNode)->getUpdateTime();
1680   rh->rp_x = iNode->X(); // x coordinate
1681   rh->rp_y = iNode->Y(); // y coordinate
1682
1683   if (now - lastUpdateTime == 0) { // if not updated, just in case
1684    rh->rp_accel = lastAccel;
1685   } else {
1686    // acceleration = delta speed / delta time
1687    rh->rp_accel = (iSpeed - lastSpeed) / (now - lastUpdateTime);
1688    lastAccel    = rh->rp_accel;
1689    lastSpeed    = iSpeed;
1690   }
1691   rh->rp_speed = iSpeed; // speed
1692   lastUpdateTime = now; // Update its latest update time
1693
1694   // Fill other HELLO message attributes then send
1695
1696  rh->rp_type = AODVTYPE_HELLO;
1697  //rh->rp_flags = 0x00;
1698  rh->rp_hop_count = 1;
1699  rh->rp_dst = index;
1700  rh->rp_dst_seqno = seqno;
1701  rh->rp_lifetime = (1 + ALLOWED_HELLO_LOSS) * HELLO_INTERVAL;
1702
1703  // ch->uid() = 0;
1704  ch->ptype() = PT_AODV;
1705  ch->size() = IP_HDR_LEN + rh->size();
1706  ch->iface() = -2;
1707  ch->error() = 0;
1708  ch->addr_type() = NS_AF_NONE;
1709  ch->prev_hop_ = index;          // AODV hack
1710
1711  ih->saddr() = index;
1712  ih->daddr() = IP_BROADCAST;
1713  ih->sport() = RT_PORT;
```

```
1714     ih->dport() = RT_PORT;
1715     ih->ttl_ = 1;
1716
1717     Scheduler::instance().schedule(target_, p, 0.0);
1718   }
```

A.3 Kode Implementasi recvRequest AODV-PNT

```
674 void
675 AODV::recvRequest(Packet *p) {
676 // struct hdr_cmn *ch = HDR_CMN(p);
677 struct hdr_ip *ih = HDR_IP(p);
678 struct hdr_aodv_request *rq = HDR_AODV_REQUEST(p);
679 aodv_rt_entry *rt;
680 bool isEligible = true;
681
682   /*
683    * Drop if:
684    *      - I'm the source
685    *      - I recently heard this request.
686    *      - I'm not in the list of eligible nodes
687    */
688
689   if(rq->rq_src == index) {
690 #ifdef DEBUG
691     fprintf(stderr, "%s: got my own REQUEST\n", __FUNCTION__);
692 #endif // DEBUG
693
694     Packet::free(p);
695     return;
696   }
697
698
699  if (id_lookup(rq->rq_src, rq->rq_bcast_id)) {
700
701 #ifdef DEBUG
702    fprintf(stderr, "%s: discarding request\n", __FUNCTION__);
703 #endif // DEBUG
704
705    Packet::free(p);
706    return;
707  }
708
709  // If list is null, then drop packet
710  if(rq->rq_eligible_nodes == NULL) {
711
712    Packet::free(p);
713    return;
714  }
715  else if (rq->rq_dst != index) {
716    isEligible = false;
717    for (u_int32_t i = 0; i < rq->nodes_list_len; ++i) {
```

```
718         // I'm on the list, so I'm eligible for this packet
719         if (rq->rq_eligible_nodes[i] == index) {
720           isEligible = true;
721           break;
722         }
723       }
724     }
725
726   // If I'm not in eligible list, then drop packet
727   if (!isEligible) {
728     Packet::free(p);
729     return;
730   }
731   // If I'm eligible, then re-compute to create new eligible list
732   else if (rq->rq_dst != index) {
733     /*
734      *  AODV-PNT re-compute TWR
735      */
736
737     // TWR list
738     std::vector<TWRContainer> listTWR;
739     std::vector<nsaddr_t> eligibleAddrList;
740     nsaddr_t *eligibleNodes;
741
742     // Get current position, speed, time
743     MobileNode *iNode;
744     iNode = (MobileNode *) (Node::get_node_by_address(index));
745     double iSpeed = ((MobileNode *) iNode)->speed();
746     double now = ((MobileNode *) iNode)->getUpdateTime();
747
748     double iAccel;
749     if (now - lastUpdateTime == 0) {
750       iAccel = lastAccel;
751     }
752     else {
753       iAccel = (iSpeed - lastSpeed) / (now - lastUpdateTime);
754       lastAccel = iAccel;
755       lastSpeed = iSpeed;
756     }
757
758     double posX = iNode->X();
759     double posY = iNode->Y();
760
761     u_int32_t fSpeed = F_SPEED;
762     u_int32_t fAccel = F_ACCEL;
763     u_int32_t fDistance = F_DISTANCE;
764     u_int32_t fQuality = F_QUALITY;
```

```
765   u_int32_t timeModifier = TIME_MOD;
766   u_int32_t maxTxRange = TX_RANGE; // ns2 default range (based on Pt_)
767   u_int32_t thresholdW = THRES_W;
768
769   // Dst fixed position (STA dst). real scenario: we can find exact
      ↪  loc via GPS
770   double xDst = DST_X;
771   double yDst = DST_Y;
772
773
774   // Traverse neighbor list
775   AODV_Neighbor *nb = nbhead.lh_first;
776
777   for(; nb; nb = nb->nb_link.le_next) {
778     // TWR Calculation
779
780     // Calculate distance between next-hop and dst
781     double nb_distance;
782     nb_distance = sqrt(pow((nb->nb_x - xDst), 2) + pow((nb->nb_y -
      ↪  yDst), 2));
783
784     // radius between this node and neighbor
785     // minimum radius -> min(√(i x - j x )^2 + (i y - j y )^2 ; r)
786     double radius = std::min(
787       sqrt(pow((nb->nb_x - posX), 2) + pow((nb->nb_y - posY), 2)),
      ↪  (double) maxTxRange);
788
789     double quality = 1.0 / (1.0 - (radius / ((double) maxTxRange +
      ↪  1.0)));
790
791     double modSpeed     = fSpeed * nb->nb_speed;
792     double modAccel     = fAccel * nb->nb_accel;
793     double modDistance  = fDistance * nb_distance;
794     double modQuality   = fQuality * quality;
795
796     // TWR = f s × |S n - S d | + f a × |A n - A d | + f d × |θ n - θ
      ↪  d | + f q × Q
797     double TWR = modSpeed + modAccel + modDistance + modQuality;
798
799     // Calculate future TWR for next [timeModifier] seconds
800     // Future speed v' = v + a × t
801     double nb_speedFuture = nb->nb_speed + (nb->nb_accel *
      ↪  timeModifier);
802
803     // Future position will be used to calc future direction
804     // Formula: x' = x + v0 t × 0.5at^2
805     // Future neighbor position
```

```
806     double nb_xFuture = nb->nb_x +
807                 (nb->nb_speed * timeModifier)
808                 + (0.5 * nb->nb_accel * timeModifier * timeModifier);
809     double nb_yFuture = nb->nb_y +
810                 (nb->nb_speed * timeModifier)
811                 + (0.5 * nb->nb_accel * timeModifier * timeModifier);
812
813     // Future this_node position
814     double iXFuture = posX +
815                 (iSpeed * timeModifier)
816                   + (0.5 * iAccel * timeModifier * timeModifier);
817     double iYFuture = posY +
818                 (iSpeed * timeModifier)
819                   + (0.5 * iAccel * timeModifier * timeModifier);
820
821     // Calculate future distance between next-hop and dst
822     double futureDistance;
823     futureDistance = sqrt(pow((nb_xFuture - xDst), 2) +
    ↪   pow((nb_yFuture - yDst), 2));
824
825     // Future radius between this node and neighbor: both using
    ↪   future values
826     double futureRadius = std::min(
827       sqrt(pow((nb_xFuture - iXFuture), 2) + pow((nb_yFuture -
    ↪   iYFuture), 2)), (double) maxTxRange);
828
829     double futureQuality = 1.0 / (1.0 - (futureRadius / ((double)
    ↪   maxTxRange + 1.0)));
830
831     // Calculate future TWR
832     modSpeed     = fSpeed * nb_speedFuture;
833     modAccel     = fAccel * nb->nb_accel;
834     modDistance  = fDistance * futureDistance;
835     modQuality   = fQuality * futureQuality;
836     double futureTWR = modSpeed + modAccel + modDistance + modQuality;
837
838     // Store TWR futureTWR in a list
839     TWRContainer container;
840     container.TWR = TWR;
841     container.futureTWR = futureTWR;
842     container.address = nb->nb_addr;
843     container.isOptimal = false;
844     container.isStable = false;
845     container.isFutureBetter = false;
846
847     listTWR.push_back(container);
848   }
```

```
849
850   // Relay decision making based on TWR and futureTWR using some rules
851   // ascending sort listTWR based on TWR value
852   std::sort(listTWR.begin(), listTWR.end(), minTWR());
853
854   // minimum TWR in the list
855   listTWR[0].isOptimal = true;
856
857   for (u_int32_t i = 0; i < listTWR.size(); i++) {
858     // TWR yang akan datang dikatakan "lebih baik" apabila nilainya
      ↪   lebih
859     // rendah dari TWR sekarang. TWR yang akan datang dikatakan "lebih
860     // buruk" apabila nilainya lebih besar dari TWR sekarang.
861     if (listTWR[i].futureTWR < listTWR[i].TWR) {
862       listTWR[i].isFutureBetter = true;
863     }
864
865     // Jika ΔTWR lebih kecil dari threshold W, maka node tersebut
866     // dianggap sebagai "stable".
867     // Jika ΔTWR lebih besar dari threshold W, maka node tersebut
868     // dianggap sebagai "unstable".
869     double deltaTWR = fabs(listTWR[i].TWR - listTWR[i].futureTWR);
870     if (deltaTWR < thresholdW) {
871       listTWR[i].isStable = true;
872     }
873
874     // Optimal, unstable, better -> Relay
875     if (listTWR[i].isOptimal && !listTWR[i].isStable &&
      ↪   listTWR[i].isFutureBetter) {
876       eligibleAddrList.push_back(listTWR[i].address);
877     }
878     // Optimal, stable, not better -> Relay
879     else if (listTWR[i].isOptimal && listTWR[i].isStable &&
      ↪   !listTWR[i].isFutureBetter) {
880       eligibleAddrList.push_back(listTWR[i].address);
881     }
882     // Suboptimal, unstable, better -> Relay
883     else if (!listTWR[i].isOptimal && !listTWR[i].isStable &&
      ↪   listTWR[i].isFutureBetter) {
884       eligibleAddrList.push_back(listTWR[i].address);
885     }
886     // Suboptimal, stable, not better -> Relay
887     else if (!listTWR[i].isOptimal && listTWR[i].isStable &&
      ↪   !listTWR[i].isFutureBetter) {
888       eligibleAddrList.push_back(listTWR[i].address);
889     }
890   }
```

```
891
892   // replace rq_eligible_nodes with the re-computed list
893   rq->nodes_list_len = eligibleAddrList.size();
894   eligibleNodes = new nsaddr_t[rq->nodes_list_len];
895   for (u_int32_t i = 0; i < rq->nodes_list_len; i++) {
896     eligibleNodes[i] = eligibleAddrList[i];
897   }
898   rq->rq_eligible_nodes = eligibleNodes;
899
900  }
901
902
903   /*
904    * Cache the broadcast ID
905    */
906   id_insert(rq->rq_src, rq->rq_bcast_id);
907
908
909
910   /*
911    * We are either going to forward the REQUEST or generate a
912    * REPLY. Before we do anything, we make sure that the REVERSE
913    * route is in the route table.
914    */
915   aodv_rt_entry *rt0; // rt0 is the reverse route
916
917     rt0 = rtable.rt_lookup(rq->rq_src);
918     if(rt0 == 0) { /* if not in the route table */
919     // create an entry for the reverse route.
920       rt0 = rtable.rt_add(rq->rq_src);
921     }
922
923     rt0->rt_expire = max(rt0->rt_expire, (CURRENT_TIME +
      ↪   REV_ROUTE_LIFE));
924
925     if ( (rq->rq_src_seqno > rt0->rt_seqno ) ||
926          ((rq->rq_src_seqno == rt0->rt_seqno) &&
927       (rq->rq_hop_count < rt0->rt_hops)) ) {
928     // If we have a fresher seq no. or lesser #hops for the
929     // same seq no., update the rt entry. Else don't bother.
930  rt_update(rt0, rq->rq_src_seqno, rq->rq_hop_count, ih->saddr(),
931                 max(rt0->rt_expire, (CURRENT_TIME + REV_ROUTE_LIFE))
      ↪   );
932       if (rt0->rt_req_timeout > 0.0) {
933       // Reset the soft state and
934       // Set expiry time to CURRENT_TIME + ACTIVE_ROUTE_TIMEOUT
935       // This is because route is used in the forward direction,
```

```
936       // but only sources get benefited by this change
937         rt0->rt_req_cnt = 0;
938         rt0->rt_req_timeout = 0.0;
939         rt0->rt_req_last_ttl = rq->rq_hop_count;
940         rt0->rt_expire = CURRENT_TIME + ACTIVE_ROUTE_TIMEOUT;
941       }
942
943       /* Find out whether any buffered packet can benefit from the
944        * reverse route.
945        * May need some change in the following code - Mahesh 09/11/99
946        */
947       assert (rt0->rt_flags == RTF_UP);
948       Packet *buffered_pkt;
949       while ((buffered_pkt = rqueue.deque(rt0->rt_dst))) {
950         if (rt0 && (rt0->rt_flags == RTF_UP)) {
951      assert(rt0->rt_hops != INFINITY2);
952           forward(rt0, buffered_pkt, NO_DELAY);
953         }
954       }
955     }
956     // End for putting reverse route in rt table
957
958
959   /*
960    * We have taken care of the reverse route stuff.
961    * Now see whether we can send a route reply.
962    */
963
964   rt = rtable.rt_lookup(rq->rq_dst);
965
966   // First check if I am the destination ..
967
968   if(rq->rq_dst == index) {
969
970  #ifdef DEBUG
971     fprintf(stderr, "%d - %s: destination sending reply\n",
972                     index, __FUNCTION__);
973  #endif // DEBUG
974
975
976     // Just to be safe, I use the max. Somebody may have
977     // incremented the dst seqno.
978     seqno = max(seqno, rq->rq_dst_seqno)+1;
979     if (seqno%2) seqno++;
980
981     sendReply(rq->rq_src,           // IP Destination
982               1,                    // Hop Count
```

```
983                 index,                  // Dest IP Address
984                 seqno,                  // Dest Sequence Num
985                 MY_ROUTE_TIMEOUT,       // Lifetime
986                 rq->rq_timestamp);      // timestamp
987
988       delete[] rq->rq_eligible_nodes;
989       rq->rq_eligible_nodes = NULL;
990       Packet::free(p);
991     }
992
993     // I am not the destination, but I may have a fresh enough route.
994
995     else if (rt && (rt->rt_hops != INFINITY2) &&
996              (rt->rt_seqno >= rq->rq_dst_seqno) ) {
997
998       //assert (rt->rt_flags == RTF_UP);
999       assert(rq->rq_dst == rt->rt_dst);
1000      //assert ((rt->rt_seqno%2) == 0);    // is the seqno even?
1001      sendReply(rq->rq_src,
1002                rt->rt_hops + 1,
1003                rq->rq_dst,
1004                rt->rt_seqno,
1005              (u_int32_t) (rt->rt_expire - CURRENT_TIME),
1006              //             rt->rt_expire - CURRENT_TIME,
1007                rq->rq_timestamp);
1008      // Insert nexthops to RREQ source and RREQ destination in the
1009      // precursor lists of destination and source respectively
1010      rt->pc_insert(rt0->rt_nexthop); // nexthop to RREQ source
1011      rt0->pc_insert(rt->rt_nexthop); // nexthop to RREQ destination
1012
1013    #ifdef RREQ_GRAT_RREP
1014
1015      sendReply(rq->rq_dst,
1016                rq->rq_hop_count,
1017                rq->rq_src,
1018                rq->rq_src_seqno,
1019              (u_int32_t) (rt->rt_expire - CURRENT_TIME),
1020              //             rt->rt_expire - CURRENT_TIME,
1021                rq->rq_timestamp);
1022    #endif
1023
1024    // TODO: send grat RREP to dst if G flag set in RREQ using
       ↪  rq->rq_src_seqno, rq->rq_hop_counT
1025
1026    // DONE: Included gratuitous replies to be sent as per IETF aodv
       ↪  draft specification. As of now, G flag has not been
       ↪  dynamically used and is always set or reset in aodv-packet.h
       ↪  --- Anant Utgikar, 09/16/02.
```

```
1027
1028        Packet::free(p);
1029      }
1030      /*
1031       * Can't reply. So forward the  Route Request
1032       */
1033     else {
1034
1035       ih->saddr() = index;
1036       ih->daddr() = IP_BROADCAST;
1037       rq->rq_hop_count += 1;
1038       // Maximum sequence number seen en route
1039       if (rt) rq->rq_dst_seqno = max(rt->rt_seqno, rq->rq_dst_seqno);
1040       forward((aodv_rt_entry*) 0, p, DELAY);
1041
1042     }
1043
1044   }
```

A.4 Kode *awk* Perhitungan *Packet Delivery Ratio*

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

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

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

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

A.5 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;
```

```
45        if ( start < end ) {
46            sigma_duration += packet_duration;
47            count++;
48        }
49      }
50      if ( count == 0 ) {
51        printf("no_packet_counted\n");
52      }
53      else {
54        printf("Average Delay: %.4f\n", sigma_duration / count);
55      }
56    }
```

A.6 Kode *awk* Perhitungan *Routing Overhead*

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

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

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

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

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

A.7 Kode Pelacakan Rute Paket Data CBR

```
import collections
import sys

# Usage: python trace_cbr.py <tracefile.tr>
filename = sys.argv[1]

lines = [line.rstrip('\n') for line in open(filename)]
links = dict()
drop_cbk = 0
drop_nrte = 0
drop_arp = 0
for line in lines:
    line = line.split()
    node = line[2]
    packet_id = line[5]
    layer = line[6]

    if layer == "cbr":
        if packet_id in links:
            if node not in links[packet_id]:
                links[packet_id].append(node)
        else:
            links[packet_id] = [node]
        if line[0] == "D":
            drop_cause = "DROP-" + line[4]
            links[packet_id].append(drop_cause)
            if drop_cause == "DROP-CBK":
                drop_cbk += 1
            elif drop_cause == "DROP-NRTE":
                drop_nrte += 1
            elif drop_cause == "DROP-ARP":
                drop_arp += 1

links_sorted = collections.OrderedDict(sorted(links.items()))
for key, value in links_sorted.iteritems():
    print key, value

print "Drop CBK: ", drop_cbk
print "Drop NRTE: ", drop_nrte
print "Drop IFQ ARP: ", drop_arp
print "Total Drop: ", (drop_cbk + drop_nrte + drop_arp)
```

BAB 6

PENUTUP

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

6.1 Kesimpulan

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

1. *Packet delivery ratio* dari AODV-PNT meningkat dari 11,8% hingga 23% pada skenario *grid* dan dari 7,4% hingga 11,2% pada skenario riil dibandingkan dengan AODV. *Packet delivery ratio* dari AODV-PNT dan AODV meningkat seiring bertambahnya kepadatan kendaraan dalam jaringan.
2. *Average end-to-end delay* dari AODV-PNT menurun seiring bertambahnya kepadatan kendaraan dalam jaringan sedangkan AODV cenderung meningkat seiring bertambahnya kepadatan kendaraan. Namun pada puncaknya (400 *node*), AODV-PNT juga mengalami peningkatan *delay*.
3. *Routing overhead* pada AODV-PNT lebih sedikit daripada AODV dikarenakan AODV-PNT melakukan seleksi kestabilan terhadap *next-hop node* sedangkan AODV melakukan *broadcast* sehingga setiap *neighbor node* menerima RREQ.

6.2 Saran

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

1. Studi lebih lanjut mengenai pengaruh dari koefisien faktor pengali dan *threshold* W yang digunakan pada rumus TWR. Ji-

ka ada pengaruh pada skenario tertentu, maka koefisien faktor pengali dapat dibuat menjadi adaptif.

2. 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.

DAFTAR PUSTAKA

[1] Xi Yu, Huaqun Guo, and Wai-Choong Wong. A Reliable Routing Protocol for VANET Communications. pages 1748--1753, July 2011. doi: 10.1109/IWCMC.2011.5982800.

[2] Xiaowei Shen, Yi Wu, Zhexin Xu, and Xiao Lind. AODV-PNT: An Improved Version of AODV Routing Protocol with Predicting Node Trend in VANET. pages 91--97, November 2014. doi: 10.1109/ICAIT.2014.7019536.

[3] Christoph Sommer and Falko Dressler. *Vehicular Networking*. Cambridge University Press, 2015. ISBN 1107046718.

[4] NS3 Code. VANET Project. `http://ns3-code.com/ns3-vanet-projects/`. Accessed: 2016-05-17.

[5] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc On-Demand Distance Vector (AODV) Routing. RFC 3561, RFC Editor, July 2003. http://www.rfc-editor.org/rfc/rfc3561.txt.

[6] Jacek Rak. LLA: A New Anypath Routing Scheme Providing Long Path Lifetime in VANETs. pages 281--284, December 2013. doi: 10.1109/LCOMM.2013.120413.132285.

[7] Daniel Krajzewicz, Jakob Erdmann, Michael Behrisch, and Laura Bieker. Recent Development and Applications of SUMO - Simulation of Urban MObility. *International Journal On Advances in Systems and Measurements*, 5:128--138, December 2012.

[8] OpenStreetMap. `https://www.openstreetmap.org/`. Accessed: 2016-05-17.

[9] Steven McCanne and Sally Floyd. ns Network Simulator. `http://www.isi.edu/nsnam/ns/`. Accessed: 2016-05-17.

[10] David Wetherall. OTcl. http://otcl-tclcl.sourceforge.net/otcl/, 1997. Accessed: 2016-05-17.

BIODATA PENULIS

Reyhan Arief dilahirkan di kota Samarinda pada tanggal 20 Oktober tahun 1994. Penulis menempuh pendidikan SD IT Cordova Samarinda (2000-2006), SMP IT Cordova Samarinda (2006-2009), SMA Negeri 1 Samarinda (2009-2012). Pada tahun 2012, penulis mengikuti jalur Program Kemitraan dan Mandiri (PKM) dan diterima di Strata Satu Jurusan Teknik Informatika, Fakultas Teknologi Informasi, Institut Teknologi Sepuluh Nopember Surabaya angkatan 2012 yang terdaftar dengan NRP 5112100175. Di Jurusan Teknik Informatika ini, penulis mengambil bidang minat Arsitektur dan Jaringan Komputer (AJK). Selama menempuh kuliah, penulis juga pernah aktif sebagai anggota Departemen Riset dan Teknologi di Himpunan Mahasiswa Teknik Computer-Informatika (HMTC). Penulis dapat dihubungi melalui alamat *e-mail* reyhan.arief.biz@gmail.com.