TESIS -KI2501

Studi Kinerja DSR-PNT dengan Penambahan Faktor Jumlah Node Tetangga pada VANETs

BAGUS GEDE KRISHNA YUDISTIRA
5115201035

DOSEN PEMBIMBING
Dr.Eng. Radityo Anggoro S.Kom., M.Sc.
Prof.Ir. Supeno Djanali, M.Sc., Ph.D.

PROGRAM MAGISTER
BIDANG KEAHLIAN KOMPUTASI BERBASIS JARINGAN
JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2017

Tesis disusun untuk memenuhi salah satu syarat memperoleh gelar
Magister Komputer (M.Kom.)
di
Institut Teknologi Sepuluh Nopember Surabaya

oleh:
Bagus Gede Krishna Yudistira
Nrp. 5115201035

Dengan judul :
Studi Kinerja DSR-PNT dengan Penambahan Faktor Jumlah Node Tetangga pada VANETs

Tanggal Ujian : 13-7-2017
Periode Wisuda : 2016 Genap

Disetujui oleh:

Dr.Eng. Radityo Anggoro, S.Kom, M.Sc
NIP. 198410162008121000 2 (Pembimbing 1)

Prof.Ir.Supeno Djanali, M.Sc, Ph.D
NIP. 194806191973011001 (Pembimbing 2)

Waskitho Wibisono, S.Kom, M.Eng, Ph.D
NIP. 19741022200003100 (Penguji 1)

Royyana Muslim I, S.Kom, M.Kom, Ph.D
NIP. 197708242006041001 (Penguji 2)

Tohari Ahmad, S.Kom, MIT, Ph.D
NIP. 197505252003121002 (Penguji 3)

Dekan FTIf – ITS

Dr. Agus Zainal Arifin, S. Kom., M. Kom.
NIP. 197208091995121001

(halaman ini sengaja dikosongkan)

STUDI KINERJA DSR-PNT DENGAN PENAMBAHAN FAKTOR JUMLAH NODE TETANGGA PADA VANETS

Nama Mahasiswa	: Bagus Gede Krishna Yudistira
NRP	: 5115201035
Pembimbing	: Dr.Eng. Radityo Anggoro S.Kom., M.Sc. Prof.Ir. Supeno Djanali, M.Sc., Ph.D.

ABSTRAK

Vehicular Ad Hoc Networks (VANETs) merupakan sebuah bagian dari *Mobile Ad Hoc Networks* (MANETs). Kendaraan yang menjadi sebuah node dalam VANETs biasanya bergerak dengan cepat dan kecepatan yang bervariabel sehingga topologinya dapat dengan cepat berubah. Salah satu permasalahan yang umum terjadi pada VANETs adalah konektifitas antar kendaraan dan pemilihan kendaraan mana yang paling tepat untuk menjadi perantara antara pengirim paket dan tujuan.

Untuk menentukan *node* yang paling sesuai untuk menjadi *relaying node*, dibutuhkan studi yang mempelajari faktor apa yang membuat sebuah *node* menjadi *node* yang paling pantas dipilih sebagai *forwarder*. Faktor tersebut dapat berupa kecepatan kendaraan maupun akselerasinya, arah pergerakan dari kendaraan, maupun kualitas *link* antar kendaraan. Setelah mengetahui ketiga faktor tersebut pada setiap *neighbour node* yang tersedia, *future Total Weight Route* (TWR) dapat dikalkulasi sehingga dapat diketahui *route* dari *source* ke *destination* yang paling optimal.

Pada penelitian ini, packet delivery ratio dari metode yang disuulkan (DSR-PNT) unggul sebanyak 20% dari protokol DSR. Hal ini disebabkan oleh seleksi relaying node yang paling layak dipilih sebagai forwarder dengan perhitungan TWR sehingga keberhasilan pengiriman paket dapat berjalan secara optimal. Penelitian ini membahas dampak penambahan *parameter neighbour* node pada metrik routing dalam menentukan nilai TWR serta pengaruh node density terhadap *packet delivery ratio*.

Kata kunci: Connectivity Model, Connectivity Pattern, Relay Node, VANETs, VANETs Scenario, Vehicular Networks.

(halaman ini sengaja dikosongkan)

DSR-PNT PERFORMANCE STUDY WITH ADDITION OF NEIGHBORS ON NODE NUMBER OF FACTORS VANETS

Nama Mahasiswa	: Bagus Gede Krishna Yudistira
NRP	: 5115201035
Pembimbing	: Dr.Eng. Radityo Anggoro S.Kom., M.Sc. Prof.Ir. Supeno Djanali, M.Sc., Ph.D.

ABSTRACT

Vehicular Ad Hoc Networks (VANETs) is a part of Mobile Ad Hoc Networks (MANETs). Node in VANETs usually moves quickly and its speed continuously changed so that the topology also quickly changed. One of the problems that commonly occur in VANETs is connectivity between the vehicle and the selection of the most appropriate vehicle to be an intermediary between the sender of the packet and the destination.

To determine the most appropriate node to be relaying node, requires study of what factors make a node being a node most electable as a forwarder. Such factors may include vehicle speed and acceleration, direction of movement of the vehicle, as well as the quality of the links between vehicles. After knowing these three factors at each neighbor node is available, futures Total Weight Route (TWR) can be calculated so that the most optimal route from source to destination can be known.

In This Study, proposed method (DSR-PNT) improved approximately 20% of packet delivery ratio from the original DSR. This situation could be achieved with the selection process of relaying node, which is the relaying node must be the most optimal node in the neighbour list. The most optimal node and their suboptimal node calculated with TWR process. This study discusses the impact of neighbor node parameters added to the routing metric in determining the value of TWR and the impact of node density to packet delivery ratio.

Kata kunci: Connectivity Model, Connectivity Pattern, Relay Node, VANETs, VANETs Scenario, Vehicular Networks.

(halaman ini sengaja dikosongkan)

KATA PENGANTAR

Om Swastiastu.

Atas berkat dan rahmat dari Ida Sang Hyang Widhi Wasa serta doa dan motivasi yang diberikan oleh keluarga penulis, maupun rekan-rekan seperjuangan, sehingga penulis dapat menyelesaikan thesis yang berjudul "**Studi Kinerja DSR-PNT dengan Penambahan Faktor Jumlah Node Tetangga pada VANETs**" dengan baik dan tepat waktu.

Dalam pelaksanaan dan pembuatan penelitian ini, tentunya banyak sekali bantuan yang telah penulis terima dari berbagai pihak. Tanpa mengurangi rasa hormat penulis ingin mengucapkan terima kasih yang sebesar-besarnya kepada:

1. Ida Sang Hyang Widhi Wasa atas limpahan berkah dan pencerahannya sehingga penulis dapat menyelesaikan penelitian ini dengan baik.
2. Kedua Orang Tua Penulis, I Nyoman Kanca dan Ni Wayan Sukarmini yang telah memberikan support penuh kepada penulis baik secara moral, spiritual, dan finansial. Serta doa yang selalu dipanjatkan kepada Tuhan agar penulis selalu berpikiran lurus dan bijak dalam bertutur maupun bertindak.
3. Adik Penulis, Luh Made Wisnu Satyaninggrat dan Luh Komang Monika Paramarthika yang selalu mensupport agar penulis tetap termotivasi.
4. Keluarga Sapta Kerti, Keluarga besar penulis yang tidak bisa disebutkan satu-persatu yang selalu memberikan support beserta nasehat agar penulis selalu semangat dalam mengikuti jenjang pendidikan.
5. Bapak Pembimbing Dr.Eng. Radityo Anggoro S.Kom., M.Sc. dan Prof.Ir. Supeno Djanali, M.Sc., Ph.D. yang selalu sabar membimbing penulis dalam menyelesaikan penelitian ini.
6. Waskitho Wibisono, S.Kom., M.Eng., Ph.D. yang selalu sabar membimbing dan memberi masukan. Serta mengajak berlatih band di studio TC bersama Bu Nanik Suciati, Pak Bagus, dan Pak Rizki Januar.
7. Rekan-Rekan The Error Project, Budi KCX, Agus Yana, Iwak Qyong, Yuda Krisnantara yang selalu menghibur dan mengejar deadline penggarapan album Dunia dalam Harapan sehingga penulis termotivasi untuk mengerjakan penelitian ini.
8. Rekan-Rekan seperjuangan S2-Informatika Angkatan 2015.
9. Rekan-Rekan seperjuangan Grezio, Adhiarta, Phebhe, Wiryo, Tsabbit.
10. Serta semua pihak yang belum sempat disebutkan satu per satu yang telah membantu terselesaikannya penelitian ini.

Penulis mengetahui bahwa penelitian ini jauh dari kata sempurna. Oleh karena itu penulis memohon saran dan kritik yang membangun dari pembaca.

Surabaya, Juli 2017

Bagus Gede Krishna Yudistira

(halaman ini sengaja dikosongkan)

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

(halaman ini sengaja dikosongkan)

BAB I

PENDAHULUAN

Pada Bab ini akan dijelaskan mengenai beberapa hal dasar dalam penelitian yang meliputi latar belakang, perumusan masalah, tujuan, manfaat, kontribusi penelitian, dan batasan masalah.

1.1 Latar belakang

VANETs (*Vehicular Ad Hoc Networks*) merupakan sebuah jaringan bergerak berbasis ad hoc network yang menjalankan prinsip dari MANETs (*Mobile Ad Hoc Network*) (Toh & K, 2002). Jaringan VANETs terbentuk berdasarkan konsep komputasi bergerak dimana kendaraan yang menjadi nodenya. Dewasa ini, VANETs dikembangkan dengan tujuan komunikasi antar kendaraan pada jalan raya maupun daerah perkotaan.

Pada jaringan yang bersifat statis, pengiriman *data* secara *wireless* memiliki halangan yang lebih sedikit daripada VANETs yang sifatnya dinamis. VANETs (*Vehicular Ad Hoc Networks*) adalah sebuah jaringan yang bersifat dinamis dan sangat rentan mengalami *delay* saat pengiriman *data* dari pengirim ke penerima karena *node* dari VANETs ini terus berubah-ubah (komunikasinya *multi-hop* pada kendaraan dan *Road side unit*). Oleh karena itu dibutuhkan sebuah metode yang mampu meningkatkan efisiensi pengiriman *data* pada VANETs. Secara umum, VANETs membutuhkan RSU (Road side unit) yang berperan sebagai jembatan/penghubung antar *node* (kendaraan) yang ada pada VANETs. Pada dunia nyata, tentu saja ada sebuah kondisi dimana jalan sama sekali tidak ada kendaraan, jalan sangat padat, dan jalan yang sepi. Keadaan-keadaan inilah yang dapat mempengaruhi efisiensi dari pengiriman *data* pada VANETs.

1.2 Perumusan Masalah

Untuk meningkatkan efisiensi pengiriman *data* pada jaringan bergerak seperti VANETs, terdapat beberapa hal yang perlu diperhatikan yaitu: tingkat kepadatan *traffic*, metode yang digunakan untuk mengirim *data*, dan lokasi yang ditetapkan sebagai studi kasus. Dalam hal ini, penulis mengambil daerah

perkotaan (*urban*) sebagai lokasi untuk studi kasus. Daerah perkotaan merupakan daerah yang memiliki tingkat kepadatan *traffic* yang tinggi dan pengemudi di daerah perkotaan sangat membutuhkan informasi di jalan raya. Informasi ini akan membantu pengemudi untuk mengetahui rute yang tepat untuk dilalui dan dapat mengurangi kemacetan di daerah perkotaan. Terlalu padatnya *traffic* pada daerah perkotaan dapat menjadi suatu permasalahan dalam efisiensi pengiriman *data* pada jaringan VANETs. Kondisi ini membutuhkan adanya suatu metode yang dapat mengoptimasi penyaluran *data* pada saat informasi dikirimkan dari *node* pengirim hingga *node* penerima. Hal ini akan membuat pengiriman *data* pada jaringan VANETs dapat berjalan secara efisien dan memiliki tingkat keberhasilan pengiriman *data* yang tinggi.

1.3 Tujuan Penelitian

Tujuan yang akan dicapai pada penelitian ini adalah :

1. Meningkatkan performa DSR diukur dengan menggunakan *routing metric (Packet Delivery Ratio, Routing Overhead, delay)*
2. Mencari nilai *Total weight of the route* (TWR) dengan *parameter* tambahan yaitu neighbour *node* sebagai acuan untuk menentukan *next hop node* yang reliabel.

1.4 Manfaat Penelitian

Manfaat penelitian ini adalah sebagai berikut :

1. Meningkatkan rasio dari pengiriman paket yang sukses pada metode sebelumnya, serta mengurangi adanya *congestion* pada jaringan yang disebabkan oleh terlalu banyaknya paket yang di*broadcast* agar paket sampai ke tujuan.

1.5 Kontribusi

Kontribusi penelitian ini adalah :

1. Meningkatkan performa pengiriman paket *data* dengan menambahkan *parameter neighbour node* pada DSR-PNT dalam pemilihan *routing node* (*node forwarder*, *next hop*) untuk meningkatkan PDR (*packet delivery ratio*).

   .

1.6 Batasan Penelitian

Dalam penelitian ini, dibutuhkan batasan masalah agar materi yang dibahas pada penelitian ini menjadi lebih terarah dan lebih spesifik. Adapun batasan masalah dalam penelitian ini yaitu:

1. *Routing* Protokol yang digunakan adalah DSR.
2. Implementasi dan uji coba menggunakan *network simulator* NS-2.

(halaman ini sengaja dikosongkan)

BAB 2

LANDASAN TEORI DAN KAJIAN PUSTAKA

2.1 VANETs

VANETs (*Vehicular Ad Hoc Networks*) merupakan sebuah jaringan bergerak bebasis *ad hoc network* yang menjalankan prinsip dari MANETs (*Mobile Ad Hoc Network*). Tujuan utama VANETs adalah membantu kendaraan berkomunikasi dan bertukar informasi di jalan raya tanpa menggunakan *control base station*.

VANETs bertanggung jawab untuk menangani komunikasi antar kendaraan pada suatu daerah tertentu. Kendaraan dapat berhubungan dengan kendaraan lain (*Vehicle to Vehicle*) atau berhubungan dengan *Road Side Unit* (*Vehicle to Roadside*) tergantung dari *availabillity neighbour node* pada saat itu.

2.2 Routing Protokol DSR

DSR (*Dynamic Source Routing*) merupakan protokol yang didesain secara khusus untuk menangani *node* yang bergerak secara terus menerus. Dengan menggunakan DSR, jaringan akan secara otomatis berubah dan menkonfigurasi dirinya sendiri tanpa adanya infrastruktur jaringan maupun *administrator* (Mohamed, Mustapha, Walid, & Abdelhamid, 2007). *Node* pada jaringan DSR berhubungan satu sama lain dan bekerja sama dalam pengiriman paket menuju ke tujuan melalui beberapa *hop*. Pergerakan *node* sangat cepat berubah, dan *sequence node* untuk pengiriman paket sampai ketujuan yang dapat berubah kapan saja akan membuat topologi jaringan sering berubah-ubah.

Protokol DSR memperbolehkan *node* secara dinamis mencari sumber paket beserta *multi hop*nya hingga tujuan paket tersebut. Setiap *data* paket membawa *header* yang berisi informasi list dari *node* yang telah dilaluinya, sehingga paket routing tidak mengalami *looping* yang dapat menyebabkan *congestion*. Dengan adanya *header* ini, *node* tetangga yang memiliki tugas yang sama dapat menyimpan informasi pada *header* untuk keperluan yang akan datang.

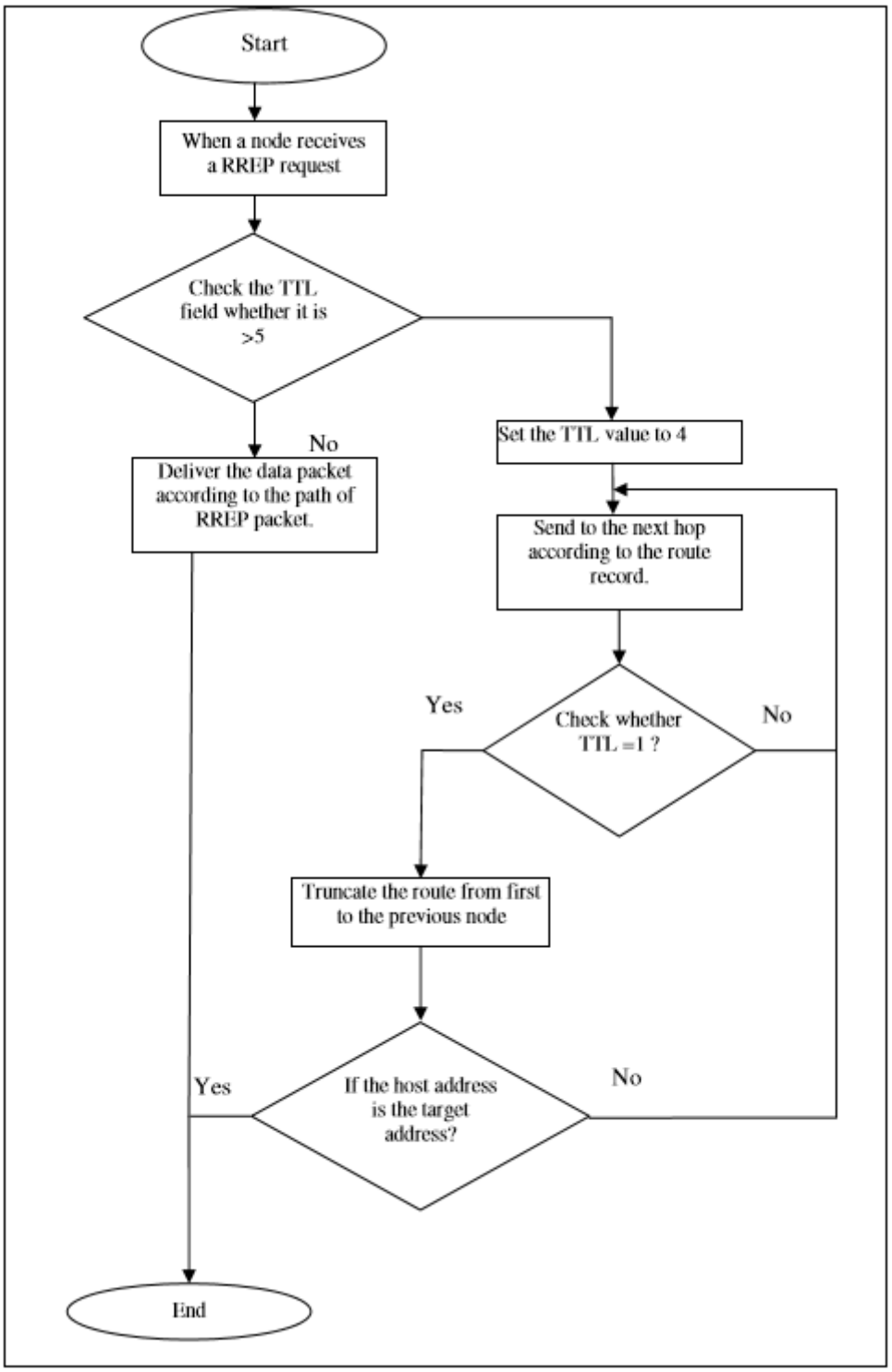
2.2.1 Enhanced DSR

Enhanced DSR merupakan pengembangan dari protokol DSR. Protokol Enhanced DSR ini dibangun untuk mengurangi *packet broadcast* yang dilakukan oleh *source node* sehingga dapat mengurangi *congestion* pada *network*. Pengurangan *control packet overhead* ini dilakukan dengan mengubah metode penerimaan RREQ packet pada suatu *node* (Sharmin, Salma, Nazma, & Ahsan, 2010). Pada saat suatu *node* menerima RREQ *packet*, *node* tersebut akan melakukan beberapa proses berikut:

1. *Node* tersebut akan mencari *list* seluruh *neighbour* pada *neighbour* tabel *node* tersebut.
2. *Node* tersebut akan hanya akan memilih *node* yang berada pada *neighbour tabel* namun belum pernah menerima RREQ. *Node* penerima RREQ tidak akan memilih *node* tetangga yang sebelumnya telah menerima RREQ.
3. *Node* tersebut akan meneruskan paket RREQ berdasarkan *node* yang telah ditemukan pada proses sebelumnya.

Dengan adanya proses penyeleksian *neighbour node* untuk pengiriman RREQ, metode E-DSR dapat mengurangi *control packet overhead* sehingga *routing overhead* pada E-DSR cenderung lebih kecil daripada original DSR.

Protokol DSR pada umumnya akan mencari *route* menuju *destination*, melakukan pencatatan terhadap *rute* yang akan ditempuh dari *source node* menuju ke *destination*. Setelah itu, *packet* akan dikirimkan berdasarkan *list* yang telah ditentukan pada *source node*. Pada Enhanced DSR, *list* rute *packet* yang ditentukan oleh *source node* dibatasi berdasarkan jumlah TTL pada rute tersebut. Proses pengurangan *packet header length* dapat dilihat pada Gambar 2.1.



Gambar 2.1 Flowchart proses pengurangan *packet header length* pada E-DSR

2.2.2 Q-DSR

Protokol Q-DSR merupakan pengembangan dari protokol DSR. Protokol Q-DSR ini dibangun dengan tujuan meningkatkan *packet delivery ratio* dan mengurangi kemungkinan *packet loss* apabila dibandingkan dengan protokol DSR. Hal ini dilakukan dengan mempertimbangkan faktor QoS dan *speed* dari node pada saat pemilihan *relaying node*. Dengan mempertimbangkan faktor QoS dan *speed* pada *relaying node*, Q-DSR dapat mempertahankan *reability* pada *network*.

Algoritma pemilihan jalur pada Q-DSR ditentukan oleh *Quality of service* pada neighbour *node*. Algoritma ini akan memilih *node* yang memiliki nilai QoS yang paling tinggi. Pada Gambar 2.2 dapat dilihat proses pemilihan rute pada Q-DSR dalam bentuk *pseudocode*.

Algorithm 1 Path Selection algorithm

Part 1 – Dissemination phase

```
1: procedure DISSEMINATION PHASE
2:     for each Source node (S) do
3:         Broadcast Route Request (RREQ)
4:         for each intermediate node(N) do
5:             include the QoS(N) in the RREQ
6:         end for
7:     end for
8: end procedure
```

Part 2 – Finding phase

```
9: procedure FINDING PHASE
10:    for each destination D do
11:        Find the QoS for each path s.t
12:        QoS (path) = min (QoS(N1), QoS(N2), ..., QoS(Nn))
13:        Path with max QoS is selected
14:        Send RREP to the source (S)
15:    end for
16: end procedure
```

Gambar 2.2 Proses pemilihan rute pada Q-DSR

2.3 PNT (Predicting Node Trend) Concept

PNT (*Predicting Node Trend*) merupakan sebuah metode prediksi pergerakan node sehingga dalam pengiriman paket, node forwarder dapat memilih node yang paling optimal untuk meneruskan paket hingga sampai ke tujuan (XiaoWei & YiWu, 2014). Improvement pada PNT adalah sebagai berikut:

1. *Routing Metric Improvement* dan menghitung *total weight of the route* (TWR).
2. Memprediksi *node future* dari TWR dan mengkalkulasi *stable threshold* sehingga dapat ditentukan apakah *relay node* tersebut stabil.

2.3.1 Perubahan metrik routing dan perhitungan TWR

Ada empat faktor utama yang dijadikan *parameter* dalam metrik *routing*. Berdasarkan empat faktor tersebut, TWR dihitung untuk menemukan *next-hop node* yang paling optimal. Detail dari empat faktor tersebut adalah sebagai berikut:

1. Kecepatan dan percepatan kendaraan

   Semakin besar perbedaan kecepatan dan percepatan antara dua kendaraan akan menghasilkan TWR yang semakin besar. Alasannya adalah untuk mengantisipasi kerusakan tautan dikarenakan oleh perbedaan kecepatan kendaraan. Kendaraan-kendaraan yang bergerak dalam kecepatan dan percepatan yang relatif sama akan berada di dalam jarak komunikasi radio dalam waktu yang lebih lama.
2. Arah pergerakan kendaraan

   Secara logika, kendaraan-kendaraan yang bergerak pada arah yang sama akan berada di dalam jarak komunikasi radio dalam waktu yang lebih lama. Karena itu, vektor arah juga menjadi krusial dalam perhitungan TWR. Faktanya, *parameter* arah adalah sangat penting dalam menentukan pilihan rute. Nilai perbedaan arah didapat dari perhitungan perbedaan sudut arah berkendara.

3. Kualitas tautan antar kendaraan

   Parameter lain yang perlu dipertimbangkan adalah kualitas tautan antara *node* sumber dengan *next-hop*. Dalam VANETs, kendaraan lain, bangunan, dan objek-objek lain dapat mempengaruhi kualitas tautan antar kendaraan.

Berdasarkan informasi dari pergerakan kendaraan, kita dapat mendapatkan indeks stabilitas s_{ij} dari tautan (i, j). (Donna, et al., 2015)

$$S_{ij} = 1 - \frac{\min(\sqrt{(i_x - j_x)^2 + (i_y - j_y)^2}; r)}{r} \quad (1)$$

Keterangan:

r = jarak komunikasi maksimum antar dua node yang berdekatan.

i_x, i_y = koordinat dari node i.

j_x, j_y = koordinat dari node j.

Nilai kualitas tautan $Q = \frac{1}{S_{ij}}$

TWR dari node sumber ke next-hop node dihitung menggunakan persamaan berikut:

$$TWR = f_s \times | S_n - S_d | + f_a \times | A_n - A_d | + f_d \times | \Theta_n - \Theta_d | + f_q \times Q \quad (2)$$

Keterangan:

S_n, A_n, dan Θ_n = Kecepatan, percepatan, dan arah dari next-hop node.

S_d, A_d, dan Θ_d = Kecepatan, percepatan, dan arah dari node tujuan.

f_s = bobot pengali kecepatan.

f_a = bobot pengali percepatan.

f_d = bobot pengali jarak.

f_q = bobot pengali kualitas tautan.

Q = kualitas tautan antara node sumber dengan next-hop node.

2.3.2 Prediksi Future TWR

VANETs memiliki karakteristik yaitu topologi yang seringkali berubah. Sebuah *node* dalam topologi bisa jadi akan pergi dalam waktu singkat. Jadi, future TWR (3 sampai 5 detik berikutnya) akan dihitung untuk memastikan *node* yang stabil yang akan dipilih di masa yang akan datang. *Future* TWR dihitung berdasarkan faktor-faktor berikut ini:

1. Prediksi kecepatan dan percepatan kendaraan

   Penelitian ini memakai asumsi bahwa percepatan dari sebuah *node* adalah konstan selama periode waktu perhitungan *future* TWR. Kecepatan *node* berubah sesuai dengan rumus percepatan.

2. Prediksi arah pergerakan kendaraan

   Di dunia nyata, jika terdapat tikungan di jalan, kendaraan akan mempunyai akselerasi negatif dan menyalakan lampu sein. Kendaraan yang memenuhi kondisi tersebut akan menghitung vektor arah dengan bantuan GPS.

3. Prediksi kualitas tautan antar kendaraan

   Berdasarkan *data* kecepatan, percepatan, dan arah kendaraan, metode ini akan mendapatkan koordinat dari sebuah *node* di waktu yang akan datang. Kualitas tautan antar *node* dapat dihitung berdasarkan koordinat dari kedua *node* tersebut.

Setelah mendapatkan *data* tersebut, future TWR dari *node* akan dapat dihitung. Nilai ini akan digunakan sebagai sebuah *parameter* untuk menentukan apakah sebuah *node* layak untuk dijadikan *relay node*. Selanjutnya, penilain kelayakan sebuah *node* untuk dijadikan *relay node* dijelaskan pada Tabel 2.1.

Tabel 2.1 Penilaian kelayakan relay node

TWR saat ini	Status	Future TWR	Penilaian
Optimal	Tidak stabil	Lebih baik	Pilih node
Optimal	Stabil		Pilih node
Suboptimal	Tidak stabil	Lebih baik	Pilih node
Suboptimal	Stabil		Pilih node
Kondisi lain			Abaikan node

TWR saat ini : semakin kecil nilai TWR adalah semakin bagus. Node dengan nilai TWR paling kecil adalah “Optimal”, sedangkan *node* lain dengan nilai TWR bagus disebut “Suboptimal”.

Status : Sebuah nilai *W* ditentukan sebagai nilai ambang batas. Jika nilai absolut dari perbedaan TWR saat ini dan future TWR (Δ TWR) kurang dari *W*, maka node tersebut dianggap sebagai node yang stabil. Jika Δ TWR lebih dari *W*, maka node tersebut dianggap tidak stabil.

2.4 Neighbour Node Discovery

Protokol *reactive* akan melakukan *broadcast hello message* untuk mengetahui *node* yang berada disekitar *node source* (RREQ). Setelah menerima *Route reply* (RREP), *routing overhead* dapat ditentukan agar paket dari *source* dapat diteruskan sampai ke *destination*.

Seringkali setiap *node* melakukan komunikasi secara langsung dengan *node* yang berada disekitarnya. Tanpa adanya *central controller*, setiap *node* harus mencari *node* yang berada disekitarnya sebelum *routing* yang efisien dapat ditentukan oleh *central controller*. Sebuah proses mencari *node* yang berada disekitar *node* pembawa paket ini disebut proses *neighbour discovery* yang merupakan langkah awal yang paling krusial pada saat membangun jaringan *wireless ad hoc network* yang reliabel (S Feroz Ahammad, 2014).

2.5 Path Duration pada VANETs

Terdapat tujuh faktor yang dapat mempengaruhi *path duration* pada VANETs (Vikas Toor, Oktober 2012). Ketujuh hal tersebut adalah:

1. *Least Remaining Distance and Shortest Path*

   Least Remaining Distance forwarding memilih *node* yang paling dekat dan memiliki jarak paling minimum kepada *node* tujuan. Sedangkan *Shortest Path* memilih jarak yang paling pendek serta *hop* yang paling sedikit.

2. *Link Residual Life*

   Link Residual Life terjadi pada saat adanya direct link antara dua *node* yang aktif dan merupakan bagian dari route transmisi paket. Jarak maksimal jangkauan transmisi dari *node source* dan *next hop node* dapat didefinisikan sebagai berikut: $t = \frac{d}{v_r}$

   Keterangan:

   d = jarak transmisi *maksimum* yang dapat dijangkau *source node* ke *next hop node*.

   Vr = kecepatan relatif kedua *node* (*source* dan *next hop*).

3. *Link Distance*

   Link Distance adalah jarak antara kedua *node* yang didefinisikan berdasarkan *link* yang tersedia antara kedua *node* tersebut. Faktor ini sangat dipengaruhi oleh protokol yang digunakan pada VANETs. *Link distance* akan bertambah apabila *next hop node* yang dipilih adalah *node* yang berada pada jarak maksimum transmisi dari *source node*.

4. *Node Density*

   Node density adalah jumlah kendaraan pada suatu area yang tercakup dalam range transmisi *source node*. Pada daerah yang *node densitynya* rendah, durasi *link* dan *quality link* menjadi hal yang paling penting yang mempengaruhi *packet delivery ratio*. Path *failure* dapat terjadi apabila *node* yang dituju sebagai *next hop* keluar dari *range* transmisi *source node*. Sedangkan pada daerah yang *node densitynya* tinggi, rentan terjadi *congestion* dan *continuous loop* pada jaringan.

5. *Velocity of Nodes*

   Kecepatan dan arah pergerakan *node* merupakan hal yang sangat krusial dalam perhitungan *path duration* antara *source node* dengan *next hop node*. Kecepatan relatif dan arah pergerakan *node* mempengaruhi kualitas link yang didasari oleh jarak kedua *node* tersebut.

6. *Number of Hops*

   Number of Hop merupakan jumlah *node* yang ada dari *source node* menuju ke *destination node*. Number of *hops* dipengaruhi oleh *Least Remaining Distance*, S*hortest Path, Link Residual Life, Link Distance, Node Density*, dan *Velocity of Nodes*. Number of *hop* yang optimal adalah number of hop yang paling minimum untuk mencegah *link failure*.

7. *Average Progress per Hop*

   Average progress per hop merupakan jarak rata-rata antar *hop*. Hal ini akan mempengaruhi jumlah *hop* dan jalur yang akan ditempuh ke *node* selanjutnya.

2.6 SUMO

SUMO adalah sebuah *simulator packet* lalu lintas berbasis *open source*, SUMO dapat digunakan untuk membantu memodelkan manajemen lalu lintas, kendaraan, dan rute. Pada simulasi SUMO kendaraan dapat dimodelkan dengan detail secara eksplisit yang memungkinkan setiap kendaraan bergerak dan *route* sendiri-sendiri.

SUMO pertama kali diimplementasikan pada tahun 2001 dan dapat digunakan secara *open source* pada tahun 2002. SUMO dikembangkan untuk mendukung riset pada bidang lalu-lintas. Pada tahun 2012 Krajzewicz dkk (Krajzewicz, Jakob, Michael, & Laura, 2012) melakukan publikasi terbaru tentang pengembangan dan aplikasi SUMO

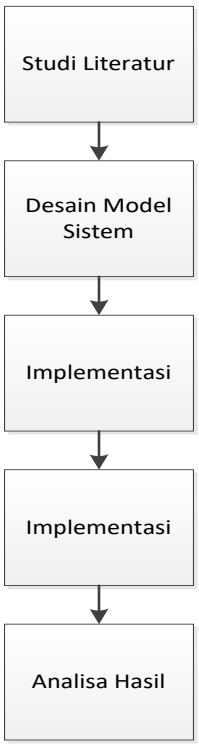
2.7 NS-2

NS-2 (*Network Simulator* 2) merupakan sebuah simulator jaringan yang mensupport simulasi TCP, *routing*, dan protokol *multicast* baik secara nirkabel maupun *local*. NS-2 merupakan simulator yang masih dalam tahap pengembangan namun telah banyak digunakan dalam penelitian khususnya pada bidang komputasi bergerak. (isi.edu, 2011)

BAB 3

METODE PENELITIAN

Metode Penelitian ini melalui beberapa tahap yang meliputi (1) Studi Literatur, (2) Desain Model Sistem, (3) Implementasi, (4) Uji Coba dan Simulasi, (5) Analisa Hasil. Alur tahapan-tahapan tersebut dapat dilihat pada Gambar 3.1



Gambar 3.1 Alur Metode Penelitian

3.1 Studi Literatur

Pada tahap studi literatur dilakukan pencarian informasi yang mendukung penelitian yang akan dilakukan. Dilakukan pula pencarian informasi lainnya yang berkaitan dengan penelitian yang akan dilakukan. Berikut adalah beberapa referensi yang dibutuhkan dalam penelitian ini:

1. VANETs
2. *Routing* Protokol DSR
3. *Predicting Node Trend Concept*
4. *Neighbour discovery method*

Pada Tabel 3.1 di bawah, dapat dilihat *roadmap* pada penelitian yang akan dilakukan oleh penulis. *Roadmap* ini berisi perkembangan penelitian pada bidang VANETs dan protokol DSR.

Tabel 3.1 *Roadmap* penelitian protokol DSR

Metode (Tahun)	Kelebihan	Kekurangan
DSR (1998)	- *Route* ke destinasi hanya dibentuk apabila dibutuhkan - *Node* perantara menggunakan *cache* yang berisi informasi routing sehingga mencegah *routing overhead* - Bebas dari *infinite loop*	- Mekanisme *routing* tidak dapat memperbaiki *broken link* secara local - Pada saat *broken link* terjadi, *routing cache* menjadi tidak reliable pada saat pembentukan r*oute* baru
E-DSR Enhanced DSR (2010)	- Mengurangi *Control packet overhead* untuk mengurangi *routing overhead* (Sharmin, Salma, Nazma, & Ahsan, 2010) - Meningkatkan PDR dengan membatasi jumlah *hop* dari *node source* ke *next-hop* dan prosedur diulang sampai paket sampai ke tujuan.	- Pada jaringan yang berukuran relatif kecil, membatasi jumlah hop hanya akan menambah durasi pengiriman paket ke tujuan
PNT Concept (2014)	- Mempertimbangkan kestabilan node pada masa yang akan datang	- Faktor jarak antara *next-hop node* dengan destinasi tidak

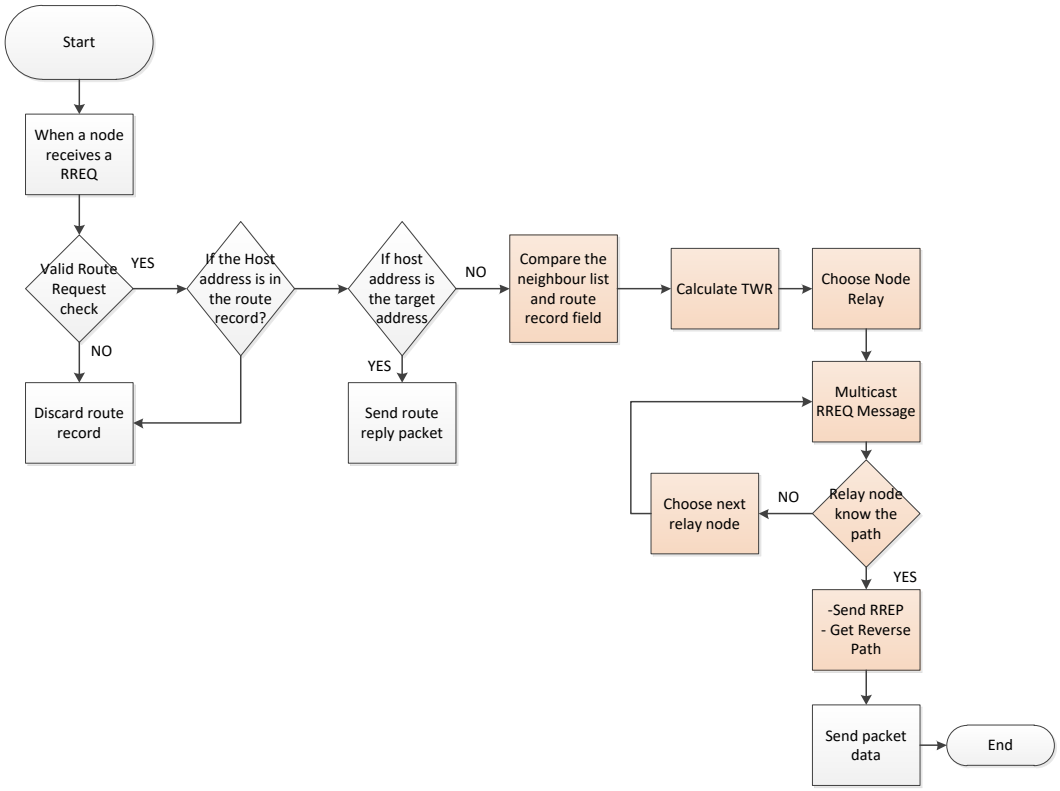
	dengan menghitung *future* TWR dan membandingkan ΔTWR dengan nilai ambang batas *W*	dipertimbangkan pada PNT Concept.
Q-DSR QoS DSR (2015)	- Memperhitungkan *parameter bandwidth* sebagai *parameter* pembantu dalam *path selection*, sehingga meningkatkan PDR dan mengurangi *packet loss.*	- Sangat terpengaruh pada *link quality* dan jarak antar *node* untuk mendapatkan *bandwidth* yang optimal dalam pengiriman paket

3.2 Desain Model Sistem

Pada tahap ini dilakukan perancangan desain model sistem serta digambarkan sistem yang akan dibangun. Pada Tabel 3.2 di bawah, dapat dilihat perbandingan dari beberapa *parameter* yang digunakan dalam metode yang diacu oleh penulis yaitu metode PNT-*Concept* dengan metode yang diajukan oleh penulis. Pada Gambar 3.2 dapat dilihat desain model system secara keseluruhan pada penelitian ini.

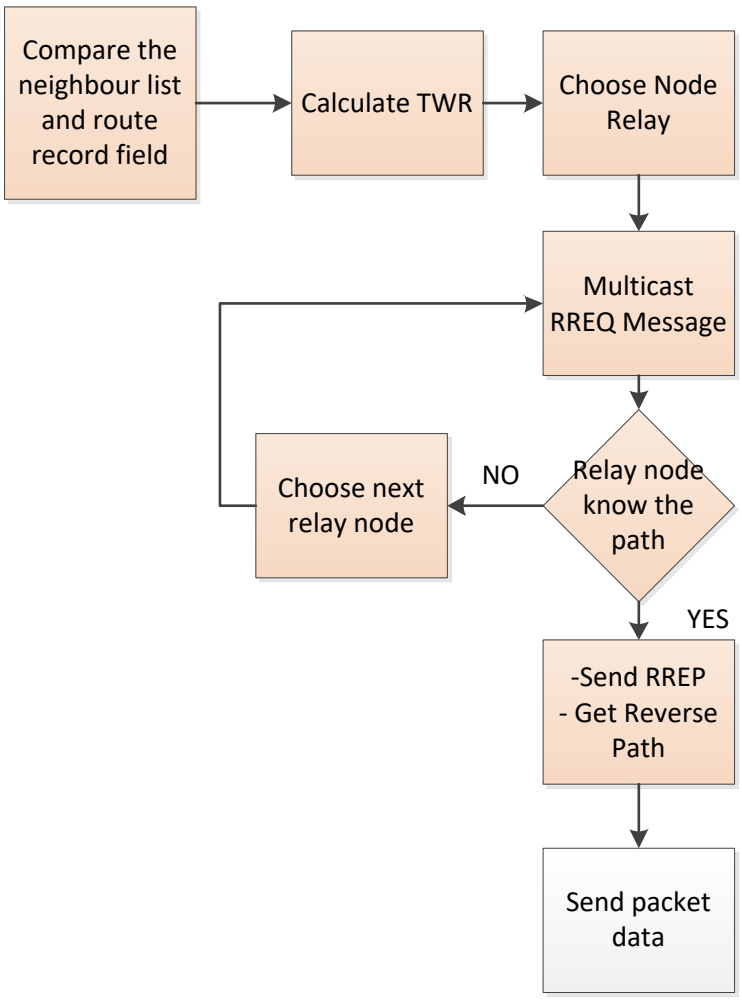
Tabel 3.2 Perbandingan PNT concept dengan metode yang diajukan

Perbedaan	PNT-Concept	Metode yang diajukan
Faktor yang mempengaruhi bobot pemilihan rute (TWR)	*Link Quality, Speed, Acceleration, Direction*	*Link Quality, Speed, Acceleration, Direction, Neighbour node*
Koefisien bobot masing masing faktor	Konstan	Berubah-ubah sesuai dengan densitas dan mobilitas *node*
Ambang batas untuk penentuan stabilitas next-hop node	Diterapkan pada perubahan total bobot (ΔTWR)	Setiap faktor memiliki ambang batas sendiri



Gambar 3.2 Desain model system

Pada Gambar 3.2 dapat dilihat desain model *system* secara keseluruhan. Pada gambar diatas, modifikasi yang diajukan pada penelitian ini dapat dilihat pada proses yang diwarnai orange. Modifikasi yang dilakukan meliputi penambahan informasi pada paket yang dikirim sebagai RREP dan RREQ, perhitungan TWR serta *future* TWR. Pada perhitungan TWR dan *future* TWR, dilakukan penyeleksian terhadap node tetangga mana yang layak dijadikan sebagai *relaying node*. Dengan adanya penyeleksian berdasarkan perhitungan TWR, diharapkan *node* yang menjadi *relaying node* adalah *node* yang paling reliable. Untuk penjelasan lebih lanjut tentang pengiriman packet berdasarkan *eligible node*, dapat dilihat pada gambar 3.3.



Gambar 3.3 Proses Pengiriman packet berdasarkan eligible node

Pada Gambar 3.3 dapat dilihat proses sebuah pengiriman *packet* berdasarkan *eligible node* sebagai *node forwarder*. Pada proses pertama, *node* yang akan mengirim paket akan melakukan pengecekan apakah *node* tersebut termasuk *neighbour list* atau tidak. Setelah itu apabila *node* tersebut merupakan *node neighbour*, dilakukan perhitungan TWR yang akan menentukan apakah *node* tersebut layak dijadikan *relaying node*. Setelah proses pemilihan *relaying node* dilakukan, proses *multicast* RREQ *message* dilakukan. Apabila *relaying node* tidak mengetahui alur pengiriman *node* ke tujuan maka dilakukan pemilihan *relay node* lainnya. Sedangkan apabila *relay node* mengetahui alur pengiriman *node* ke tujuan, maka dikirimkan RREP ke *node* sumber, beserta alur dari tujuan ke sumber disimpan dan dijadikan sebuah informasi didalam *packet* yang dikirim sebagai RREP.

3.3 Implementasi

Pada tahap ini, dilakukan implementasi terhadap desain yang telah dirancang sebelumnya. Implementasi pada penelitian ini dilakukan dengan menggunakan sistem operasi Linux dan perangkat lunak NS-2 dengan *scenario* implementasi pada jalan raya di daerah perkotaan.

Pada tahap implementasi ini, protokol yang digunakan merupakan protokol DSR yang telah dimodifikasi. Berikut akan dijelaskan keunggulan protokol DSR original dalam melakukan tugasnya. Routing protokol DSR memiliki beberapa keunggulan sebagai protokol yang bersifat reaktif. Keuntungan dari routing protokol DSR adalah *Route* ke destinasi hanya dibentuk apabila dibutuhkan. Hal ini akan mengurangi jumlah traffic didalam jaringan yang dapat mengurangi network congestion.

Keuntungan lainnya adalah *node* perantara menggunakan *cache* yang berisi informasi routing sehingga mencegah *routing overhead.* Informasi routing yang terdapat pada protokol DSR hanya berisi tentang rute node dari sumber ke tujuan, yang digunakan sebagai acuan rute untuk mengirimkan paket dari sumber ketujuan maupun untuk mendapatkan reverse path dari tujuan ke sumber node. Oleh sebab itu pada penelitian ini dilakukan penambahan informasi pada routing cache *agar pengiriman paket data dapat berjalan secara lebih efektif.*

Pada protokol DSR terdapat beberapa kekurangan yang dapat mempengaruhi *packet delivery ratio* pada protokol ini. Hal ini meliputi mekanisme *routing* tidak dapat memperbaiki *broken link* secara local. Pada saat *broken link* terjadi, *routing cache* menjadi tidak reliable pada saat pembentukan r*oute* baru. Hal ini dapat diantisipasi pada perhitungan TWR yang diajukan pada penelitian ini. Dimana dengan adanya perhitungan ini, pemilihan *node* yang paling tepat untuk melakukan tugasnya sebagai *relaying node* dapat dipilih berdasarkan perhitungan dari TWR.

3.4 Uji coba dan Simulasi

Pada tahap ini, dilakukan uji coba dari sistem yang telah dibangun. Pada penelitian ini, uji coba dilakukan berdasarkan beberapa faktor yang dapat

mempengaruhi rasio keberhasilan pengiriman packet. Hal tersebut adalah kecepatan kendaraan dan akselerasinya, arah pergerakan kendaraan, dan *link quality* antar kendaraan. Simulasi dilakukan dengan memperhatikan rasio optimal antara bobot kecepatan kendaraan dan akselerasinya, arah pergerakan kendaraan, dan *link quality* antar kendaraan serta pengaruhnya terhadap *package delivery ratio*. Rasio bobot yang optimal berpotensi untuk meningkatkan *packet delivery ratio* serta meningkatkan *link quality* pengiriman data pada *source node* ke *node* selanjutnya.

3.5 Analisa Hasil

Pada tahap ini, dilakukan analisis dari hasil uji coba yang telah dilakukan sebelumnya. Serta kesimpulan yang dapat diambil dari penelitian ini berdasarkan uji coba dan simulasi yang telah dilakukan.

(halaman ini sengaja dikosongkan)

BAB 4

IMPLEMENTASI

Pada bab ini akan dipaparkan informasi tentang hasil dari pengujian yang telah dilakukan. Pada setiap hasil yang diperoleh akan dilakukan analisa dan pembahasan terhadap hal yang mempengaruhi hasil dari suatu informasi yang didapatkan pada penelitian ini. Informasi yang didapatkan akan mempengaruhi hasil kesimpulan.

4.1 Implementasi pada Network Simulator NS-2

Untuk melakukan simulasi VANET pada *Network Simulator*, dibutuhkan sebuah file OTcl yang berisi deskripsi beserta skenario yang digunakan pada lingkungan simulasi. Pada file OTcl berisikan pengaturan terhadap setiap *node* dan beberapa event yang dapat diatur agar berjalan pada waktu tertentu. Potongan *code* file OTcl dapat dilihat pada Gambar 4.1

```
# simulasi 50 jumlah simpul DSR

# Define options
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)            CMUPriQueue                ;# interface queue type // set opt Queue/DropTail/PriQueue
set val(ll)             LL                         ;# link layer type
set val(ant)            Antenna/OmniAntenna        ;# antenna model
set val(ifqlen)         50                         ;# max packet in ifq
set val(nn)             50                         ;# number of mobilenodes
set val(rp)             DSR                       ;# routing protocol
set val(x)              500                        ;# X dimension of topography
set val(y)              400                        ;# Y dimension of topography
set val(stop)           150                        ;# time of simulation end

set ns          [new Simulator]
set tracefd       [open testDSR.tr w]
set windowVsTime2 [open win.tr w]
set namtrace      [open testDSR.nam w]

$ns trace-all $tracefd
$ns namtrace-all-wireless $namtrace $val(x) $val(y)

# set up topography object
set topo       [new Topography]

$topo load_flatgrid $val(x) $val(y)

```

Gambar 4.1 Potongan *Code* Pengaturan *Node*

Penjelasan dari pengaturan *node* dapat dilihat pada Tabel 4.1. Pengaturan lain yang dilakukan pada file OTcl tersebut adalah lokasi penyimpanan *trace file*,

lokasi *file* mobilitas *node*, konfigurasi *source node* dan *destination node*, serta konfigurasi *event* pengiriman paket data.

Tabel 4.1 Penjelasan parameter pada file OTcl

Parameter	**Value**	**Penjelasan**
llType	LL	Standard Link Layer
mactType	Mac/802_11	Komunikasi data bersifat *wireless*
ifqType	CMUPriQueue	Memanage paket berdasarkan prioritas
ifqLen	50	Jumlah paket maksimal pada antrian
antType	Antenna/OmniAntenna	Jenis antenna yang digunakan
propType	Propagation/TwoRayGround	Tipe propagasi sinyal *wireless*
phyType	Phy/WirelessPhy	Komunikasi menggunakan *wireless*
topoInstance	$topo	Topologi yang digunakan saat skenario dijalankan
agentTrace	ON	Pencatatan aktifitas pada agent diaktifkan
RouterTrace	ON	Pencatatan aktifitas pada routing protokol diaktifkan

Skenario simulasi dijalankan dengan perintah pada Gambar 4.2. Setelah simulasi selesai, *file trace* yang berisikan hasil simulasi akan di *generate* sehingga dapat dianalisa. Isi dari *file trace* tersebut adalah catatan seluruh *event* yang terjadi pada setiap *node* dalam lingkungan simulasi. Perintah untuk menjalankan file OTcl adalah ns scenario-dsr.tcl.

4.2 Implementasi PNT pada protokol DSR

Pada penelitian ini, protokol yang digunakan adalah DSR. Modifikasi yang dilakukan pada protokol DSR di NS2 adalah sebagai berikut:

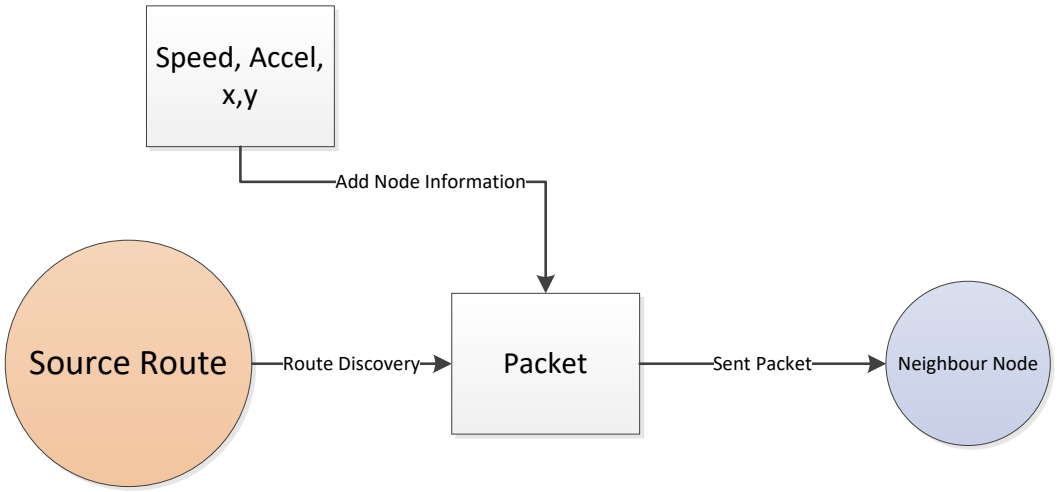
- Penambahan atribut kelas pada DSR
- Struktur *Neighbour Cache*
- Penanganan paket RREP dan RREQ

Kode yang berisi implementasi dari protokol DSR pada NS2 versi 2.35 dapat ditemukan pada direktori ns2/dsr. Daftar kode yang akan dimodifikasi adalah sebagai berikut:

- Dsragent.cc untuk modifikasi penanganan RREQ dan RREP
- Requesttable.h untuk mengubah struktur *neighbour node*
- Dsragent.h untuk penambahan atribut kelas pada protokol DSR

4.2.1 Modifikasi Struktur Paket Hello

Modifikasi dilakukan pada hdr_sr.h dalam struct route_reply. Terdapat tiga parameter yang perlu ditambahkan pada struktur paket hello yang terdapat pada routing protokol DSR. Parameter tersebut adalah kecepatan, akselerasi, koordinat x dan koordinat y yang direpresentasikan sebagai sr_speed, sr_accel, sr_x, dan sr_y pada *struct* route_reply. Pada Gambar 4.2 akan diilustrasikan penambahan parameter pada paket hello saat *source node* melakukan *route discovery*.



Gambar 4.2 Ilustrasi penambahan parameter pada packet hello dalam proses route discovery

4.2.2 Modifikasi Class DSR

Terdapat beberapa informasi penting yang didapatkan dari objek MobileNode yaitu Position_update_time yang berfungsi untuk mencatat pada detik berapa terjadi pergerakan node. Berdasarkan informasi bergeraknya *node* yang diberikan oleh Position_update_time, perhitungan akselerasi dan kecepatan dapat dilakukan. Oleh karena itu dibutuhkan penambahan atribut lastUpdateTime,

lastSpeed, dan lastAccel pada class DSR. Potongan code modifikasi class DSR dapat dilihat pada Gambar 4.3.

```
176         nsaddr_t     index; //ip address
177         u_int32_t    seqno; //Sequence number
178         int          bid;   //BroadcastID
179
180         double lastUpdateTime;
181         double lastAccel;
182         double lastSpeed;
```

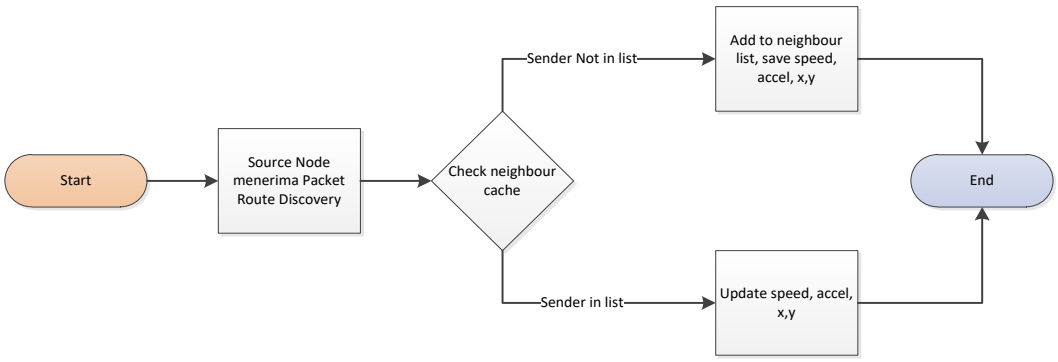
Gambar 4.3 Penambahan atribut pada class DSR

4.2.3 Modifikasi *Neighbour Cache*

Informasi pada Neighbour Cache akan diperbaharui setiap kali menerima paket HELLO. Pada fungsi add_route dalam mobicache.cc dapat dilihat isi paket yang menyimpan isi *cache* (Hogan, 2010). Untuk melihat *neighbour cache* diperlukan statement yang menampilkan seluruh rute yang akan dilalui setiap fungsi add_route dipanggil. Pada fungsi add_route dapat ditambahkan atribut sr_speed, sr_accel, sr_x, dan sr_y yang dapat menyimpan informasi kecepatan, akselerasi dan koordinat x dan y.

4.2.4 Modifikasi Proses Penerimaan paket Route Discovery

Pada saat *source node* menerima paket saat *Route discovery*, dilakukan pengecekan terhadap *neighbour cache*. Apabila pengirim paket tersebut tidak ada didalam *list*, maka *node* tersebut ditambahkan kedalam *list* dan informasi kecepatan, akselerasi dan koordinatnya akan disimpan. Apabila pengirim paket tersebut sudah ada didalam list, maka informasi kecepatan, akselerasi dan koordinat pada *cache* akan diperbaharui. Pada Gambar 4.4 dapat dilihat ilustrasi proses penerimaan paket saat *route discovery* dilakukan.



Gambar 4.4 Proses penerimaan paket dalam route discovery

4.2.5 Modifikasi Proses Pengiriman paket *Route Discovery*

Pada saat *source node* melakukan proses pengiriman paket untuk melakukan *route discovery*, ditambahkan informasi yang berisi kecepatan, akselerasi, dan koordinat dari *source node*. Struktur paket yang telah dimodifikasi tersebut dapat ditambahkan pada fungsi sendOutRtReq pada dsragent.cc. Pada Gambar 4.5 dapat dilihat potongan code proses pengiriman paket route discovery.

```
1565  void
1566  DSRAgent::sendOutRtReq(SRPacket &p, int max_prop)
1567    // turn p into a route request and launch it, max_prop of request is
1568    // set as specified
1569    // p.pkt is freed or handed off
1570  {
1571       //speed accel posx y
1572      MobileNode *iNode;
1573      iNode = (MobileNode *) (Node::get_node_by_address(index));
1574      double iSpeed = ((MobileNode *) iNode)->speed();
1575      double now = ((MobileNode *) iNode)->getUpdateTime();
1576      double posX = iNode->X();
1577      double posY = iNode->Y();
1578
1579      hdr_sr *srh =  hdr_sr::access(p.pkt);
1580      assert(srh->valid());
1581      //printf("sendoutroutereq\n");
1582      srh->route_request() = 1;
1583      srh->rtreq_seq() = route_request_num++;
1584      srh->max_propagation() = max_prop;
1585      p.route.reset();
1586      p.route.appendToPath(net_id);
1587
1588      srh->sr_speed = iSpeed;
1589      if (now - lastUpdateTime == 0) {
1590      srh->sr_accel = lastAccel;
1591      }
1592      else {
1593      srh->sr_accel = iSpeed / (now - lastUpdateTime); // a = v/t
1594      lastAccel    = sr->sr_accel;
1595      }
1596
1597      srh->sr_x     = posX;
1598      srh->sr_y     = posY;
1599
1600      Scheduler::instance().schedule(target_, p, 0.0);
1601
1602      // Update its latest update time
1603      lastUpdateTime = now;
1604  }
```

Gambar 4.5 Proses pengiriman paket *route discovery*

4.2.6 Implementasi Perhitungan TWR dan Future TWR

Perhitungan TWR membutuhkan informasi kecepatan, akselerasi, dan koordinat dari *source node* dan *destination*. Informasi tersebut dapat dilihat pada fungsi add_route dalam mobicache.cc. Sebelum perhitungan TWR dilakukan, telebih dahulu dilakukan inisialisasi terhadap nilai *threshold* W, faktor pengali

kecepatan, percepatan, jarak dan *link quality* dari *source node*. Potongan code implementasi TWR dapat dilihat pada Gambar 4.6

```
// 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));

// radius between this node and neighbor
// minimum radius -> min(√(i x − j x )^2 + (i y − j y )^2 ; r)
double radius = std::min(
  sqrt(pow((nb->nb_x - posX), 2) + pow((nb->nb_y - posY), 2)), (double) maxTxRange);

double quality = 1.0 / (1.0 - (radius / ((double) maxTxRange + 1.0)));

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 × |θ n − θ d | + f q × Q
double TWR = modSpeed + modAccel + modDistance + modQuality;
```

Gambar 4.6 Potongan code implementasi TWR

Pada node tujuan, nilai kecepatan dan percepatan adalah nol karena node tujuan bersifat *stationary*. Node tujuan hanya memiliki koordinat yang dituliskan langsung pada kode OTcl.

Apabila informasi pada *source node*, *neighbour node*, serta *destination node* telah didapatkan, maka dilakukan perhitungan TWR. Perhitungan TWR dilakukan dengan menghitung jarak dari *next hop node* dengan *destination node*. Setelah itu, *link quality* antara *source node* dengan *next hop node* dihitung dengan formula indeks stabilitas. Kemudian dilakukan perhitungan terhadap kecepatan, akselerasi, jarak, dan *link quality* dikalikan dengan faktor pengali dan dijumlahkan sehingga menghasilkan nilai TWR.

Perhitungan *future* TWR dilakukan dengan memanfaatkan informasi yang telah didapatkan sebelumnya pada perhitungan TWR. Hal ini dilakukan dengan melakukan perhitungan gerak lurus untuk memprediksi pergerakan *node* berdasarkan koordinat *node* tersebut. Pada Gambar 4.7 dapat dilihat potongan code Future TWR.

Setelah TWR dan *Future* TWR didapatkan, maka kedua nilai tersebut akan disimpan dalam *vector* yang nilainya akan diklasifikasikan seperti pada Tabel 2.1. Alamat dari setiap node yang memenuhi kriteria *relay* akan disimpan pada *array*

bertipe nsaddr_t. untuk lebih lengkapnya, implementasi perhitungan TWR dan *Future* TWR dapat dilihat pada Lampiran 3 dan 4.

```
798     // Future speed v' = v + a × t
799     double nb_speedFuture = nb->nb_speed + (nb->nb_accel * timeModifier);
800
801     // Future position will be used to calc future direction
802     // Formula: x' = x + v0 t × 0.5at^2
803     // Future neighbor position
804     double nb_xFuture = nb->nb_x +
805                 (nb->nb_speed * timeModifier)
806                   + (0.5 * nb->nb_accel * timeModifier * timeModifier);
807     double nb_yFuture = nb->nb_y +
808                 (nb->nb_speed * timeModifier)
809                   + (0.5 * nb->nb_accel * timeModifier * timeModifier);
810
811     // Future this_node position
812     double iXFuture = posX +
813                 (iSpeed * timeModifier)
814                   + (0.5 * iAccel * timeModifier * timeModifier);
815     double iYFuture = posY +
816                 (iSpeed * timeModifier)
817                   + (0.5 * iAccel * timeModifier * timeModifier);
```

Gambar 4.7 Potongan *code* implementasi *Future* TWR

4.2.7 Modifikasi Proses Pengiriman RREQ

Pada saat *source node* akan melakukan proses pengiriman RREQ, terlebih dahulu dilakukan perhitungan TWR dan Future TWR seperti yang telah dijelaskan sebelumnya. Pada Gambar 4.8 dibawah ditunjukkan bagaimana proses pengiriman RREQ saat *source node* melakukan *route discovery* dalam bentuk *pseudocode.*

```
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 packets;
fill up RREQ fields;
rqlist_length<- list.size();
rqrelay_node_set<-list;
broadcast rq;
```

Gambar 4.8 *Pseudocode* modifikasi pemrosesan RREQ yang akan dikirim

Setelah *route discovery* dilakukan dan *relay node list* telah terbentuk. Paket RREQ akan dikirim sesuai dengan *relay node list* dengan cara *broadcast*.

Ketika *neighbour node* menerima paket RREQ, perlu dilakukan pengecekan apakah *node* tersebut ada didalam *relay node list*. Oleh karena itu perlu ditambahkan *array* rq_eligible_nodes yang bertugas untuk menyimpan *relay node list.* Apabila *node* tersebut merupakan *node* yang termasuk didalam *relay node list* maka proses selanjutnya akan dilakukan. apabila *node* tersebut tidak termasuk didalam *relay node list*, maka paket RREQ akan di drop.

4.2.8 Modifikasi Proses Penerimaan RREQ

Ketika paket RREQ diterima oleh suatu *node*, perlu dilakukan beberapa pengecekan seperti apakah *node* tersebut merupakan *source node*, apakah *node* tersebut merupakan *node* tujuan, apakah *node* tersebut sudah pernah menerima RREQ, dan lain-lain. Oleh karena itu dibutuhkan kondisi tambahan yaitu apabila node tersebut merupakan *node* yang termasuk didalam *relay node list* maka proses selanjutnya akan dilakukan. apabila *node* tersebut tidak termasuk didalam *relay node list*, maka paket RREQ akan di drop. Pada Gambar 4.9 dibawah ditunjukkan bagaimana proses penerimaan RREQ beserta kondisi tambahan yang telah dijelaskan diatas dalam bentuk *pseudocode*.

```
Input: rreq
.
.
index <- address of this node;
if index is not in rreqrelay_node_set then
    drop packet;
else
    if rreqdst 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;
        rreqrelay_node_set<-new_list;
    end
end
```

Gambar 4.9 *Pseudocode* proses penerimaan RREQ

4.3 Implementasi Metrik Analisis

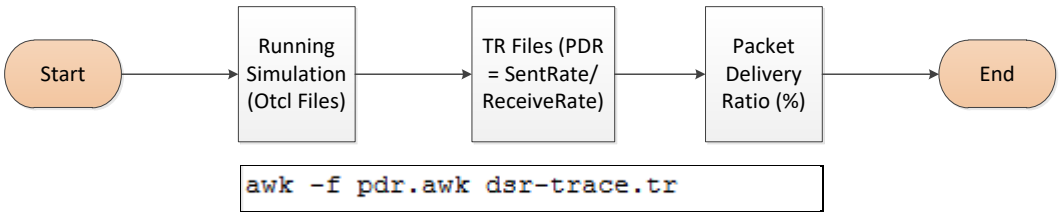
Simulasi dari skenario pada NS-2 akan menghasilkan sebuah trace file yang digunakan sebagai bahan analisis performa dari metode yang telah diajukan.

Terdapat tiga metrik yang akan menjadi parameter analisis pada penelitian ini yaitu *packet delivery ratio*, *average end to end delay*, dan *routing overhead*.

4.3.1 Implementasi *Packet Delivery Ratio*

Packet delivery ratio merupakan rasio keberhasilan pengiriman packet yang dapat dihitung dari perbandingan total paket yang dikirim dengan total paket yang diterima. Informasi didapat dengan menyaring informasi dalam trace file yang mengandung string AGT karena *event* ini berhubungan dengan paket data.

Pada Gambar 4.10 adalah contoh perintah untuk memanggil skrip awk *packet delivery ratio* untuk menganalisis *trace file* beserta proses yang terjadi dalam implementasi *packet delivery ratio*.



Gambar 4.10 Proses Implementasi *Packet delivery ratio* beserta perintah untuk menjalankan skrip awk *packet delivery ratio*

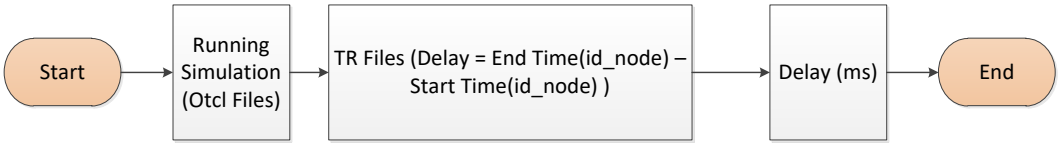
Pada Gambar 4.10 dapat dilihat bahwa pada implementasi *packet delivery ratio* berjalan, pertama-tama hal yang dilakukan adalah menjalankan simulasi berdasarkan skenario penelitian yang telah dibuat. Skenario yang dibuat pada penelitian ini berbentuk OTcl files yang berisi konfigurasi skenario penelitian ini. Setelah files OTcl dijalankan, akan dihasilkan sebuah *trace file* yang berisi infomasi yang terjadi pada saat skenario dijalankan. Untuk mendapatkan *packet delivery ratio* dalam *trace files*, dibutuhkan sebuah *file* awk yang berisi formula *packet delivery ratio*. Setelah awk dijalankan maka akan menghasilkan *packet delivery ratio* pada skenario yang telah dijalankan.

4.3.2 Implementasi *Average End to End Delay*

Average end to end delay merupakan waktu jeda antara *event* pengiriman paket dan penerimaan paket pada suatu *node*. Dalam *end to end delay* terdapat beberapa hal yang harus diperhatikan yaitu *event* terjadinya pengiriman dan penerimaan paket, waktu terjadinya *event*, informasi komunikasi paket pada suatu

layer, ID paket dan tipe paket tersebut. *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.

Pada Gambar 4.11 adalah contoh perintah untuk memanggil skrip awk *delay* untuk menganalisis *trace file* beserta proses yang terjadi dalam implementasi *average end to end delay*.




```
awk -f e2edelay.awk dsr-trace.tr
```

Gambar 4.11 Proses Implementasi *Average end to end delay* beserta perintah untuk menjalankan skrip awk *end to end delay*

Pada Gambar 4.11 dapat dilihat bahwa pada implementasi *average end to end delay* berjalan, pertama-tama hal yang dilakukan adalah menjalankan simulasi berdasarkan skenario penelitian yang telah dibuat. Skenario yang dibuat pada penelitian ini berbentuk OTcl files yang berisi konfigurasi skenario penelitian ini. Setelah files OTcl dijalankan, akan dihasilkan sebuah *trace file* yang berisi infomasi yang terjadi pada saat skenario dijalankan. Untuk mendapatkan *average end to end delay* dalam *trace files*, dibutuhkan sebuah *file* awk yang berisi formula *average end to end delay*. Setelah awk dijalankan maka akan menghasilkan *average end to end delay* pada skenario yang telah dijalankan.

4.3.3 Implementasi Routing Overhead

Routing Overhead didapatkan dengan cara menyaring setiap baris pada trace file yang mengandung string REQUEST, REPLY, dan ERROR. Setiap ditemukan string tersebut, dilakukan increment untuk menghitung jumlah paket routing yang tersebar di jaringan.

Pada Gambar 4.12 adalah contoh perintah untuk memanggil skrip awk *routing overhead* untuk menganalisis *trace file* beserta proses yang terjadi dalam implementasi *routing overhead*.


```
awk -f ro.awk dsr-trace.tr
```

Gambar 4.12 Proses Implementasi *Routing Overhead* beserta perintah untuk menjalankan skrip awk *routing overhead*

Pada Gambar 4.12 dapat dilihat bahwa pada implementasi *routing overhead* berjalan, pertama-tama hal yang dilakukan adalah menjalankan simulasi berdasarkan skenario penelitian yang telah dibuat. Skenario yang dibuat pada penelitian ini berbentuk OTcl files yang berisi konfigurasi skenario penelitian ini. Setelah files OTcl dijalankan, akan dihasilkan sebuah *trace file* yang berisi infomasi yang terjadi pada saat skenario dijalankan. Untuk mendapatkan *routing overhead* dalam *trace files*, dibutuhkan sebuah *file* awk yang berisi formula *routing overhead*. Setelah awk dijalankan maka akan menghasilkan *routing overhead* pada skenario yang telah dijalankan.

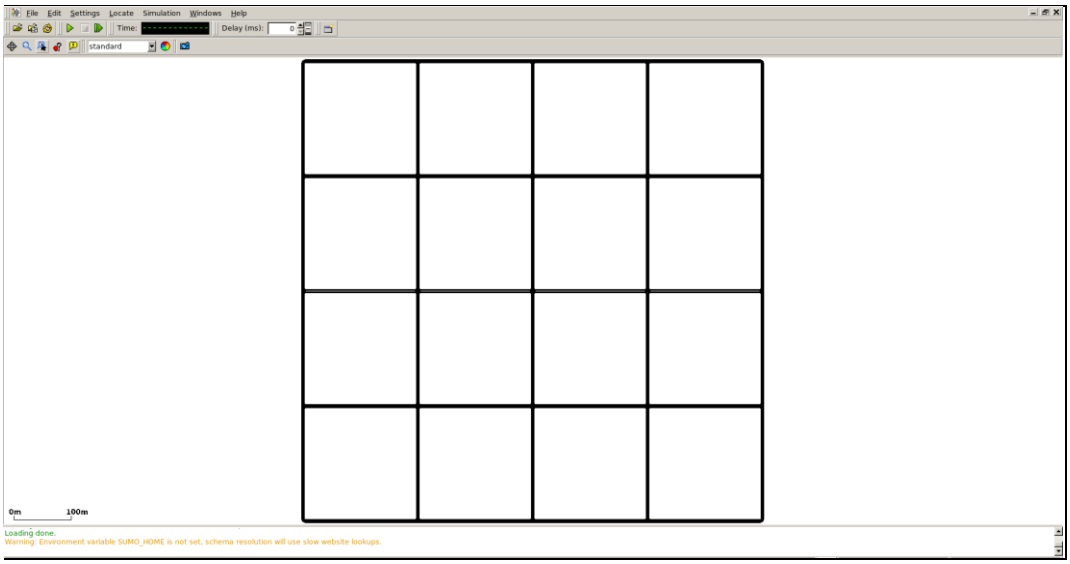
4.4 Implementasi Skenario Grid

Skenario Grid dibuat melalui tool netgenerate dari SUMO. Skenario grid dibuat dengan panjang jalan 200m dan luas peta 800 m x 800 m. Kecepatan kendaraan diatur sebesar 15 m/s. sedangkan jumlah titik persimpangan antara jalan vertical sebanyak 5 titik x 5 titik. Pada Gambar 4.13 dapat dilihat perintah untuk membuat skenario grid.

```
netgenerate --grid --grid.number=5 --grid.length=200 -default.speed=15
--tls.guess=1 --output-file=map.net.xml
```

Gambar 4.13 Perintah untuk membuat skenario grid

Gambar peta yang telah dibuat dengan netgenerate dapat dilihat pada Gambar 4.14



Gambar 4.14 Peta grid hasil perintah netgenerate

Setelah peta terbentuk, selanjutnya dilakukan pembuatan titik awal dan titik tujuan secara random dengan menggunakan randomTrips.py seperti pada Gambar 4.15

```
phyton $SUMO_HOME/tools/randomTrips.py --seed=$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 4.15 Perintah untuk mengenerate titik awal dan titik tujuan kendaraan

Pada tahap selanjutnya, dibuat rute yang akan dilalui oleh kendaraan berdasarkan peta yang telah dibuat seperti pada Gambar 4.14. Perintah untuk membuat rute kendaraan dapat dilihat pada Gambar 4.16

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

Gambar 4.16 Perintah untuk membuat rute kendaraan

Pada tahap selanjutnya, dibuat file .sumocfg yang digunakan untuk meninisialisasi lokasi file net.xml dan trips.xml. file .sumocfg diletakkan di direktori yang sama dengan net.xml dan trip.xml. Setelah itu, file .sumocfg dapat

dijalankan pada SUMO simulator. Isi dari .sumocfg dapat dilihat pada Gambar 4.17

```
<?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>
<begin value ="0"/>
```

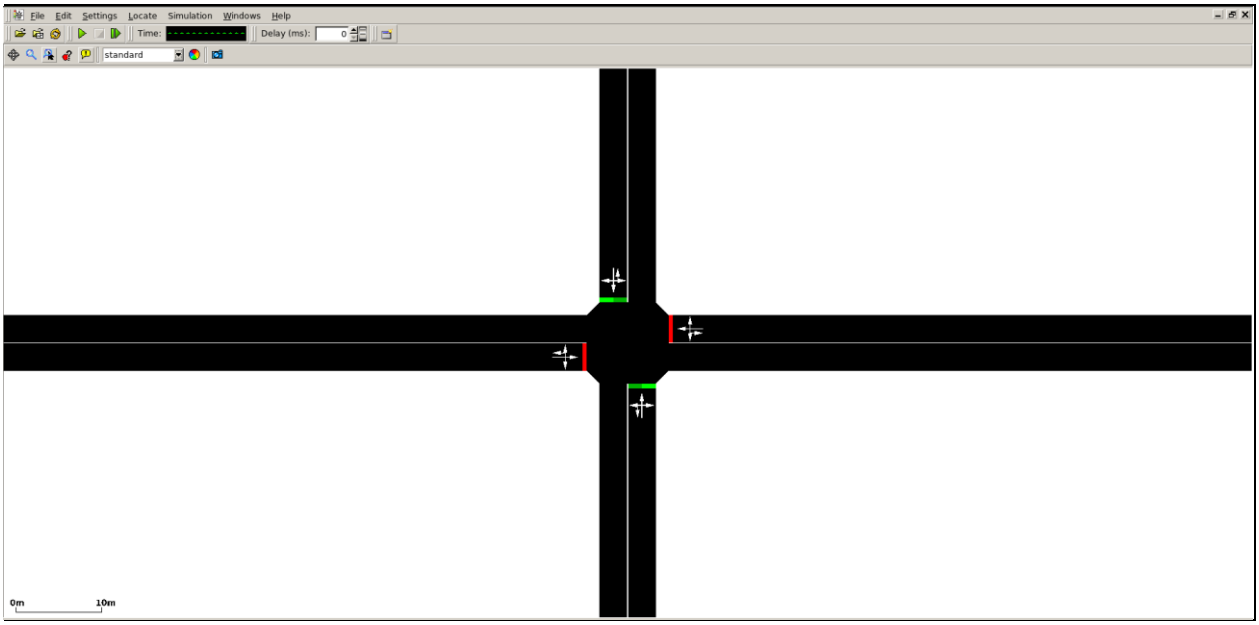
Gambar 4.17 isi dari .sumocfg

Pada tahap selanjutnya dilakukan simulasi lalu lintas yang dapat dilakukan dengan perintah sumo –c file.sumocfg –fcd-output simulation-result.xml.

Agar dapat digunakan di NS-2, keluaran perintah SUMO harus dikonversi ke format yang dapat dipahami oleh NS-2 melalui perintah pada Gambar 4.18 Pada Gambar 4.19 dapat dilihat cuplikan tampilan pada SUMO GUI beserta simulasi lalu lintas yang telah dibuat sebelumnya.

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

Gambar 4.18 Perintah untuk mengkonversi keluaran dari SUMO



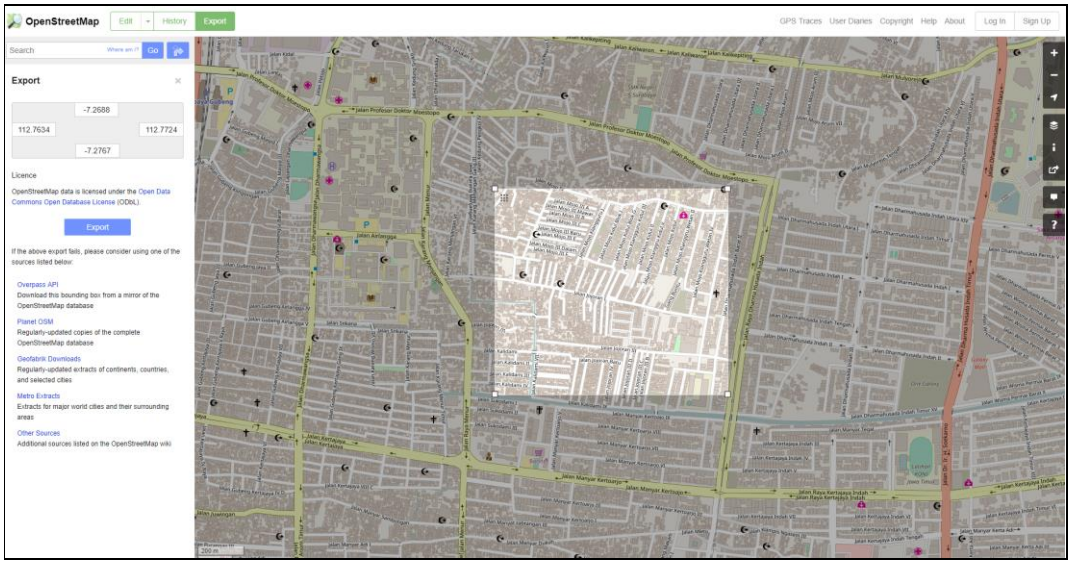
Gambar 4.19 Visualisasi tampilan pada SUMO GUI

4.5 Implementasi Skenario Riil

Simulasi Skenario riil diambil dari OpenStreetMap. Peta diambil dengan cara membuat area seleksi pada suatu wilayah kemudian diekspor sehingga menghasilkan file osm. Bagian peta skenario riil yang digunakan pada penelitian ini adalah sebuah peta wilayah kota Surabaya. Contoh proses pengambilan peta dari OpenStreetMap dapat dilihat pada Gambar 4.20.

Peta yang telah diekspor dari OpenStreetMap kemudian disunting melalui program JOSM. Tujuan penyuntingan ini adalah untuk menghapus jalan yang tidak digunakan, dan memperbaiki jalan buntu sehingga kepadatan jalan tetap stabil dan tidak ada area yang jarang dikunjungi kendaraan

Setelah proses penyuntingan peta selesai, peta tersebut dikonversi ke dalam format .net.xml menggunakan tool netconvert. Perintah konversi dapat dilihat pada Gambar 4.21.



Gambar 4.20 Proses pengambilan area simulasi riil pada Open Street Map

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

Gambar 4.21 Perintah konversi peta skenario berbentuk osm files ke xml files

(halaman ini sengaja dikosongkan)

BAB 5
UJI COBA DAN EVALUASI

Pada Bab ini akan membahas 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 penelitian ini dapat dilihat pada Tabel 5.1.

Tabel 5.1 Spesifikasi Perangkat Keras

Komponen	**Spesifikasi**
CPU	Intel Core i5-6600 @3.30GHz (4 CPUs) ~ 3.30GHz
OS	Ubuntu 64bit 14.04
RAM	8 GB DDR4
Storage	1 TB

Berikut adalah beberapa perangkat lunak beserta versi yang digunakan dalam penelitian ini:

- NS-2 versi 2.35 untuk simulasi skenario VANETs.
- JOSM versi 12450 untuk penyuntingan peta Open Street Map.
- SUMO versi 0.28.0 untuk pembuatan skenario mobilitas VANETs.

Parameter lingkungan uji coba yang digunakan pada NS-2 2.35 dalam penelitian ini dapat dilihat pada Tabel 5.2.

Dalam penelitian ini, pengujian dilakukan dengan menjalankan skenario melalui NS-2. Skenario yang telah dijalankan menghasilkan sebuah *tracefile* yang akan dianalisis dengan menggunakan skrip AWK.

Tabel 5.2 Parameter Simulasi VANETs

Number of nodes	100, 120, 140, 160
Channel type	wireless
Routing Protocol	DSR
Mac Type	802.11p
Source of traffic	Constant bit rate (CBR)
Package size	512 bytes
Network Simulator	NS-2
Simulation environment	Urban (urban)
Propagation models	Two Ray Ground
Model mobility generator	SUMO (Krajzewicz, Jakob, Michael, & Laura, 2012)
Simulation Area	800 m x 800 m Grid, 800m x 600m Real
Source & Destination node	Tetap (Stationary)
Parameter Factor	f_s=15, f_a = 10, f_d=20, f_q=30, f_n= 19, W=100.
Speed	15 ms, 20 ms

5.2 Hasil Uji Coba

Hasil uji coba pada penelitian ini dibagi menjadi dua bagian yaitu skenario grid dan skenario riil. Hasil uji coba pada penelitian ini dapat dilihat pada sub-bab dibawah ini.

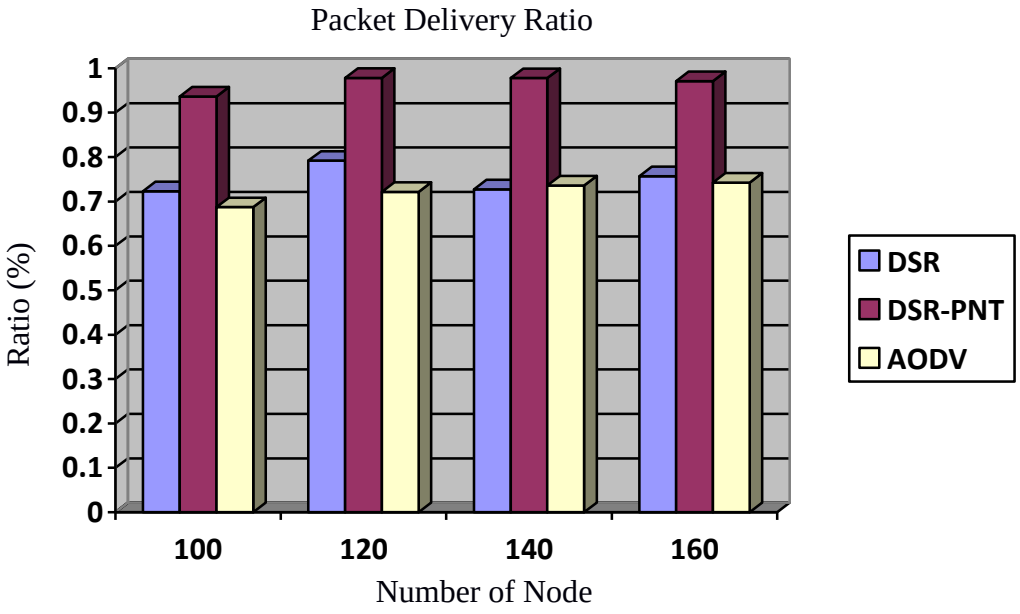
5.2.1 Hasil Uji Coba Skenario Grid.

Uji coba skenario grid pada penelitian ini dilakukan sebanyak 10 kali dengan skenario mobilitas *node random* pada peta grid berukuran 1000 m x1000 m. Jumlah node pada penelitian ini adalah 100, 120, 140, dan 160. 160 node merupakan jumlah node maksimal yang dapat digunakan pada skenario ini.

Tabel 5.3 Packet Delivery Ratio Skenario Grid speed 15ms

Jumlah Node	AODV	DSR	DSR-PNT	Perbedaan (DSR vs DSR-PNT)
100	0.6872	0.7231	0.9364	0.2133
120	0.7214	0.7918	0.9783	0.1865
140	0.7367	0.7274	0.9779	0.2505
160	0.7423	0.7569	0.9711	0.2142

Berdasarkan data pada Tabel 5.3 diatas, dibuat grafik yang merepresentasikan hasil perhitungan packet delivery ratio yang dapat dilihat pada Gambar 5.1



Gambar 5.1 Packet Delivery Ratio AODV, DSR & DSR-PNT Skenario Grid speed 15ms

Pada Gambar 5.1 dapat dilihat bahwa modifikasi DSR PNT terdapat peningkatan *packet delivery ratio* sebesar 0.2133 pada node 100, 0.1865 pada node 120, 0.2505 pada node 140, 0.2142 pada node 160. Dapat disimpulkan bahwa pada luas area 1000x1000 protokol DSR original telah memiliki *packet delivery ratio* yang sudah tinggi yaitu pada *range* 70%. Pada protokol AODV, *packet delivery ratio* pada skenario penelitian ini jatuh pada range 70%. Sementara pada DSR-PNT packet delivery ratio meningkat ke *range* 90% hal ini disebabkan oleh perhitungan *future* TWR yang dapat menentukan *relay node* yang

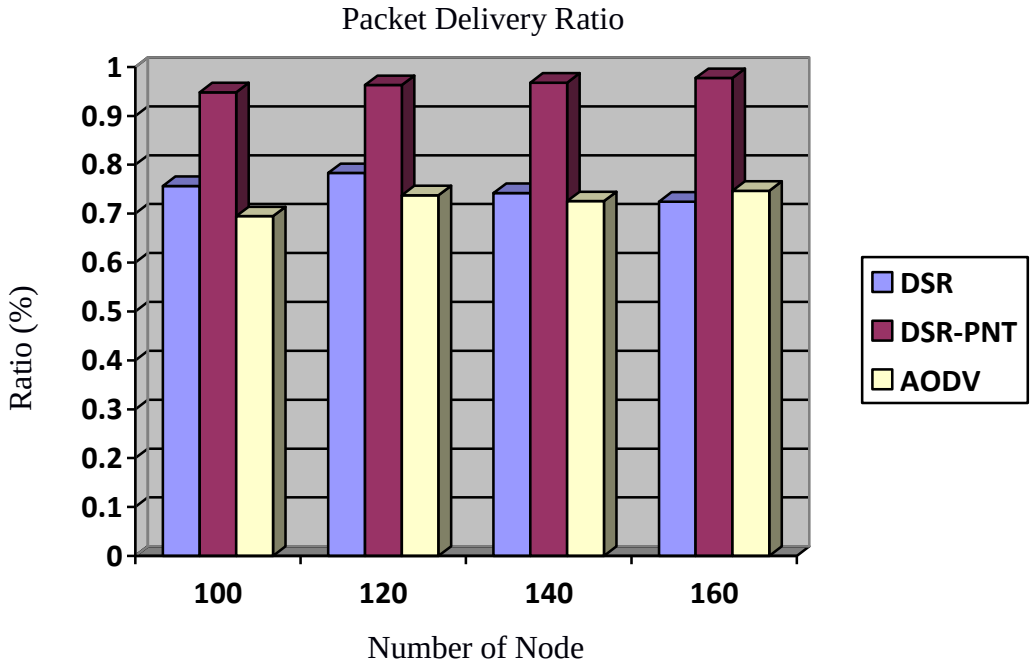
paling optimal sehingga dapat meminimalisir *route error* serta melakukan *drop packet* saat *relay node* yang tidak terdapat dalam *list* menerima RREQ.

Hal ini turut serta membantu network congestion dalam jaringan VANETs. Stabilnya *packet delivery ratio* pada *node* 160 berarti *node density* yang tinggi akan menambah jumlah *node* yang layak sebagai *relaying node* sehingga tercipta banyak alternatif rute menuju *destination* sehingga *broken link* dapat diminimalisir.

Tabel 5.4 Packet Delivery Ratio Skenario Grid speed 20ms

Jumlah Node	AODV	DSR	DSR-PNT	Perbedaan (DSR vs DSR-PNT)
100	0.6946	0.7567	0.9482	0.1915
120	0.7376	0.7830	0.9629	0.1799
140	0.7255	0.7423	0.9679	0.2256
160	0.7469	0.7246	0.9773	0.2527

Berdasarkan data pada Tabel 5.4 diatas, dibuat grafik yang merepresentasikan hasil perhitungan *packet delivery ratio* yang dapat dilihat pada Gambar 5.2



Gambar 5.2 Packet Delivery Ratio AODV, DSR & DSR-PNT Skenario Grid speed 20ms

Pada Gambar 5.2 dapat dilihat bahwa modifikasi DSR PNT terdapat peningkatan *packet delivery ratio* sebesar 0.1915 pada node 100, 0.1799 pada node 120, 0.2256 pada node 140, 0.2527 pada node 160. Dapat disimpulkan bahwa pada luas area 1000x1000 protokol DSR original telah memiliki *packet delivery ratio* yang sudah tinggi yaitu pada *range* 70%. Pada protokol AODV, *packet delivery ratio* pada skenario penelitian ini jatuh pada range 70%. Sementara pada DSR-PNT *packet delivery ratio* meningkat ke *range* 90% hal ini disebabkan oleh perhitungan future TWR yang dapat menentukan *relay node* yang paling optimal sehingga dapat meminimalisir *route error* serta melakukan drop packet saat *relay node* yang tidak terdapat dalam *list* menerima RREQ.

Hal ini turut serta membantu *network congestion* dalam jaringan VANETs. Perbedaan speed 20ms memberikan sedikit peningkatan packet delivery ratio pada protokol DSR. Stabilnya *packet delivery ratio* pada *node* 160 berarti *node density* yang tinggi akan menambah jumlah *node* yang layak sebagai *relaying node* sehingga tercipta banyak alternatif rute menuju *destination* sehingga *broken link* dapat diminimalisir.

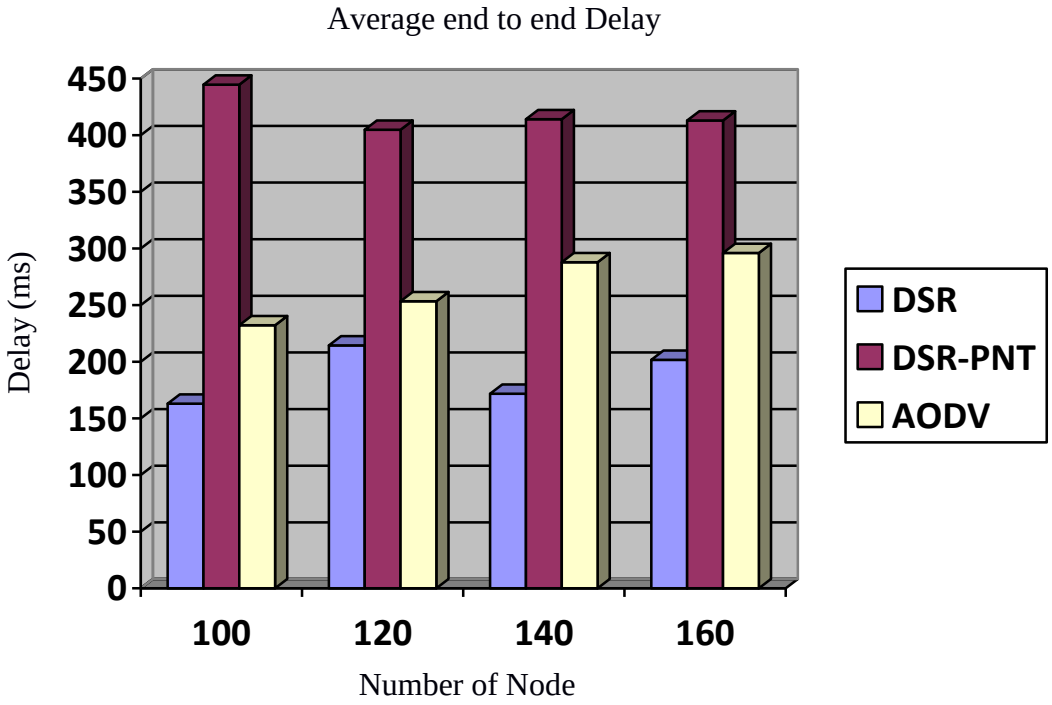
Tabel 5.5 *Average end to end delay* Skenario Grid speed 15ms

Jumlah Node	AODV	DSR	DSR-PNT	Perbedaan (DSRvsDSR-PNT)
100	232.32 ms	163.01 ms	444.70 ms	281.69 ms
120	253.45 ms	214.34 ms	404.94 ms	190.60 ms
140	287.76 ms	171.67 ms	414.14 ms	242.47 ms
160	296.17 ms	201.69 ms	413.11 ms	211.42 ms

Berdasarkan data pada Tabel 5.5 diatas, dibuat grafik yang merepresentasikan hasil perhitungan *average end to end delay* yang dapat dilihat pada Gambar 5.3.

Pada Gambar 5.3 dapat dilihat bahwa peningkatan *delay* telah terjadi pada DSR-PNT sejak node 100 sementara protokol DSR memiliki *delay* yang lebih rendah pada semua banyak *node* dalam penelitian ini. Semakin kecil jumlah *node* yang digunakan sebagai skenario, semakin kecil pula *average end to end delay*. Namun kecilnya *delay* dan jumlah *node* yang lebih sedikit tidak menjamin bahwa *packet delivery ratio* akan meningkat. Hal ini disebabkan oleh semakin sedikitnya

node yang tersedia sebagai *neighbour node*, semakin sedikit pula *node* yang memenuhi syarat sebagai *relay node*.



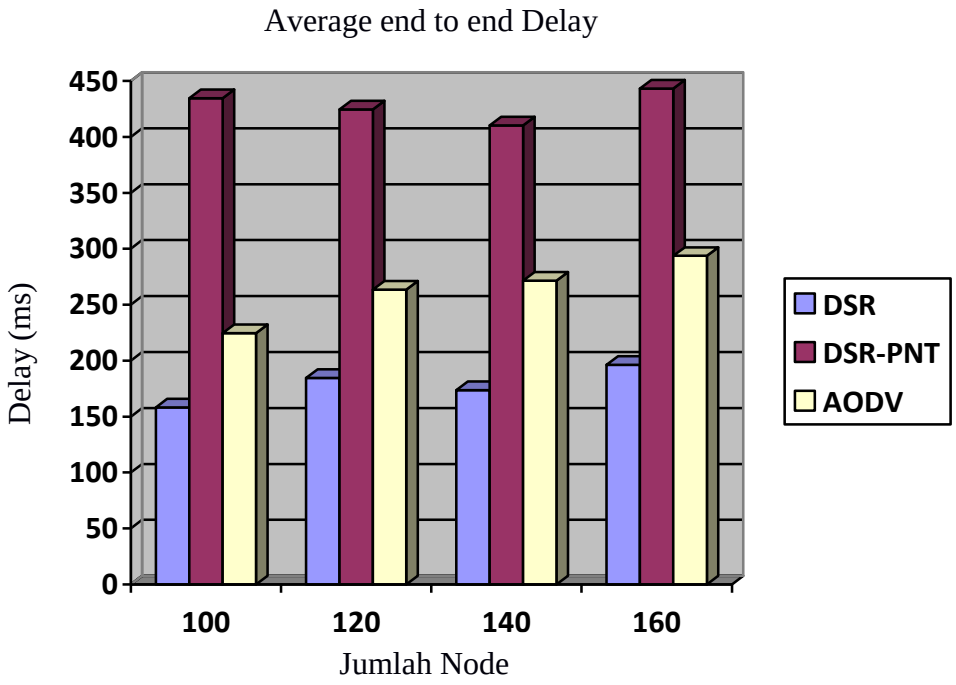
Gambar 5.3 *Average end to end delay* AODV, DSR & DSR-PNT Skenario Grid *speed* 15ms

Penambahan variasi speed pada kecepatan 15ms dan 20ms pada parameter *average end to end delay* tidak memberikan perubahan yang signifikan terhadap protokol DSR-PNT, namun memberikan sedikit perubahan pada protokol DSR dan AODV. DSR-PNT tidak terlalu terpengaruh terhadap faktor variasi kecepatan karena pemilihan *relaying node* melalui perhitungan TWR yang mengakibatkan tingginya *delay* pada DSR-PNT, sementara protokol DSR dan AODV melakukan pemilihan *relaying node* berdasarkan node yang paling cepat mengirimkan *reply* kepada *node* pengirim sehingga memiliki *delay* yang relatif rendah.

Tabel 5.6 *Average end to end delay* Skenario Grid 20ms

Jumlah Node	AODV	DSR	DSR-PNT	Perbedaan (DSRvsDSR-PNT)
100	224.67 ms	158.32 ms	432.64 ms	274.32 ms
120	263.59 ms	184.47 ms	424.43 ms	239.96 ms
140	271.42 ms	173.73 ms	410.29 ms	236.56 ms
160	293.66 ms	196.18 ms	443.12 ms	246.94 ms

Berdasarkan data pada Tabel 5.6 diatas, dibuat grafik yang merepresentasikan hasil perhitungan *average end to end delay* yang dapat dilihat pada Gambar 5.4.



Gambar 5.4 *Average end to end delay* AODV, DSR & DSR-PNT Skenario Grid *speed* 20ms

Pada Gambar 5.4 dapat dilihat bahwa peningkatan *delay* telah terjadi pada DSR-PNT sejak node 100 sementara protokol DSR memiliki *delay* yang lebih rendah pada semua banyak *node* dalam penelitian ini. Semakin kecil jumlah *node* yang digunakan sebagai skenario, semakin kecil pula *average end to end delay*. Namun kecilnya *delay* dan jumlah *node* yang lebih sedikit tidak menjamin bahwa *packet delivery ratio* akan meningkat. Hal ini disebabkan oleh semakin sedikitnya *node* yang tersedia sebagai *neighbour node*, semakin sedikit pula *node* yang memenuhi syarat sebagai *relay node*.

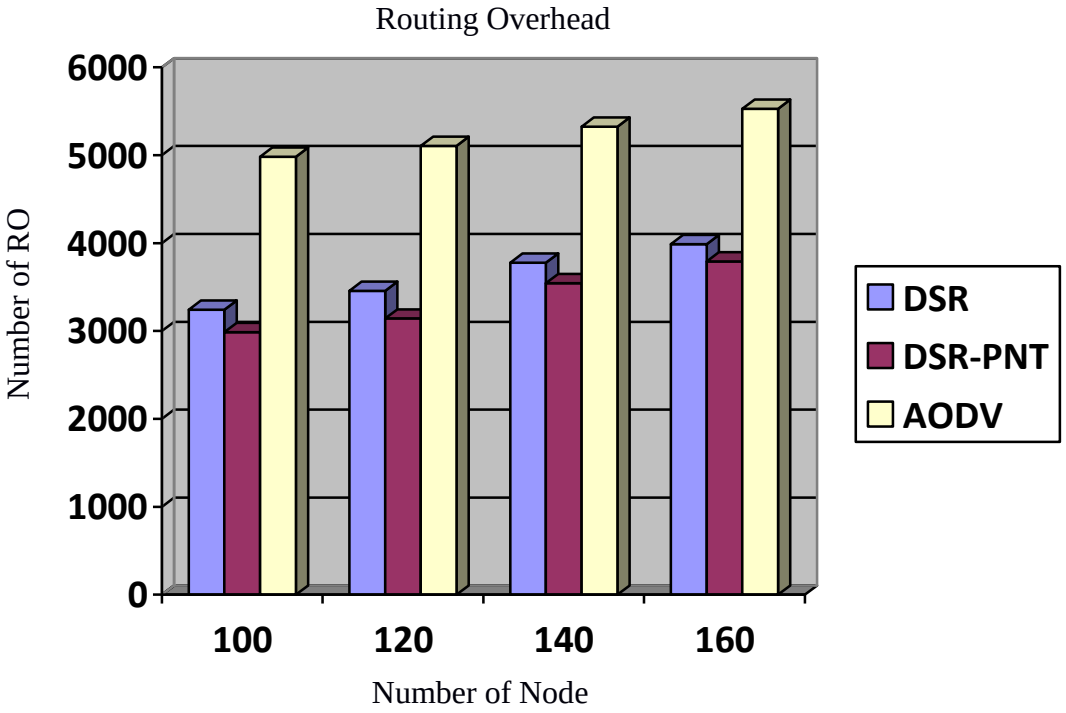
Penambahan variasi speed pada kecepatan 15ms dan 20ms pada parameter *average end to end delay* tidak memberikan perubahan yang signifikan terhadap protokol DSR-PNT, namun memberikan sedikit perubahan pada protokol DSR dan AODV. DSR-PNT tidak terlalu terpengaruh terhadap faktor variasi kecepatan karena pemilihan *relaying node* melalui perhitungan TWR yang mengakibatkan tingginya *delay* pada DSR-PNT, sementara protokol DSR dan AODV melakukan

pemilihan *relaying node* berdasarkan node yang paling cepat mengirimkan *reply* kepada *node* pengirim sehingga memiliki *delay* yang relatif rendah.

Tabel 5.7 *Routing Overhead* Skenario Grid *speed* 15ms

Jumlah Node	AODV	DSR	DSR-PNT	Perbedaan (DSR vs DSR-PNT)
100	4983	3243	2987	256
120	5107	3458	3142	316
140	5324	3776	3544	222
160	5529	3990	3793	197

Berdasarkan data yang didapat pada Tabel 5.7 maka dapat dibuat grafik yang merepresentasikan hasil perhitungan *routing overhead* yang dapat dilihat pada Gambar 5.5.



Gambar 5.5 *Routing Overhead* AODV, DSR & DSR-PNT Skenario Grid *speed* 15ms

Pada Gambar 5.5 dapat dilihat bahwa *routing overhead* dari DSR-PNT sedikit lebih unggul daripada protokol DSR. Hal ini disebabkan oleh semakin meningkatnya jumlah *node*, maka *send rate*, *receive rate* beserta *error rate* akan meningkat karena kedua metode menggunakan metode *broadcast* saat *route*

discovery. Pemilihan *node* yang tepat dalam *relaying* atau *Eligible node* yang terdapat dalam list *relay node set* membuat DSR-PNT sedikit lebih unggul. Protokol AODV memiliki *routing overhead* paling tinggi karena memiliki jumlah pengiriman paket, penerimaan paket serta paket *error* yang tinggi. Hal ini berbanding lurus dengan meningkatnya jumlah *node* dalam jaringan. Semakin tinggi jumlah *node* dalam jaringan, maka *routing overhead* AODV cenderung meningkat.

Penambahan variasi *speed* pada kecepatan 15ms dan 20ms pada parameter *routing overhead* tidak memberikan perubahan yang signifikan terhadap protokol DSR-PNT, namun memberikan sedikit perubahan pada protokol DSR dan AODV. DSR-PNT tidak terlalu terpengaruh terhadap faktor variasi kecepatan karena pemilihan *relaying node* melalui perhitungan TWR, sementara protokol DSR dan AODV melakukan pemilihan *relaying node* berdasarkan node yang paling cepat mengirimkan reply kepada node pengirim.

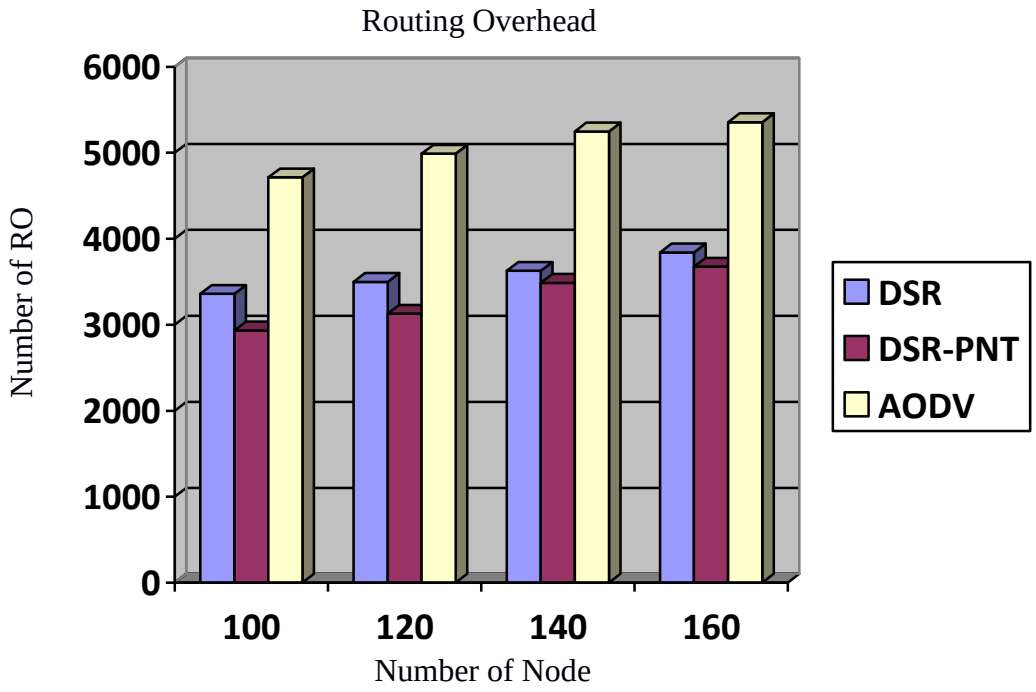
Tabel 5.8 *Routing Overhead* Skenario Grid *speed* 20ms

Jumlah Node	AODV	DSR	DSR-PNT	Perbedaan (DSR vs DSR-PNT)
100	4713	3359	2933	426
120	4989	3498	3129	369
140	5247	3627	3487	140
160	5355	3842	3676	166

Berdasarkan data yang didapat pada Tabel 5.8 maka dapat dibuat grafik yang merepresentasikan hasil perhitungan *routing overhead* yang dapat dilihat pada Gambar 5.6

Pada Gambar 5.6 dapat dilihat bahwa *routing overhead* dari DSR-PNT sedikit lebih unggul daripada protokol DSR. Hal ini disebabkan oleh semakin meningkatnya jumlah *node*, maka *send rate*, *receive rate* beserta *error rate* akan meningkat karena kedua metode menggunakan metode *broadcast* saat *route discovery*. Pemilihan node yang tepat dalam *relaying* atau *Eligible node* yang terdapat dalam *list relay node set* membuat DSR-PNT sedikit lebih unggul. Protokol AODV memiliki *routing overhead* paling tinggi karena memiliki jumlah pengiriman paket, penerimaan paket serta paket *error* yang tinggi. Hal ini

berbanding lurus dengan meningkatnya jumlah *node* dalam jaringan. Semakin tinggi jumlah *node* dalam jaringan, maka *routing overhead* AODV cenderung meningkat.



Gambar 5.6 *Routing Overhead* AODV, DSR & DSR-PNT Skenario Grid *speed* 20ms

Penambahan variasi *speed* pada kecepatan 15ms dan 20ms pada parameter *routing overhead* tidak memberikan perubahan yang signifikan terhadap protokol DSR-PNT, namun memberikan sedikit perubahan pada protokol DSR dan AODV. DSR-PNT tidak terlalu terpengaruh terhadap faktor variasi kecepatan karena pemilihan *relaying node* melalui perhitungan TWR, sementara protokol DSR dan AODV melakukan pemilihan *relaying node* berdasarkan node yang paling cepat mengirimkan reply kepada node pengirim.

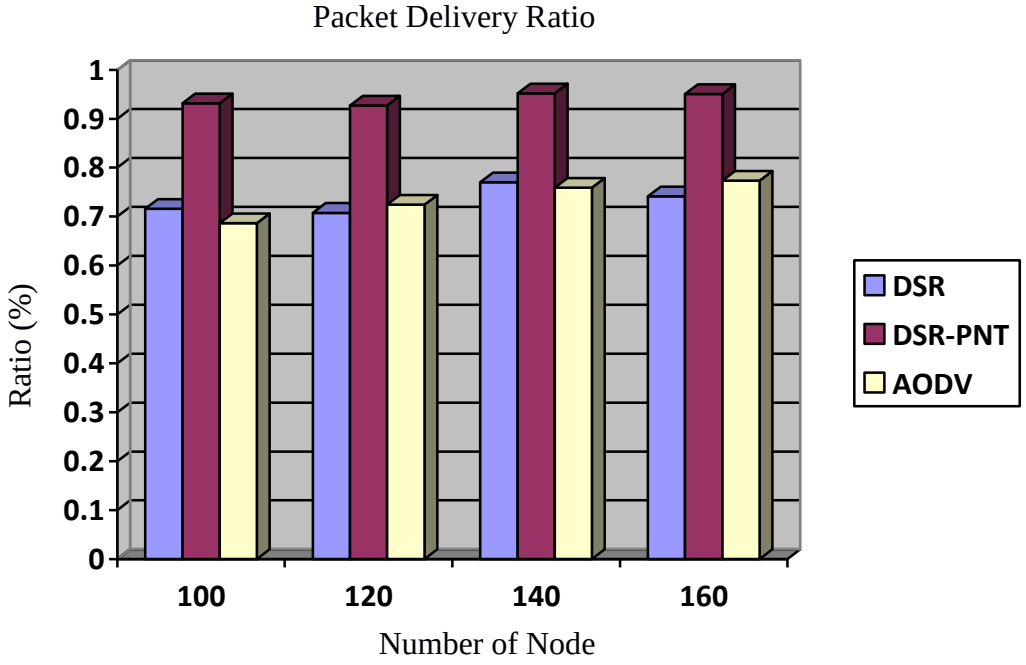
5.2.2 Hasil Uji Coba Skenario Riil.

Uji coba skenario riil pada penelitian ini dilakukan sebanyak 10 kali dengan skenario mobilitas *node random* pada peta grid berukuran 800 m x 600m. Jumlah node pada penelitian ini adalah 100, 120, 140, dan 160. 160 *node* merupakan jumlah *node* maksimal yang dapat digunakan pada skenario ini.

Tabel 5.9 *Packet Delivery Ratio* Skenario Riil speed 15ms

Jumlah Node	AODV	DSR	DSR-PNT	Perbedaan (DSRvsDSR-PNT)
100	0.6862	0.7160	0.9318	0.2158
120	0.7241	0.7077	0.9270	0.2193
140	0.7593	0.7701	0.9517	0.1816
160	0.7732	0.7414	0.9509	0.2095

Berdasarkan data pada Tabel 5.9 diatas, dibuat grafik yang merepresentasikan hasil perhitungan *packet delivery ratio* yang dapat dilihat pada Gambar 5.7.



Gambar 5.7 *Packet Delivery Ratio* AODV, DSR & DSR-PNT Skenario Riil *speed* 15ms

Pada Gambar 5.7 dapat dilihat bahwa modifikasi DSR PNT terdapat peningkatan *packet delivery ratio* sebesar 0.2158 pada node 100, 0.2193 pada *node* 120, 0,1816 pada *node* 140, 0.2095 pada *node* 160. Dapat disimpulkan bahwa pada luas area 800x600 protokol DSR original telah memiliki *packet delivery ratio* yang sudah tinggi yaitu pada *range* 70%. Pada protokol AODV, *packet delivery ratio* pada skenario penelitian ini jatuh pada range 70%. Sementara pada DSR-PNT *packet delivery ratio* meningkat ke range 90% hal ini disebabkan oleh perhitungan *future* TWR yang dapat menentukan *relay node* yang

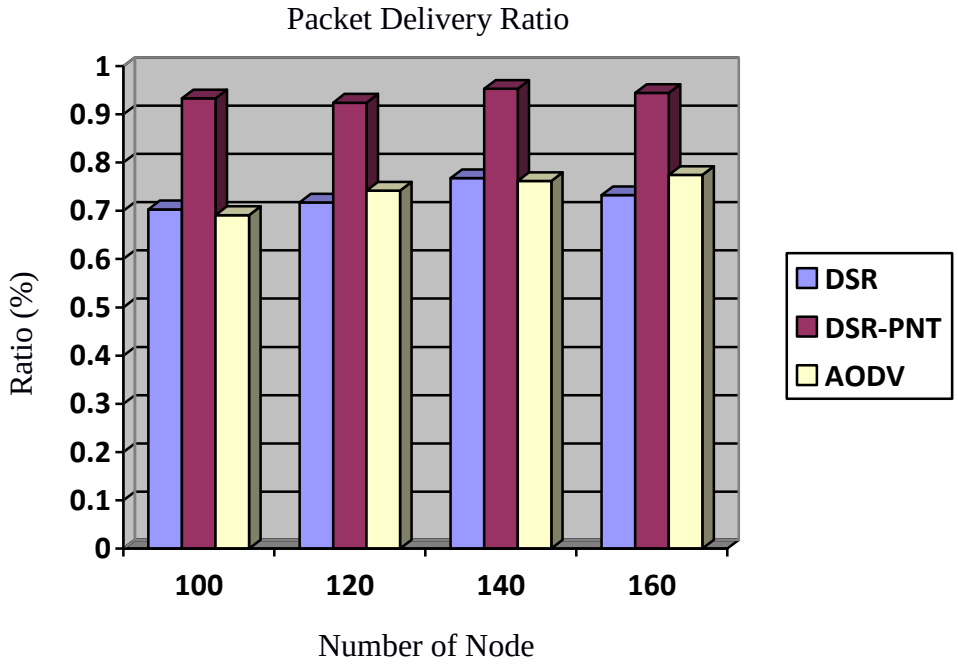
paling optimal sehingga dapat meminimalisir *route error* serta melakukan *drop packet* saat *relay node* yang tidak terdapat dalam *list* menerima RREQ.

Hal ini turut serta membantu *network congestion* dalam jaringan VANETs. Stabilnya *packet delivery ratio* pada *node* 160 berarti *node density* yang tinggi akan menambah jumlah *node* yang layak sebagai *relaying node* sehingga tercipta banyak alternatif rute menuju *destination* sehingga *broken link* dapat diminimalisir.

Tabel 5.10 *Packet Delivery Ratio* Skenario Riil speed 20ms

Jumlah Node	AODV	DSR	DSR-PNT	Perbedaan (DSRvsDSR-PNT)
100	0.6913	0.7032	0.9331	0.2299
120	0.7421	0.7174	0.9243	0.2069
140	0.7619	0.7680	0.9532	0.1852
160	0.7743	0.7328	0.9443	0.2115

Berdasarkan data pada Tabel 5.10 diatas, dibuat grafik yang merepresentasikan hasil perhitungan *packet delivery ratio* yang dapat dilihat pada Gambar 5.8.



Gambar 5.8 *Packet Delivery Ratio* AODV, DSR & DSR-PNT Skenario Riil *speed* 20ms

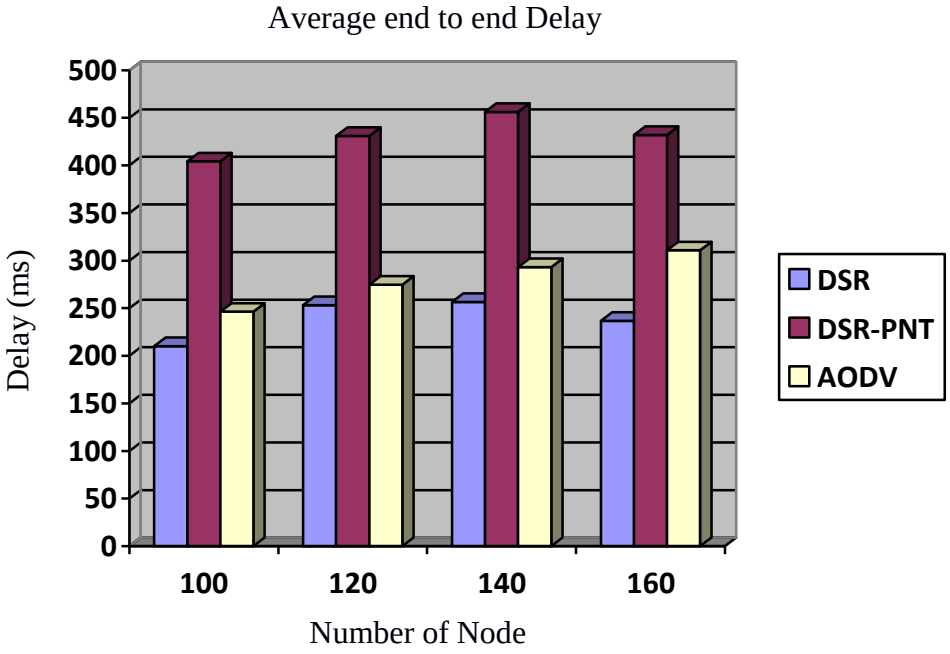
Pada Gambar 5.8 dapat dilihat bahwa modifikasi DSR PNT terdapat peningkatan *packet delivery ratio* sebesar 0.2299 pada *node* 100, 0.2069 pada *node* 120, 0,1852 pada node 140, 0.2115 pada *node* 160. Dapat disimpulkan bahwa pada luas area 800x600 protokol DSR original telah memiliki *packet delivery ratio* yang sudah tinggi yaitu pada *range* 70%. Pada protokol AODV, *packet delivery ratio* pada skenario penelitian ini jatuh pada range 70%. Sementara pada DSR-PNT *packet delivery ratio* meningkat ke *range* 90% hal ini disebabkan oleh perhitungan *future* TWR yang dapat menentukan relay node yang paling optimal sehingga dapat meminimalisir *route error* serta melakukan *drop packet* saat *relay node* yang tidak terdapat dalam list menerima RREQ.

Hal ini turut serta membantu *network congestion* dalam jaringan VANETs. Penggunaan variasi *speed* ke 20ms tidak memberikan perubahan yang signifikan pada *packet delivery ratio*. Hal ini disebabkan oleh perhitungan TWR yang melakukan seleksi terhadap *node* yang termasuk dalam *eligible node,* sehingga apabila sebuah *node* memiliki *speed* yang lebih tinggi namun tidak termasuk dalam *eligible node*, maka *node* tersebut akan melakukan *drop packet*. Stabilnya *packet delivery ratio* pada *node* 160 berarti *node density* yang tinggi akan menambah jumlah *node* yang layak sebagai *relaying node* sehingga tercipta banyak alternatif rute menuju *destination* sehingga *broken link* dapat diminimalisir.

Tabel 5.11 *Average end to end* delay Skenario Riil *speed* 15ms

Jumlah Node	AODV	DSR	DSR-PNT	Perbedaan (DSRvsDSR-PNT)
100	246.31 ms	210.28 ms	404.29 ms	194.01 ms
120	274.76 ms	253.11 ms	430.99 ms	177.88 ms
140	293.14 ms	256.51 ms	456.33 ms	199.82 ms
160	310.81 ms	236.64 ms	432.15 ms	195.51 ms

Berdasarkan data pada Tabel 5.11 di atas, dibuat grafik yang merepresentasikan hasil perhitungan *average end to end delay* yang dapat dilihat pada Gambar 5.9.



Gambar 5.9 Average end to end delay AODV, DSR & DSR-PNT Skenario Riil *speed* 15ms

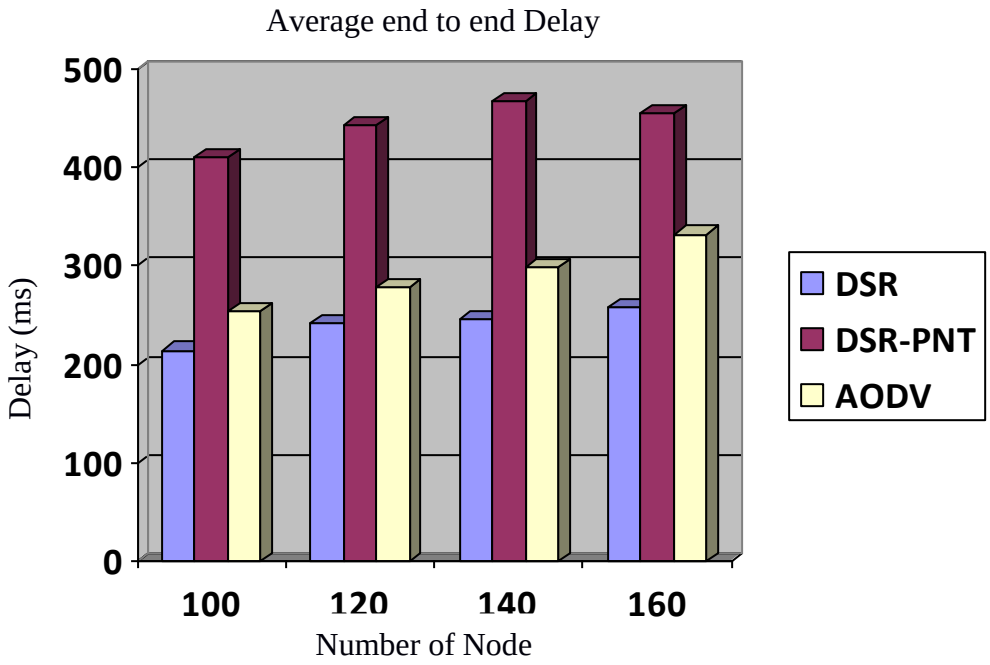
Pada Gambar 5.9 dapat dilihat bahwa peningkatan *delay* telah terjadi pada DSR-PNT sejak node 100 sementara protokol DSR memiliki *delay* yang lebih rendah pada semua banyak *node* dalam penelitian ini. Hal ini disebabkan oleh semakin sedikitnya *node* yang tersedia sebagai *neighbour node*, semakin sedikit pula *node* yang memenuhi syarat sebagai *relay node*. Semakin kecil jumlah *node* yang digunakan sebagai skenario, semakin kecil *pula average end to end delay*. Namun kecilnya *delay* dan jumlah *node* yang lebih sedikit tidak menjamin bahwa *packet delivery ratio* akan meningkat.

Penambahan variasi speed pada kecepatan 15ms dan 20ms pada parameter *average end to end delay* tidak memberikan perubahan yang signifikan terhadap protokol DSR-PNT, namun memberikan sedikit perubahan pada protokol DSR dan AODV. DSR-PNT tidak terlalu terpengaruh terhadap faktor variasi kecepatan karena pemilihan *relaying node* melalui perhitungan TWR yang mengakibatkan tingginya *delay* pada DSR-PNT, sementara protokol DSR dan AODV melakukan pemilihan *relaying node* berdasarkan node yang paling cepat mengirimkan *reply* kepada *node* pengirim sehingga memiliki *delay* yang relatif rendah.

Tabel 5.12 *Average end to end delay* Skenario Riil *speed* 20ms

Jumlah Node	AODV	DSR	DSR-PNT	Perbedaan (DSRvsDSR-PNT)
100	253.63 ms	214.49 ms	410.42 ms	195.93 ms
120	277.92 ms	242.31 ms	442.12 ms	199.81 ms
140	298.26 ms	246.18 ms	467.23 ms	221.05 ms
160	332.17 ms	258.33 ms	455.21 ms	196.88 ms

Berdasarkan data di atas, dibuat grafik yang merepresentasikan hasil perhitungan *average end to end delay* yang dapat dilihat pada Gambar 5.10.



Gambar 5.10 *Average end to end delay* AODV, DSR & DSR-PNT Skenario Riil *speed* 20ms

Pada Gambar 5.10 dapat dilihat bahwa peningkatan *delay* telah terjadi pada DSR-PNT sejak *node* 100 sementara protokol DSR memiliki *delay* yang lebih rendah pada semua banyak *node* dalam penelitian ini. Hal ini disebabkan oleh semakin sedikitnya *node* yang tersedia sebagai neighbour *node*, semakin sedikit pula *node* yang memenuhi syarat sebagai *relay node*. Semakin kecil jumlah *node* yang digunakan sebagai skenario, semakin kecil pula *average end to end delay*. Namun kecilnya *delay* dan jumlah *node* yang lebih sedikit tidak menjamin bahwa *packet delivery ratio* akan meningkat.

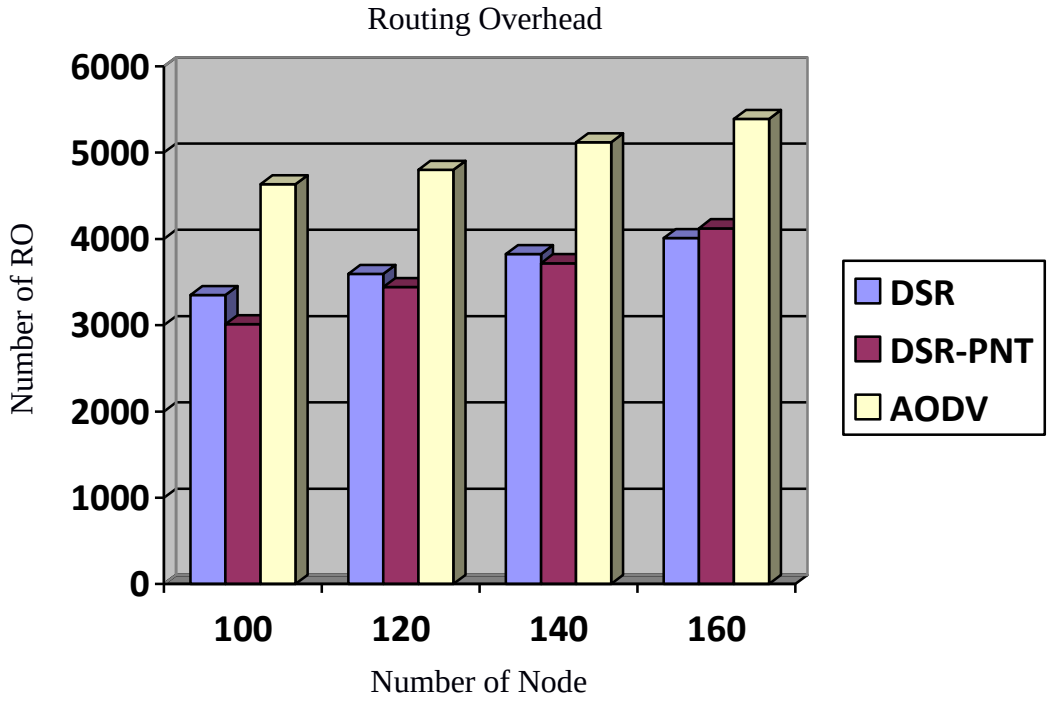
Penambahan variasi speed pada kecepatan 15ms dan 20ms pada parameter *average end to end delay* tidak memberikan perubahan yang signifikan terhadap

protokol DSR-PNT, namun memberikan sedikit perubahan pada protokol DSR dan AODV. DSR-PNT tidak terlalu terpengaruh terhadap faktor variasi kecepatan karena pemilihan *relaying node* melalui perhitungan TWR yang mengakibatkan tingginya *delay* pada DSR-PNT, sementara protokol DSR dan AODV melakukan pemilihan *relaying node* berdasarkan node yang paling cepat mengirimkan *reply* kepada *node* pengirim sehingga memiliki *delay* yang relatif rendah.

Tabel 5.13 Routing Overhead Skenario Riil speed 15ms

Jumlah Node	AODV	DSR	DSR-PNT	Perbedaan (DSRvsDSR-PNT)
100	4632	3350	3010	340
120	4801	3593	3442	151
140	5120	3824	3718	106
160	5391	4123	4009	114

Berdasarkan data yang didapat pada Tabel 5.13 maka dapat dibuat grafik yang merepresentasikan hasil perhitungan *routing overhead* yang dapat dilihat pada Gambar 5.11.



Gambar 5.11 Routing Overhead AODV, DSR & DSR-PNT Skenario Riil *speed* 15ms

Pada Gambar 5.11 dapat dilihat bahwa *routing overhead* dari DSR-PNT sedikit lebih unggul daripada protokol DSR. Hal ini disebabkan oleh semakin meningkatnya jumlah *node*, maka *send rate, receive rate* beserta *error rate* akan meningkat karena kedua metode menggunakan metode *broadcast* saat *route discovery*. Pemilihan *node* yang tepat dalam *relaying* atau *Eligible node* yang terdapat dalam *list relay node set* membuat DSR-PNT sedikit lebih unggul. Protokol AODV memiliki *routing overhead* paling tinggi karena memiliki jumlah pengiriman paket, penerimaan paket serta paket *error* yang tinggi. Hal ini berbanding lurus dengan meningkatnya jumlah *node* dalam jaringan. Semakin tinggi jumlah *node* dalam jaringan, maka *routing overhead* AODV cenderung meningkat.

Penambahan variasi *speed* pada kecepatan 15ms dan 20ms pada parameter *routing overhead* tidak memberikan perubahan yang signifikan terhadap protokol DSR-PNT, namun memberikan sedikit perubahan pada protokol DSR dan AODV. DSR-PNT tidak terlalu terpengaruh terhadap faktor variasi kecepatan karena pemilihan *relaying node* melalui perhitungan TWR, sementara protokol DSR dan AODV melakukan pemilihan *relaying node* berdasarkan node yang paling cepat mengirimkan reply kepada node pengirim.

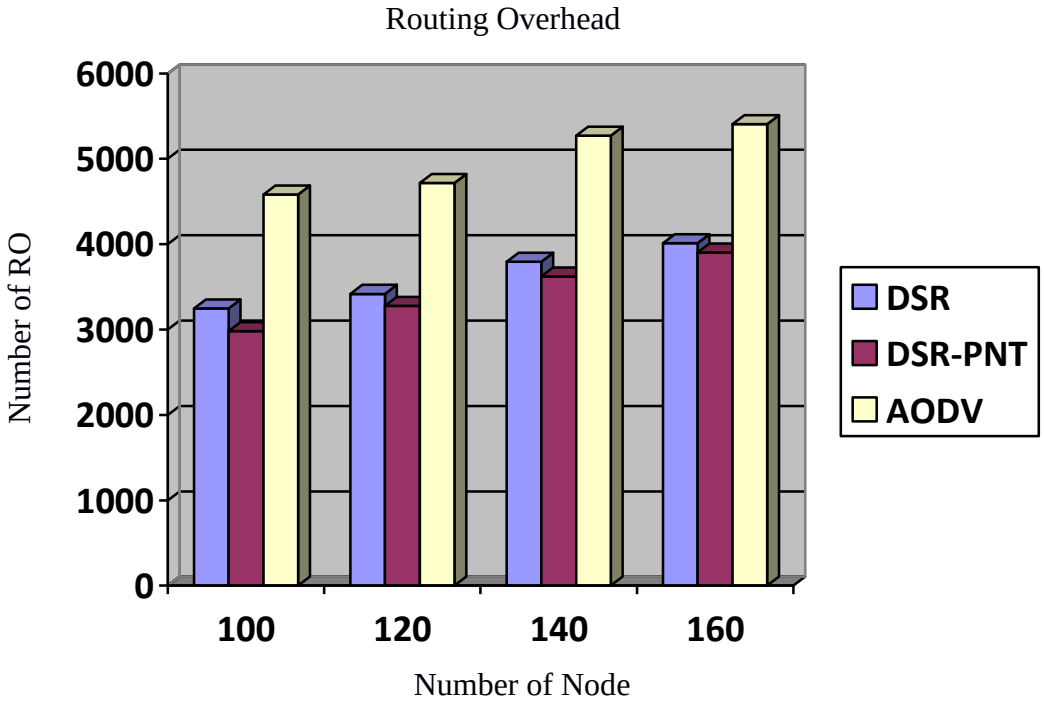
Tabel 5.14 Routing Overhead Skenario Riil speed 20ms

Jumlah Node	AODV	DSR	DSR-PNT	Perbedaan (DSRvsDSR-PNT)
100	4582	3247	2983	264
120	4718	3418	3279	139
140	5271	3793	3621	172
160	5405	4011	3903	108

Berdasarkan data yang didapat pada Tabel 5.14 maka dapat dibuat grafik yang merepresentasikan hasil perhitungan *routing overhead* yang dapat dilihat pada Gambar 5.12.

Pada Gambar 5.12 dapat dilihat bahwa *routing overhead* dari DSR-PNT sedikit lebih unggul daripada protokol DSR. Hal ini disebabkan oleh semakin meningkatnya jumlah *node*, maka *send rate, receive rate* beserta *error rate* akan

meningkat karena kedua metode menggunakan metode *broadcast* saat *route discovery*. Pemilihan *node* yang tepat dalam *relaying* atau *Eligible node* yang terdapat dalam *list relay node set* membuat DSR-PNT sedikit lebih unggul. Protokol AODV memiliki *routing overhead* paling tinggi karena memiliki jumlah pengiriman paket, penerimaan paket serta paket *error* yang tinggi. Hal ini berbanding lurus dengan meningkatnya jumlah *node* dalam jaringan. Semakin tinggi jumlah *node* dalam jaringan, maka *routing overhead* AODV cenderung meningkat.



Gambar 5.12 Routing Overhead AODV, DSR & DSR-PNT Skenario Riil speed 20ms

Penambahan variasi *speed* pada kecepatan 15ms dan 20ms pada parameter *routing overhead* tidak memberikan perubahan yang signifikan terhadap protokol DSR-PNT, namun memberikan sedikit perubahan pada protokol DSR dan AODV. DSR-PNT tidak terlalu terpengaruh terhadap faktor variasi kecepatan karena pemilihan *relaying node* melalui perhitungan TWR, sementara protokol DSR dan AODV melakukan pemilihan *relaying node* berdasarkan node yang paling cepat mengirimkan reply kepada node pengirim.

BAB 6

KESIMPULAN DAN SARAN

Pada Bab ini akan diberikan kesimpulan yang dapat diambil dalam penelitian ini beserta saran-saran tentang pengembangan yang dapat dilakukan terhadap penelitian ini di masa mendatang.

6.1 Kesimpulan

Kesimpulan yang diperoleh dari penelitian ini berdasarkan uji coba yang telah dilakukan adalah sebagai berikut:

1. *Packet Delivery Ratio* dari DSR-PNT meningkat sebesar 18% hingga 25% pada skenario grid dan riil dibandingkan dengan DSR. Semakin meningkatnya jumlah kendaraan pada jaringan cenderung meningkatkan *packet delivery ratio* pada DSR-PNT.
2. *Average end to end delay* pada DSR-PNT baik pada skenario grid maupun riil cenderung tinggi pada setiap jumlah *node* yang di skenariokan pada penelitian ini. Sedangkan pada original DSR, *average end to end delay* lebih rendah namun akan meningkat apabila jumlah kendaraan dalam jaringan meningkat.
3. *Routing overhead* dari DSR-PNT baik pada skenario grid maupun riil cenderung lebih rendah daripada *original* DSR. Hal ini disebabkan oleh semakin meningkatnya jumlah *node*, maka *send rate*, *receive rate* beserta *error rate* akan meningkat karena kedua metode menggunakan metode *broadcast* saat *route discovery*. Pemilihan *node* yang tepat dalam *relaying* atau *Eligible node* yang terdapat dalam *list relay node set* membuat DSR-PNT sedikit lebih unggul.
4. *Node density* atau jumlah *neighbour* terbukti dapat meningkatkan *packet delivery ratio* pada DSR-PNT dengan menggunakan metode pemilihan *relaying node* yang optimal pada saat pengiriman paket. Pada jumlah *node* yang lebih tinggi, *packet delivery ratio* tetap memiliki *rate* yang tinggi dengan kompensasi meningkatnya *average end to end delay*.

6.2 Saran

Saran yang dapat diberikan berdasarkan hasil uji coba yang telah dilakukan adalah perlu dilakukan studi lebih lanjut terhadap weight factor dalam TWR dan future TWR. Studi yang dimaksud terkait dengan berbagai jenis skenario yang beragam sehingga dibutuhkan weight factor yang bersifat adaptif untuk menangani berbagai jenis skenario.

DAFTAR PUSTAKA

Donna, Hadi, Hassan, Mahmoud, Raed, Shubair, et al. (2015). QDSR. *International Conference on Innovation in Information Technology (IIT)*, 162-165.

Hogan, B. (2010, February 3). *DSR FAQ*. Dipetik 3 6, 2017, dari Bryan's NS-2 Dynamic Source Routing FAQ: http://www.skynet.ie/~bryan/dsr_faq/

isi.edu. (2011, November). *The Network Simulator* . Diambil kembali dari - ns-2: http://www.isi.edu/nsnam/ns/

Krajzewicz, D. E., Jakob, Michael, B., & Laura, B. (2012). Recent Development and Applications of SUMO – Simulation of Urban MObility. *International Journal on Advances in Systems and Measurements*, *5*.

Mohamed, Mustapha, Walid, & Abdelhamid. (2007). Optimization and Performance Study of the Dynamic Source Routing Protocol. *IEEE*.

S Feroz Ahammad, M. V. (2014). Neighbour Position discovery and verification in manet. *IJIRCCE*, 6220-6225.

Sharmin, Salma, Nazma, & Ahsan. (2010). Enhanced DSR : A New Approach to Improve Performance of DSR. *IJCSIT*, 113-123.

Toh, & K, C. (2002). Ad Hoc Mobile Wireless Networks. *Prentice Hall*.

Vikas Toor, S. R. (Oktober 2012). Path Duration Analysis In Vehicular Ad-hoc Network. *International Journal on AdHoc Networking Systems (IJANS) vol 2*, 57-66.

XiaoWei, & YiWu. (2014). AODV-PNT : Improved version of AODV with predicting node trend in VANET. *IEEE*, 91-97.

(halaman ini sengaja dikosongkan)

BIOGRAFI PENULIS

Bagus Gede Krishna Yudistira dilahirkan di Singaraja, Bali pada tanggal 12 Agustus 1992 dan merupakan anak pertama dari tiga bersaudara dari pasangan I Nyoman Kanca dan Ni Wayan Sukarmini. Penulis menempuh pendidikan dasar di SD LAB IKIP Negeri Singaraja pada tahun 1998 hingga tahun 2004, lalu melanjutkan pendidikan menengah pertama di SMP Negeri 2 Singaraja pada tahun 2004 hingga tahun 2007, dan pendidikan menengah atas di SMA Negeri 1 Singaraja pada tahun 2007 hingga tahun 20010. Penulis melanjutkan jenjang perguruan tinggi pada Jurusan Teknik Informatika di Institut Teknologi Sepuluh Nopember pada tahun 2010 hingga tahun 2014. Setelah itu penulis melanjutkan jenjang magister Teknik Informatika di Institut Teknologi Sepuluh Nopember Surabaya pada tahun 2015 hingga tahun 2017.

Setelah selesai menempuh pendidikan sarjana penulis pernah mengikuti kursus Bahasa Inggris pada kelas conversation di LIA Surabaya pada tahun 2014. Penulis juga pernah mengikuti kursus di IALF surabaya pada tahun 20114 untuk mengasah kemampuan berbahasa inggris selama 1 tahun hingga akhirnya pada tahun 2015 memutuskan melanjutkan pendidikan di Institut Teknologi Sepuluh Nopember.

Selain Pendidikan *formal*, penulis juga meluangkan waktu sebagai *Session Guitarist* dan *Music Composer*. Penulis mempunyai ketertarikan dalam bidang musik seperti bermain alat musik dan juga menciptakan dan mengaransemen lagu. Penulis juga mempunyai hobi lain yaitu bermain *game dan* melakukan aktifitas jasmani seperti *badminton* dan *weightlifting*. Penulis dapat dihubungi melalui email yaitu: Krishna.mustang49@gmail.com

(halaman ini sengaja dikosongkan)

LAMPIRAN

Lampiran 1. Kode Skenario NS-2

```
# wrls1.tcl
# A 3-node example for ad-hoc simulation with DSR

# Define options
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)            CMUPriQueue                ;# 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(nn)             120                        ;# number of mobilenodes
set val(rp)             DSR                        ;# routing protocol
set val(x)              1011.65                    ;# X dimension of topography
set val(y)              1011.65                    ;# Y dimension of topography
set val(stop)           360                        ;# time of simulation end
set val(mobilityfile) "mobility.tcl"
set val(activityfile) "activity.tcl"

set ns     [new Simulator]
set tracefd       [open tracef-dsr.tr w]
set windowVsTime2 [open win.tr w]
set namtrace      [open nam-dsr.nam w]

$ns trace-all $tracefd
$ns namtrace-all-wireless $namtrace $val(x) $val(y)

# set up topography object
set topo       [new Topography]

$topo load_flatgrid $val(x) $val(y)


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

# configure the nodes
        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) \
                    -channelType $val(chan) \
                    -topoInstance $topo \
                    -agentTrace ON \
                    -routerTrace ON \
                    -macTrace OFF \
                    -movementTrace ON

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)]

# Provide initial location of mobilenodes
set sender [$ns node]
$sender set X_ 490.0
$sender set Y_ 285.0
$sender set Z_ 0.0

set receiver [$ns node]
$receiver set X_ 150.0
$receiver set Y_ 240.0
$receiver set Z_ 0.0


# Set a TCP connection between node_(0) and node_(1)
set tcp [new Agent/TCP/Newreno]
$tcp set class_ 2
set sink [new Agent/TCPSink]
$ns attach-agent $sender $tcp
$ns attach-agent $receiver $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 150.0 "$ftp start"

# Printing the window size
```

```
proc plotWindow {tcpSource file} {
global ns
set time 0.01
set now [$ns now]
set cwnd [$tcpSource set cwnd_]
puts $file "$now $cwnd"
$ns at [expr $now+$time] "plotWindow $tcpSource $file" }
$ns at 10.1 "plotWindow $tcp $windowVsTime2"


# Telling nodes when the simulation ends
for {set i 0} {$i < $val(nn) } { incr i } {
    $ns at $val(stop) "$node_($i) reset";
}

# ending nam and the simulation
$ns at $val(stop) "$ns nam-end-wireless $val(stop)"
$ns at $val(stop) "stop"
$ns at 360.01 "puts \"end simulation\" ; $ns halt"
proc stop {} {
    global ns tracefd namtrace
    $ns flush-trace
    close $tracefd
    close $namtrace
}

$ns run
```

(halaman ini sengaja dikosongkan)

Lampiran 2. Potongan Kode SendOutRtReq pada DSRagent.cc

```
1565  void
1566  DSRAgent::sendOutRtReq(SRPacket &p, int max_prop)
1567    // turn p into a route request and launch it, max_prop of request is
1568    // set as specified
1569    // p.pkt is freed or handed off
1570  {
1571       //speed accel posx y
1572      MobileNode *iNode;
1573      iNode = (MobileNode *) (Node::get_node_by_address(index));
1574      double iSpeed = ((MobileNode *) iNode)->speed();
1575      double now = ((MobileNode *) iNode)->getUpdateTime();
1576      double posX = iNode->X();
1577      double posY = iNode->Y();
1578
1579      hdr_sr *srh =  hdr_sr::access(p.pkt);
1580      assert(srh->valid());
1581      //printf("sendoutroutereq\n");
1582      srh->route_request() = 1;
1583      srh->rtreq_seq() = route_request_num++;
1584      srh->max_propagation() = max_prop;
1585      p.route.reset();
1586      p.route.appendToPath(net_id);
1587
1588      srh->sr_speed = iSpeed;
1589      if (now - lastUpdateTime == 0) {
1590      srh->sr_accel = lastAccel;
1591      }
1592      else {
1593      srh->sr_accel = iSpeed / (now - lastUpdateTime); // a = v/t
1594      lastAccel    = sr->sr_accel;
1595      }
1596
1597      srh->sr_x     = posX;
1598      srh->sr_y     = posY;
1599
1600      Scheduler::instance().schedule(target_, p, 0.0);
1601
1602      // Update its latest update time
1603      lastUpdateTime = now;
1604  }
```

(halaman ini sengaja dikosongkan)

Lampiran 3. Potongan Code TWR Calculation & Future TWR 1

```
// 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));

// radius between this node and neighbor
// minimum radius -> min(√(i x − j x )^2 + (i y − j y )^2 ; r)
double radius = std::min(
  sqrt(pow((nb->nb_x - posX), 2) + pow((nb->nb_y - posY), 2)), (double) maxTxRange);

double quality = 1.0 / (1.0 - (radius / ((double) maxTxRange + 1.0)));

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 × |θ n − θ d | + f q × Q
double TWR = modSpeed + modAccel + modDistance + modQuality;
```

```
798     // Future speed v' = v + a × t
799     double nb_speedFuture = nb->nb_speed + (nb->nb_accel * timeModifier);
800
801     // Future position will be used to calc future direction
802     // Formula: x' = x + v0 t × 0.5at^2
803     // Future neighbor position
804     double nb_xFuture = nb->nb_x +
805                 (nb->nb_speed * timeModifier)
806                   + (0.5 * nb->nb_accel * timeModifier * timeModifier);
807     double nb_yFuture = nb->nb_y +
808                 (nb->nb_speed * timeModifier)
809                   + (0.5 * nb->nb_accel * timeModifier * timeModifier);
810
811     // Future this_node position
812     double iXFuture = posX +
813                 (iSpeed * timeModifier)
814                   + (0.5 * iAccel * timeModifier * timeModifier);
815     double iYFuture = posY +
816                 (iSpeed * timeModifier)
817                   + (0.5 * iAccel * timeModifier * timeModifier);
```

(halaman ini sengaja dikosongkan)

Lampiran 4. Potongan Code recvRequest yang menangani TWR Calculation & Future TWR

```
void
DSRAgent::recvRequest(Packet *p) {
// struct hdr_cmn *ch = HDR_CMN(p);
struct hdr_ip *ih = HDR_IP(p);
struct hdr_DSRAgent_request *rq = HDR_DSRAgent_REQUEST(p);
DSRAgent_rt_entry *rt;
bool isEligible = true;

  /*
   * Drop if:
   *      - I'm the source
   *      - I recently heard this request.
   *      - I'm not in the list of eligible nodes
   */

  if(rq->rq_src == index) {
#ifdef DEBUG
    fprintf(stderr, "%s: got my own REQUEST\n", __FUNCTION__);
#endif // DEBUG

    Packet::free(p);
    return;
  }


 if (id_lookup(rq->rq_src, rq->rq_bcast_id)) {

#ifdef DEBUG
   fprintf(stderr, "%s: discarding request\n", __FUNCTION__);
#endif // DEBUG

   Packet::free(p);
   return;
 }

 // If list is null, then drop packet
 if(rq->rq_eligible_nodes == NULL) {

   Packet::free(p);
   return;
 }
 else if (rq->rq_dst != index) {
   isEligible = false;
   for (u_int32_t i = 0; i < rq->nodes_list_len; ++i) {
```

```
    // I'm on the list, so I'm eligible for this packet
    if (rq->rq_eligible_nodes[i] == index) {
      isEligible = true;
      break;
    }
   }
 }

// If I'm not in eligible list, then drop packet
if (!isEligible) {
  Packet::free(p);
  return;
}
// If I'm eligible, then re-compute to create new eligible list
else if (rq->rq_dst != index) {
  /*
   *  DSRAgent-PNT re-compute TWR
   */

 // TWR list
 std::vector<TWRContainer> listTWR;
 std::vector<nsaddr_t> eligibleAddrList;
 nsaddr_t *eligibleNodes;

 // Get current position, speed, time
 MobileNode *iNode;
 iNode = (MobileNode *) (Node::get_node_by_address(index));
 double iSpeed = ((MobileNode *) iNode)->speed();
 double now = ((MobileNode *) iNode)->getUpdateTime();

 double iAccel;
 if (now - lastUpdateTime == 0) {
   iAccel = lastAccel;
 }
 else {
   iAccel = iSpeed / (now - lastUpdateTime);
   lastAccel = iAccel;
 }

 double posX = iNode->X();
 double posY = iNode->Y();

 u_int32_t fSpeed = F_SPEED;
 u_int32_t fAccel = F_ACCEL;
 u_int32_t fDistance = F_DISTANCE;
 u_int32_t fQuality = F_QUALITY;
 u_int32_t timeModifier = TIME_MOD;
```

```
 u_int32_t maxTxRange = TX_RANGE; // ns2 default range (based on Pt_)
 u_int32_t thresholdW = THRES_W;

 // Dst fixed position (STA dst). real scenario: we can find exact loc via GPS
 double xDst = DST_X;
 double yDst = DST_Y;


 // Traverse neighbor list
 DSRAgent_Neighbor *nb = nbhead.lh_first;

 for(; nb; nb = nb->nb_link.le_next) {
  // 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));

  // radius between this node and neighbor
  // minimum radius -> min(√(i x − j x )^2 + (i y − j y )^2 ; r)
  double radius = std::min(
    sqrt(pow((nb->nb_x - posX), 2) + pow((nb->nb_y - posY), 2)), (double)
maxTxRange);

  double quality = 1.0 / (1.0 - (radius / ((double) maxTxRange + 1.0)));

  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 × |θ n − θ d | + f q × Q
  double TWR = modSpeed + modAccel + modDistance + modQuality;

  // Calculate future TWR for next [timeModifier] seconds
  // Future speed v' = v + a × t
  double nb_speedFuture = nb->nb_speed + (nb->nb_accel * timeModifier);

  // Future position will be used to calc future direction
  // Formula: x' = x + v0 t × 0.5at^2
  // Future neighbor position
  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);

  // Calculate future distance between next-hop and dst
  double futureDistance;
  futureDistance = sqrt(pow((nb_xFuture - xDst), 2) + pow((nb_yFuture - yDst),
2));

  // Future radius between this node and neighbor: both using future values
  double futureRadius = std::min(
    sqrt(pow((nb_xFuture - iXFuture), 2) + pow((nb_yFuture - iYFuture), 2)),
(double) maxTxRange);

  double futureQuality = 1.0 / (1.0 - (futureRadius / ((double) maxTxRange +
1.0)));

  // Calculate future TWR
  modSpeed     = fSpeed * nb_speedFuture;
  modAccel     = fAccel * nb->nb_accel;
  modDistance  = fDistance * futureDistance;
  modQuality   = fQuality * futureQuality;
  double futureTWR = modSpeed + modAccel + modDistance + modQuality;

  // Store TWR futureTWR in a list
  TWRContainer container;
  container.TWR = TWR;
  container.futureTWR = futureTWR;
  container.address = nb->nb_addr;
  container.isOptimal = false;
  container.isStable = false;
  container.isFutureBetter = false;

  listTWR.push_back(container);
 }

 // Relay decision making based on TWR and futureTWR using some rules
 // ascending sort listTWR based on TWR value
 std::sort(listTWR.begin(), listTWR.end(), minTWR());

 // minimum TWR in the list
 listTWR[0].isOptimal = true;
```

```
 for (u_int32_t i = 0; i < listTWR.size(); i++) {
   // 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.
   if (listTWR[i].futureTWR < listTWR[i].TWR) {
     listTWR[i].isFutureBetter = true;
   }

   // Jika ΔTWR lebih kecil dari threshold W, maka node tersebut
   // dianggap sebagai “stable”.
   // Jika ΔTWR lebih besar dari threshold W, maka node tersebut
   // dianggap sebagai “unstable”.
   double deltaTWR = fabs(listTWR[i].TWR - listTWR[i].futureTWR);
   if (deltaTWR < thresholdW) {
     listTWR[i].isStable = true;
   }

   // Optimal, unstable, better -> Relay
   if (listTWR[i].isOptimal && !listTWR[i].isStable &&
listTWR[i].isFutureBetter) {
     eligibleAddrList.push_back(listTWR[i].address);
   }
   // Optimal, stable, not better -> Relay
   else if (listTWR[i].isOptimal && listTWR[i].isStable
&& !listTWR[i].isFutureBetter) {
     eligibleAddrList.push_back(listTWR[i].address);
   }
   // Suboptimal, unstable, better -> Relay
   else if (!listTWR[i].isOptimal && !listTWR[i].isStable &&
listTWR[i].isFutureBetter) {
     eligibleAddrList.push_back(listTWR[i].address);
   }
   // Suboptimal, stable, not better -> Relay
   else if (!listTWR[i].isOptimal && listTWR[i].isStable
&& !listTWR[i].isFutureBetter) {
     eligibleAddrList.push_back(listTWR[i].address);
   }
 }

 // replace rq_eligible_nodes with the re-computed list
 rq->nodes_list_len = eligibleAddrList.size();
 eligibleNodes = new nsaddr_t[rq->nodes_list_len];
 for (u_int32_t i = 0; i < rq->nodes_list_len; i++) {
   eligibleNodes[i] = eligibleAddrList[i];
 }
 rq->rq_eligible_nodes = eligibleNodes;
```

```
}


 /*
  * Cache the broadcast ID
  */
 id_insert(rq->rq_src, rq->rq_bcast_id);



 /*
  * We are either going to forward the REQUEST or generate a
  * REPLY. Before we do anything, we make sure that the REVERSE
  * route is in the route table.
  */
 DSRAgent_rt_entry *rt0; // rt0 is the reverse route

   rt0 = rtable.rt_lookup(rq->rq_src);
   if(rt0 == 0) { /* if not in the route table */
   // create an entry for the reverse route.
     rt0 = rtable.rt_add(rq->rq_src);
   }

   rt0->rt_expire = max(rt0->rt_expire, (CURRENT_TIME +
REV_ROUTE_LIFE));

   if ( (rq->rq_src_seqno > rt0->rt_seqno ) ||
                                                ((rq->rq_src_seqno == rt0-
>rt_seqno) &&
                                                 (rq->rq_hop_count < rt0-
>rt_hops)) ) {
   // If we have a fresher seq no. or lesser #hops for the
   // same seq no., update the rt entry. Else don't bother.
rt_update(rt0, rq->rq_src_seqno, rq->rq_hop_count, ih->saddr(),
                                                        max(rt0->rt_expire,
(CURRENT_TIME + REV_ROUTE_LIFE)) );
     if (rt0->rt_req_timeout > 0.0) {
     // Reset the soft state and
     // Set expiry time to CURRENT_TIME + ACTIVE_ROUTE_TIMEOUT
     // This is because route is used in the forward direction,
     // but only sources get benefited by this change
       rt0->rt_req_cnt = 0;
       rt0->rt_req_timeout = 0.0;
       rt0->rt_req_last_ttl = rq->rq_hop_count;
       rt0->rt_expire = CURRENT_TIME + ACTIVE_ROUTE_TIMEOUT;
     }
```

```
    /* Find out whether any buffered packet can benefit from the
     * reverse route.
     * May need some change in the following code - Mahesh 09/11/99
     */
    assert (rt0->rt_flags == RTF_UP);
    Packet *buffered_pkt;
    while ((buffered_pkt = rqueue.deque(rt0->rt_dst))) {
      if (rt0 && (rt0->rt_flags == RTF_UP)) {
                                                    assert(rt0->rt_hops !=
INFINITY2);
        forward(rt0, buffered_pkt, NO_DELAY);
      }
    }
  }
  // End for putting reverse route in rt table


 /*
  * We have taken care of the reverse route stuff.
  * Now see whether we can send a route reply.
  */

 rt = rtable.rt_lookup(rq->rq_dst);

 // First check if I am the destination ..

 if(rq->rq_dst == index) {

#ifdef DEBUG
  fprintf(stderr, "%d - %s: destination sending reply\n",
               index, __FUNCTION__);
#endif // DEBUG


  // Just to be safe, I use the max. Somebody may have
  // incremented the dst seqno.
  seqno = max(seqno, rq->rq_dst_seqno)+1;
  if (seqno%2) seqno++;

  sendReply(rq->rq_src,          // IP Destination
            1,                   // Hop Count
            index,               // Dest IP Address
            seqno,               // Dest Sequence Num
            MY_ROUTE_TIMEOUT,    // Lifetime
            rq->rq_timestamp);   // timestamp
```

```
  delete[] rq->rq_eligible_nodes;
  rq->rq_eligible_nodes = NULL;
  Packet::free(p);
 }

 // I am not the destination, but I may have a fresh enough route.

 else if (rt && (rt->rt_hops != INFINITY2) &&
                                                (rt->rt_seqno >= rq-
>rq_dst_seqno) ) {

  //assert (rt->rt_flags == RTF_UP);
  assert(rq->rq_dst == rt->rt_dst);
  //assert ((rt->rt_seqno%2) == 0);          // is the seqno even?
  sendReply(rq->rq_src,
            rt->rt_hops + 1,
            rq->rq_dst,
            rt->rt_seqno,
                                                (u_int32_t) (rt->rt_expire -
CURRENT_TIME),
                                                //          rt->rt_expire -
CURRENT_TIME,
            rq->rq_timestamp);
  // Insert nexthops to RREQ source and RREQ destination in the
  // precursor lists of destination and source respectively
  rt->pc_insert(rt0->rt_nexthop); // nexthop to RREQ source
  rt0->pc_insert(rt->rt_nexthop); // nexthop to RREQ destination

#ifdef RREQ_GRAT_RREP

  sendReply(rq->rq_dst,
            rq->rq_hop_count,
            rq->rq_src,
            rq->rq_src_seqno,
                                                (u_int32_t) (rt->rt_expire -
CURRENT_TIME),
                                                //          rt->rt_expire -
CURRENT_TIME,
            rq->rq_timestamp);
#endif

// TODO: send grat RREP to dst if G flag set in RREQ using rq->rq_src_seqno,
rq->rq_hop_counT

// DONE: Included gratuitous replies to be sent as per IETF DSRAgent draft
specification. As of now, G flag has not been dynamically used and is always set
or reset in DSRAgent-packet.h --- Anant Utgikar, 09/16/02.
```

```
                                                    Packet::free(p);
 }
 /*
  * Can't reply. So forward the  Route Request
  */
 else {

   ih->saddr() = index;
   ih->daddr() = IP_BROADCAST;
   rq->rq_hop_count += 1;
   // Maximum sequence number seen en route
   if (rt) rq->rq_dst_seqno = max(rt->rt_seqno, rq->rq_dst_seqno);
   forward((DSRAgent_rt_entry*) 0, p, DELAY);

 }

}
```

(halaman ini sengaja dikosongkan)

Lampiran 5. Skrip AWK PDR

```
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);
                }
```

(halaman ini sengaja dikosongkan)

Lampiran 6. Skrip AWK end to end delay

```
#
=========================================================
=========

# AWK Script for calculating:

#     => Average End-to-End Delay.

#
=========================================================
=========



BEGIN {

   seqno = -1;

#    droppedPackets = 0;

#    receivedPackets = 0;

   count = 0;

}

{

   if($4 == "AGT" && $1 == "s" && seqno < $6) {

        seqno = $6;

   }
#         else if(($4 == "AGT") && ($1 == "r")) {

#           receivedPackets++;

#    } else if ($1 == "D" && $7 == "tcp" && $8 > 512){

#           droppedPackets++;

#  }

   #end-to-end delay
```

```
   if($4 == "AGT" && $1 == "s") {
        start_time[$6] = $2;
   } else if(($7 == "tcp") && ($1 == "r")) {
      end_time[$6] = $2;
   } else if($1 == "D" && $7 == "tcp") {
        end_time[$6] = -1;
   }
}

END {
   for(i=0; i<=seqno; i++) {
        if(end_time[i] > 0) {
           delay[i] = end_time[i] - start_time[i];
               count++;
      }
          else
          {
               delay[i] = -1;
          }
   }
   for(i=0; i<=seqno; i++) {
        if(delay[i] > 0) {
           n_to_n_delay = n_to_n_delay + delay[i];
      }
```

```
    }

   n_to_n_delay = n_to_n_delay/count;



    print "\n";

#    print "GeneratedPackets            = " seqno+1;

#    print "ReceivedPackets             = " receivedPackets;

#    print "Packet Delivery Ratio       = " receivedPackets/(seqno+1)*100
#"%";

#    print "Total Dropped Packets = " droppedPackets;

    print "Average End-to-End Delay    = " n_to_n_delay * 1000 " ms";

    print "\n";

}
```

(halaman ini sengaja dikosongkan)

Lampiran 7. Skrip AWK 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);
                }
```