PENERAPAN ALGORITMA SEMUT PADA PROTOKOL ROUTING AOMDV UNTUK OPTIMASI PENCARIAN RUTE DI JARINGAN VANET

Tugas Akhir

Untuk memenuhi sebagian persyaratan
mencapai derajat Sarjana S-1 Program Studi Teknik Informatika

Disusun Oleh:

NI DESAK KETUT PUJIKA DEWI

F1D014066

PROGRAM STUDI TEKNIK INFORMATIKA

FAKULTAS TEKNIK

UNIVERSITAS MATARAM

2020

TUGAS AKHIR

PENERAPAN ALGORITMA SEMUT PADA PROTOKOL ROUTING AOMDV UNTUK OPTIMASI PENCARIAN RUTE DI JARINGAN VANET

Oleh :

NI DESAK KETUT PUJIKA DEWI

F1D 014 066

Telah diperiksa oleh Tim Pembimbing :

1. Pembimbing Utama

   Tanggal : 31/05/2020

   Andy Hidayat Jatmika, S.T., M.Kom.

   NIP. 19831209 201212 1001

2. Pembimbing Pendamping

   Tanggal : 02/06/2020

   Ariyan Zubaidi, S.Kom., M.T.

   NIP. 19860913 201504 1001

Mengetahui,

Ketua Program Studi Teknik Informatika

Fakultas Teknik

Universitas Mataram

Prof. Dr. Eng. I Gede Pasek Suta Wijaya, ST., MT.

NIP. 19731130 200003 1001

TUGAS AKHIR

PENERAPAN ALGORITMA SEMUT PADA PROTOKOL ROUTING AOMDV UNTUK OPTIMASI PENCARIAN RUTE DI JARINGAN VANET

Oleh :

NI DESAK KETUT PUJIKA DEWI
F1D 014 066

Telah diujikan di depan penguji
Pada tanggal 19 Mei 2020
Dan dinyatakan telah memenuhi syarat mencapai derajat Sarjana S-1
Program Studi Teknik Informatika

Susunan Tim Penguji :

1. Penguji 1

   Tanggal : 31/05/2020

   Dr.Eng. I Gde Putu Wirarama Wedashwara Wirawan, ST., MT.

   NIP. 19840919 201803 1001

2. Penguji 2

   Tanggal : 01/06/2020

   Ahmad Zafrullah M., S.T., M.Eng.

3. Penguji 3

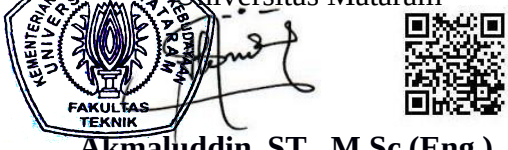
   

   Tanggal : 01/06/2020

   Gibran Satya Nugraha, S.Kom., M.Eng.

   NIP. 19920323 201903 1012

Mataram, 4 Juni 2020
Dekan Fakultas Teknik
Universitas Mataram



Akmaluddin, ST., M.Sc.(Eng.), Ph.D.

NIP. 19681231 1994121 001

HALAMAN PERNYATAAN KEASLIAN TUGAS AKHIR

Saya yang bertanda tangan di bawah ini menyatakan bahwa dalam Tugas Akhir ini tidak terdapat karya yang pernah diajukan untuk memperoleh gelar kesarjanaan di suatu Perguruan Tinggi sepanjang pengetahuan saya, juga tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali secara tertulis diacu dalam naskah ini dan disebutkan dalam daftar pustaka.

Mataram, 1 Juni 2020

Ni Desak Ketut Pujika Dewi

PRAKATA

Puji syukur penulis panjatkan kehadirat Tuhan Yang Maha Pengasih dan Penyayang atas segala berkat, bimbingan, dan karunia-Nya, sehingga penulis dapat menyelesaikan penyusunan Tugas Akhir dengan judul "*Penerapan Algoritma Semut pada Protokol Routing AOMDV untuk Optimasi Pencarian Rute di Jaringan VANET* ".

Tugas Akhir ini dilaksanakan secara online oleh program studi Teknik Informatika Universitas Mataram. Tujuan dari Tugas Akhir ini adalah untuk menerapkan algoritma semut pada protokol *routing* AOMDV, sehingga dapat diketahui kinerja dari protokol *routing* AOMDV yang telah diterapkan algoritma semut dalam melakukan pencarian rute yang optimal di jaringan VANET. Tugas Akhir ini juga merupakan salah satu persyaratan kelulusan guna mencapai gelar kesarjanaan di Program Studi Teknik Informatika, Fakultas Teknik UNRAM.

Akhir kata semoga tidaklah terlampau berlebihan, bila penulis berharap agar karya ini dapat bermanfaat bagi pembaca.

Mataram, 1 Juni 2020

Penulis

UCAPAN TERIMA KASIH

Tugas Akhir ini dapat diselesaikan berkat bimbingan dan dukungan ilmiah maupun materil dari berbagai pihak, oleh karena itu pada kesempatan ini penulis menyampaikan ucapan terimakasih yang setulus-tulusnya kepada :

1. Kedua orang tua saya, dan seluruh keluarga yang telah mendukung dan mendoakan.
2. Bapak Akmaluddin, ST., M.Sc.(Eng.), Ph.D., selaku Dekan Fakultas Teknik Universitas Mataram.
3. Bapak Prof. Dr. Eng. I Gede Pasek Suta Wijaya, ST., MT. selaku Ketua Program Studi Teknik Informatika Fakultas Teknik Universitas Mataram.
4. Bapak Andy Hidayat Jatmika, ST., M. Kom. selaku Dosem Pembimbing Utama yang telah memberikan bimbingan, arahan, serta motivasi kepada penulis selama mengerjakan Tugas Akhir.
5. Bapak Ariyan Zubaidi, S.Kom., M.T. selaku Dosen Pembimbing Pendamping yang telah turut memberikan ide, saran, arahan, serta motivasi kepada penulis dalam mengerjakan Tugas Akhir.
6. Bapak Dr.Eng. I Gde Putu Wirarama Wedashwara Wirawan, ST., MT. selaku Dosen Penguji I, Bapak Ahmad Zafrullah M., S.T., M.Eng., selaku Dosen Penguji II, Bapak Gibran Satya Nugraha, S.Kom., M.Eng., selaku Dosen Penguji III, yang telah memberikan saran dan arahan kepada penulis dalam mengerjakan Tugas Akhir.
7. Teman-teman pejuang MANET, yang turut mendukung dan membantu.
8. Teman-teman seperjuangan Teknik Informatika angkatan 2014, yang turut mendukung dan membantu.
9. Semua pihak yang tidak dapat penulis sebutkan satu persatu, yang telah memberikan bimbingan dan dukungan kepada penulis dalam menyelesaikan Tugas Akhir ini.

Semoga Tuhan Yang Maha Esa memberikan imbalan yang setimpal atas bantuan yang diberikan kepada penulis.

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

ABSTRAK

VANET merupakan salah satu turuanan dari jaringan MANET. Pada jaringan VANET lebih spesifik digunakan untuk berkomunikasi antar kendaraan satu dengan yang lainnya. Dengan adanya jaringan VANET diharapkan mampu meningkatkan keamanan pengemudi di jalan raya. Pada penerapan jaringan VANET dibutuhkannnya sebuah protokol *routing*. Protokol AOMDV merupakan salah satu contoh protokol *routing* yang dapat membantu kinerja dari jaringan VANET. Tujuan dari penelitian ini di lakukan untuk mengoptimalkan proses pencarian rute pada protokol *routing* AOMDV.

Proses pemilihan rute pada protokol *routing* AOMDV berdasarkan jumlah hop yang paling sedikit, sehingga rute tersebut kemungkinan cepat terputus. Untuk mengatasi kekurangan dari protokol *routing* AOMDV dilakukan dengan menerapkan algoritma semut. Keunggulan dari algoritma semut yaitu dalam melakukan pencarian rute dilakukan dengan memperhitungkan jarak antar *node*, sehingga rute tidak cepat terputus. Pemilihan rute pada algoritma semut berdasarkan feromon yang paling banyak, feromon merupakan jejak kaki semut. Penulis melakukan modifikasi pada beberapa kerangka bagian protokol *routing* AOMDV yang diberi nama AOMDV-semut.

Pada penelitian dilakukan 5 kali percobaan. Parameter uji coba yang digunakan untuk menilai kinerja jaringan pada penelitian ini yaitu *throughput*, *Packet delivery ratio (PDR)* dan *end to end delay*. Dari hasil ujicoba yang telah dilakukan pada protokol *routing* AOMDV dengan menggunakan algoritma semut mampu meningkatkan kinerja dari parameter ujicoba *throughput* sebesar 9.7649 Kbps dan kinerja *Packet delivery ratio* sebesar 11.2838%. Sedangkan untuk parameter *average end to end delay* mampu menurunkan *delay* sebesar 13.3093 ms.

Kata kunci: VANET, AOMDV, Algoritma Semut, Protokol *Routing*, Pencarian Rute

ABSTRACT

VANET is a derivative of the MANET network. VANET networks are more specifically used to communicate between one vehicle and another vehicle. With the existence of the VANET network, it is expected to be able to improve the safety of drivers on the highway. In the application of the VANET network, a routing protocol is needed. The AOMDV protocol is an example of a routing protocol that can help the performance of a VANET network. The purpose of this research is to optimize the route search in the AOMDV routing protocol.

The route selection process in the AOMDV routing protocol is based on the least number of hops, so the route is likely to be disconnected quickly. To overcome the shortcomings of AOMDV routing protocol, the ant colony optimization algorithm is applied. The advantage of an ant colony optimization algorithm is that in conducting a route search, performed by calculating the distance between nodes so that the route is not quickly interrupted. Selection of the route on the ant colony optimization algorithm based on the most pheromones, pheromones are ant footprints. The author modifies some of the frameworks of the AOMDV routing protocol section named AOMDV-ant.

In the study conducted 5 experiments. The trial parameters used to assess network performance is throughput, packet delivery ratio (PDR), and average end-to-end delays. From the results of trials that have been carried out on the AOMDV routing protocol using the ant colony optimization algorithm, it can improve the performance of the test throughput parameters of 9.7649 Kbps and the performance of the Packet delivery ratio of 11.2838%. Whereas the average end to end delay parameter can reduce the delay by 13.3093ms.

Keywords: VANET, AOMDV, Algoritma Semut, Protokol *Routing*, Pencarian Rute

BAB I
PENDAHULUAN

1.1 Latar Belakang

Vehicular Ad Hoc Network (VANET) merupakan salah satu jaringan *wireless* yang merupakan turunan dari jaringan MANET. Jaringan VANET adalah suatu jaringan *ad hoc* yang digunakan untuk berkomunikasi antar kendaraan satu dengan kendaraan lainnya [1]. Performa suatu jaringan ditentukan oleh kinerja protokol *routing* yang digunakan, *routing* ialah suatu proses yang dilakukan untuk menetukan rute dari *node* sumber ke *node* tujuan sehingga pada protokol *routing* inilah yang akan bertugas untuk menentukan suatu rute. *Ad Hoc On-Demand Multipath Distance Vector* (AOMDV) merupakan salah satu contoh dari protokol *routing* yang digunakan pada jaringan VANET.

Protokol *routing* AOMDV pada saat *route discovery* atau proses pencarian rute dari *node* sumber ke *node* tujuan dilakukan dengan mengirimkan paket RREQ (*Route Request*) ke *node* tetangganya. Jika paket RREQ telah diterima oleh *node* tujuan, maka *node* tujuan akan mengirimkan paket RREP (*Route Reply*) ke *node* sumber. Pada protokol *routing* AOMDV setiap RREP akan dipertimbangkan oleh *node* tujuan, sehingga beberapa *path* bisa ditemukan dalam satu proses pencarian rute [2]. Proses pemilihan rute pada protokol *routing* AOMDV dilakukan dengan mempertimbangkan jumlah *hop* pada rute tersebut. Jumlah *hop* yang paling sedikit yang akan digunakan untuk memilih rute pada protokol AOMDV. Kekurangan dari protokol *routing* AOMDV yaitu rute yang dipilih tidak memperhitungkan jarak antar *node*, sehingga dapat mengakibatkan rute tersebut memiliki kemungkinan terputus. Untuk mengatasi kekurangan dari protokol *routing* AOMDV tersebut diperlukannya sebuah metode. Algoritma semut merupakan salah satu metode yang digunakan untuk proses pencarian rute yang optimal.

Algoritma semut (*Ant Colony Optimization)* merupakan algoritma yang diambil dari perilaku semut atau koloni semut untuk mencari makanan. Pada algoritma semut memperhitungkan jarak antar *node*, menghitung probabilitas untuk menentukan arah rute yang akan dituju sehingga lintasan semut dapat diketahui. Dalam algoritma semut terdapat istilah feromon yaitu jejak kaki semut yang melintasi rute, koloni semut dapat menemukan rute terpendek dengan mengikuti jejak kaki semut sebelumnya yang telah melintasi rute tersebut. Pada feromon tersebut dilakukan pembaharuan feromon (*update* feromon) untuk mendapatkan rute yang paling sering dilintasi oleh semut. Semakin

banyak semut yang melewati rute tersebut maka semakin jelas feromon atau jejak kaki semut pada lintasan tersebut. Tujuan dari penelitian ini yaitu untuk menemukan proses pencarian rute yang optimal dengan menerapkan algoritma semut pada protokol *routing* AOMDV. Rute yang dipilih adalah berdasarkan jumlah feromon yang paling banyak yang dimiliki oleh algoritma semut, bukan lagi berdasarkan jumlah *hop* yang dimiliki oleh protokol *routing* AOMDV. Protokol *routing* yang telah dimodifikasi diberi nama AOMDV-Semut.

Oleh karena itu pada penelitian ini akan membandingkan kinerja dari protokol *routing* AOMDV standar dengan protokol *routing* AOMDV yang telah dimodifikasi dengan algoritma semut (AOMDV-Semut). Pemilihan peta pada penelitian ini yaitu mengambil peta jalan Kota Mataram, daerah tersebut dipilih karena memiliki karakteristik jalan persimpangan yang lebih dari satu. Simulasi dilakukan dengan menggunakan *Network Simulator* 2 (NS-2) versi 2.35, *Java OpenSteetMap Editor* (JOSM) dan *Simulation of Urban Mobility* (SUMO). NS2 merupakan perangkat lunak didesain secara spesifik untuk penelitian dalam bidang jaringan komunikasi komputer yang bersifat *open source*. JOSM merupakan aplikasi yang digunakan untuk meng-*edit* peta yang diperoleh melalui *OpenStreetMap*. SUMO merupakan aplikasi yang bersifat *open source* digunakan untuk membuat skenario *mobilitas* serta dapat diintegrasikan dengan *OpenStreetMap* untuk menetukan lokasi simulasi yang diinginkan. Parameter uji coba yang digunakan untuk menilai kinerja jaringan adalah *throughput*, *Packet delivery ratio (PDR)* dan *end to end delay*.

1.2 Rumusan Masalah

Berdasarkan uraian latar belakang yang ada maka dirumuskan masalah dalam tugas akhir ini yaitu :

1. Bagaimana menerapkan algoritma semut pada protokol *routing* AOMDV di jaringan VANET?
2. Bagaimana kinerja protokol *routing* AOMDV yang telah diterapkan algoritma semut dalam melakukan pencarian rute yang optimal di jaringan VANET?

1.3 Batasan Masalah

Untuk membatasi ruang lingkup dari permaslahan yang ada, serta agar mencapai tujuan dan sasaran berdasarkan pada rumusan masalah, maka diberikan beberapa batasan masalah yaitu:

1. Protokol *routing* yang digunakan adalah AOMDV (*Ad Hoc On-Demand Multipath Distance Vector*).
2. Metode yang digunakan adalah algoritma semut (*Ant Colony Optimization*).
3. Jumlah *node* yang digunakan 50 *node*, 70 *node* dan 100 *node*.
4. Kecepatan *node* yang digunakan sebesar 30 km/jam, 60 km/jam, 90 km/jam, 120km/jam, 150km/jam.
5. Parameter uji kinerja adalah *Average End to end delay*, *Throughput* dan *Packet delivery ratio* (PDR).
6. Simulasi dilakukan dengan menggunakan *Network Simulator 2* (NS2) versi 2.35, *Simulation of Urban Mobility* (SUMO) versi 0.32.0 dan *Java Open Street Map editor* (JOSM).

1.4 Tujuan Penelitian

Tujuan dari pembuatan tugas akhir ini yaitu sebagai berikut :

1. Menerapkan algoritma semut pada protokol *routing* AOMDV di jaringan VANET.
2. Mengetahui kinerja protokol *routing* AOMDV yang telah diterapkan algoritma semut dalam melakukan pencarian rute yang optimal di jaringan VANET.

1.5 Manfaat Penelitian

Hasil dari penelitian ini diharapkan dapat memperoleh manfaat dalam meningkatkan proses pencarian rute yang optimal dengan menerapkan algoritma semut pada protokol *routing* AOMDV di jaringan VANET.

1.6 Sistematika Penulisan

1. BAB I PENDAHULUAN

Bab ini berisi latar belakang penulisan tugas akhir, rumusan masalah, batasan masalah, tujuan penelitian, manfaat penelitian, dan sistematika penulisan.

2. **BAB II TINJAUAN PUSTAKA DAN LANDASAN TEORI**

   Bab ini menjelaskan mengenai penelitian yang dilakukan sebelumnya dan teori yang berkaitan dengan judul/masalah tugas akhir.

3. **BAB III METODE PENELITIAN**

   Bab ini berisi perencanaan simulasi jaringan dan membahas langkah-langkah penelitian.

4. **BAB IV HASIL DAN PEMBAHASAN**

   Bab ini berisi pelaksanaan simulasi dan hasil analisis data simulasi jaringan.

5. **BAB V KESIMPULAN DAN SARAN**

   Bab ini berisi beberapa kesimpulan yang didapat dan saran-saran berdasarkan hasil analisis dan simulasi jaringan.

BAB II
TINJAUAN PUSTAKA DAN LANDASAN TEORI

2.1 Tinjauan Pustaka

Pada sub bab ini akan dijelaskan mengenai tinjauan terhadap penelitian yang sudah ada yang berkaitan dengan topik tugas akhir dan penelitian yang diusulkan.

2.1.1 Penelitian Terkait

Penelitian yang berjudul "Analisis Performansi *Routing Protocol* OLSR dan AOMDV pada *Vehicular Ad Hoc Network* (VANET)" [2] yaitu melakukan penelitian dengan membandingkan protokol *routing* OLSR dan AOMDV di jaringan VANET. Pengujian simulasi dilakukan dengan menggunakan simulator NS-2.34 dengan luas area 1000m x 1000m, dengan jumlah *node* sebanyak 80 *node*, 120 *node*, 160 *node*, dan 200 *node*, dengan kecepatan *node* 20 km/jam, 30 km/jam, 40 km/jam dan 50 km/jam. Parameter uji coba yang digunakan yaitu *Packet delivery ratio, average throughput, Average End to end delay, normalized routing load, dan routing overhead.* Setelah dilakukan simulasi dan analisa terhadap kedua algoritma *routing protocol* yaitu OLSR dan AOMDV, maka dapat diambil kesimpulan bahwa AOMDV lebih unggul hampir pada semua metrik performansi dengan nilai nilai rata-rata PDR 87.804%, *throughput* 449.565 kbps, *routing overhead* 0.9773, dan NRL 1.1108. Sedangkan pada OLSR memiliki rata-rata PDR 83.539%, *throughput* 427.735 kbps, *routing overhead* 1.4523, NRL 1.7436. Pada metrik performansi *end to end delay* OLSR lebih unggul dengan memiliki nilai rata-rata 4.765 ms. Sedangkan pada AOMDV memiliki nilai 8.215 ms. Karena AOMDV lebih unggul empat dari lima metrik performansi yang diujikan ini dapat menunjukkan bahwa protokol *routing* AOMDV lebih efisien diterapkan pada jaringan *vehicular ad-hoc network* (VANET) pada kondisi perkotaan.

Penelitian yang berjudul "*Performance Improvement of Dynamic Source Routing (DSR) Protocol using Ant Colony Optimization for Vehicular Ad-hoc Network (VANET)"* [3] yaitu melakukan analisis terhadap protokol *routing* DSR standar dengan protokol *routing* DSR yang telah dimodifikasi dengan menggunakan algoritma semut (DSR-Ant). Pengujian simulasi pada protokol DSR dilakukan dengan menggunakan simulator NS-2 dengan luas area 300m x 300m, dan jumlah *node* sebanyak 10 – 40 *node*. Prameter uji coba yang dihitung yaitu *delay*, *jitter*, konsumsi energi, *routing load* dan *Packet delivery*

ratio. Hasil uji simulasi *delay, jitter, routing load,* PDR pada DSR standar dengan nilai rata sebesar 2.25 ms, 1.2 ms, 0.36 Kbps, 83.5%. Sedangkan hasil simulasi *delay, jitter, routing load,* PDR untuk DSR yang telah dimodifikasi (DSR-Ant) mendapatkan nilai rata-rata sebesar 2.025 ms, 0.9 ms, 0.3425 Kbps, 86.75%. Pada hasil simulasi konsumsi energi, DSR standar memiliki kinerja yang lebih baik yaitu 15.75 *joule*, sedangkan DSR-Ant memiliki konsumsi energi yang lebih besar yaitu 18 *joule*. Dengan demikian dapat disimpukan bahwa pada DSR yang telah dimodifikasi dengan algoritma semut lebih unggul empat dari lima parameter uji yang telah di simulasikan, ini dapat menunjukan kinerja DSR-Ant lebih baik dibandingkan DSR standar.

Penelitian yang berjudul *"Comparative Analysis of Various Routing Protocols in VANET"* [4] yaitu melakukan penelitian dengan membandingkan kinerja dari protokol *routing* AODV, AOMDV, DSR dan DSDV di jaringan VANET. Pengujian simulasi dengan menggunakan simulator NS-2.34, jumlah *node* sebanyak 10 *node*, 20 *node*, 30 *node*, 40 *node*, 50 *node* dan 60 *node,* luas area simulasi sebesar 1700m x 1700 m dan parameter uji yang digunakan yaitu *delay, Packet delivery ratio, throughput, normalized routing load, packet loss.* Hasil simulasi *delay* pada protokol *routing* AODV, AOMDV, DSR dan DSDV dengan nilai rata-rata yaitu sebesar 4.48179 m/s, 4.2686 m/s, 3.09288 m/s dan 4.41658 m/s. Hasil simulasi *throughput* pada protokol *routing* AODV, AOMDV, DSR dan DSDV dengan nilai rata-rata yaitu sebesar 858.965 Kbps, 756.248 Kbps, 1833.24 Kbps dan 921.18 Kbps. Hasil simulasi PDR pada protokol *routing* AODV, AOMDV, DSR dan DSDV dengan nilai rata-rata yaitu sebesar 99.2629%, 98.55498%, 76.50597% dan 96.61478%. Hasil simulasi *packet loss* pada protokol *routing* AODV, AOMDV, DSR dan DSDV dengan nilai rata-rata yaitu sebesar 2.02397%, 1.4512%, 24.10258% dan 1.3820%. Hasil simulasi NRL pada protokol *routing* AODV, AOMDV, DSR dan DSDV dengan nilai rata-rata yaitu sebesar 0.501 , 0.5558, 0.462 dan 0.503. Dengan demikian dapat disimpulkan bahwa AOMDV lebih unggul dari hasil uji yang telah dilakukan, dilihat dari hasil uji PDR, *packet loss* dan NRL, sehingga ini dapat menunjukkan bahwa protokol *routing* AOMDV lebih efisien diterapkan pada jaringan *vehicular ad-hoc network* (VANET).

Penelitian yang berjudul "*Ant Colony Optimization based Modified* AOMDV *for Multipath Routing in MANET"* [5] yaitu menerapkan algoritma semut pada *routing* protokol AOMDV di jaringan MANET. Pada penelitian ini memodifikasi protokol *routing* AOMDV dengan menggunakan algoritma semut yang dimana menggunakan

AODV-Ant sebagai perbandingannya. Pengujian simulasi pada dilakukan dengan menggunakan simulator NS-2.34 dengan luas area 800m x 600m, dan jumlah *node* sebanyak 20 *node*. Prameter uji coba yang dihitung yaitu UDP data *transmission analysis,* UDP data *Receives analysis, routing overhead, normalized routing load, Packet delivery ratio, no.of dropped data (packets).* Hasil simulasi UDP data *transmission analysis,* UDP data *Receives analysis, routing overhead, normalized routing load, Packet delivery ratio,* dan *no.of dropped data (packets)* pada AODV-Ant memiliki nilai rata-rata sebesar 5758 Bps, 4870 Bps, 5651 Bps, 1.16, 84.58%, 821 paket *drop*. Sedangkan hasil simulasi UDP data *transmission analysis,* UDP data *Receives analysis, routing overhead, normalized routing load, Packet delivery ratio,* dan *no.of dropped data (packets)* pada AOMDV-Ant memiliki nilai rata-rata sebesar 6694 Bps, 6325 Bps, 4088 Bps, 0.65, 94.49%, 364 paket *drop.* Karena AOMDV-Ant unggul dari ke enam parameter uji yang telah dilakukan, dapat ditarik kesimpulan bahwa AOMDV-Ant memiliki kinerja yang lebih baik dibandingkan dengan AODV-Ant.

Penelitian yang berjudul "*AODV Extension using Ant Colony Optimization for Scalable Routing in VANETs*" [6] yaitu melakukan modifikasi protokol *routing* AODV dengan menggunakan algoritma semut. Algoritma semut memiliki potensi untuk mengoptimalkan protokol *routing* AODV. Tujuan dilakukannya modifikasi ini untuk menghindari *delay* saat berkomunikasi karena terjadinya *disconnect* dalam *routing*, dengan mengunakan algoritma semut untuk mendapatkan rute yang optimal. Penerapan optimasi koloni semut ke dalam protokol *routing* AODV digunakan untuk menentukan rute dan melakukan perbaikan jika terjadi kegagalan rute. Modifikasi AODV dengan algoritma semut dapat mengurangi *routing overhead* dan meningkatkan kinerja dengan menghindari kegagalan rute yang sering terjadi. Simulator yang digunakan pada penelitian ini dengan menggunakan *Network Simulator* 2 (NS-2). Dari hasil simulasi dapat disimpulkan bahwa kinerja algoritma semut pada protokol *routing* AODV dapat meningkatkan proses pencarian rute.

2.1.2 Penelitian yang diusulkan

Pada penelitian terkait sebelumnya telah dijelaskan beberapa keunggulan dari protokol *routing* AOMDV tanpa menggunakan metode dan keunggulan AOMDV dengan mengunakan algoritma semut di jaringan MANET, sehingga pada penelitian ini ingin menerapkan algoritma semut pada protokol *routing* AOMDV di jaringan VANET.

Penerapan AOMDV dengan menggunakan algoritma semut pada jaringan MANET tentu saja berbeda dengan penerapan AOMDV dengan algoritma semut dijaringan VANET. Beberapa perbedaan mencolok antara MANET dan VANET adalah letak dari skenario mobilitasnya, yaitu di dalam MANET memiliki mobilitas yang sangat acak dan lebih susah diprediksi, namun dengan kecepatan yang relatif rendah, tetapi VANET memiliki mobilitas yang sudah terprediksi (sesuai dengan kondisi lingkungan jalan raya) dan memiliki kecepatan yang relatif rendah sampai tinggi (tergantung kepadatan jalan di suatu perkotaan) [7]. VANET sendiri adalah suatu kelompok jaringan yang sama seperti MANET, bedanya yaitu di dalam VANET terdiri dari perangkat bergerak seperti truk, mobil, motor, bus, ataupun yang tidak bergerak seperti lampu lalu-lintas. Hal ini yang menyebabkan perbedaan pola pergerakan antara MANET dan VANET. Penelitian dilakukan dengan menggunakan peta jalan Kota Mataram, peta diambil melalui aplikasi *Java OpenStreetMap Editor*.

Tujuan dilakukan penelitian ini untuk mengoptimalkan proses pencarian rute pada protokol AOMDV dengan memperhitungkan jarak antar *node* yang dimiliki oleh algoritma semut, sehingga rute optimal dapat ditemukan dengan menggunakan nilai feromon pada algoritma semut. Rute yang memiliki jumlah feromon atau jejak kaki terbanyak yang akan dipilih sebagai rute yang optimal, tidak lagi berdasarkan jumlah *hop* pada *routing* AOMDV. Penelitian yang dilakukan akan membandingkan kinerja antara protokol *routing* AOMDV standar (tanpa modifikasi) dengan protokol *routing* AOMDV yang telah dimodifikasi dengan menggunakan algoritma semut. *Output* yang diperoleh dari setiap hasil simulasi berupa *file trace.* Semua *file trace* akan di-*filter* menggunakan AWK *Script* berdasarkan parameter uji yang akan digunakan, yakni *Average End to end delay, throughput* dan *packet delivey ratio* (PDR).

2.2 Landasan Teori

Pada sub bab ini akan dijelaskan tentang teori-teori penunjang jaringan *ad hoc* dan juga teori pemrograman NS2 secara umum.

2.2.1 *Vehicular Ad Hoc Network* (VANET)

VANET merupakan turunan dari MANET (*Mobile Ad Hoc Network*), jaringan VANET dibuat dengan menggunakan kendaraan sebagai *node*. Komunikasi pada VANET meliputi komunikasi kendaraan antar kendaraan (*vehicle to vehicle*) dan komunikasi kendaraan antar perangkat yang terletak dipinggir jalan raya (*vehchile to*

infrastuctur). Pada VANET dilengkapi dengan sensor atau perangkat yang terpasang pada kendaraan, sensor tersebut adalah OBU (*On Board Units*). Begitu pula infrastruktur jalanan yang dapat diikuti dalam komunikasi atau dikenal dengan RSU (*Road Side Unit*) [8]. VANET juga dapat dikatakan sebagai sebuah jaringan terorganisir yang dibentuk dengan menghubungkan antar kendaraan dan RSU (*Road Side Unit*). Lebih lanjut RSU terhubung ke jaringan *backbone* berkecepatan tinggi melalui koneksi jaringan. Kepentingan peningkatan baru-baru ini telah diajukan pada aplikasi melalui komunikasi V2V (*Vehicle to Vehicle*) dan V2I (*Vehicle to Infrastructur*), bertujuan untuk meningkatkan keselamatan mengemudi dan manajemen lalu lintas sementara bagi para pengemudi dan penumpang. Dalam VANET, RSU (*Road Side Unit*) dapat memberikan bantuan dalam menemukan fasilitas seperti restoran dan pom bensin serta mem-*broadcast* pesan yang terkait seperti kecepatan kendaraan kepada pengendara lain. Sebagai contoh sebuah kendaraan dapat terhubung dengan lampu lalu lintas melalui komunikasi V2I dan lampu lalu lintas dapat memberikan informasi ke kendaraan ketika dalam keadaan lampu ke kuning atau merah. Ini dapat berfungsi sebagai tanda pemberitahuan kepada pengemudi dan akan sangat membantu para pengendara ketika mereka sedang berkendara selama kondisi cuaca buruk atau di daerah asing, hal ini dapat mengurangi terjadinya kecelakaan. Melalui komunikasi V2V, pengendara bisa mendapatkan informasi yang lebih cepat dan lebih baik serta mengambil tindakan awal untuk menghadapi situasi yang abnormal. Untuk mencapai hal ini, suatu OBU (*On-Board Unit*) secara teratur menyiarkan pesan yang terkait dengan informasi dari posisi pengendara, waktu sekarang, arah pengemudian, kecepatan, status rem, lampu sen, percepatan/perlambatan, kondisi jalan [1]. Jaringan VANET kelak akan berperan penting dalam pengembangan teknologi *Intelligent Transport System* (ITS) untuk menyediakan aplikasi keamanan bagi pengendara jalan.

Ada beberapa karakteristik dari jaringan VANET yaitu sebagai berikut [9] [10] :

1. Jumlah *node* besar dan mobilitas tinggi yaitu jumlah *node* dan kecepatan kendaraan yang tinggi pada VANET, kecepatan ini nantinya akan mempengaruhi karaktristik dari lapisan komunikasi dan performansi jaringan.
2. Putusnya koneksi jaringan secara berkala yaitu pada jaringan dinamis, putusnya koneksi yang disebabkan perubahan informasi (posisi, arah, dll).
3. Topologi jaringan sering berubah atau dinamis yaitu topologi dari VANET yang cepat berubah disebabkan oleh pergerakan mobil pada keceparan tinggi.

4. Daya dan kapasitas penyimpanan yaitu berbeda dengan MANET. Pada VANET kendaraan modern memiliki daya baterai dan kapasitas *storage* yang tak terbatas sehingga sangat membantu dalam proses komunikasi dan *routing*.
5. Model *mobilitas* dan prediksi yaitu *mobilitas* dari tiap mobil tergantung pada kondisi lalu lintas, kondisi jalan, kecepatan kendaraan, perilaku pengemudi dalam berkendara, dan lain-lain.
6. *Hard Delay Constraints* yaitu aspek keamanan dari aplikasi VANET (seperti kecelakaan, masalah rem, dan lain-lain) harus menjamin kecepatan pengiriman pesan ke *node-node* yang relevan.
7. Interaksi dengan sensor *onboard* yaitu posisi terkini dan pergerakan *node* dapat dengan mudah dikirim dengan sensor *onboard* seperti GPS.

2.2.2 Protokol *Routing*

Protokol *routing* adalah standarisasi yang melakukan kontrol terhadap arah pergerakan *node* dalam meneruskan paket diantara perangkat komputasi, dimana protokol *routing* berfungsi untuk mencarikan *route link* (jalur rute) yang terbaik dari jalur yang akan dilalui melalui mekanisme pembentukan tabel *routing*. Pemilihan *router* terbaik tersebut didasarkan atas beberapa pertimbangan seperti bandwith link dan jaraknya [11].

Dalam jaringan VANET terdapat beberapa kategori protokol *routing* yaitu [12]:

1. Protokol *routing* proaktif

Protokol *routing* proaktif yaitu masing-masing *node* akan memiliki table *routing* yang lengkap. Sebuah *node* dalam antrian akan mengetahui semua rute *node* lain yang berada dalam jaringan tersebut. Setiap *node* secara periodik akan melakukan *update table routing* yang dimilikinya, sehingga perubahan topologi jaringan dapat diketahui setiap interval waktu. Contoh protokol *routing* proaktif yaitu DSDV (*Destination Sequenced Distance Vector*), CSGR (*Cluster Switch Gateway Routing*), WRP (*Wireless Routing Protocol*), dan OLSR (*Optimized Linkstate Routing*).

2. Protokol *routing* reaktif

Protokol *routing* reaktif merupakan proses pencarian rute hanya akan dilakukan ketika dibutuhkan komunikasi antar *node* sumber dengan *node* tujuan. Tabel *routing* yang dimiliki oleh sebuah *node* berisi informasi rute ke *node* tujuan saja. Contoh protokol *routing* reaktif yaitu DSR (*Dynamic Source Routing*), AODV (*Ad Hoc On-demand*

Distance Vector), TORA (*Temporally Ordered Routing Algorithm*), ABR (*Associaty Based Routing*), dan SSR (*Stabillity Routing*).

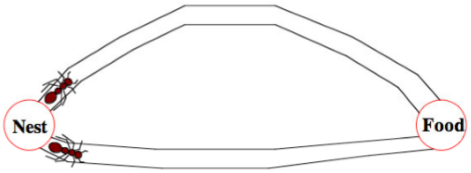
2.2.3 Protokol *Routing* AOMDV

Protokol *routing Ad Hoc On-Demand Multipath Distance Vector* (AOMDV) memiliki banyak persamaan karakteristik dengan Protokol *routing* AODV. AOMDV memiliki konsep berbasis vektor dan menggunakan pendekatan *hop by hop.* AOMDV juga memiliki dua fitur utama yang mirip dengan AODV yaitu *route discovery* dan *route maintenance*. Perbedaan utama antara AODV dan AOMDV adalah jumlah rute yang ditemukan di setiap pencarian rute atau *route discovery* [13]. *Route discovery* dilakukan jika *node* sumber memerlukan rute untuk melakukan komunikasi dengan *node* tujuan, maka *node* sumber akan mengirimkan paket REEQ (*Route Request*) secara *broadcast* ke *node-node* tetangga di dalam jaringan. Pada protokol *routing* AOMDV ini menerima semua salinan paket RREQ untuk *reverse path*. Ketika *intermediate node* menerima paket RREQ, *node* ini akan mengecek apakah ada satu atau lebih *forward path* ke *node* tujuan yang *valid*. Jika ada, *node* ini akan membuat paket RREP (*Route Reply*) dan mengirim kembali ke *node* sumber melalui *reverse path*. Jika tidak ada, maka *intermediate node* akan meneruskan paket RREQ hingga ke *node* tujuan, kemudian *node* tujuan akan membalas dengan paket RREP.

2.2.4 Algoritma Semut

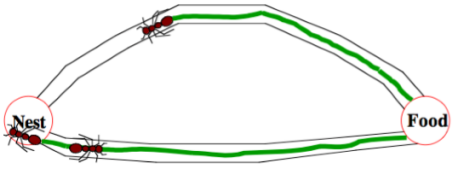
Algoritma semut diperkenalkan oleh Moyson dan Manderick dan secara meluas dikembangkan oleh Marco Dorigo. *Ant colony optimization* atau algoritma semut termasuk dalam kelompok *Swarm Intelligence*, yang merupakan salah satu jenis pengembangan paradigma yang digunakan untuk menyelesaikan masalah optimasi dimana inspirasi yang digunakan untuk memecahkan masalah tersebut berasal dari perilaku kumpulan atau kawanan (*swarm*) serangga. *Swarm intelligence* (SI) adalah kecerdasan buatan yang didasarkan pada perilaku kolektif dari sistem yang terdesentralisasi dan otonom. Algoritma semut biasanya digunakan untuk menyelesaikan *discrete optimization problems* dan persoalan yang kompleks dimana terdapat banyak variabel. Algoritma semut sudah diterapkan di berbagai masalah seperti penjadwalan proyek dengan sumberdaya terbatas, data mining, penjadwalan pekerjaan (*job scheduling*) dan beberapa masalah kombinatorial yang lain [14].

Algoritma ini terinspirasi oleh perilaku semut dalam menemukan jalur dari koloninya menuju makanan. Semut mampu mengindera lingkungannya yang kompleks untuk mencari makanan dan kemudian kembali ke sarangnya dengan meninggalkan zat feromon pada jalur-jalur yang mereka lalui. Feromon merupakan zat kimia yang berasal dari kelenjar endokrin dan digunakan oleh makhluk hidup untuk mengenali sesama jenis, individu lain, kelompok, dan untuk membantu proses reproduksi. Feromon menyebar ke luar tubuh dan hanya dapat dikenali oleh individu lain yang sejenis (satu spesies). Proses peninggalan feromon ini dikenal sebagai *stigmergy*. *Stigmergy* ini merupakan sebuah proses memodifikasi lingkungan yang tidak hanya bertujuan untuk mengingat jalan pulang ke sarang, tetapi juga memungkinkan para semut berkomunikasi dengan koloninya.



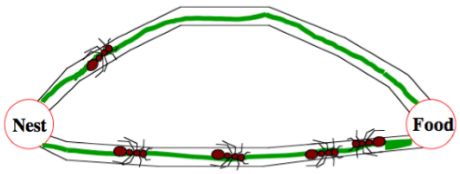
Gambar 2.1 Semut menyebar melewati rute menuju sumber makanan [3]

Pada Gambar 2.1 merupakan ilustrasi saat semut menyebar mencari rute menuju sumber makanan. Semula semut akan menyebar ke setiap rute secara acak hingga menemukan makanan.



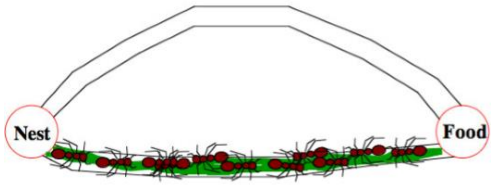
Gambar 2.2 Semut kembali ke sarang membawa makanan [3]

Pada Gambar 2.2 merupakan ilustrasi saat semut kembali ke sarang membawa makanan. Setelah menemukan makanan, semut akan kembali ke sarangnya untuk memberikan tanda dengan jejak feromon bahwa rute tersebut telah dilalui, feromon ini menjadi semacam sinyal bagi sesama semut. Jika semut lain menemukan jejak feromon pada rute tersebut, mereka tidak akan berpergian secara acak, tetapi mengikuti jejak yang sudah ada.



Gambar 2.3 Semut kembali menuju sumber makanan [3]

Pada Gambar 2.3 merupakan ilustrasi saat semut kembali menuju sumber makanan mengikuti jejak feromon sebelumnya. Semut akan kembali dan menguatkan jejak tersebut untuk menemukan makanan. Jika terdapat semut yang secara tidak sengaja menemukan rute baru yang lebih optimal yang akan menempuh rute lebih cepat dari sebelumnya dan rute tersebut lebih sering dilalui maka dengan sendirinya meninggalkan feromon lebih banyak dari rute yang lebih lambat ditempuh.



Gambar 2.4 Semut menemukan rute terpendek menuju sumber makanan [3]

Pada Gambar 2.4 merupakan ilustrasi saat semut telah menemukan rute terpendek menuju sumber makanan. Pada saat semut berikutnya hendak melewati suatu rute menuju sumber makanan maka semut akan melihat terlebih dahulu rute yang akan dilalui. Semut akan memilih rute yang memiliki sinyal yang paling kuat sehingga rute terpendek dapat ditemukan. Semakin banyak semut yang lewat suatu rute, semakin kuat sinyal pada rute tersebut.

2.2.5 *Simulation of Urban Mobility* (SUMO)

SUMO adalah paket simulasi lalu lintas mikroskopis portabel yang dirancang untuk menangani jaringan jalan besar. Paket simulasi lalu lintas pada SUMO bersifat *open source* yaitu memiliki kelengkapan fitur dan permodelannya, termasuk kemampuan jalannya jaringan untuk membaca format yang berbeda, permintaan dengan skala besar hingga pengaturan *routing* dan semua yang berhubungan dengan penelitian pergerakan dalam lalu lintas yang berfokus pada daerah penduduk padat (*urban*) [15]. Alasan utama untuk pengembangan *open source*, simulasi lalu lintas jalan mikroskopis yaitu untuk mendukung komunitas riset lalu lintas dengan alat di mana algoritma sendiri dapat diimplementasikan dan dievaluasi. Ini memungkinkan pengguna untuk membuat topologi jalan dengan gerakan kendaraan sesuai dengan kebutuhan pengguna dan juga pengguna dapat menentukan jalur yang digunakan, kecepatan, atau posisi dapat ditentukan.

2.2.6 *OpenSreetMap* (OSM)

OSM merupakan sebuah contoh peta dunia yang bersifat *open source* yang dapat diakses atau dapat mengubah data untuk mebuat peta yang lebih akurat terperinci dan *up-to-date* secara gratis oleh siapapun. OSM sebagai "Wikipedianya Peta" memiliki peta yang lebih rinci daripada *Google Maps*, karena setiap menit terdapat pembaharuan pada peta OSM. Hal ini terjadi karena lebih dari dua juta yang terdaftar *editor* OSM yang terus menerus membuat perubahan peta di seluruh dunia. OSM juga dapat digunakan untuk pemetaan jalan dan jalan satu arah dan petunjuk untuk area tertentu. Peta adalah simbol visual dunia kita, OSM juga digunakan untuk mengembangkan model 3D bangunan [16].

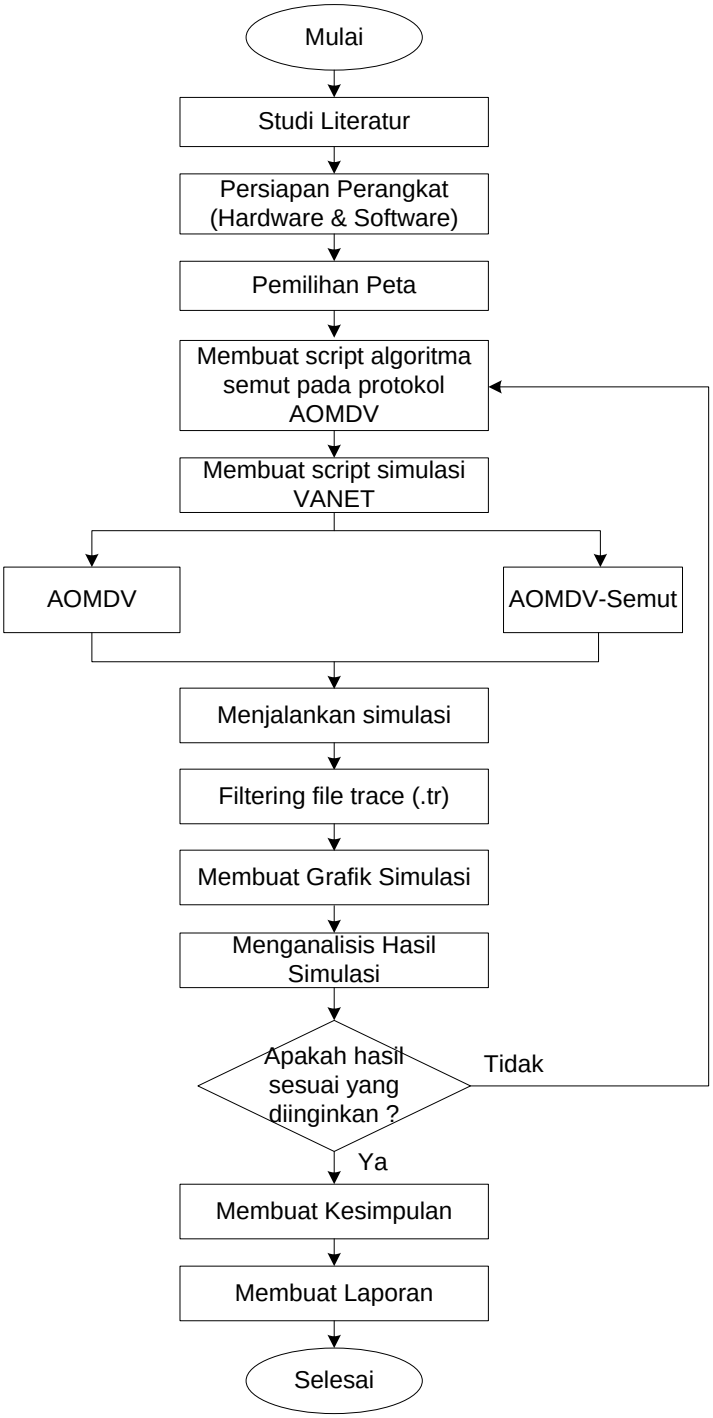
2.2.7 *Network Simulator* 2 (NS2)

Network Simulator (NS-2) adalah suatu interpreter yang *object-oriented*, dan *discrete event-driven* yang dikembangkan oleh *University of California Berkeley* dan USC ISI sebagai bagian dari projek *Virtual INternet Testbed* (VINT). NS menjadi salah satu *tool* yang sangat berguna untuk menunjukkan simulasi jaringan melibatkan Local *Area Network* (LAN), *Wide Area Network* (WAN), tapi fungsi dari *tool* ini telah berkembang selama beberapa tahun belakangan ini untuk memasukkan didalamnya jaringan nirkabel (*wireless*) dan juga jaringan *ad hoc*. *Network Simulator* pertama kali dibangun sebagai varian dari REAL *Network Simulator* pada tahun 1989 di *University of California Berkeley* (UCB). Pada tahun 1995 pembangunan *Network Simulator* didukung oleh *Defense Advanced Research Project Agency* (DARPA) melalui VINT *Project*, yaitu sebuah tim riset gabungan yang beranggotakan tenaga-tenaga ahli dari beberapa instansi ternama. Ada beberapa keuntungan menggunakan NS sebagai perangkat lunak simulasi pembantu analisi dalam riset, antara lain adalah NS dilengkapi dengan *tool* validasi yang digunakan untuk menguji kebenaran pemodelan yang ada pada NS. Secara default, semua pemodelan NS akan dapat melewati proses validasi ini. Pemodelan media, protokol dan komponen jaringan yang lengkap dengan perilaku trafiknya sudah disediakan pada *library* NS. NS juga bersifat *open source* dibawah *Gnu Public License* (GPL), sehingga NS dapat di-*download* dan digunakan secara gratis melalui *website* NS yaitu http://www.isi.edu/nsnam/. Sifat *open source* juga mengakibatkan pengembangan NS menjadi lebih dinamis [17].

BAB III
METODE PENELITIAN

3.1 Diagram Alir Penelitian

Berikut ini merupakan alur kerja dari penelitian yang akan dilakukan dapat dilihat pada Gambar 3.1.



Gambar 3.1 Diagram alir penelitian

Pada Gambar 3.1 merupakan langkah-langkah kerja yang akan dilakukan dalam penelitian tugas akhir ini. Adapun alur kerjanya sebagai berikut:

a. Studi Literatur

Pada penelitian ini dilakukan pembelajaran terhadap penelitian-penelitian terkait yang telah dilakukan sebelumnya sebagai dasar dalam melakukan penelitian yang akan dilakukan. Sumber penelitian sebelumnya dapat berupa paper, skripsi, tesis, maupun buku yang dapat menunjang serta mempermudah penelitian yang akan dilakukan.

b. *Hardware* dan *Sofware*

Perangkat keras yang digunakan pada penelitian ini dengan menggunakan Laptop ASUS A455L dengan spesifikasi perangkat yang dapat dilihat pada Tabel 3.1:

Tabel 3.1 Spesifikasi perangkat keras yang digunakan

Komponen	Spesifikasi
CPU	Intel® Core i3-4030U CPU @ 1.9 GHz
Sistem Operasi	*Linux Ubuntu* -14.04.4 LTS 32 *bit*
Memori	RAM 6 GB
Harddisk	500 GB

Adapun perangkat lunak yang digunakan pada penelitian ini adalah sebagai berikut :

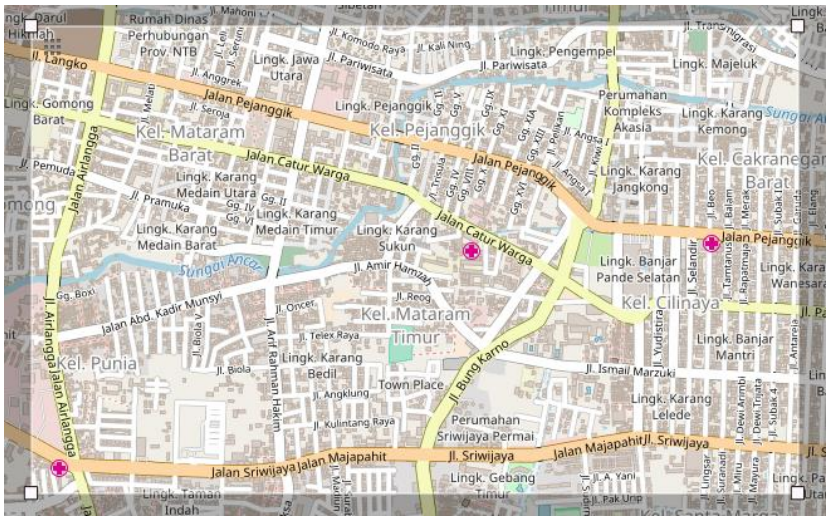
- *Network Simulator 2* (NS2) versi 2.35 untuk melakukan simulasi VANET.
- *Simulation of Urban Mobility* (SUMO) untuk membuat skenario mobilitas VANET.
- *Java OpenStreetMap* (JOSM) untuk menyunting peta yang didapat dari *OpenStreetMap*.
- *Microsoft Office Excel* untuk membuat grafik hasil uji coba.

c. Pemilihan Peta

Pada Gambar 3.2 merupakan pemilihan peta yang akan digunakan pada penelitian yang akan dilakukan dengan mengambil peta dari *OpenStreetMap* (OSM). Berdasarkan observasi yang telah dilakukan, pada kota Mataram terdapat ruas jalan yang menjadi kemacetan lalu lintas. Salah satu contoh titik terjadinya kepadatan kendaraan pada kota Mataram yaitu khususnya jalan yang terhubung antara Jl. Pejanggik, Jl. Catur Warga, Jl. Panca Usaha, Jl. Bung Karno, Jl. Sriwijaya Majapahit.

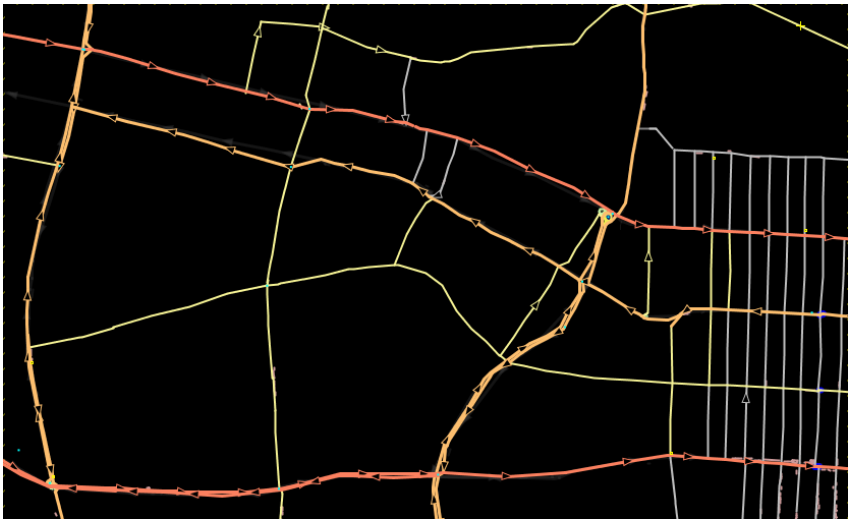
Kemacetan bisa saja terjadi karena pada area tersebut merupakan area perkantoran dan perdagangan, sehingga banyak yang melewati jalan tersebut. Peta jalan dapat dilihat

dari website *http://openstreetmap.org*. Hasil dari proses ini berupa peta daerah Kota Mataram yang telah dipilih dengan format .osm.



Gambar 3.2 Pengambilan peta jalan di *openstreetmap.org*

Pada Gambar 3.3 merupakan peta yang telah di-*edit* pada aplikasi JOSM. Peta tersebut yang nantinya akan digunakan untuk konfigurasi mobilitas sesuai skenario yang telah dirancang dengan menggunakan SUMO.



Gambar 3.3 Peta jalan yang digunakan

d. Membuat *Script* Algoritma Semut pada Protokol AOMDV

Pada tahap ini, dilakukan modifikasi *script* pada protokol AOMDV dengan menggunakan algoritma semut. Modifikasi ini diharapkan dapat mengoptimalkan kinerja protokol AOMDV dalam melakukan pencarian rute pada jaringan VANET. Modifikasi yang dilakukan yaitu dengan melakukan perubahan pada proses *route discovery* AOMDV.

e. Membuat *Script* Simulasi VANET

Pada tahap ini dilakukan perancangan simulasi jaringan yaitu pada jaringan VANET. Dalam membuat *script* simulasi jaringan digunakan bahasa pemerograman TCL (*Tool Command Language*). Bahasa pemerograman TCL merupakan bahasa pemerograman yang berdasarkan pada *string.*

f. *Filtering File Trace*

Pada tahap ini didapatkan hasil simulasi jaringan dengan menggunakan NS-2 dan menghasilkan *file trace* (.tr). Setelah itu dilakukan *filtering* terhadap *file trace* dengan menggunakan bahasa pemrograman AWK. Kemudian hasil *filtering* dari *file trace* berupa parameter uji. Parameter uji yang digunakan yaitu *Average End to end delay*, *throughput* dan *Packet delivery ratio*.

g. Membuat Grafik Simulasi

Pada tahap ini nilai yang didapatkan setelah dilakukan *filtering file trace* kemudian dibuat dalam bentuk grafik agar memudahkan dalam melakukan analisis. Dalam membuat grafik hasil uji coba simulasi dengan mengguanakan *Microsoft Excel*, dimana datanya diperoleh dari proses *filtering* dari *file trace* yang berisikan parameter uji.

h. Menganalisis Hasil Simulasi

Pada tahap ini dilakukan proses analisis terhadap hasil yang diperoleh dari penelitian yang dilakukan. Hasil yang diperoleh untuk mengetahui mengetahui bagaimana pengaruh algoritma semut pada protokol AOMDV dalam jaringan VANET. *Script* AWK digunakan untuk menganalisis parameter uji kinerja yang menjadi acuan yaitu *throughput, Average End to end delay* dan *Packet delivery ratio.* Jika hasil yang diperoleh tidak sesuai memenuhi kualitas kinerja yang baik yang ditentukan dari parameter uji maka akan dilakukan perancangan simulasi kembali.

i. Membuat Kesimpulan

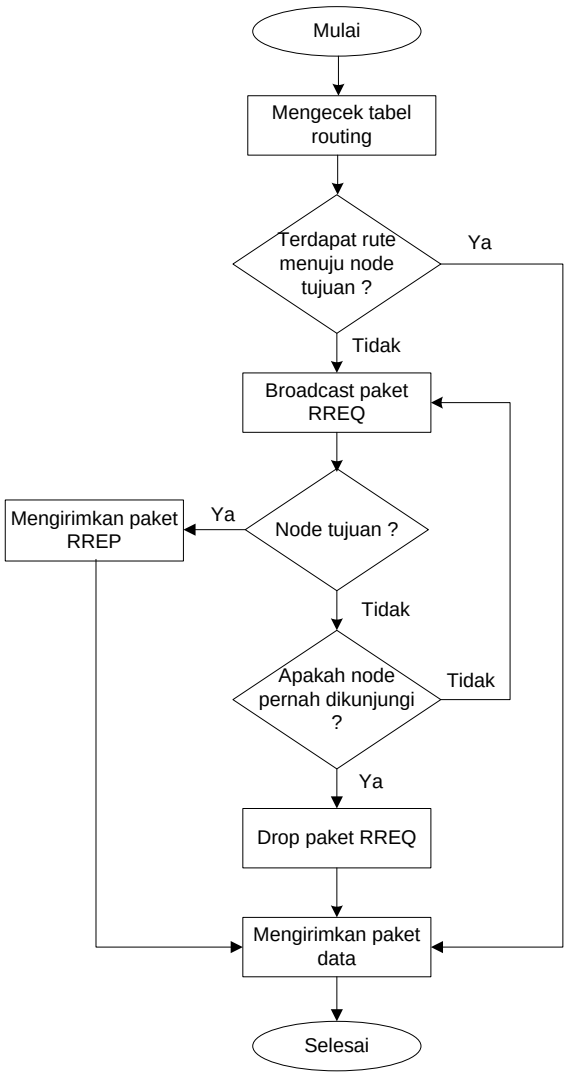
Pada tahap ini akan dilakukan penarikan kesimpulan terhadap penelitian yang telah dilakukan. Penarikan kesimpulan dilakukan agar dapat mengetahui kelebihan dan kekurangan dari suatu penelitian, sehingga kinerja protokol *routing* AOMDV dengan algoritma semut di jaringan VANET dapat diketahui.

j. Membuat Laporan

Pada tahap ini membuat laporan atau dokumentasi terhadap penelitian yang dilakukan. Dokumentasi laporan ini diharapkan agar dapat membantu dalam penelitian berikutnya yang berkaitan dengan penelitian yang dilakukan.

3.2 Algoritma Penemuan Rute Menggunakan Protokol *Routing* AOMDV

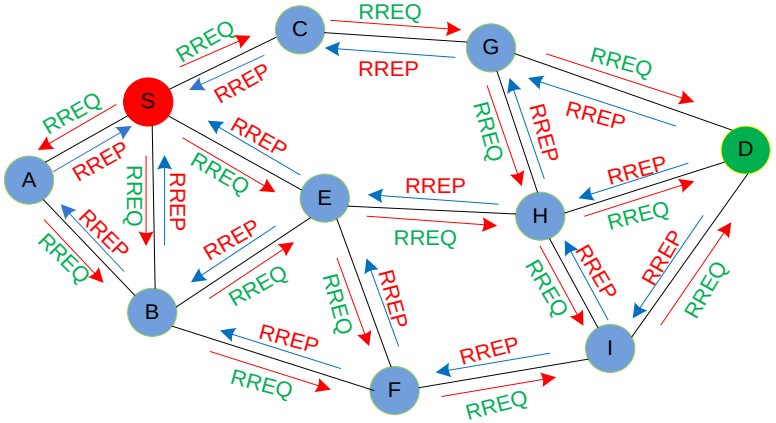
Protokol *routing* AOMDV (*Ad Hoc On-Demand Multipath Distance Vector Routing*) adalah protokol *routing* reaktif yang merupakan pengembangan dari protokol *routing* AODV. AOMDV juga menyediakan dua mekanisme kerja yaitu *route discovery* dan *maintenance*. Pada protokol *routing* AOMDV berbasis vektor dan menggunakan pendekatan *hop-by-hop*. AOMDV juga hanya melakukan pencarian rute ketika dibutuhkan dengan menggunakan prosedur *route discovery*.



Gambar 3.4 Diagram alir proses *route discovery* AOMDV

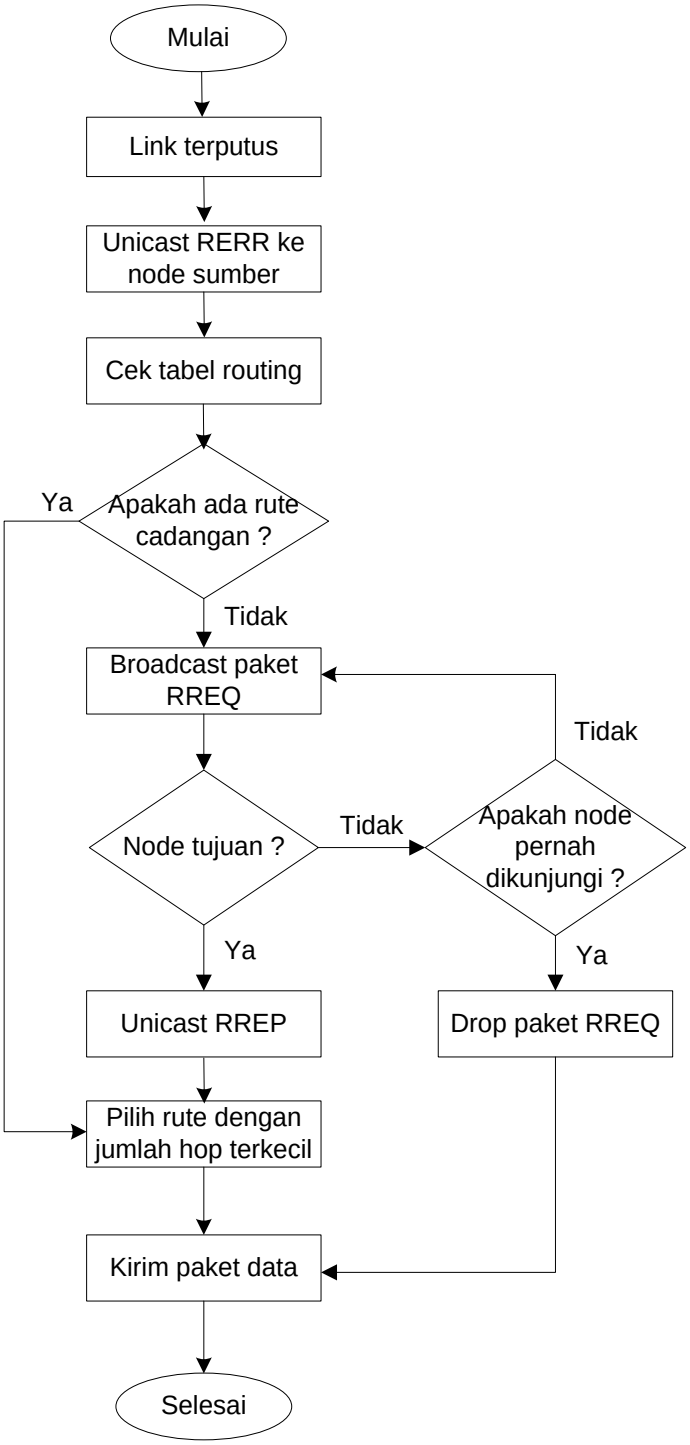
Pada Gambar 3.4 merupakan diagram alir proses pencarian rute pada protokol *routing* AOMDV ketika *node* sumber memerlukan rute untuk mengirimkan paket ke *node* tujuan. *Node* sumber akan memeriksa pada tabel *routing* terlebih dahulu untuk mengirimkan paket data menuju *node* tujuan. Jika rute tersedia maka akan dipilih rute terdekat berdasarkan jumlah *hop* terkecil untuk mengirimkan paket data. Jika tidak ada

maka akan dilakukan pencarian rute dengan cara mem-*broadcast* paket RREQ *node* terdekatnya dan melakukan *set up reverse path*. Setiap *node* yang dikunjungi akan diperiksa terlebih dahulu sebelum melanjutkan *broadcast* RREQ. Jika *node* yang dikunjungi merupakan *node* tujuan, maka akan dilakukan pengiriman RREP melalui *reverse path* yang sudah di *set up* sebelumnya. Jika bukan *node* tujuan maka akan dicek kembali apakah *node* tersebut sudah pernah dikunjungi sebelumnya. Jika pernah dikunjungi, maka paket RREQ akan di *drop*. Jika belum pernah dikunjungi, maka proses *broadcast* RREQ akan dilanjutkan sampai *node* tujuan ditemukan.



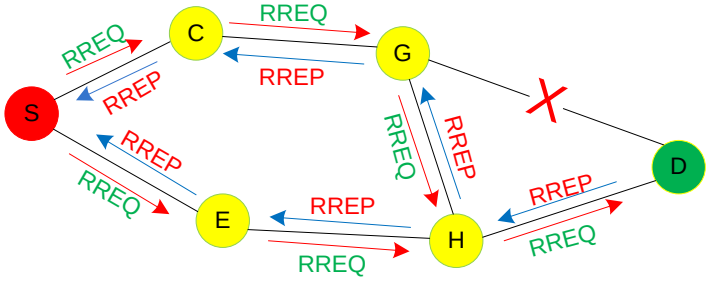
Gambar 3.5 Ilustrasi *route discovery* AOMDV

Pada Gambar 3.5 merupakan ilustrasi proses pencarian rute pada protokol *routing* AOMDV. *Node* S akan mengirimkan paket data ke *node* D, karena pada tabel *routing node* S tidak terdapat rute cadangan ke *node* D, maka *node* harus mem-*broadcast* paket RREQ ke *node-node* disekitarnya untuk menemukan rute menuju *node* D. *Node* S mem-*broadcast* paket RREQ ke *node* A, B, C dan E, kemudian *node* A, B, C, E juga akan mem-*broadcast* paket RREQ ke *node-node* tetangga mereka masing-masing hingga sampai di *node* D. *Node-node* yang berada diantara *node* S dan *node* D merupakan *intermediate node* yang akan melakukan *set up reverse path* untuk mengingat *node-node* yang telah dilewati untuk mencapai *node* D. Selanjutnya, *node* D akan mengirimkan paket RREP sebagai balasan dari paket RREQ ke *node* S secara *unicast* melalui masing-masing rute. Dengan demikian, diperoleh empat rute, yakni S→A→B→F→I→D, S→B→F→I→D, S→C→G→D dan S→E→H→D. Rute yang terpilih adalah rute yang memiliki jumlah *hop* terkecil, yakni rute S→C→G→D dan S→E→H→D. Sedangkan, rute yang tidak terpilih akan dijadikan rute cadangan.



Gambar 3.6 Diagram alir proses *route maintenance* AOMDV

Pada Gambar 3.6 merupakan proses route maintenance dari ptotokol *routing* AOMDV. *Route maintenance* merupakan proses pemeliharaan rute. *Node* akan mengirimkan RERR (*Route Error*) ke *node* sumber secara *unicast* jika link yang menghubungkan antara *node* sumber dan *node* tetangganya terputus. Kemudian tabel *routing node* sumber akan di cek, apakah terdapat rute cadangan ke *node* tujuan yang sama. Jika ada, maka rute dengan jumlah *hop* terkecil akan dipilih dan paket data dapat langsung dikirimkan. Namun, apabila tidak terdapat rute cadangan, maka akan dilakukan proses *route discovery*.



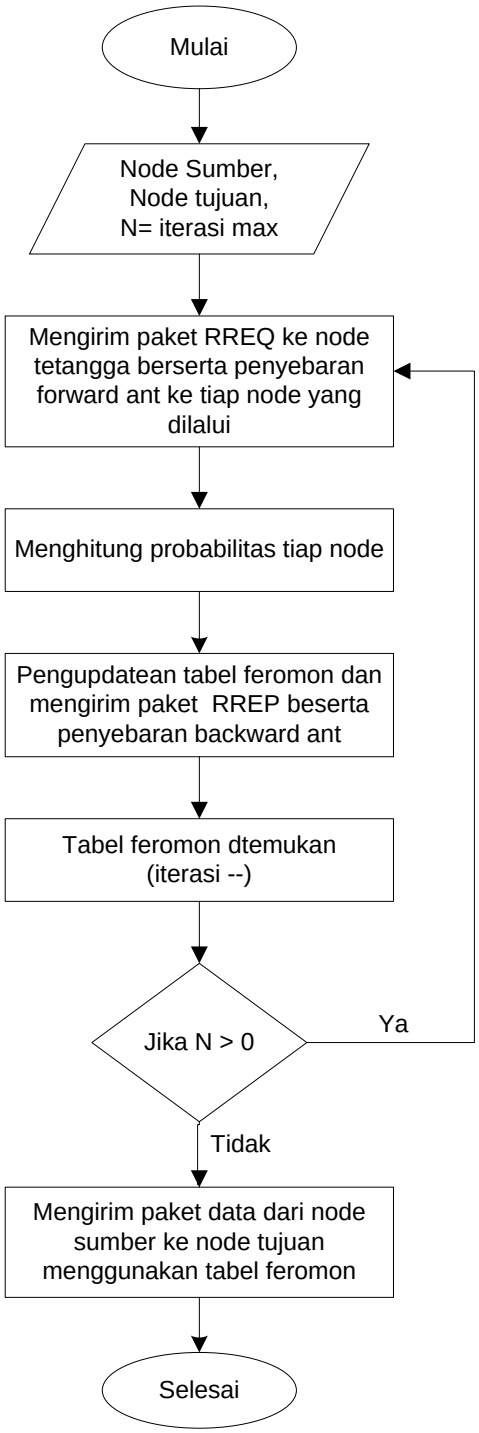
Gambar 3.7 Ilustrasi proses *route maintenance* AOMDV

Pada Gambar 3.7 merupakan ilustrasi dari proses *route maintenance* pada protokol *routing* AOMDV. Dimana pada saat terjadi kegagalan rute, maka akan dikirim paket *Route Error* (RERR), jika terdapat jalur yang rusak, protokol AOMDV akan memilih rute cadangan (*multipath*) apabila ada, kemudian *node* sumber akan kembali melakukan pencarian rute.

3.3 Algoritma AOMDV-Semut

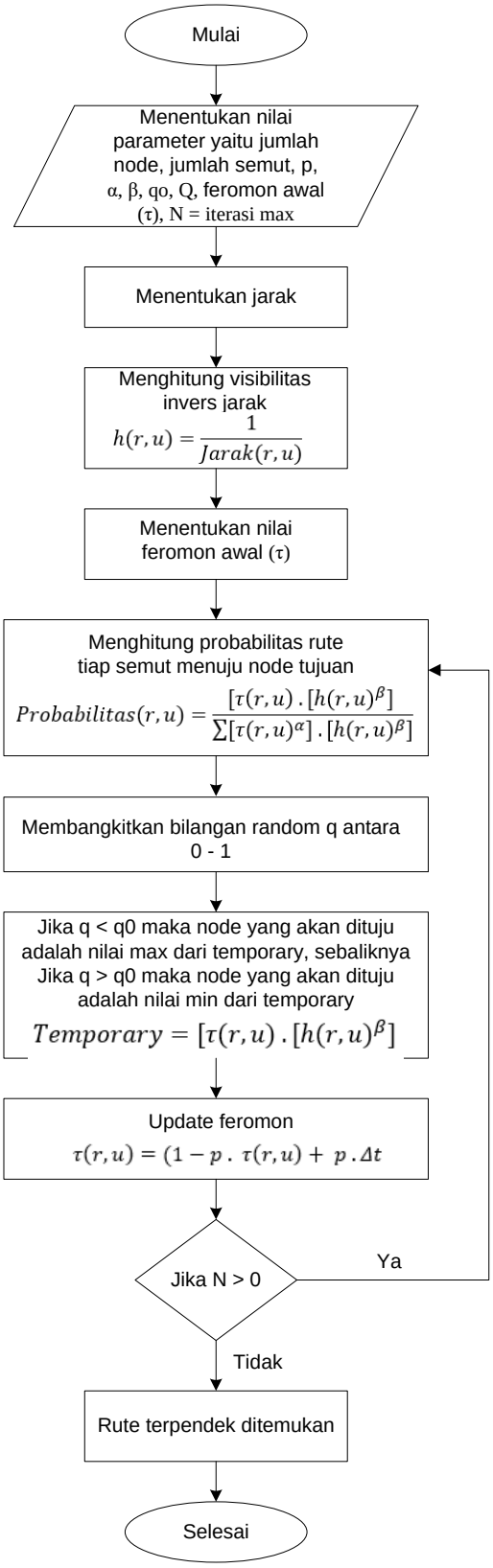
Pada protokol *routing* AOMDV mempunyai tujuan yang sama dengan algoritma semut yaitu sama-sama mencari rute yang paling optimal dari *node* sumber ke *node* tujuan. Pada algoritma semut memiliki dua mekanisme kerja yaitu *forward ant* dan *backward ant. Forward ant* sama halnya dengan paket RREQ pada protokol *routing* AOMDV dan *backward ant* sama halnya dengan paket RREP pada protokol *routing* AOMDV. Penerapan pada *routing* jaringan, sarang semut merupakan *node* sumber dan sumber makanan adalah *node* tujuan.

Pada Gambar 3.8 merupakan proses pencarian rute pada protokol *routing* AOMDV dengan menggunakan algoritma semut.



Gambar 3.8 Diagram alir proses AOMDV-Semut

Adapun proses kerja dari algoritma semut dijelaskan pada Gambar 3.9 yaitu sebagai berikut :



Gambar 3.9 Diagram alir proses algoritma semut

Pada Gambar 3.9 merupakan proses kerja dari algoritma semut. Adapun tahap-tahap yang dilakukan yaitu:

1. Tahap pertama yaitu menentukan atau menginisialisasikan nilai parameter awal, parameter-parameter tersebut adalah [18] [19]:
   a. Banyak *node* atau jumlah *node*
   b. Banyak semut
   c. Tetapan siklus-semut (Q)
   d. Tetapan pengendali intensitas jejak semut (α), nilai $\alpha \geq 0$
   e. Tetapan pengendali visibilitas (β), nilai $\beta \geq 0$
   f. Tetapan penguapan jejak semut (ρ), nilai ρ harus > 0 dan < 1 untuk mencegah jejak feromone yang tak terhingga.
   g. Parameter perbandingan eksploitasi terhadap eksplorasi (q0)
   h. Nilai feromon awal (τ)
2. Kemudian memperhitungkan koordinat (x,y) atau jarak antar *node* untuk mengetahui jarak *node* satu dengan *node* lainnya.
3. Setelah semut diinisialisaikan kemudian semut ditempatkan pada *node* pertama tertentu secara acak. Semut ini disebut paket *forward ant*, karena akan menelusuri rute hingga ke *node* tujuan. Ketika sudah menemukan *node* tujuan, semut ini akan kembali melewati rute yang sama. Pada kondisi ini semut tersebut disebut *backward ant*.
4. Selanjutnya yaitu menghitung *invers* jarak atau *visibilitas* antar *node* yang akan dilakukan pengisian kedalam tabu *list* yang berisikan informasi dari *visibilitas* antar *node*. Dapat dilihat pada persamaan 3.1 yaitu sebagai berikut [18]:

$$h(r,u) = \frac{1}{Jarak(r,u)} \tag{3.1}$$

5. Menentukan nilai feromon awal dan memasukkannya kedalam tabu list. Nilai dari semua feromon (τ) pada awal perhitungan ditetapkan dengan angka awal yang sangat kecil, misal 0.001.
6. Selanjutnya menghitung probabilitas rute tiap semut menuju *node* tujuan. Koloni semut yang sudah terdistribusi ke sejumlah atau setiap *node*, akan mulai melakukan perjalanan dari *node* pertama masing-masing sebagai *node* asal dan salah satu *node*

lainnya sebagai *node* tujuan. Untuk menentukan *node* tujuan dapat dilihat pada persamaan 3.2 yaitu sebagai berikut [18]:

$$Probabilitas(r,u) = \frac{[\tau(r,u)\ .[h(r,u)^{\beta}]}{\sum[\tau(r,u)^{\alpha}]\ .[h(r,u)^{\beta}]} \quad (3.2)$$

7. Membangkitkan bilangan *random* q yaitu koloni semut akan melanjutkan perjalanan dengan memilih salah satu dari *node - node* yang tidak terdapat pada tabu *list* sebagai *node* tujuan selanjutnya. Perjalanan koloni semut berlangsung terus menerus sampai semua *node* satu persatu dikunjungi atau telah menempati tabu *list*.
8. Membandingkan nilai dari bilangan *random* q dengan bilangan q0 yang telah ditetapkan diawal. Jika q>q0 maka *node* selanjutnya yang dipilih adalah nilai *temporary* yang paling kecil, jika q<q0 maka *node* selanjutnya yang dipilih adalah nilai *temporary* yang paling besar. Dapat dilihat pada persamaan 3.3 yaitu sebagai berikut [18]:

$$Temporary = [\tau(r,u)\ .[h(r,u)^{\beta}] \quad (3.3)$$

9. Setelah rute dari semut 1 hingga ke-n dari siklus pertama ditemukan, maka dilakukan pembaharuan feromon atau *update* feromon. Koloni semut akan meninggalkan jejak-jejak kaki pada lintasan antar *node* yang dilaluinya. Adanya penguapan dan perbedaan jumlah semut yang lewat, menyebabkan kemungkinan terjadinya perubahan harga intensitas jejak kaki semut antar *node*. Dapat dilihat pada persamaan 3.4 yaitu sebagai berikut [18]:

$$\tau(r,u) = (1 - p\ .\ \tau(r,u) + p\ .\Delta t \quad (3.4)$$

Δt adalah perubahan harga intensitas jejak kaki semut antar *node* setiap semut yang dihitung berdasarkan persamaan 3.5 [19] :

$$\Delta t = \frac{Q}{Lk} \quad (3.5)$$

Lk merupakan panjang jarak dari rute yang telah ditemukan.

10. Lakukan ulang langkah 6 hingga iterasi atau perulangan selesai, sehingga rute terpendek dapat ditemukan dengan menilai dari banyak feromon yang dilalui semut pada rute tersebut.

3.4 Ruang Lingkup Simulasi

Ruang lingkup simulasi pada penelitian ini terdiri dari dua parameter yaitu parameter *Quality of Service* (QoS) dan parameter skenario simulasi. Pada sub bab 3.4.1 dan 3.4.2 akan menjelaskan tentang parameter tersebut.

3.4.1 Parameter Uji atau *Quality of Service* (QoS)

Parameter QoS merupakan metode pengukuran untuk mengetahui tingkat kinerja atau kualitas dari ssuatu jaringan dilihat dari hasil kolektif dan beragam performansi yang digunakan sebagai patokan. Beberapa parameter yang akan digunakan untuk menguji kinerja pada penelitian ini yaitu sebagai berikut:

1. *Average End-to-end delay*

Delay merupakan waktu pengiriman paket data dari sumber ke penerima, *delay* akan meningkat ketika kecepatan gerak *node* bertambah besar. Hal ini disebabkan pada saat kecepatan tinggi, maka kondisi jaringan akan semakin tidak stabil. Hal ini akan mengakibatkan semakin banyak paket RREQ yang dikirimkan, sehingga akan mengakibatkan peluang tabrakan antar paket semakin besar. Pada saat *node* bergerak dengan kecepatan tinggi, *route* semakin tidak tersedia sehingga paket data akan tertahan di *buffer* sementara sampai *route* baru di temukan. Hal ini akan mengakibatkan rata-rata *delay* pengiriman paket data akan semakin besar. Secara umum *average end-to-end delay* dapat dilihat pada persamaan 3.6 dengan rumus sebagai berikut [11]:

$$\textit{Average Delay}\ (\text{m/s}) = \frac{\text{Total delay}}{\text{Total paket yang diterima}} \tag{3.6}$$

2. *Throughput*

Throughput merupakan suatu istilah yang mendefinisikan banyaknya bit yang diterima dalam selang waktu tertentu. Secara umum *throughput* dinyatakan dalam persamaan 3.7 dengan rumus sebagai berikut [19]:

$$\textit{Throughput}\ (\text{Kbps}) = \frac{\sum paket\ data\ diterima}{\sum waktu} \tag{3.7}$$

3. *Packet delivery ratio* (PDR)

Packet delivery ratio merupakan perbandingan antara paket data yang berhasil diterima oleh *node* tujuan dengan total paket data yang dikirimkan oleh *node* sumber. *Packet delivery ratio* akan menurun pada saat kecepatan gerak *node* bertambah besar. Hal ini disebabkan bertambahnya kecepatan gerak *node* maka akan sering terjadi perubahan posisi *node* sehingga rute rusak, menyebabkan hilangnya paket data yang lewat (*drop*).

Secara umum *Packet delivery ratio* dapat dilihat pada persamaan 3.8. dengan rumus sebagai berikut [11]:

$$\textit{Packet delivery ratio}\ (\%) = \frac{\text{Paket data diterima}}{\text{Paket data dikirim}} \tag{3.8}$$

3.4.2 Parameter Skenario Simulasi

Parameter skenario simulasi merupakan skenario simulasi yang akan dilakukan pada penelitian ini. Simulasi dilakukan pada protokol *routing* AOMDVyang akan diimplementasikan dengan menggunakan *network* simulator 2 atau NS-2 versi 2.35. Jumlah *node* yang akan digunakan pada penelitian ini sebanyak 50 *node*, 70 *node* dan 100 *node*. *Layer* yang digunakan adalah berdasarkan pada standar IEEE 802.11p. *Node-node* akan bergerak dalam kecepatan 30 km/jam, 60 km/jam dan 90 km/jam, 120 km/jam dan 150 km/jam. Parameter ini merupakan parameter secara umum yang akan digunakan dalam simulasi penelitian ini. Adapun parameter simulasi lebih lengkap dapat dilihat pada Tabel 3.2.

Tabel 3.2 Parameter skenario.

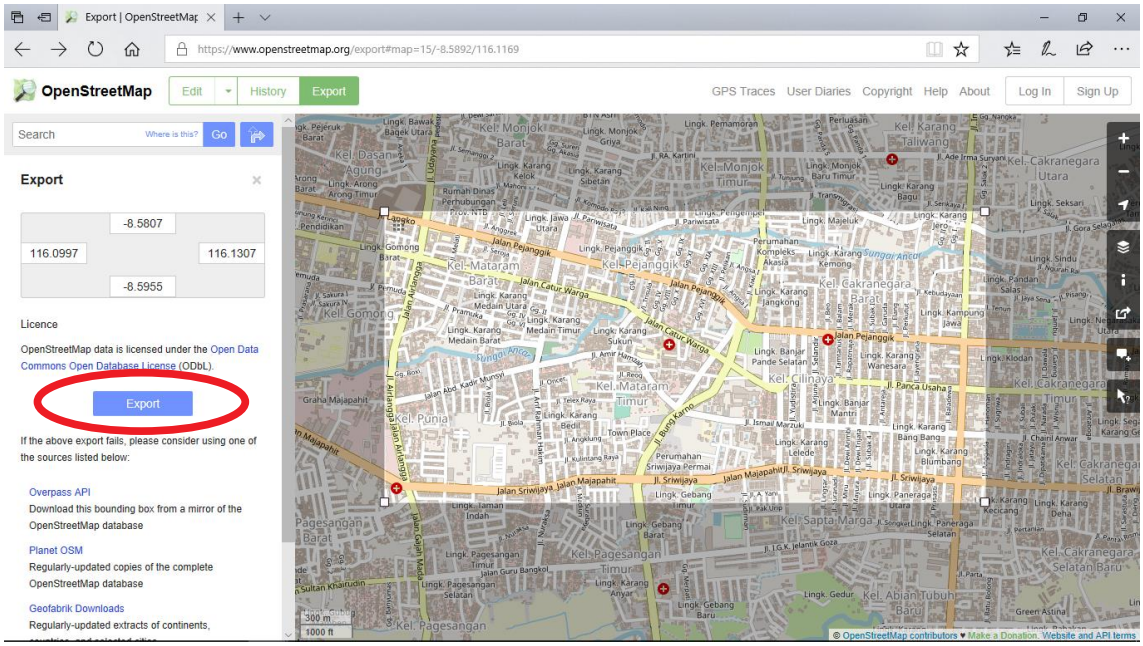
Parameters	***Values***
Protokol *routing*	AOMDV
Jumlah *node*	50 *node*, 70 *node*, 100 *node*
Waktu simulasi	200 s
Kecepatan *node*	30 km/jam, 60 km/jam, 90 km/jam, 120 km/jam dan 150 km/jam
Metode	Algoritma Semut
Propagation	*Two Ray Ground*
Pola trafik	CBR
Antena	*Omni antenna*
MAC Type	IEEE 802.11p
Peta	Kota Mataram
Network Simulator	NS-2.35

BAB IV

HASIL DAN PEMBAHASAN

4.1 Implementasi Skenario

Skenario simulasi pada penelitian ini menggunakan peta jalan Kota Mataram yaitu pada wilayah yang terhubung antara Jl. Pejanggik, Jl. Catur Warga, Jl. Panca Usaha, Jl. Bung Karno, Jl. Sriwijaya Majapahit. Pengambilan peta skenario dilakukan dengan menggunakan *Openstreetmap* melalui *web browser*. Berikut ini adalah gambar yang memperlihatkan proses pengambilan peta yang akan digunakan melalui situs *www.openstreetmap.org.*



Gambar 4.1 Pemilihan lokasi peta melalui *openstreetmap*

Setelah memilih peta yang akan digunakan dalam skenario simulasi, selanjutnya area yang telah dipilih akan di-*eksport* sehingga menjadi sebuah *file* yang berekstensi .osm. Proses *eksport* peta yang telah dipilih dilakukan dengan menekan tombol *"Export"* seperti yang terlihat pada Gambar 4.1 di atas. Ketika tombol *"Export"* ditekan, proses penyimpanan peta yang telah dipilih akan dilakukan. Pada penelitian ini, peta yang digunakan diberi nama *"map.osm"*.

Setelah peta tersimpan, maka terdapat beberapa langkah yang harus dilakukan agar peta tersebut dapat ditampilkan pada SUMO. Langkah-langkah tersebut dijelaskan sebagai berikut :

a. Melakukan proses peng-*import*-an jaringan jalan yang terdapat pada *file map.osm* yang telah di-*download*.

Proses peng-*import*-an jaringan jalan dilakukan dengan menggunakan perintah berikut ini.

```
netconvert --osm-files map.osm -o map.net.xml
```

Perintah *netconvert* digunakan untuk mengkonversi jaringan jalan yang terdapat pada *file .osm* menjadi *file .xml*, dimana pada kasus ini *file .xml* diberi nama *"map.net.xml"*. Setelah proses selesai, maka akan tercipta *file map.net.xml* yang merupakan *file* jaringan jalan yang akan digunakan dalam proses simulasi.

b. Melakukan proses peng-*import*-an *polygon* pada jaringan jalan yang digunakan

File "map.osm" yang telah di *download* tidak hanya terdiri atas jaringan jalan saja, akan tetapi berisi beberapa komponen seperti sungai, gedung, dan lain sebagainya. Untuk menambahkan komponen-kompenen tersebut, maka langkah yang perlu dilakukan adalah melakukan peng-*import*-an *polygon*. Sebelum melakukan proses peng-*import*-an, maka hal pertama yang harus dilakukan adalah menciptakan *file .xml* yang berisikan jenis-jenis *polygon* yang dikenali oleh SUMO. Pada kasus ini *file* tersebut diberi nama *"typemap.xml"*. Pendefinisian file .xml yang berisi data polygon pada sebuah file osm dilakukan dengan menggunakan format berikut ini.

```
<polygonTypes>
   <polygonType id="id polygon" name="nama poligon" color="warna
poligon" layer="no layer"/>

</polygonTypes>
```

Pendefinisian data-data polygon di dalam file .xml dilakukan didalam tag **<polygonTypes> .. </polygonTypes>**. Di dalam tag tersebut, data polygon untuk setiap jenis komponen didefinisikan dengan menggunakan tag **<polygonType>**. Tag **<polygonType>** memiliki beberapa parameter yaitu:

- **id** : merupakan parameter identitas unik untuk setip jenis polygon yang ditambahkan.
- **name** : merupakan parameter nama dari polygon yang ditambahkan.
- **color** : merupakan parameter yang digunakan untuk menentukan warna dari polytgon yang ditambahkan.
- **layer** : merupakan parameter yang bersifat opsional yang digunakan untuk menentukan posisi dari polygon yang ditambahkan.

Berikut adalah tipe-tipe polygon yang terdapat di dalam SUMO.

Tabel 4.1 Data *polygon* pada SUMO

No	Polygon	Deskripsi
1	water	Merupakan tipe polygon yang digunakan untuk mendefinisikan komponen air di dalam peta.
2	natural	Merupakan tipe polygon yang digunakan mendefinisikan komponen alam di dalam peta.
3	forest	Merupakan tipe polygon yang digunakan untuk mendefinisikan komponen hutan.
4	land	Merupakan tipe polygon yang digunakan untuk mendefinisikan komponen daratan.
5	landuse	Merupakan tipe polygon yang digunakan untuk mendefinisikan komponen daratan yang terpakai atau yang memiliki penghuni.
6	park	Merupakan tipe polygon yang digunakan untuk mendefinisikan komponen taman.
7	residential	Merupakan tipe polygon yang digunakan untuk mendefinisikan komponen perumahan atau area yang dipenuhi penduduk.
8	commercial	Merupakan tipe polygon yang digunakan untuk mendefinisikan komponen wilayah pertokoan.
9	industrial	Merupakan tipe polygon yang digunakan untuk mendefinisikan komponen wilayah industry/area industri.
10	military	Merupakan tipe polygon yang digunakan untuk mendefinisikan komponen area militer.
11	farm	Merupakan tipe polygon yang digunakan untuk mendefinisikan komponen sawah.
12	tourism	Merupakan tipe polygon yang digunakan untuk mendefinisikan komponen area pariwisata.
13	sport	Merupakan tipe polygon yang digunakan untuk mendefinisikan komponen area olahraga.

14	leisure	Merupakan tipe polygon yang digunakan untuk mendefinisikan komponen area tempat bersantai.
15	aerialway	Merupakan tipe polygon yang digunakan untuk mendefinisikan komponen area jalur udara.
16	shop	Merupakan tipe polygon yang digunakan untuk mendefinisikan komponen area tempat berbelanja.
17	historic	Merupakan tipe polygon yang digunakan untuk mendefinisikan komponen area tempat bersejarah.
18	building	Merupakan tipe polygon yang digunakan untuk mendefinisikan komponen bangunan.
19	parking	Merupakan tipe polygon yang digunakan untuk mendefinisikan komponen area tempat parkir.
20	power	Merupakan tipe polygon yang digunakan untuk mendefinisikan komponen pembangkit listrik.
21	highway	Merupakan tipe polygon yang digunakan untuk mendefinisikan komponen jalan raya.
22	railway	Merupakan tipe polygon yang digunakan untuk mendefinisikan komponen kreta api.
23	boundary	Merupakan tipe polygon yang digunakan untuk mendefinisikan komponen pembatas/batas area.

Setelah menciptakan *file* tersebut, maka proses penambahan komponen-komponen *polygon* pada jaringan dapat dilakukan. Berikut adalah perintah yang digunakan untuk melakukan hal tersebut.

```
polyconvert --osm-files map.osm --net-file map.net.xml --type-file
typemap.xml -o typemap.poly.xml
```

Perintah di atas menghasilkan *file "typemap.poly.xml"* yang berisikan komponen-komponen *polygon* pada jaringan jalan yang digunakan.

c. Menciptakan rute lalu lintas pada jaringan lalu lintas

Penciptaan rute pada jaringan lalu lintas yang digunakan dilakukan untuk melihat simulasi lalu lintas pada area atau peta yang digunakan. Proses penciptaan rute lalu lintas dilakukan menggunakan perintah berikut ini.

```
python /home/Ica/Documents/sumo-0.26.0/tools/randomTrips.py -n
map.net.xml -r map.rou.xml -e 50 -l
```

File "randomTrips.py" digunakan untuk membangkitkan lalu lintas kendaraan secara acak pada jaringan lalu lintas. Perintah di atas akan menciptakan *file "map.rou.xml"* yang berisi *route* lalu lintas kendaraan pada jaringan lalu lintas yang digunakan dengan jumlah 50 kendaraan.

Untuk membangkitkan atau menciptakan rute kendaraan dengan kecepatan tertentu, maka dapat dilakukan dengan menciptakan *file* tambahan terlebih dahulu. Pada kasus ini *file* tersebut diberi nama *"type.add.xlm"* yang di dalamnya mendefinisikan kecepatan kendaraan dan maksimum kecepatan yang digunakan dalam jaringan lalu lintas. Berikut adalah perintah yang terdapat pada *file* tersebut.

```
<additional>
<vType id="myType" accel="8.33" decel="0.8" sigma="0.5"
maxSpeed="55.5"/>
</additional>
```

Perintah di atas mendefinisikan kecepatan kendaraan sebesar 8.33 m/s atau setera dengan 30 km/jam. Adapun maksimum kecepatan kendaraan yaitu 55.5 m/s atau setara dengan 200 km/jam.

Setelah menciptakan *file* tambahan, maka proses pembangkitan rute lalu lintas pada jaringan dapat dilakukan dengan menggunakan perintah berikut ini.

```
python /home/Ica/Documents/sumo-0.26.0/tools/randomTrips.py -n
map.net.xml -r map.rou.xml -e 50 -l --trip-
attributes="type=\"myType\"" --additional-file type.add.xml
```

d. Membuat *file* konfigurasi untuk simulasi jaringan lalu lintas

Pembuatan *file* konfigurasi merupakan langkah terakhir yang dilakukan agar simulasi jaringan lalu lintas pada SUMO dapat dijalankan. Pada kasus ini, *file* konfigurasi yang diciptakan diberi nama *"map.sumo.cfg".* Berikut adalah perintah lengkap yang terdapat pada *file* tersebut.

```
<configuration>
     <input>
         <net-file value="map.net.xml"/>
```

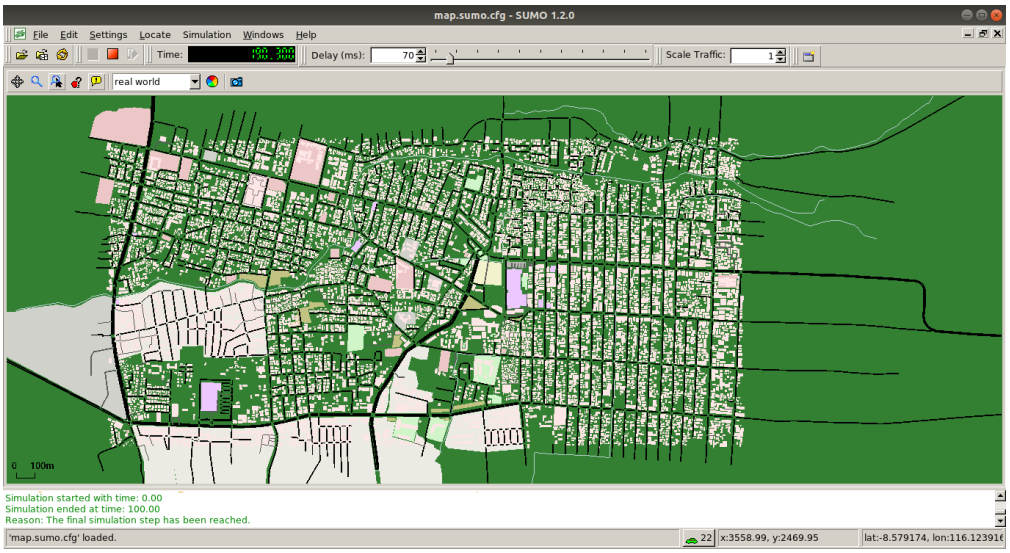
```
      <route-files value="map.rou.xml"/>
         <additional-files value="typemap.poly.xml"/>
     </input>
      <time>
      <begin value="0"/>
      <end value="200"/>
      <step-length value="0.1"/>
      </time>
</configuration>
```

e. Menjalankan simulasi

Untuk menjalankan simulasi terhadap jaringan lalu lintas yang telah diciptakan dapat dilakukan menggunakan perintah berikut ini.

```
sumo-gui map.sumo.cfg
```

Berikut ini adalah bentuk gambar peta pada aplikasi SUMO yang memperlihatkan tampilan jaringan lalu lintas pada area yang telah ditentukan. Pada simulasi ini *node-node* akan berjalan di area simulasi sesuai kondisi jalanan yang telah di tentukan, pengambilan area simulasi menggunakan *open street map*.



Gambar 4.2 Simulasi jaringan lalu lintas pada SUMO : *mode real world*

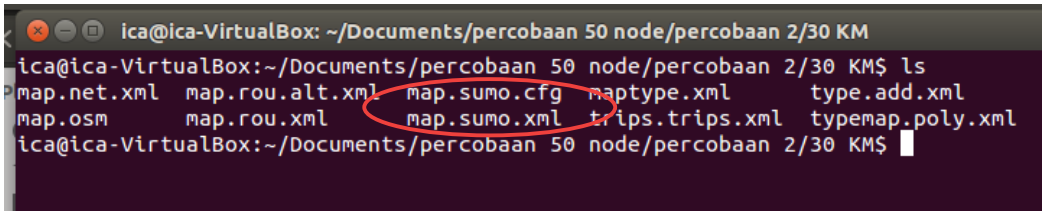
4.2 Implementasi VANET

Setelah menciptakan sekenario jaringan lalu lintas yang akan digunakan, maka langkah selanjutnya yaitu mengintergrasikan jaringan lalu lintas tersebut dengan VANET. Berikut adalah langkah-langkah yang dilakukan dalam melakukan hal tersebut.

a. Mengkonversi *file* konfigurasi SUMO menjadi *file .xml* dengan menggunakan perintah berikut ini.

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

Perintah di atas mengubah *file* konfigurasi *map.sumo.cfg* menjadi *file .xml* yang diberi nama *map.sumo.xml*. Seluruh konfigurasi skenario lalu lintas yang ada pada *file* konfigurasi SUMO akan dikonversi menjadi *file .xml*.



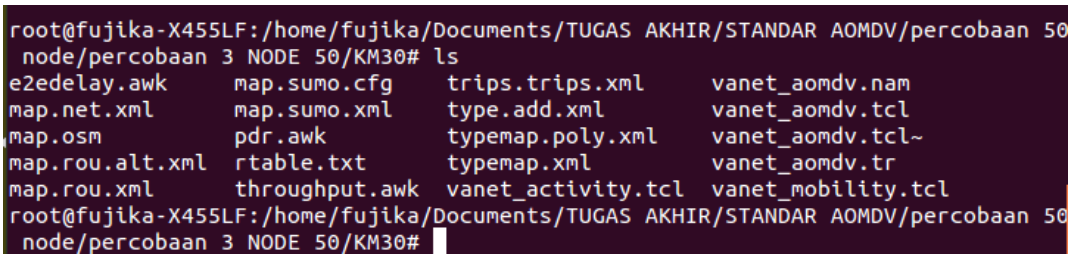
Gambar 4.3 *File map.sumo.xml*

b. Mengekstraksi *file* konfigurasi *.xml* untuk menciptakan *file-file* yang dibutuhkan dalam simulasi VANET di NS2

Setelah mengkonversi *file* konfigurasi SUMO menjadi *file .xml*, maka langkah selanjutnya adalah mengkestraksi *file* tersebut untuk menciptakan *node-node* yang akan digunakan pada NS2. Berikut adalah perintah yang digunakan untuk melakukan hal tersebut.

```
python /usr/share/sumo/tools/traceExporter.py --fcd-input map.sumo.xml
--ns2config-output vanet_aomdv.tcl --ns2mobility-output
vanet_mobility.tcl --ns2activity-output vanet_activity.tcl
```

Perintah di atas akan menciptakan tiga buah *file* yaitu *vanet_aomdv.tcl* yang digunakan sebagai *file* konfigurasi vanet, *file vanet_mobility.tcl* yang berisi *node-node* yang telah dibangkitkan pada simulasi SUMO dan *file* yang terakhir yaitu *file vanet_activity.tcl* yang berisi aktifitas pergerakan *node-node* yang digunakan.



Gambar 4.4 Proses ekstrasi *node* pada *file map.sumo.xml*

Gambar 4.4 di atas memperlihatkan bahwa penciptaan *file* konfigurasi untuk VANET, *file mobilitas node* dan *file* aktifitas *node* yang akan digunakan dalam simulasi VANET telah berhasil digunakan.

c. Melakukan konfigurasi *file vanet_aomdv.tcl*

Sebelum memulai simulasi VANET, maka langkah terakhir yang dilakukan adalah melakukan konfigurasi *file vanet_aomdv.tcl* yang telah diciptakan. Konfigurasi ini bertujuan untuk menentukan spesifikasi jaringan yang akan digunakan. Berikut adalah langkah-langkah yang digunakan untuk melakukan hal tersebut :

- Menentukan parameter konfigurasi jaringan VANET

Berikut adalah perintah yang digunakan untuk menentukan konfigurasi jaringan VANET yang digunakan.

```
#menentukan parameter konfigurasi VANET
set val(chan)         Channel/WirelessChannel    ;# tipe channel
set val(prop)         Propagation/TwoRayGround   ;# model radio-
propagation
set val(netif)        Phy/WirelessPhy            ;# tipe perangkat
jaringan
set val(mac)          Mac/802_11                 ;# tipe MAC
set val(ifq)          Queue/DropTail/PriQueue    ;# interface queue
type
set val(ll)           LL                         ;# tipe link layer
set val(ant)          Antenna/OmniAntenna        ;# antenna model
set val(ifqlen)       50                         ;# max packet in ifq
set val(nn)           50                         ;# jumlah node
set val(rp)           AOMDV                      ;# routing protocol
set opt(x) 6193.71
set opt(y) 3930.62
```

- Menciptakan objek dari *Network Simulator*

Setelah mendefinisikan parameter-parameter yang akan digunakan untuk mengkonfigurasi jaringan VANET, maka langkah selanjutnya adalah menciptakan objek *Network Simulator* yang akan digunakan untuk menjalankan simulasi.

```
set ns_ [new Simulator]
```

- Meciptakan objek perekaman

Objek perekaman berfungsi untuk merekam seluruh aktifitas yang terjadi pada jaringan VANET. Perekaman tersebut juga ditujukan untuk melihat nilai *Packet delivery ratio, Troughput,* dan nilai *End to end delay* yang terjadi pada jaringan VANET. Berikut adalah perintah yang digunakan untuk melakukan hal tersebut.

```
set tracefd [open vanet_aomdv.tr w]
$ns_ trace-all $tracefd
set namf [open vanet_aomdv.nam w]
$ns_ namtrace-all-wireless $namf $opt(x) $opt(y)
```

- Menciptakan objek topologi

Objek topologi berfungsi untuk menciptakan topologi berdasarkan ukuran map yang digunakan. Berikut adalah perintah yang digunakan untuk menciptakan topologi jaringan vanet sesuai dengan ukuran map yang digunakan.

```
# Setup objek topologi
set topo [new Topography]
$topo load_flatgrid $opt(x) $opt(y)
# menciptakan God
create-god $val(nn)
set god_ [God instance]
```

- Melakukan konfigurasi VANET dan menentukan posisi awal *node*

Berikut adalah perintah yang digunakan untuk melakukan konfigurasi jaringan pada VANET dan proses penentuan posisi awal *node-node* yang terdapat di dalamnya.

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

#penciptaan node, dan inisialisasii posisi awal node
 for {set i 0} {$i < $val(nn) } {incr i} {
  set node_($i) [$ns_ node]
  $node_($i) random-motion 0  ;# mendisable random motion
  $ns_ initial_node_pos $node_($i) 30
 }
```

- Mendefinisikan prosedur pengiriman data

Setelah melakukan konfigurasi jaringan VANET dan menentukan posisi awal *node*, maka langkah selanjutnya yang dilakukan adalah mendefinisikan prosedur yang akan digunakan untuk melakukan pengiriman data. Dalam hal ini prosedur tersebut diberinama *"send-data"*. Berikut adalah perintah yang digunakan untuk melakukan hal tersebut.

```
#mendefinisikan prosedur atau fungsi untuk pengiriman packet data
```

```
proc send-packet { node sink size interval rate } {
   #membuat objek simulator
   set ns [Simulator instance]
    #Membuat UPD agent dan menambahkannya pada node
    set source [new Agent/UDP]
    $ns attach-agent $node $source
    #mendepinisikan paket trafik
    set traffic [new Application/Traffic/CBR]
    $traffic set packetSize_ $size
    $traffic set rate_ $rate
    $traffic set maxpkts_ 4028
    $traffic set interval_ $interval
    $traffic set random_ 1
    $traffic set type_        cbr

    # menambahkan trafik paket ke node sumber
    $traffic attach-agent $source
    #menghubungkan node sumber ke node tujuan
    $ns connect $source $sink
    return $traffic
}
```

- Mendefinisikan prosedur pemberhentian

Prosedur pemberhentian digunakan untuk menghentikan seluruh aktifitas yang terjadi pada jaringan VANET. Pada kasus ini prosedur pemberhentian yang diciptakan diberi nama *"stop"*. Berikut adalah perintah yang terdapat pada prosedur ini.

```
#mendefinisikan prosedur/fungsi stop
proc stop {} {
   global ns_  tracefd namf
   $ns_ flush-trace
   close $tracefd
   close $namf
   #menjalankan file
    exec nam vanet_aomdv.nam &
    exec awk -f throughput.awk vanet_aomdv.tr &
    exec awk -f e2edelay.awk vanet_aomdv.tr &
    exec awk -f pdr.awk vanet_aomdv.tr &
    exit 0
}
```

- Melakukan proses pengiriman data

Setelah mendefinisikan prosedur-prosedur di atas, maka langkah selanjutnya adalah menciptakan *agent-agent* yang akan digunakan untuk melakukan proses pengiriman data. Berikut adalah perintah yang digunakan untuk melakukan hal tersebut.

```
#menciptakan agent node penerima
set sink0 [new Agent/LossMonitor]
```

```
$ns_ attach-agent $node_(18) $sink0
$node_(18) color red
$ns_ at 1.0 "$node_(18) color red"

set sink1 [new Agent/LossMonitor]
$ns_ attach-agent $node_(1) $sink1
$node_(1) color blue
$ns_ at 1.0 "$node_(1) color blue"

set sink2 [new Agent/LossMonitor]
$ns_ attach-agent $node_(32) $sink2
$node_(32) color green
$ns_ at 1.0 "$node_(32) color green"

#melakukan pengiriman packet
set source0 [send-packet $node_(16) $sink0 512 1 1024k]
$node_(16) color red
$ns_ at 1.0 "$node_(16) color red"

set source1 [send-packet $node_(9) $sink1 512 1 1024k]
$node_(9) color blue
$ns_ at 1.0 "$node_(9) color blue"

set source2 [send-packet $node_(22) $sink2 512 1 1024k]
$node_(22) color green
$ns_ at 1.0 "$node_(22) color green"
```

- Menentukan waktu untuk proses simulasi

Sebelum menejalankan simulasi, maka tahap yang harus dilakukan adalah menentukan waktu kapan proses simulasi dimulai dan kapan simulasi akan berhenti. Berikut adalah perintah yang digunakan untuk melakukan hal tersebut.

```
#memulai pengiriman paket
$ns_ at 1.0 "$source0 start"
$ns_ at 1.0 "$source1 start"
$ns_ at 1.0 "$source2 start"
# #mengakhiri pengiriman packet
$ns_ at 199.0 "$source0 stop"
$ns_ at 199.0 "$source1 stop"
$ns_ at 199.0 "$source2 stop"
$ns_ at 200 "stop"
$ns_ at 200.01 "puts \"NS SELESAI...\" ; $ns_ halt"

for {set i 0} {$i < $val(nn) } {incr i} {
    $ns_ at 200.0 "$node_($i) reset";
}
```

- Menjalankan *Network Simulator*

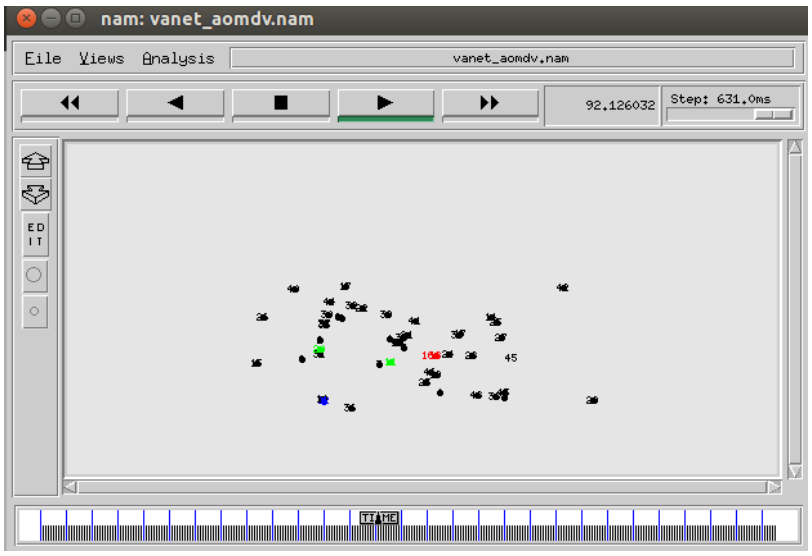
Langkah terakhir yang dilakukan pada *file vanet_aomdv.tcl* adalah menjalankan simulasi dengan menggunakan perintah berikut ini.

```
puts "Memulai Simulasi…"
$ns_ run
```

Setelah pengaturan terhadap *file vanet_aomdv* telah selesai dilakukan, maka simulasi dapat dijalankan dengan menggunakan perintah berikut ini.

```
ns vanet_aomdv.tcl
```

Berikut adalah tampilan pada saat *file* dijalankan. Setelah *file* TCL dijalankan berdasarkan skenario simulasi akan menghasilkan dua buah *file* yaitu *file*.nam (network animator)* digunakan untuk memperlihatkan animasi simulasi jaringan terlihat pada Gambar 4.5 dan *file*.tr* (*trace file*) digunakan untuk memunculkan nilai data statistik menggunakan skrip awk. Untuk mengetahui kinerja dari protokol *routing* AOMDV, maka dilakukan beberapa analisis pada *file*.tr* sesuai dengan pengukuran kinerja yang telah ditentukan yaitu *throughput*, *average end-to-end delay* dan *packet delivery ratio* (seluruh skrip perhitungan awk terlampir).

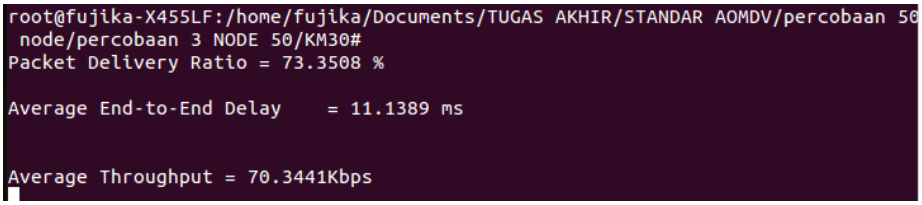


Gambar 4.5 *Network animator* pada jaringan VANET dengan 50 *node* AOMDV-standar

Gambar 4.5 memperlihatkan simulasi visual jaringan VANET protocol AOMDV standar dengan jumlah *node* 50. Pada gambar tersebut terlihat *node-node* yang yang saling berkomunikasi yang direferesentasikan dengan tiga warna yaitu warna merah, warna hijau, dan warna biru. Masing masing warna mereferesentasikan komunikasi

antara dua buah *node*. Dimana pada kasus ini, warna merah mereferesentasikan komonikasi antara *node* 16 dan *node* 18. Adapun warna hijau mereferesentasikan komunikasi antara *node* 22 dan *node* 32. Dan warna biru mereferesentasikan komunikasi antara *node* 9 dan *node* 1.

Adapun hasil perhtingunan parameter *average end to end delay, average throughput* dan *Packet delivery ratio* dapat dilihat pada gambar 4.6 di bawah ini.



Gambar 4.6 Hasil perhitungan parameter *average end to end delay, pakcet lost, average throughput* dan *Packet delivery ratio*

Pada Gambar 4.6 di atas terlihat bahwa pada ujicoba dengan dengan skenario 50 *node*, dan kecepatan 30 km/jam didapatkan nilai *Average End to end delay* yang diperoleh 11.1389 ms, *average throughput* = 70.3441 Kbps, dan *Packet delivery ratio* = 73.3508%.

4.3 Implementasi Algoritma Semut (*Ant Colony Optimization*)

Secara umum, proses optimasi protokol *routing* pada jaringan VANET dengan menggunakan algoritma semut (*Ant Colony Optimization*) terdiri atas beberapa tahapan yaitu sebagai berikut.

a. Inisialisasi parameter semut

   Inisialisasi parameter-parameter algoritma semut pada jaringan VANET meliputi jumlah jumlah *node*, waktu start simulasi, durasi waktu simulasi, dan interpal waktu untuk pembangkitan semut.

b. Pengiriman packet RREQ ke *node* tetangga dan penyebaran *forward ant*

   Proses pencarian rute dengan menggunakan algoritma semut diawali dengan pengiriman packet RREQ dari *node* sumber ke *node* tentangga dengan menggunakan forward ant. Proses ini bertujuan untuk mencari nilai peromon dari setiap agent/semut terhadap *node-node* tetangga.

c. Pembaruan nilai feromon, pengiriman paket RREP dan penyebaran *backward ant*

   Ketika *forward ant* telah mencapai tujuannya, maka pengiriman paket RREP beserta penyebaran *backward ant* akan dilakukan guna memperbarui tabel peromon yang nantinya akan digunakan sebagai acuan rute dalam proses pengiriman data.

d. Pembentukan tabel peromon

Tahap terakhir yang dilakukan setelah proses *forward ant* dan *backward ant* dilakukan adalah pembentukan tabel peromon berdasarkah nilai peromon yang paling optimal. Tabel peromon inilah yang digunakan sebagai rute dalam proses pengiriman packet data dari *node* sumber ke *node* tujuan.

Berikut adalah pseudocode penerapan algoritma semut untuk optimasi protocol *routing* pada jaringan VANET.

```
Node := jumlah node
t:= waktu start
tend:= total waktu simulasi
Δt:=interval waktu


foreach Node
s=node sumber, d=node tujuan,
c=current node
while (t≤tend)
   if ((t mod Δt)=0)
      d:=memilih_node_tujuan()
      menjalankan_forward_ant(d)
      menjalankan_backward_ant(d,s)
      memperbarui_tabel_peromon(c,s,stack_data)
   endif
endwhile
endfor
```

Berdasrakan pseudocode diatas dapat dijelaskan bahwa proses proses pencarian ruter terpendek algoritma semut akan terus berlangsung selama waktu yang ditentukan. Jadi proses pembaruan tabel peromon akan terus dilakukan selama waktu simulasi berlangsung.

4.4 Implementasi Algoritma Semut Pada Protokol AOMDV

Pengimplementasian algoritma semut *(ant colony)* terhadap jaringan VANET pada *routing* protokol AOMDV yang telah dibuat dapat dilakukan dengan menggunakan langkah-langkah berikut ini.

a. Menciptakan *Agent* Semut

Langkah pertama yang dilakukan untuk mengimplementasikan algoritma semut adalah menciptakan *agent* semut dengan menggunakan perintah berikut ini.

```
set sz 75
for {set i 0} {$i < $sz} {incr i} {
     set  na($i)  [new Agent/Antnet $i]
}
```

b. Menghubungkan *Node* dan *Agent*

Setelah menciptakan *agent* semut, langkah selanjutnya adalah menghubungkan *agent-agent* tersebut dengan *node-node* yang telah ada. Berikut adalah perintah yang digunakan untuk melakukan hal tersebut.

```
for {set j 0} {$j < $sz} {incr j} {
    for {set i 0} {$i < $val(nn)} {incr i} {
         $ns_ attach-agent  $node_($i)  $na($j)
    }
}

for {set i 0} {$i < $sz} {incr i} {
     for {set j 0} {$j < $sz} {incr j} {
        $ns_ connect $na($i) $na($j)
    }
}
```

c. Menentukan Posisi *Agent*

Setelah menghubungkan *agent* dengan *node*, langkah selanjutnya adalah menentukan posisi *agent-agent* terhadap *node*. Berikut adalah perintah yang digunakan untuk menentukan posisi *agent* terhadap *node*.

```
for {set i 0} {$i < [expr $val(nn)-1]} {incr i} {
     for {set j 0} {$j < $sz } {incr j} {
        for {set k 0} {$k < [expr $val(nn)-1]} {incr k} {
            $ns_ at now "$na(0) add-neighbor $node_($i) $node_($k)"
        }
    }
}
```

d. Menentukan Parameter *Agent* Semut

Setelah menentukan posisi *agent*, langkah selanjutnya yaitu menentukan parameter dari algoritma semut. Berikut adalah perintah yang digunakan untuk melakukan hal tersebut.

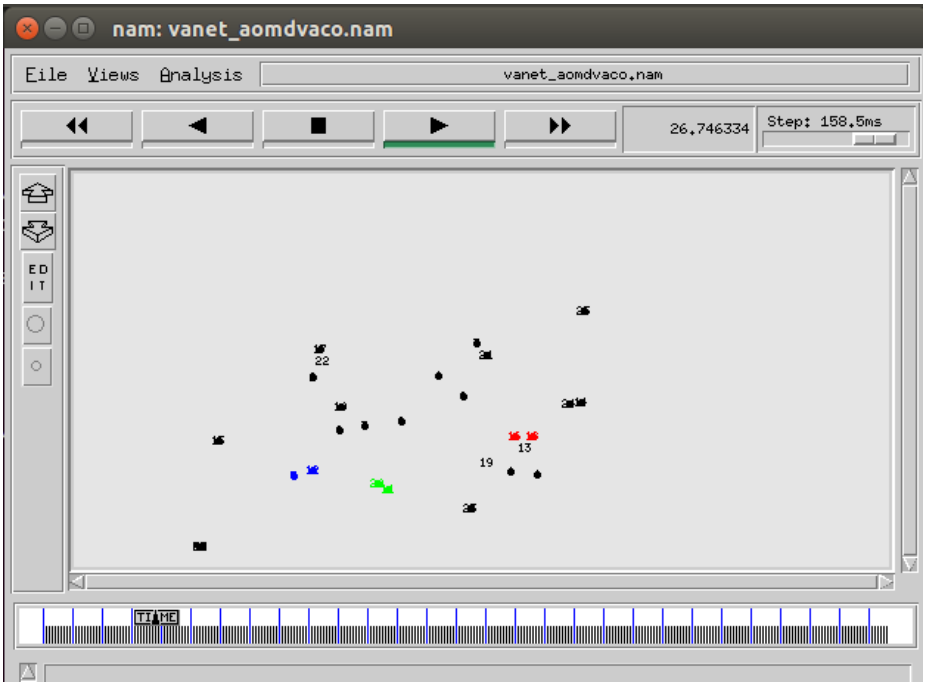
```
for {set i 0} {$i < $sz} {incr i} {

      $na($i) set num_nodes_ $sz

      $na($i) set timer_ant_ 5000

      $na($i) set r_factor_ 0.01

      $ns_ at 0.00005 "$na($i) start"

}
```

e. Menghentikan *Agent*

Langkah terakhir dalam meng-implementasi-kan algoritma semut pada jaringan VANET adalah menentukan kapan waktu *agent-agent* tersebut berhenti bekerja. Berikut adalah perintah yang digunakan untuk melakukan hal tersebut.

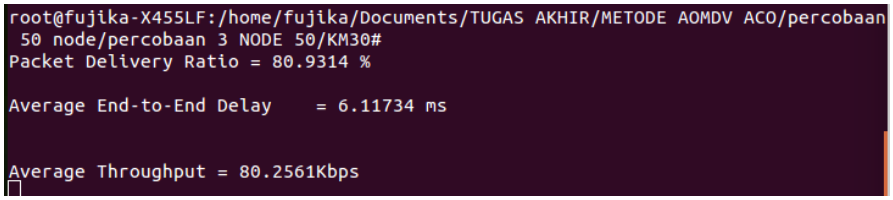
```
for {set i 0} {$i < $sz} {incr i} {
    $ns_ at 200.0 "$na($i) stop"
}
```

Berikut adalah hasil peng-implementasi-an algoritma semut pada protokol *routing* AOMDV, sama seperti percobaan AOMDV standar setelah *file* TCL dijalankan berdasarkan skenario simulasi akan menghasilkan dua buah *file* yaitu *file*.nam (network animator)* digunakan untuk memperlihatkan animasi simulasi jaringan terlihat pada Gambar 4.5 dan *file*.tr* (*trace file*) digunakan untuk memunculkan nilai data statistik menggunakan skrip awk. Untuk mengetahui kinerja dari protokol *routing* AOMDV, maka dilakukan beberapa analisis pada *file*.tr* sesuai dengan pengukuran kinerja yang telah ditentukan yaitu *throughput*, *average end-to-end delay* dan *packet delivery ratio* (seluruh skrip perhitungan awk terlampir).



Gambar 4.7 *Network animator* jaringan VANET dengan 50 *node* AOMDV-Semut

Adapun hasil perhitungan parameter *Average End to end delay*, *average throughput* dan *Packet delivery ratio* dapat dilihat pada gambar 4.8 di bawah ini.



Gambar 4.8 Hasil perhitungan parameter *Average End to end delay, pakcet lost, average throughput* dan *Packet delivery ratio*

Pada Gambar 4.8 di atas terlihat bahwa pada ujicoba dengan dengan skenario 50 *node* AOMDV-Semut, kecepatan 30 km/jam didapatkan nilai *Average End to end delay* yang diperoleh 6.11734 ms, *average throughput* = 80.2561 Kbps dan *Packet delivery ratio* = 80.9314 %.

4.5 Hasil Pengujian Simulasi

Proses pengujian dalam Tugas Akhir ini dilakukan pada jaringan VANET untuk optimasi pencarian rute pada protokol *routing* AOMDV Standar dibandingkan dengan AOMDV dengan mengimplementasikan algoritma semut. Uji coba dilakukan sebanyak 5 kali percobaan untuk mendapatkan akurasi data selama uji coba terhadap perbandingan protokol *routing* AOMDV dan AOMDV-semut. Parameter yang digunakan dalam melakukan analisis kinerja protokol *routing* adalah *throughput*, *average end to end delay* dan *packet delivery ratio*. Berikut ini merupakan sample yang diambil berdasarkan data yang memiliki hasil terbaik yaitu untuk percobaan 50 *node* mengambil sample data uji coba ke 3, untuk percobaan 70 *node* mengambil sample data uji coba ke 3, dan untuk percobaan 100 *node* mengambil sample data uji coba ke 5 [seluruh pengujian terlampir].

Hasil percobaan dengan 50, 70 dan 100 *node*, dengan kecepatan minimum 30 km/jam, 60 km/jam, 90 km/jam, 120 km/jam dan 150 km/jam pada protokol *routing* AOMDV standar dan AOMDV-semut dapat dilihat pada sub bab 4.5.1, sub bab 4.5.2 dan sub bab 4.5.3.

4.5.1 Hasil Pengujian *Throughput*

Skenario uji coba pertama adalah mencari rata-rata jumlah paket data yang berhasil diterima disisi penerima setiap detiknya (*throughput*) karena adanya mobilitas *node* pada VANET. Tabel 4.2 merupakan nilai kinerja dari protokol *routing* AOMDV berdasarkan rata-rata *throughput* didapat dari hasil uji coba yang telah dilakukan. Pengujian dilakukan dengan jumlah *node* dan kecepatan yang berbeda, dengan jumlah jumlah *node* sebanyak 50, 70, dan 100 dengan variasi kecepatan maksimal *node* (*maximal speed*) yaitu 30 km/jam, 60 km/jam,

90 km/jam, 120 km/jam dan 150 km/jam. Berikut adalah detail terkait pengujian-pengujian yang dilakukan pada uji coba ini.

Tabel 4.2 Hasil uji coba *throughput*

THROUGHPUT						
Kecepatan (km/jam)	**AOMDV Standar**			**AOMDV Semut**		
	50 *Node*	**70 *Node***	**100 *Node***	**50 *Node***	**70 *Node***	**100 *Node***
30	70.3441	79.6826	84.8455	80.2561	88.6867	95.2705
60	70.2903	77.2819	82.7662	78.4725	86.2804	93.2567
90	70.2615	75.4424	81.4772	78.2878	84.2496	91.1559
120	67.4707	72.7276	77.5971	73.5535	79.8462	90.1071
150	67.1962	70.5575	74.3487	73.1816	79.2816	88.3469

Berdasarkan ujicoba yang telah dilakukan, beberapa contoh rute yang dihasilkan pada *throughput* AOMDV standar dan AOMDV-semut yaitu :

- *Throughput* AOMDV Standar

  50 *node* : *Node* sumber = 16 dan *node* tujuan = 18

  70 *node* : *Node* sumber = 14 dan *node* tujuan = 9

  100 *node* : *Node* sumber = 24 dan *node* tujuan = 56

Tabel 4.3 Hasil rute *throughput* AOMDV standar

Node	Kecepatan	Rute
50 *node*	30 km	16 → 8 → 15 → 6 → 1 → 3 → 7 → 25 → 10 → 13 → 11 → 19 → 21 → 27 → 18
70 *node*	30 km	14 → 6 → 4 → 8 → 12 → 11 → 3 →2 →22 →25 → 28 →18 → 30 → 17 → 9
100 *node*	30 km	24 → 2 → 5 → 4 → 25 →23 →29 → 11 →15 → 20 → 17 → 19 → 56

- *Throughput* AOMDV-Semut

  50 *node* : *Node* sumber = 16 dan *node* tujuan = 18

  70 *node* : *Node* sumber = 14 dan *node* tujuan = 9

  100 *node* : *Node* sumber = 24 dan *node* tujuan = 56

Tabel 4.4 Hasil rute *throughput* AOMDV- semut

Node	Kecepatan	Rute	Nilai feromon
50 *node*	30 km	16 → 8 → 15 → 6 → 1 → 3 → 14 → 22 → 18	0.09135 → 0.08654 → 0.08173 → 0.07692 → 0.07212 → 0.06731 → 0.06614 → 0.06230
70 *node*	30 km	14 → 4 → 8 →10 →7 → 9 → 11 → 9	0.09875 → 0.09201 → 0.08979 → 0.08237 → 0.07970 → 0.07231 → 0.06981

100 *node*	30 km	24 → 9 → 8 → 60 →25 → 28 → 56	0.09934 → 0.09733 → 0.09231 → 0.0844 →0.07950 → 0.07442

Berdasarkan Tabel 4.2 dapat diperoleh nilai rata-rata *throughput* protokol *routing* AOMDV tanpa modifikasi pada 50 *node* yaitu sebesar 69.11256Kbps. Untuk nilai rata-rata *throughput* pada protokol *routing* AOMDV-semut pada 50 *node* yaitu sebesar 76.7503Kbps. Adapun selisih rata-rata yang diperoleh antara AOMDV standar dengan AOMDV-Semut pada parameter *throughput* pada 50 *node* yaitu sebesar 7.63774Kbps. Sedangkan nilai rata-rata *throughput* protokol *routing* AOMDV standar pada 70 *node* yaitu sebesar 75.1384Kbps. Untuk nilai rata-rata *throughput* protokol *routing* AOMDV-semut pada 70 *node* yaitu sebesar 83.6689Kbps. Adapun selisih rata-rata yang diperoleh antara AOMDV standar dengan AOMDV-Semut pada 70 *node* yaitu sebesar 10.2366Kbps. Kemudian nilai rata-rata *throughput* protokol AOMDV standar pada 100 *node* yaitu sebesar 80.20694Kbps. Dan nilai rata-rata *throughput* protokol *routing* AOMDV-semut pada 100 *node* yaitu sebesar 91.62742Kbps. Adapun selisih rata-rata yang diperoleh pada parameter *throughput* antara AOMDV standar dengan AOMDV-Semut pada 100 *node* yaitu sebesar 11.42048Kbps.

4.5.2 Hasil Pengujian *Average End to End Delay*

Pengujian ini dilakukan pada jaringan VANET dengan AMODV standar dan AOMDV dengan menggunakan algoritma semut. Pada tabel 4.7 merupakan hasil nilai rata-rata yaitu mencari nilai rata-rata selang waktu mulai dari paket dikirimkan oleh *node* sumber sampai paket data tersebut berhasil diterima oleh *node* tujuan, menggunakan data hasil uji coba protokol *routing* skenario VANET. Pengujian dilakukan dengan jumlah *node* dan kecepatan yang berbeda, dengan jumlah jumlah *node* sebanyak 50, 70, dan 100 dengan variasi kecepatan maksimal *node* (*maximal speed*) yaitu 30 km/jam, 60 km/jam, 90 km/jam, 120 km/jam dan 150 km/jam. Pengujian ini bertujuan untuk melihat bagaimana kinerja protokol AMODV pada VANET ketika jumlah *node* dan kecepatan *node* vervariasi. Berdasarkan ujicoba yang telah dilakukan, beberapa contoh rute yang dihasilkan pada *Average end to end delay* AOMDV standar dan AOMDV-semut yaitu :

- *Average end to end delay* AOMDV Standar

  50 *node* : *Node* sumber = 16 dan *node* tujuan = 18

  70 *node* : *Node* sumber = 14 dan *node* tujuan = 9

  100 *node* : *Node* sumber = 24 dan *node* tujuan = 56

Tabel 4.5 Hasil rute *average end to end delay* AOMDV standar

Node	Kecepatan	Rute
50 *node*	30 km	16 → 8 → 15 → 6 → 1 → 3 → 7 → 25 → 10 → 13 → 17 → 11 → 18
70 *node*	30 km	14 → 6 → 4 → 8 → 12 → 11 → 3 →2 →22 →25 → 28 →18 → 30 → 17 → 9
100 *node*	30 km	24 → 44 → 70 → 30 → 18 → 42 → 28 → 20 →22 →13 → 12 → 56

- *Average end to end delay* AOMDV-Semut

  50 *node* : *Node* sumber = 16 dan *node* tujuan = 18

  70 *node* : *Node* sumber = 14 dan *node* tujuan = 9

  100 *node* : *Node* sumber = 24 dan *node* tujuan = 56

Tabel 4.6 Hasil rute *average end to end delay* AOMDV- semut

Node	Kecepatan	Rute	Nilai feromon
50 *node*	30 km	16 → 8 → 15 → 6 → 1 → 3 → 16 → 10 → 18	0.09230 → 0.08534 → 0.08125 → 0.07988 → 0.07893 → 0.07436 → 0.06989 → 0.06793
70 *node*	30 km	14 → 4 → 8 →10 →7 → 3 → 9 → 11 → 15 → 9	0.09877 → 0.09755 → 0.09327 → 0.09149 → 0.08876 → 0.08328 → 0.07894 → 0.07699 → 0.07230
100 *node*	30 km	24 → 9 → 6 → 4 → 8 → 12 → 11 →22 → 28 → 56	0.09934 → 0.09776 → 0.09238 → 0.08867 → 0.08659 → 0.08219 → 0.08159 → 0.07641 → 0.7325

Berikut adalah detail terkait pengujian-pengujian yang dilakukan pada uji coba ini.

Tabel 4.7 Hasil uji coba *average end to end delay*

AVERAGE END TO END DELAY						
Kecepatan (km/jam)	**AOMDV Standar**			**AOMDV Semut**		
	50 *Node*	**70 *Node***	**100 *Node***	**50 *Node***	**70 *Node***	**100 *Node***
30	11.1389	27.7074	39.5456	6.11734	21.3647	26.4555
60	26.0846	37.5506	49.8008	10.1725	30.6043	38.5872
90	32.8883	44.2726	68.9381	26.0124	38.0947	52.7615
120	34.4813	51.2477	90.7636	32.5626	45.2455	65.6145
150	42.8105	61.8224	126.283	35.0368	47.9456	76.9903

Berdasarkan Tabel 4.7 diperoleh nilai rata-rata *end to end delay* pada AOMDV standar dengan 50 *node* yaitu sebesar 29.48072 ms. Sedangkan nilai rata-rata *end to end delay* pada AOMDV semut dengan 50 *node* yaitu sebesar 21.980328ms. Adapun selisih

rata-rata yang diperoleh antara AOMDV standar dengan AOMDV-Semut dengan 50 *node* pada parameter *end to end delay* sebesar 7.500392ms. Dan nilai rata-rata *end to end delay* pada AOMDV standar dengan 70 *node* yaitu sebesar 44.52014 ms sedangkan nilai rata-rata *end to end delay* pada AOMDV semut dengan 70 *node* yaitu sebesar 36.65096ms. Adapun selisih rata-rata yang diperoleh antara AOMDV standar dengan AOMDV-Semut dengan 70 *node* pada parameter *end to end delay* sebesar 9.443016ms. Dan nilai rata-rata *end to end delay* pada AOMDV standar dengan 100 *node* yaitu sebesar 75.06622 ms. Sedangkan nilai rata-rata *end to end delay* pada AOMDV semut dengan 100 *node* yaitu sebesar 52.0818ms. Adapun selisih rata-rata yang diperoleh antara AOMDV standar dengan AOMDV-Semut dengan 100 *node* pada parameter *end to end delay* sebesar 22.98442ms. Hal ini menunjukkan bahwa implementasi algoritma semut pada protokol AOMDV menyebabkan penurunan nilai *end to end delay* pada protocol *routing* AOMDV standar. Sehingga dari hasil ujicoba yang telah dilakukan pada protokol *routing* AOMDV dengan menggunakan algoritma semut untuk parameter *average end to end delay* mampu menurunkan *delay* sebesar 13.3093 ms.

4.5.3 Hasil Pengujian *Packet Delivery Ratio*

Pengujian ini dilakukan pada jaringan VANET AMODV standar dan AOMDV semut. Pada Tabel 4.8 mencari nilai perbandingan antara paket data yang terkirim (paket data yang berhasil diterima oleh *node* tujuan) dengan jumlah paket data yang dikirimkan oleh *node* sumber (paket data yang di-*generate node* sumber). Pengujian dilakukan dengan jumlah *node* dan kecepatan yang berbeda. Pengujian dilakukan dengan jumlah *node* dan kecepatan yang berbeda, dengan jumlah jumlah *node* sebanyak 50, 70, dan 100 dengan variasi kecepatan maksimal *node* (*maximal speed*) yaitu 30 km/jam, 60 km/jam, 90 km/jam, 120 km/jam dan 150 km/jam. Pengujian ini bertujuan untuk melihat bagaimana kinerja protokol AMODV pada VANET ketika jumlah *node* dan kecepatan *node* vervariasi. Berikut adalah detail terkait pengujian-pengujian yang dilakukan pada uji coba ini. Berdasarkan ujicoba yang telah dilakukan, beberapa contoh rute yang dihasilkan pada *packet delivery ratio* AOMDV standar dan AOMDV-semut yaitu :

- *Packet delivery ratio* AOMDV Standar

  50 *node* : *Node* sumber = 16 dan *node* tujuan = 18

  70 *node* : *Node* sumber = 14 dan *node* tujuan = 9

  100 *node* : *Node* sumber = 24 dan *node* tujuan = 56

Tabel 4.8 Hasil rute *packet delivery ratio* AOMDV standar

Node	Kecepatan	Rute
50 *node*	30 km	16 → 35 → 47 → 30 → 28 → 12 → 3 → 6 → 15 → 3 → 29 → 19 → 18
70 *node*	30 km	14 → 61 → 55 → 63 → 30 → 24 → 18 → 11 → 20 → 7 → 10 → 15 → 9
100 *node*	30 km	24 → 50 → 83 → 61 → 63 → 78 → 52 → 41 → 32 → 21 → 26 → 20 → 56

- *Packet delivery ratio* AOMDV-Semut

  50 *node* : *Node* sumber = 16 dan *node* tujuan = 18

  70 *node* : *Node* sumber = 14 dan *node* tujuan = 9

  100 *node* : *Node* sumber = 24 dan *node* tujuan = 56

Tabel 4.9 Hasil rute *packet delivery ratio* AOMDV- semut

Node	Kecepatan	Rute	Nilai feromon
50 *node*	30 km	16 → 29 → 17 → 24 → 17 →12 → 7 → 9 → 17 → 18	0.09230 → 0.09122 → 0.08921 → 0.08654 → 0.08432 → 0.08203 → 0.07960 → 0.07877 → 0.6997
70 *node*	30 km	14 → 40 → 32 → 26 → 37 → 23 → 19 → 17 → 9	0.09875 → 0.09201 → 0.08979 → 0.08237 →0.07970 → 0.07231 → 0.06981 → 0.6779
100 *node*	30 km	24 → 29 → 31 → 60 → 81 → 92 → 29 → 56	0.09944 → 0.09877 → 0.09334 → 0.08998 → 0.08116 → 0.07996 → 0.07359

Tabel 4.10 Hasil uji coba *packet delivery ratio*

PACKET DELIVERY RATIO						
Kecepatan (km/jam)	**AOMDV Standar**			**AOMDV Semut**		
	50 *Node*	**70 *Node***	**100 *Node***	**50 *Node***	**70 *Node***	**100 *Node***
30	73.3508	76.1568	87.7979	80.9314	89.4178	98.5002
60	70.6947	74.4346	84.7441	79.4733	87.3611	96.3119
90	69.0826	74.0849	83.2487	76.7649	87.2247	94.0121
120	65.4284	73.7668	80.8347	75.9742	84.1091	90.4153
150	63.2545	71.7292	78.9041	75.3467	80.5853	88.6366

Berdasarkan Tabel 4.10 di peroleh jumlah rata-rata *Packet delivery ratio* pada AOMDV standar dengan 50 *node* yaitu sebesar 68.3622%. Sedangkan jumlah rata-rata *Packet delivery ratio* pada AOMDV semut dengan 50 *node* yaitu sebesar 77.6981%. Adapun selisih rata-rata yang diperoleh antara AOMDV standar dengan AOMDV-Semut pada parameter *Packet delivery ratio* dengan 50 *node* yaitu sebesar 9.3359%. Dan jumlah

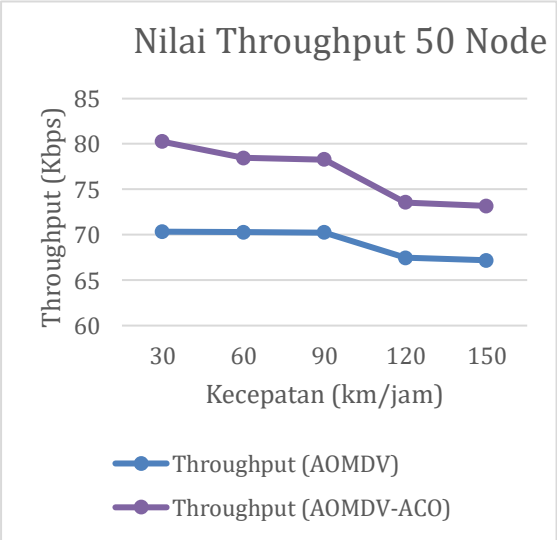
rata-rata *Packet delivery ratio* pada AOMDV standar dengan 70 *node* yaitu sebesar 74.03446%. Sedangkan jumlah rata-rata *Packet delivery ratio* AOMDV semut dengan 70 *node* yaitu sebesar 85.7396%. Adapun selisih rata-rata yang diperoleh antara AOMDV standar dengan AOMDV-Semut dengan 70 *node* pada parameter *Packet delivery ratio* sebesar 14.046168%. Dan jumlah rata-rata *Packet delivery ratio* pada AOMDV standar dengan 100 *node* yaitu sebesar 83.1059%. jumlah rata-rata *Packet delivery ratio* pada AOMDV semut dengan 100 *node* yaitu sebesar 93.57522%. Adapun selisih rata-rata yang diperoleh antara AOMDV standar dengan AOMDV-Semut pada parameter *Packet delivery ratio* dengan 100 *node* yaitu sebesar 10.46932%. Hal ini menunjukkan bahwa implementasi algoritma semut pada protokol AOMDV menyebabkan peningkatan persentase *Packet delivery ratio*. Dari hasil ujicoba yang telah dilakukan pada protokol *routing* AOMDV dengan menggunakan algoritma semut mampu meningkatkan kinerja dari parameter ujicoba *Packet delivery ratio* sebesar 11.2838%.

4.6 Analisis Hasil Pengujian Simulasi

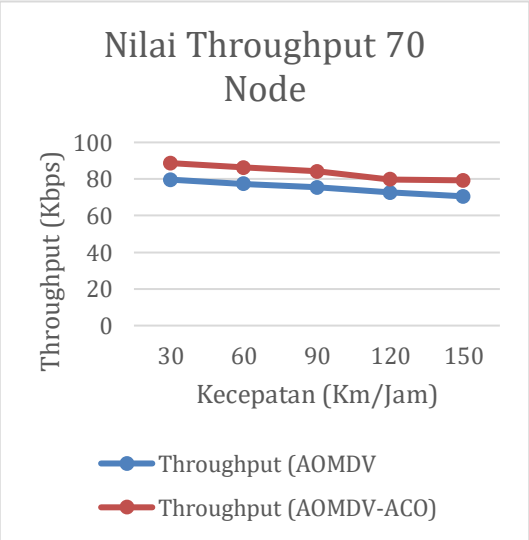
Pada sub bab ini, hasil uji coba yang telah didapatkan akan dianalisis untuk mengetahui tingkat kinerja dari protokol *routing* yang diajukan dan protokol *routing* standar. Analisis yang dilakukan adalah analisis terhadap parameter *throughput*, *Average End to end delay* dan *Packet delivery ratio*.

4.6.1 Analisis Hasil Pengujian *Throughput*

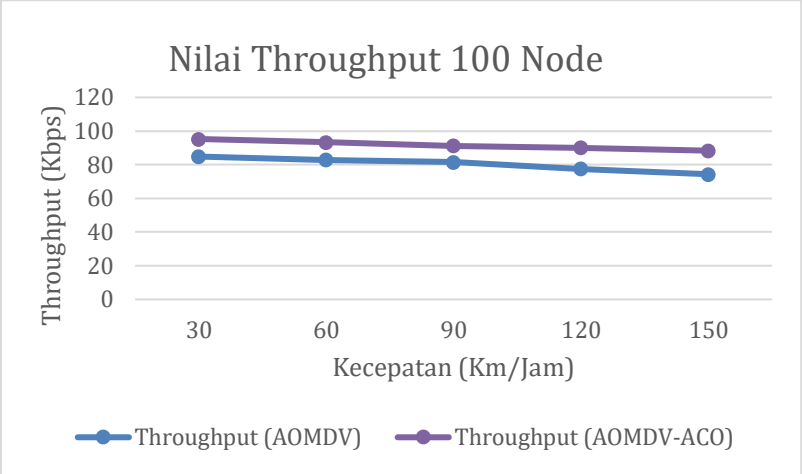
Grafik nilai *throughput* yang telah didapatkan dari skenario percobaan dengan penggunaan jumlah *node* dari 50, 70 dan 100 *node* dengan kecepatan 30 km/ham, 60 km/jam, 90 km/jam, 120 km/jam dan 150 km/jam.



(a)



(b)



(c)

Gambar 4.9 Grafik perbandingan *throughput* protokol AOMDV Standar dan AOMDV-Semut (a) Grafik perbandingan nilai *throughput* pada 50 *node*, (b) Grafik perbandingan nilai *throughput* pada 70 *node*, (c) Grafik perbandingan nilai *throughput* pada 100 *node*.

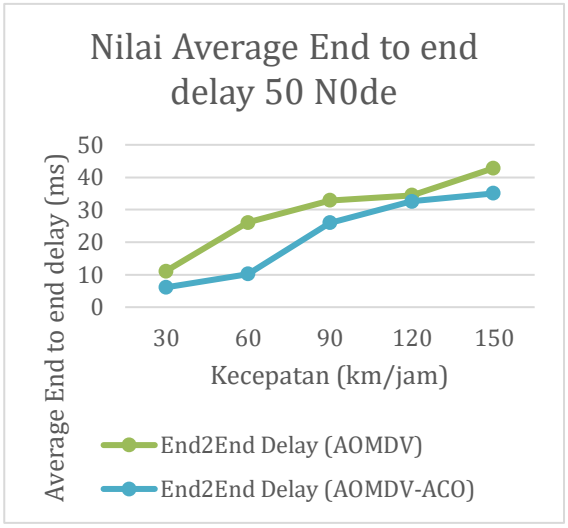
Throughput merupakan ukuran kecepatan yang diperlukan untuk melakukan transfer data dari *node* sumber ke *node* tujuan. Kinerja *throughput* dapat dikatakan baik apabila kecepatan transfer data paket yang tinggi dari *node* sumber menuju *node* tujuan. Berdasarkan seluruh ujicoba yang telah dilakukan pada percobaan AOMDV standar, maka dapat diketahui bahwa pada parameter *throughput* nilai yang di peroleh relatif tinggi. Pengiriman paket data oleh *throughput* dapat dipengaruhi oleh paket RREQ yang dikirimkan seperti jumlah *node* yang banyak atau jumlah *node* yang sedikit untuk sampai ke *node* tujuan atau kecepatan transfer yang digunakan. Hal ini disebabkan oleh sifat reaktif yang dimiliki oleh protokol *routing* AOMDV, sehingga pencarian rute hanya dilakukan pada saat rute dari *node* sumber ke *node* tujuan tidak terdapat di dalam tabel *routing*.

Adanya penambahan jumlah *node* menyebabkan nilai *throughput* pada protokol AOMDV mengalami peningkatan, hal ini disebabkan oleh banyak *node* yang terdapat pada lingkungan simulasi maka lingkungan simulasi akan semakin padat sehingga semakin kecil kemungkinan terjadinya link terputus pada jalur komunikasi dan menyebabkan daya tahan link menjadi lebih tahan lama. Kondisi ini bebanding terbalik dengan adanya penambahan kecepatan *node*, semakin tinggi mobilitas *node* maka semakin rendah nilai *throughput* yang dihasilkan, karena semakin tinggi mobilitas setiap *node* maka pergerakan *node* akan semakin cepat, sehingga link yang sebelumnya telah

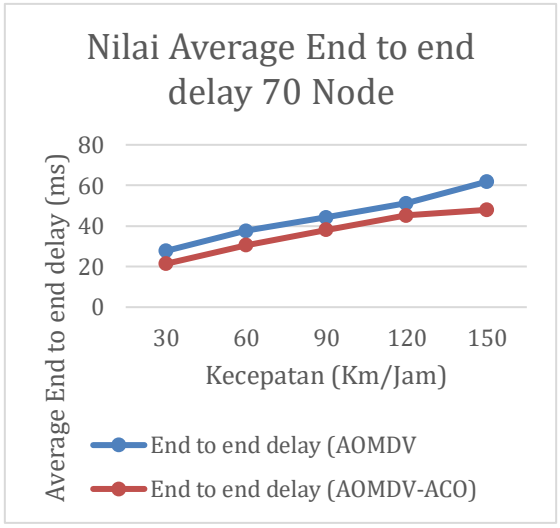
terbentuk akan semakin cepat putus, sehingga untuk dapat mengirimkan paket dari sumber ke tujuan diperlukan proses *broadcast* kembali.

Jadi berdarkan Gambar 4.9 menunjukan nilai *throughput* pada protocol *routing* AOMDV dengan diterapkan algoritma semut memberikan peningkatan nilai sebesar 11.42048Kbps, sehingga dengan peningkatan nilai tersebut dapat dikatakan ujicoba ini berhasil.

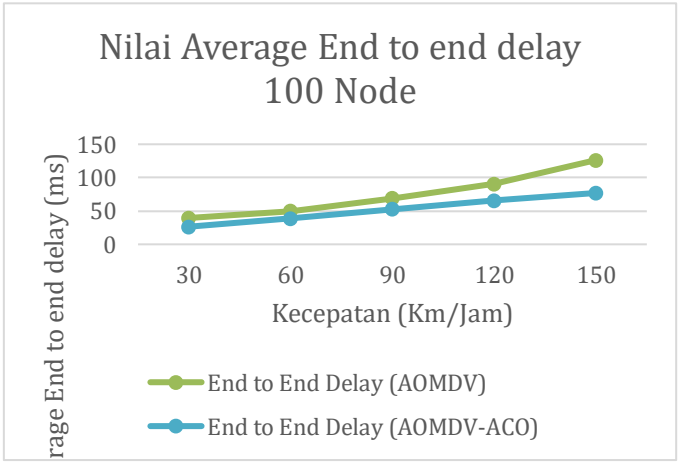
4.6.2 Analisis Hasil Pengujian *Average End to end delay*



(a)



(b)



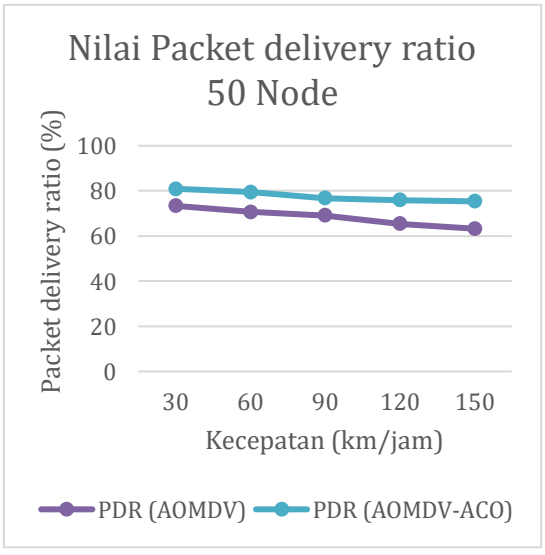
(c)

Gambar 4.10 Grafik perbandingan *average end to end delay* protokol AOMDV Standar dan AOMDV-Semut (a) Grafik perbandingan nilai *average end to end delay* pada 50 *node*, (b) Grafik perbandingan nilai *average end to end delay* pada 70 *node*, (c) Grafik perbandingan nilai *average end to end delay* pada 100 *node*.

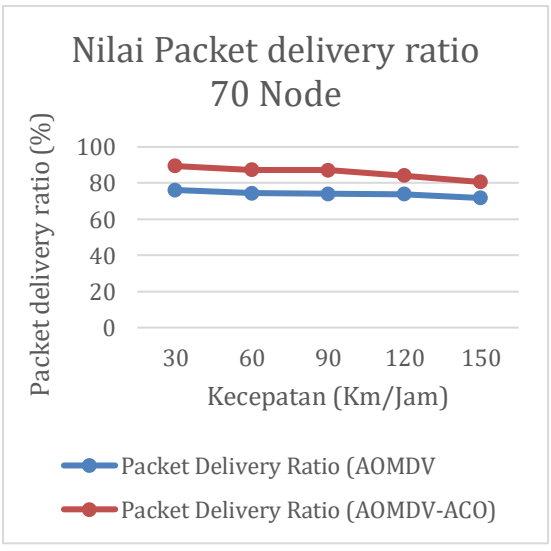
Average End to end delay merupakan jumlah waktu yang diperlukan untuk sebuah paket yang dikirim oleh *node* sumber sampai ke *node* tujuan. Kinerja dari *end to end delay* dapat dikatakan baik apabila waktu yang dibutuhkan oleh protokol *routing* untuk mengirimkan paket dengan waktu yang lebih sedikit. Pada parameter *end to end delay* yaitu waktu yang dibutuhkan dalam pengiriman data dari *source node* ke *destination node*. Dalam proses pengiriman data, akan ada banyak faktor yang mengakibatkan *delay*. Peningkatan *end to end delay* dapat terjadi ketika jumlah komunikasi bertambah (kepadatan kendaraan bertambah). Semakin padatnya kendaraan (*node*) dapat menyebabakn semakin banyak *node* perantara yang terpakai untuk mengirimkan layanan dari sumber ke tujuan, sehingga paket-paket yang dikirimkan harus melalui buffer dan menyebabkan waktu pengiriman semakin lama.

Berdasarkan Gambar 4.10 menunjukan hasil rata-rata *average end to end delay* pada percobaan 50 *node*, 70 *node* dan 100 *node*. Hasil menunjukan nilai delay pada protocol *routing* AOMDV tanpa menggunakan algoritma dibandingkan dengan nilai delay pada protocol *routing* dengan menggunakan algoritma semut. Adanya penerapan algoritma semut pada protocol *routing* AOMDV terbukti dapat menurunkan hasil delay sebesar 13.3093 ms, sehingga dapat dikatakan uji coba pada penelitian ini berhasil.

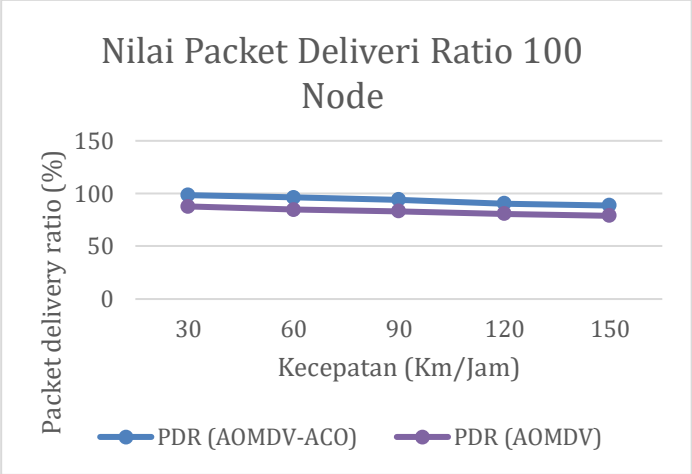
4.6.3 Analisis Hasil Pengujian *Packet delivery ratio*



(a)



(b)



(c)

Gambar 4.11 Grafik perbandingan *Packet delivery ratio* protokol AOMDV Standar dan AOMDV-Semut (a) Grafik perbandingan nilai *Packet delivery ratio* pada 50 *node*, (b) Grafik perbandingan nilai *Packet delivery ratio* pada 70 *node*, (c) Grafik perbandingan nilai *Packet Delivery Ratio* pada 100 *node*.

Packet delivery ratio merupakan ukuran perbandingan antara paket yang berhasil diterima oleh *node* tujuan dengan total paket dari *node* sumber. Pada parameter *Packet delivery ratio* (PDR) menunjukkan keberhasilan protokol dalam mengirim data, tinggi nilai *Packet delivery ratio* salah satunya disebabkan oleh berhasilnya sebuah protokol dalam melakukan pencarian dan pemeliharaan rutenya. Berdasarkan Gambar 4.11 menunjukan hasil dengan jumalh *node* 100 menunjukan hasil yang lebih tinggi dibandingkan dengan 50 *node* dan 70 *node*, hal ini dikarenakan semakin padat jumlah *node* pada suatu simulasi dapat diperoleh hasil yang semakin tinggi karena link tidak cepat terputus.

Pada grafik (a,b,c) menunjukan hasil nilai semakin tinggi kecepatan pada *node* 50, 70 maupun 100 memberikan hasil yang semakin menurun pada protokol AOMDV. Hal ini membuktikan bahwa semakin padatnya lingkungan simulasi yang diikuti dengan semakin lambatnya kendaraan bergerak menyebabkan rute yang terbentuk tidak cepat terputus, sehingga protokol tidak sering melakukan pembaharuan rute dan semakin sedikit terjadinya link terputus maka semakin besar nilai *Packet delivery ratio* pada suatu protokol. Pada penelitian ini penerapan algoritma semut pada protocol *routing* AOMDV memberikan peningkatan nilai *Packet delivery ratio* sebesar 11.2838%, sehingga dapat dikatakan uji coba ini berhasil.

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Dari penjelasan di atas dapat ditarik beberapa kesimpulan yaitu :

1. Pada uji coba AOMDV dengan jumlah *node* yang semakin banyak (50 *node*, 70 *node* 100 *node*) didapatkan hasil yang menunjukan bahwa semakin banyak jumlah *node*, maka nilai parameter *throughput*, *Packet delivery ratio* yang diperoleh semakin meningkat hal ini di sebabkan karena semakin padatnya lingkungan simulasi sehingga link tidak cepat terputus.
2. Pada *avegare end to end delay* semakin besar jumlah *node* maka semakin besar nilai yang di peroleh karena banyaknya *node* yang harus di lewati untuk mengirimkan paket data.
3. Pada penelitian dilakukan 5 kali percobaan. Hasil menunjukkan bahwa uji coba yang telah dilakukan pada protocol *routing* AOMDV dengan menggunakan algoritma semut dapat meninggkatkan kinerja dari parameter *throughput* sebesar sebesar 9.7649 Kbps.
4. Untuk parameter dari *Packet delivery ratio* dengan menerapkan algoritma semut pada protocol *routing* AOMDV menunjukan hasil peningkatan dengan nilai sebesar 11.2838%.
5. Sedangkan untuk parameter *average end to end delay* mampu menurunkan *delay* pada protocol *routing* AOMDV dengan nilai sebesar 13.3093 ms.
6. Penentuan parameter, kecepatan *node*, serta jumlah *node* serta jumlah semut pada algoritma semut yang digunakan, dapat mempengaruhi hasil optimasi pencarian rute pada protokol *routing* AOMDV.

5.2 Saran

Berdasarkan kesimpulan di atas, maka peneliti dapat memberikan saran-saran sebagai berikut:

1. Mengimplementasikan algoritma optimasi lainnya pada protokol *routing* AOMDV
2. Melakukan penelitian dengan menggunakan protokol *routing* yang berbeda pada algoritma semut.
3. Melakukan penelitian yang sama dengan mengubah parameter-parameter penelitian.

4. Berdasarkan kendala yang dialami penulis, penelitian dapat dilakukan dengan mengembangkan sendiri algoritma optimasi, sehingga dapat digunakan sesuai dengan kebutuhan dan dapat dikostumasi.
5. Untuk penelitian selanjutnya dapat digunakan *whitebox testing* untuk menjabarkan efisiensi rute yang dihasilkan.

DAFTAR PUSTAKA

[1] F. Nutrihadi, R. Anggoro and R. M. Ijtihadie, "Studi Kinerja VANET Scenario Generators: SUMO dan Vanet Mobisim untuk Implementasi *Routing* Protocol AODV menggunakan Network Simulator 2 (NS-2)," *Jurnal Teknik ITS* , vol. 5, no. 1, pp. A19-A24, 2016.

[2] R. Anisia, R. Munadi and R. M. Negara, "Analisis Performansi *Routing* Protocol OLSR dan AOMDV pada Vehicular Ad Hoc Network (VANET)," *Jurnal Nasional Teknik Elektro* , vol. 5, no. 1, pp. 87-97, 2016.

[3] A. Deshmukh and S. Dorle, "Performance Improvement of Dynamic Source *Routing* (DSR) Protocol using Ant Colony Optimization for Vehicular Ad-hoc Network (VANet)," *International Journal of Scientific Research,* vol. 5, no. 1, pp. 171-173, 2016.

[4] S. Singh, P. Kumari and S. Agrawal, "Comparative Analysis of Various *Routing* Protocols in VANET," *International Conference on Advanced Computing & Communication Technologies,* pp. 315-319, 2015.

[5] C. Kanani and A. Sinhal, "Ant Colony Optimization based Modified AOMDV for Multipath *Routing* in MANET," *International Journal of Computer Applications,* vol. 82, no. 10, pp. 14-19, 2013.

[6] R. Chauhan and A. Dahiya, "AODV Extension using Ant Colony Optimization for Scalable *Routing* in VANETs," *Journal of Emerging Trends in Computing and Information Sciences,* vol. 3, no. 2, pp. 241-244, 2012.

[7] Li, F., & Wang, Y. (2007). *Routing* in vehicular ad hoc networks: A survey. IEEE Vehicular Technology Magazine, 2(2), 12-22.

[8] S. L. O. B. Correia, J. C. J´unior and O. Cherkaoui, "Mobility-aware Ant Colony Optimization *Routing* for Vehicular Ad Hoc Networks," *IEEE WCNC,* pp. 1125-1130, 2011.

[9] A. D. Devangavi and D. R. Gupta, "*Routing* Protocols in VANET - A Survey," *International Conference On Smart Technology for Smart Nation,* pp. 163-167, 2017.

[10] I. N. Dewi, R. Munadi and L. V. Y., "Simulasi dan Analisis Performansi dari Protokol *Routing* Berbasis Posisi GPRS Dan GYTAR untuk Vehicle

Communication pada Vehicular Ad Hoc Network (VANET)," *Jurnal Penelitian dan Pengembangan Telekomunikasi, Kendali, Komputer, Elektrik, dan Elektronika (TEKTRIKA)* , vol. 1, no. 1, pp. 24-31, 2016.

[11] H. E. Wahanani, "Kinerja Protokol DSR Pada Jaringan MANET dengan Metode *Node* Disjoint And Alternative Multipath *Routing*," in *Seminar Nasional Teknik Informatika (SANTIKA)* , Universitas Pembangunan Nasional "Veteran" Jawa Timur , 2013.

[12] Y. A. Arinatal, Analisis Kecepatan Inisialisasi Jaringan Ad Hoc pada *Routing* Protocol AODV, OLSR, dan ZRP dengan NS2, Yogyakarta: Universitas Sanata Dharma, 2015.

[13] B. P. Bahteradi and R. Anggoro, "Studi Kinerja Multipath AODV dengan Menggunakan Network simulator 2 (NS-2)," *Jurnal Teknik ITS* , vol. 5, no. 2, pp. A652-A656, 2016.

[14] Karjono, Moedjiono and D. Kurniawan, "Ant Colony Optimization," *Jurnal TICOM,* vol. 4, no. 3, pp. 119-125, 2016.

[15] D. D. and M. A., "Road-Based *Routing* Protocol for Vanet Using Sumo and Move," *International Journal of Computer Sciences and Engineering,* vol. 4, no. 2, pp. 151-155, 2016.

[16] K. Kaur, "SpatialHadoop: For OpenStreetMap Data," *International Conference on Inventive Systems and Control ,* 2017.

[17] R. F. Sari, A. Syarif and B. Budiardjo, "Analisis Kinerja Protokol *Routing* Ad Hoc On-Demand Distance Vector (AODV) pada Jaringan Ad Hoc Hybrid: Perbandingan Hasil Simulasi dengan Ns-2 dan Implementasi pada Testbed dengan PDA," *MAKARA, Teknologi,* vol. 12, no. 1, p. 7-18, 2008.

[18] R. Amalia, "Pencarian Jalur Terpendek Menggunakan Ant Colony System (Kasus: Pariwisata Kota Bogor)," *Universitas Indraprasta PGRI,* pp. 290-304, 2015.

[19] V. L. Ratrindra, Analisis Performansi Destination Sequenced Distance Vector (DSDV) dan Zone *Routing* Protocol (ZRP) Berbasis Algoritma Ant pada Jaringan Mobile Ad Hoc, Bandung: Institut Teknologi Telkom, 2010.

LAMPIRAN 1

KONFIGURASI FILE POTOKOL *ROUTING* AOMDV STANDAR

1. 1 Script File TCL Protokol AOMDV Standar

```
#menentukan parameter konfigurasi VANET
set val(chan)           Channel/WirelessChannel    ;# tipe channel
set val(prop)           Propagation/TwoRayGround   ;# model radio-
propagation
set val(netif)          Phy/WirelessPhy            ;# tipe perangkat
jaringan
set val(mac)            Mac/802_11                 ;# tipe MAC
set val(ifq)            Queue/DropTail/PriQueue    ;# interface queue
type
set val(ll)             LL                         ;# tipe link layer
set val(ant)            Antenna/OmniAntenna        ;# antenna model
set val(ifqlen)         50                         ;# max packet in
ifq
set val(nn)             50                         ;# jumlah node
set val(rp)             AOMDV                       ;# routing
protocol
set opt(x) 6193.71
set opt(y) 3930.62

# Inisialisasi Variabel-variabel global
set ns_ [new Simulator]

set tracefd [open vanet_aomdv.tr w]
$ns_ trace-all $tracefd

set namf [open vanet_aomdv.nam w]
$ns_ namtrace-all-wireless $namf $opt(x) $opt(y)
# Setup objek topologi
set topo [new Topography]

$topo load_flatgrid $opt(x) $opt(y)

# menciptakan God
create-god $val(nn)
set god_ [God instance]

# configure node

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

#penciptaan node, dan inisialisasii posisi awal node
 for {set i 0} {$i < $val(nn) } {incr i} {
  set node_($i) [$ns_ node]
  #$node_($i) color blue
  #$ns_ at 0.0 "$node_($i) color blue"
```

```
  $node_($i) random-motion 0  ;# mendisable random motion
  $ns_ initial_node_pos $node_($i) 30
 }

# Mendefinisikan Warna
$ns_ color 1 dodgerblue
$ns_ color 2 blue
$ns_ color 3 cyan
$ns_ color 4 green
$ns_ color 5 yellow
$ns_ color 6 black
$ns_ color 7 magenta
$ns_ color 8 gold
$ns_ color 9 red

#Menciptkana array untuk menyimpan warna
set colorname(0) blue
set colorname(1) cyan
set colorname(2) green
set colorname(3) red
set colorname(4) gold
set colorname(5) magenta

# mendefinisikan prosedur untuk memberi warna pada node secara
dinamis
proc dynamic-color {} {
      global ns_ val node_ colorname
      set time 0.3
      set now [$ns_ now]
      set Rand [expr round(rand()*5)]
      for {set i 0} {$i < $val(nn) } {incr i } {
            $node_($i) color $colorname($Rand)
            $ns_ at $now "$node_($i) color $colorname($Rand)"
      }
      $ns_ at [expr $now+$time] "dynamic-color"
}

source vanet_mobility.tcl

#mendefinisikan prosedur/fungsi stop
proc stop {} {
    global ns_ tracefd namf

      $ns_ flush-trace
    close $tracefd
      close $namf

      #menjalankan file
    exec nam vanet_aomdv.nam &
    exec awk -f throughput.awk vanet_aomdv.tr &
    exec awk -f e2edelay.awk vanet_aomdv.tr &
    exec awk -f pdr.awk vanet_aomdv.tr &

    exit 0
}

#mendefinisikan prosedur atau fungsi untuk pengiriman packet data
proc send-packet { node sink size interval rate } {
      #membuat objek simulator
      set ns [Simulator instance]
```

```
      #Membuat UPD agent dan menambahkannya pada node
      set source [new Agent/UDP]
      $ns attach-agent $node $source

      #mendepinisikan paket trafik
      set traffic [new Application/Traffic/CBR]
      $traffic set packetSize_ $size
      $traffic set rate_ $rate
    $traffic set maxpkts_ 4028
    $traffic set interval_ $interval
    $traffic set random_ 1
    $traffic set type_        cbr

    # menambahkan trafik paket ke node sumber
    $traffic attach-agent $source
      #menghubungkan node sumber ke node tujuan
      $ns connect $source $sink
      return $traffic
}
#menciptakan agent node penerima
set sink0 [new Agent/LossMonitor]
$ns_ attach-agent $node_(18) $sink0
$node_(18) color red
$ns_ at 1.0 "$node_(18) color red"

set sink1 [new Agent/LossMonitor]
$ns_ attach-agent $node_(1) $sink1
$node_(1) color blue
$ns_ at 1.0 "$node_(1) color blue"

set sink2 [new Agent/LossMonitor]
$ns_ attach-agent $node_(32) $sink2
$node_(32) color green
$ns_ at 1.0 "$node_(32) color green"

#melakukan pengiriman packet
set source0 [send-packet $node_(16) $sink0 512 1 1024k]
$node_(16) color red
$ns_ at 1.0 "$node_(16) color red"

set source1 [send-packet $node_(9) $sink1 512 1 1024k]
$node_(9) color blue
$ns_ at 1.0 "$node_(9) color blue"

set source2 [send-packet $node_(22) $sink2 512 1 1024k]
$node_(22) color green
$ns_ at 1.0 "$node_(22) color green"



for {set i 0} {$i < $val(nn) } {incr i} {
    $ns_ at 200.0 "$node_($i) reset";
}

#memulai pengiriman paket
$ns_ at 1.0 "$source0 start"
$ns_ at 1.0 "$source1 start"
$ns_ at 1.0 "$source2 start"
# #mengakhiri pengiriman packet
```

```
$ns_ at 199.0 "$source0 stop"
$ns_ at 199.0 "$source1 stop"
$ns_ at 199.0 "$source2 stop"
$ns_ at 200 "stop"
$ns_ at 200.01 "puts \"NS SELESAI...\" ; $ns_ halt"
puts "Memulai Simulasi..."
$ns_ run
```

LAMPIRAN 2

KONFIGURASI FILE ALGORITMA SEMUT

2.1 Implementasi Algoritma Semut (*Ant Colony Optimization*)

Algoritma semut memiliki beberapa tahapan dalam melakukan proses optimasi yaitu sebagai berikut.

a. Inisialisasi parameter semut

```
Antnet::Antnet(nsaddr_t   id)   :   Agent(PT_ANT),   ant_timer_(this),
dmux_(0){

      bind("num_nodes_", &num_nodes_);
      bind("num_nodes_x_", &num_nodes_x_);
      bind("num_nodes_y_", &num_nodes_y_);
      bind("r_factor_", &r_factor_);
      bind("timer_ant_", &timer_ant_);

      ra_addr_ = id;
      ant_seq_num_ = 0;
}
```

b. Pengiriman packet RRQ ke *node* tetangga dan penyebaran *forward ant*

```
void Antnet::send_ant_pkt() {
      nsaddr_t next, dest;
      Packet* p = allocpkt();
      struct hdr_cmn* ch = HDR_CMN(p);
      struct hdr_ip* ih = HDR_IP(p);
      struct hdr_ant_pkt* ah = HDR_ANT_PKT(p);

      ah->pkt_type() = FORWARD_ANT;
      ah->pkt_src() = addr();
      ah->pkt_len() = ANT_SIZE;
      ah->pkt_seq_num() = ant_seq_num_++;
      ah->pkt_start_time() = CURRENT_TIME;
      dest = rtable_.calc_destination(addr());
      ah->pkt_dst() = dest;
      ah->pkt_mem_size() = 0;
      ah->pkt_memory_[0].node_addr = addr();
      ah->pkt_memory_[0].trip_time = 0.0;
      ah->pkt_mem_size()++;
      ch->ptype() = PT_ANT;
      ch->direction() = hdr_cmn::DOWN;
      ch->size() = IP_HDR_LEN + ah->pkt_len();
      ch->error() = 0;
      ch->addr_type() = NS_AF_INET;
      // generate next hop as per AntNet algorithm
      next = rtable_.calc_next(addr(), ah->pkt_dst(), addr());
      // if next hop same as this node, release packet
      if(next == addr()) {
            Packet::free(p);
            return;
      }
      ch->next_hop() = next;

      ih->saddr() = addr();
      ih->daddr() = next;
      ih->ttl() = 2 * (NUM_NODES);
      if(DEBUG)
```

```
          fprintf(stdout,"sending antnet packet from %d to %d next
         hop %d\n", ah->pkt_src(), ah->pkt_dst(), ih->daddr());

         target_->recv(p);    // send forward ant packet
}


void Antnet::forward_ant_pkt(Packet* p){
      struct hdr_ip* ih = HDR_IP(p);
      struct hdr_cmn* ch = HDR_CMN(p);
      struct hdr_ant_pkt* ah = HDR_ANT_PKT(p);

      nsaddr_t parent = ih->saddr();
      // find next hop node as per AntNet algorithm
      nsaddr_t next = rtable_.calc_next(addr(), ah->pkt_dst(),
     parent);

      if(next == addr() || next == parent){
            Packet::free(p);
            return;
      }

      ch->next_hop() = next;
      ih->saddr() = addr();
      ih->daddr() = next;
      if(DEBUG)
         fprintf(stdout,"forwarding antnet packet from %d source %d
        dest %d next hop %d\n", addr(), ah->pkt_src(), ah->pkt_dst(),
        ih->daddr());
      // send packet to next hop node
      target_->recv(p);
}
```

c. Menghitung probabilitas setiap *node*

```
nsaddr_t  antnet_rtable::calc_next(nsaddr_t  source,  nsaddr_t  dest,
nsaddr_t parent) {
      nsaddr_t next, nextn;
      double thisph;
      double thisqueue;
      double thisprob;
      double maxprob = 0.0;
      double maxph = 0.0;
      double lrange = 0.0, urange = 0.0;
      // find routing table entry for destination node
      rtable_t::iterator iter = rt_.find(dest);
      double qtotal = 0.0;
      if(DEBUG)
           fprintf(stdout,"in calc_next at source %d dest %d parent
           %d\n",source,dest,parent);
      if(iter != rt_.end()) {
            pheromone_matrix vect_pheromone;

            if(DEBUG) {
                  vect_pheromone = (*iter).second;
                  fprintf(stdout,"neighbors of %d:\t",source);
                  for(pheromone_matrix::iterator iterPh =
                vect_pheromone.begin(); iterPh !=
                vect_pheromone.end(); iterPh++) {
```

```
                fprintf(stdout,"%d\t%f\t",(*iterPh).neighbor,
                (*iterPh).phvalue);
            }
                  fprintf(stdout,"\n");
      }


            vect_pheromone = (*iter).second;
            for(pheromone_matrix::iterator iterPh =
                vect_pheromone.begin(); iterPh !=
                vect_pheromone.end(); iterPh++) {
                    next = (*iterPh).neighbor;
                    Node *node1 = node1->get_node_by_address(source);
                    Node *node2 = node2->get_node_by_address(next);
                    int temp_len = get_queue_length(node1,node2);
                    qtotal += temp_len;
               }
            if(qtotal == 0.0) {
                  qtotal = 1.0;
            }

// calculate probability range for parent link
            lrange = 0.0;
            urange = 0.0;
            for(pheromone_matrix::iterator iterPh =
                vect_pheromone.begin(); iterPh !=
                vect_pheromone.end(); iterPh++) {
                  thisph = (*iterPh).phvalue;
                  next = (*iterPh).neighbor;
                  Node *node1 = node1->get_node_by_address(source);
                  Node *node2 = node2->get_node_by_address(next);
                  int thisqueue = get_queue_length(node1,node2);
                  thisprob = (thisph + ALPHA*(1 - thisqueue/qtotal))
                   / (1 + ALPHA*(N-1));
                  //thisprob = thisph;
                  if(next == parent) {
                        urange = lrange + (thisph);
                        break;
                  }
                  lrange += (thisph);
            }
            if(urange == 0.0)
                  urange = 1.0;

            // dead end, loopback
            if(lrange == 0.0 && urange == 1.0) {
            //    printf("return parent %d\n",parent);
                  return parent;
            }


            double tmp_double;
            do {
                  tmp_double = rnum->uniform(1.0);
                  }
               while(tmp_double >= lrange && tmp_double < urange);
                  lrange = 0.0;
                  urange = 0.0;
            for(pheromone_matrix::iterator iterPh =
                vect_pheromone.begin(); iterPh !=
```

```
                vect_pheromone.end(); iterPh++) {
                   thisph = (*iterPh).phvalue;
                   next = (*iterPh).neighbor;
                   Node *node1 = node1->get_node_by_address(source);
                   Node *node2 = node2->get_node_by_address(next);
                   int thisqueue = get_queue_length(node1,node2);
                   thisprob = (thisph + ALPHA*(1 - thisqueue/qtotal))
                    / (1 + ALPHA*(N-1));
                   urange += (thisph);
                   if(tmp_double >= lrange && tmp_double < urange) {
                         //printf("return next %d\n",next);
                         return next;
                   }
                   lrange = urange;
             }
      }
}
```

d. Pembaruan nilai feromon dan pengiriman paket RREP

```
void antnet_rtable::update(nsaddr_t dest, nsaddr_t next) {

      pheromone_matrix *vect_pheromone;
      pheromone_matrix temp;

      // read ruoitng table entry for destination
      rtable_t::iterator iterRt = rt_.find(dest);
      if(iterRt != rt_.end()) {
            vect_pheromone = &((*iterRt).second);
            pheromone_matrix::iterator iterPh = vect_pheromone-
       >begin();
            for(; iterPh != vect_pheromone->end(); iterPh++) {
                  double oldph = (*iterPh).phvalue;
                  if((*iterPh).neighbor == next)
                        (*iterPh).phvalue = oldph + r*(1 - oldph);
                  else
                        (*iterPh).phvalue = (1-r)*oldph;
            }
      }
}

void Antnet::backward_ant_pkt(Packet* p) {
      struct hdr_ip* ih = HDR_IP(p);
      struct hdr_cmn* ch = HDR_CMN(p);
      struct hdr_ant_pkt* ah = HDR_ANT_PKT(p);

      // find node previous to this node in memory
      int index;
      for(int i = ah->pkt_mem_size()-1; i >= 0; i--) {
            if(ah->pkt_memory_[i].node_addr == addr()) {
                  index = i-1;
                  break;
            }
      }
      // next hop node determined from memory
      ch->next_hop() = ah->pkt_memory_[index].node_addr;
      ch->direction() = hdr_cmn::UP;       // backward ant
      ch->ptype() = PT_ANT;    // packet type = Ant
      ih->saddr() = addr();    // source address
```

```
      ih->daddr() = ch->next_hop(); // destination addres

      if(DEBUG)
          fprintf(stdout,"forwarding backward antnet packet from %d
           source %d dest %d next hop %d\n", addr(), ah->pkt_src(),
           ah->pkt_dst(), ih->daddr());
      // send backward ant to next hop
      target_->recv(p);
}
```

e. Pembentukan tabel feromon

```
void Antnet::update_traffic(Packet* p) {
      //update mean, variance, best.
      struct traffic_matrix temp_traffic;
      nsaddr_t dest, next;
      double tt, oldtt;
      double oldvar;
      double varsigma = VARSIGMA;

      struct hdr_ant_pkt* ah = HDR_ANT_PKT(p);
      int i;
      for(i=0; ah->pkt_memory_[i].node_addr != addr(); i++);
         double initialtt = ah->pkt_memory_[i].trip_time;
         i++;
         next = ah->pkt_memory_[i].node_addr;

      for(int index = i; index < ah->pkt_mem_size(); index++) {
            dest = ah->pkt_memory_[index].node_addr;
            tt = ah->pkt_memory_[index].trip_time - initialtt;

            /* update sample window */
            window_t::iterator iterWin = window_.find(dest);
            if(iterWin != window_.end()) {      // destination entry
             exists, add to it in window
                  (*iterWin).second.push_back(tt);
            }
            else {      // destination entry does not exist, add new
                 dest entry to window
                  triptime_t win_tt;
                  win_tt.push_back(tt);
                  window_[dest] = win_tt;
            }
      }

      /* update traffic */
      for(int index = i; index < ah->pkt_mem_size(); index++) {

            dest = ah->pkt_memory_[index].node_addr;
            tt = ah->pkt_memory_[index].trip_time - initialtt;

            /* find best trip time from this node to dest */
            window_t::iterator iterWin = window_.find(dest);
            triptime_t win_tt = (*iterWin).second;
            triptime_t::iterator itertt = win_tt.begin();
            double mintt = (*itertt);
            for(; itertt != win_tt.end(); itertt++) {
                  if((*itertt) < mintt)
```

```
                        mintt = (*itertt);
                }

                /* update traffic */
                state_t::iterator iterFind = state_.find(dest);
                if(iterFind != state_.end()) {
                        // update existing entry
                        oldtt = (*iterFind).second.mean_tt;
                        (*iterFind).second.mean_tt = oldtt + varsigma * (tt
                        - oldtt);
                        oldvar = (*iterFind).second.var_tt;
                        (*iterFind).second.var_tt = oldvar*oldvar + varsigma
                        * ((tt - oldtt)*(tt - oldtt) -  oldvar*oldvar);
                        (*iterFind).second.best_tt = mintt;
                }
                else {
                        // add map entry
                        temp_traffic.mean_tt = tt;
                        temp_traffic.var_tt = tt;
                        temp_traffic.best_tt = mintt;
                        state_[dest] = temp_traffic;
                }
        }

        /* find r and update pheromone */
        for(int index = i; index < ah->pkt_mem_size(); index++) {

                dest = ah->pkt_memory_[index].node_addr;
                tt = ah->pkt_memory_[index].trip_time - initialtt;

                /* find r */
                double W_best = state_[dest].best_tt;
                double I_inf = W_best;
                double mu = state_[dest].mean_tt;
                double sigma = sqrt(state_[dest].var_tt);
                int w = get_win_size(dest);
                double I_sup = mu + zee * (sigma/sqrt(w));
                if(I_sup == I_inf && I_inf == tt)
                        r = 0.0;
                else
                        r = c1*(W_best/tt) + c2 * ((I_sup - I_inf) / ((I_sup
                        - I_inf) + (tt - I_inf) ));

                if(DEBUG) {
                        printf("r = %f\n", r);
                }
        }
}
```

LAMPIRAN 3

KONFIGURASI FILE POTOKOL *ROUTING* AOMDV-SEMUT

3.1 Script File TCL AOMDV-Semut

```
#menentukan parameter konfigurasi VANET
set val(chan)           Channel/WirelessChannel    ;# tipe channel
set val(prop)           Propagation/TwoRayGround   ;# model radio-
propagation
set val(netif)          Phy/WirelessPhy            ;# tipe perangkat
jaringan
set val(mac)            Mac/802_11                 ;# tipe MAC
set val(ifq)            Queue/DropTail/PriQueue    ;# interface queue
type
set val(ll)             LL                         ;# tipe link layer
set val(ant)            Antenna/OmniAntenna        ;# antenna model
set val(ifqlen)         50                         ;# max packet in
ifq
set val(nn)             50                         ;# jumlah node
set val(rp)             AOMDV                       ;# routing
protocol
set opt(x) 6193.71
set opt(y) 3930.62


# Inisialisasi Variabel-variabel global
set ns_ [new Simulator]

set tracefd [open vanet_aomdv.tr w]
$ns_ trace-all $tracefd

set namf [open vanet_aomdvaco.nam w]
$ns_ namtrace-all-wireless $namf $opt(x) $opt(y)
# Setup objek topologi
set topo [new Topography]

$topo load_flatgrid $opt(x) $opt(y)

# menciptakan God
create-god $val(nn)
set god_ [God instance]


# configure node

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

#penciptaan node, dan inisialisasii posisi awal node
 for {set i 0} {$i < $val(nn) } {incr i} {
  set node_($i) [$ns_ node]
```

```
  #$node_($i) color blue
  #$ns_ at 0.0 "$node_($i) color blue"
  $node_($i) random-motion 0  ;# mendisable random motion
  $ns_ initial_node_pos $node_($i) 30
 }

#Menciptakan Agent Semut
set sz 36
$ns_ multicast
for {set i 0} {$i < $sz} {incr i} {
      set  na($i)  [new Agent/Antnet $i]

}

#Menghubungkan Agen dan Node
for {set j 0} {$j < $sz} {incr j} {
    for {set i 0} {$i < $val(nn)} {incr i} {
          $ns_ attach-agent  $node_($i)  $na($j)
    }
}

# Menghubungkan Agent dan Node
for {set i 0} {$i < $sz} {incr i} {
      for {set j 0} {$j < $sz} {incr j} {
        $ns_ connect $na($i) $na($j)
    }
}

#Create connection between the nodes
for {set i 0} {$i < [expr $val(nn)-1]} {incr i} {
      for {set j 0} {$j < $sz } {incr j} {
        #$ns_ connect $na($i) $na($j)
        #Add neighbors
        for {set k 0} {$k < [expr $val(nn)-1]} {incr k} {
            $ns_ at now "$na(0) add-neighbor $node_($i) $node_($k)"
        }
    }
}

# Set parameters and start time
for {set i 0} {$i < $sz} {incr i} {
      $na($i) set num_nodes_ $sz
      $na($i) set timer_ant_ 6000
      $na($i) set r_factor_ 0.001
      $ns_  at  0.005 "$na($i) start"
}

# Mendefinisikan Warna
$ns_ color 1 dodgerblue
$ns_ color 2 blue
$ns_ color 3 cyan
$ns_ color 4 green
$ns_ color 5 yellow
$ns_ color 6 black
$ns_ color 7 magenta
$ns_ color 8 gold
$ns_ color 9 red

#Menciptkana array untuk menyimpan warna
set colorname(0) blue
```

```
set colorname(1) cyan
set colorname(2) green
set colorname(3) red
set colorname(4) gold
set colorname(5) magenta

# mendefinisikan prosedur untuk memberi warna pada node secara
dinamis
proc dynamic-color {} {
      global ns_ val node_ colorname
      set time 0.3
      set now [$ns_ now]
      set Rand [expr round(rand()*5)]
      for {set i 0} {$i < $val(nn) } {incr i } {
            $node_($i) color $colorname($Rand)
            $ns_ at $now "$node_($i) color $colorname($Rand)"
      }
      $ns_ at [expr $now+$time] "dynamic-color"
}

source vanet_mobility.tcl

#mendefinisikan prosedur/fungsi stop
proc stop {} {
    global ns_ tracefd namf

      $ns_ flush-trace
    close $tracefd
      close $namf

      #menjalankan file
    exec nam vanet_aomdvaco.nam &
    exec awk -f throughput.awk vanet_aomdv.tr &
    exec awk -f e2edelay.awk vanet_aomdv.tr &
    exec awk -f pdr.awk vanet_aomdv.tr &

    exit 0
}

#mendefinisikan prosedur atau fungsi untuk pengiriman packet data
proc send-packet { node sink size interval rate } {
      #membuat objek simulator
      set ns [Simulator instance]

      #Membuat UPD agent dan menambahkannya pada node
      set source [new Agent/UDP]
      $ns attach-agent $node $source

      #mendepinisikan paket trafik
      set traffic [new Application/Traffic/CBR]
      $traffic set packetSize_ $size
      $traffic set rate_ $rate
    $traffic set maxpkts_ 4028
    $traffic set interval_ $interval
    $traffic set random_ 1
    $traffic set type_        cbr

    # menambahkan trafik paket ke node sumber
    $traffic attach-agent $source
      #menghubungkan node sumber ke node tujuan
```

```
      $ns connect $source $sink
      return $traffic
}

#menciptakan agent node penerima
set sink0 [new Agent/LossMonitor]
$ns_ attach-agent $node_(18) $sink0
$node_(18) color red
$ns_ at 1.0 "$node_(18) color red"

set sink1 [new Agent/LossMonitor]
$ns_ attach-agent $node_(1) $sink1
$node_(1) color blue
$ns_ at 1.0 "$node_(1) color blue"

set sink2 [new Agent/LossMonitor]
$ns_ attach-agent $node_(32) $sink2
$node_(32) color green
$ns_ at 1.0 "$node_(32) color green"

#melakukan pengiriman packet
set source0 [send-packet $node_(16) $sink0 512 1 1024k]
$node_(16) color red
$ns_ at 1.0 "$node_(16) color red"

set source1 [send-packet $node_(9) $sink1 512 1 1024k]
$node_(9) color blue
$ns_ at 1.0 "$node_(9) color blue"

set source2 [send-packet $node_(22) $sink2 512 1 1024k]
$node_(22) color green
$ns_ at 1.0 "$node_(22) color green"



for {set i 0} {$i < $val(nn) } {incr i} {
    $ns_ at 200.0 "$node_($i) reset";
}

#Set stop time for AntNet algorithm
for {set i 0} {$i < $sz} {incr i} {
    $ns_ at 200.0 "$na($i) stop"
}

#Print routing tables generated by AntNet
for {set i 0} {$i < $sz} {incr i} {
      $ns_ at 1.0 "$na($i) print_rtable"
}

#memulai pengiriman paket
$ns_ at 1.0 "$source0 start"
$ns_ at 1.0 "$source1 start"
$ns_ at 1.0 "$source2 start"
# #mengakhiri pengiriman packet
$ns_ at 199.0 "$source0 stop"
$ns_ at 199.0 "$source1 stop"
$ns_ at 199.0 "$source2 stop"
$ns_ at 200 "stop"
$ns_ at 200.01 "puts \"NS SELESAI...\" ; $ns_ halt"
```

```
puts "Memulai Simulasi..."
$ns_ run
```

LAMPIRAN 4

KODE AWK SCRIPT

4.1 Kode awk Perhitungan *Throughput*

```
BEGIN {
       recvdSize = 0
       startTime = 400
       stopTime = 0
  }

  {
   event = $1
   time = $2
   node_id = $3
   pkt_size = $8
   level = $4

  # menyimpan waktu start
  if ((level == "AGT") && (event == "s") && pkt_size >= 128) {
    if (time < startTime) {
              startTime = time
              }
       }

  # memperbarui total paket yang diterima dan menyimpan waktu
penerimaann
  if ((level == "AGT" || level == "PTR") && (event == "r") &&
pkt_size >= 128) {
       if (time > stopTime) {
             stopTime = time
       }
       # Rip off the header
       hdr_size = pkt_size % 128
       pkt_size -= hdr_size

       #menyimpan ukuran paket yang diterima
       recvdSize += pkt_size
       }
  }

  END {
       print ""
       print "Average Throughput = "(recvdSize/(stopTime-
startTime))*(8/1000) "Kbps"
  }
```

4.2 Kode awk Perhitungan *Average End to end delay*

```
BEGIN {
    droppedPackets = 0;
    paketsize=0;
    seqno = -1;
    count = 0;
}
    #end-to-end delay
    if($4 == "AGT" && $1 == "s") {
          start_time[$6] = $2;

    } else if(($7 == "cbr") && ($1 == "r")) {
        end_time[$6] = $2;
```

```
    } else if($1 == "D" && $7 == "cbr") {
          end_time[$6] = -1;
    }
}


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

    }

    for(i=0; i<=seqno; i++) {
          if(delay[i] > 0) {
              n_to_n_delay = n_to_n_delay + delay[i];
        }
    }

   n_to_n_delay = n_to_n_delay/count;


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

}
```

4.3 Kode awk Perhitungan *Packet delivery ratio*

```
# Packet delivery ratio

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

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

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

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

END {
            print "";
```

```
        print "Packet delivery ratio = " (recvLine/sendLine)*100 "
%";
            #printf "cbr s:%d r:%d, r/s Ratio:%.4f, f:%d \n",
sendLine, recvLine, (recvLine/sendLine),fowardLine;
}
```

LAMPIRAN 5

HASIL PENGUJIAN SKENARIO PENELITIAN

5.1 Hasil Pengujian Kinerja Protokol AOMDV Standar

5.1.1 Percobaan 50 *Node* AOMDV Standar

a. Percobaan 1 AOMDV Standar

Tabel 1 Hasil ujicoba percobaan 1, 50 *node*

Kecepatan (Km/Jam)	***Average Throughput* (Kbps)**	***Average End to end delay* (ms)**	***Packet delivery ratio* (%)**
30	67.1168	9.50251	68.9662
60	65.9883	23.1761	66.8984
90	64.8499	33.8944	65.0978
120	64.0149	40.5115	63.3174
150	62.0448	53.7026	62.7451

b. Percobaan 2 AOMDV Standar

Tabel 2 Hasil ujicoba percobaan 2, 50 *node*

Kecepatan (Km/Jam)	***Average Throughput* (Kbps)**	***Average End to end delay* (ms)**	***Packet delivery ratio* (%)**
30	64.0069	22.8603	65.8975
60	63.0865	32.8614	65.1398
90	62.8499	38.8944	62.0975
120	62.6435	56.6318	61.0857
150	59.6303	66.7851	60.5536

c. Percobaan 3 AOMDV Standar

Tabel 3 Hasil ujicoba percobaan 3, 50 *node*

Kecepatan (Km/Jam)	***Average Throughput* (Kbps)**	***Average End to end delay* (ms)**	***Packet delivery ratio* (%)**
30	70.3441	11.1389	73.3508
60	70.2903	26.0846	70.6947
90	70.2615	32.8883	69.0826
120	67.4707	34.4813	65.4284
150	67.1962	42.8105	63.2545

d. Percobaan 4 AOMDV Standar

Tabel 4 Hasil ujicoba percobaan 4, 50 *node*

Kecepatan (Km/Jam)	***Average Throughput* (Kbps)**	***Average End to end delay* (ms)**	***Packet delivery ratio* (%)**
30	68.9175	17.4705	64.3301
60	67.1991	24.4133	62.9258
90	63.2037	29.3096	61.3824
120	61.2634	36.2576	58.0421
150	57.9833	42.2857	55.3457

e. Percobaan 5 AOMDV Standar

Tabel 5 Hasil ujicoba percobaan 5, 50 *node*

Kecepatan (Km/Jam)	***Average Throughput* (Kbps)**	***Average End to end delay* (ms)**	***Packet delivery ratio* (%)**
30	69.9426	24.3533	65.2792
60	63.4172	36.4299	63.8787
90	63.1117	43.6834	60.8804
120	57.9888	51.9314	59.2469
150	57.8683	59.6196	57.7148

4.1.2 Percobaan 70 *Node* AOMDV Standar

a. Percobaan 1 AOMDV Standar

Tabel 6 Hasil ujicoba percobaan 1, 70 *node*

Kecepatan (Km/Jam)	***Average Throughput* (Kbps)**	***Average End to end delay* (ms)**	***Packet delivery ratio* (%)**
30	69.6943	38.1294	73.5903
60	68.8862	42.9108	71.3876
90	67.0526	56.3556	70.7277
120	64.9468	59.0868	68.0056
150	62.0526	85.3556	66.7277

b. Percobaan 2 AOMDV Standar

Tabel 7 Hasil ujicoba percobaan 2, 70 *node*

Kecepatan (Km/Jam)	***Average Throughput* (Kbps)**	***Average End to end delay* (ms)**	***Packet delivery ratio* (%)**
30	74.6185	33.2132	72.2339
60	73.9057	49.7569	72.2908
90	73.6563	52.6204	71.4211
120	71.5309	68.8013	69.3607
150	70.8338	95.6063	67.6535

c. Percobaan 3 AOMDV Standar

Tabel 8 Hasil ujicoba percobaan 3, 70 *node*

Kecepatan (Km/Jam)	***Average Throughput* (Kbps)**	***Average End to end delay* (ms)**	***Packet delivery ratio* (%)**
30	79.6826	27.7074	76.1568
60	77.2819	37.5506	74.4346
90	75.4424	44.2726	74.0849
120	72.7276	51.2477	73.7668
150	70.5575	61.8224	71.7292

d. Percobaan 4 AOMDV Standar

Tabel 9 Hasil ujicoba percobaan 4, 70 *node*

Kecepatan (Km/Jam)	***Average Throughput* (Kbps)**	***Average End to end delay* (ms)**	***Packet delivery ratio* (%)**
30	78.8511	31.3561	73.2852
60	78.0117	48.9625	72.9855
90	75.9473	53.3374	70.0392
120	71.9993	57.8957	68.7445
150	70.9473	88.3374	68.0392

e. Percobaan 5 AOMDV Standar

Tabel 10 Hasil ujicoba percobaan 5, 70 *node*

Kecepatan (Km/Jam)	***Average Throughput* (Kbps)**	***Average End to end delay* (ms)**	***Packet delivery ratio* (%)**
30	79.8213	43.9649	75.3356
60	76.1576	52.2551	72.8376
90	73.3559	59.5629	71.8222
120	72.0429	64.1421	69.7883
150	71.3972	96.0704	67.9634

4.1.2 Percobaan 100 *Node* AOMDV Standar

a. Percobaan 1 AOMDV Standar

Tabel 11 Hasil ujicoba percobaan 1, 100 *node*

Kecepatan (Km/Jam)	***Average Throughput* (Kbps)**	***Average End to end delay* (ms)**	***Packet delivery ratio* (%)**
30	77.3631	48.9405	81.2739
60	76.5189	63.0383	81.1926
90	74.4772	76.9381	80.2487
120	72.5971	82.7618	79.8347
150	72.3487	102.281	77.9041

b. Percobaan 2 AOMDV Standar

Tabel 12 Hasil ujicoba percobaan 2, 100 *node*

Kecepatan (Km/Jam)	***Average Throughput* (Kbps)**	***Average End to end delay* (ms)**	***Packet delivery ratio* (%)**
30	82.7619	37.4243	83.5494
60	79.8951	54.7916	82.3171
90	77.6592	75.4776	79.8493
120	77.5207	84.6751	76.4041
150	74.6446	131.697	75.1181

c. Percobaan 3 AOMDV Standar

Tabel 13 Hasil ujicoba percobaan 3, 100 *node*

Kecepatan (Km/Jam)	***Average Throughput* (Kbps)**	***Average End to end delay* (ms)**	***Packet delivery ratio* (%)**
30	83.0914	57.4325	85.6125
60	81.3448	69.6324	82.2339
90	81.2322	75.2452	80.0798
120	79.3068	96.2541	78.6968
150	79.2111	137.482	77.9333

d. Percobaan 4 AOMDV Standar

Tabel 14 Hasil ujicoba percobaan 4, 100 *node*

Kecepatan (Km/Jam)	***Average Throughput* (Kbps)**	***Average End to end delay* (ms)**	***Packet delivery ratio* (%)**
30	82.9999	58.2025	84.7697
60	81.7297	76.1765	82.2125
90	80.1057	82.4055	79.8082
120	79.4065	97.6548	78.0394
150	75.4282	130.579	74.3617

e. Percobaan 5 AOMDV Standar

Tabel 15 Hasil ujicoba percobaan 5, 100 *node*

Kecepatan (Km/Jam)	***Average Throughput* (Kbps)**	***Average End to end delay* (ms)**	***Packet delivery ratio* (%)**
30	84.8455	39.5456	87.7979
60	82.7662	49.8008	84.7441
90	81.4772	68.9381	83.2487
120	77.5971	90.7636	80.8347
150	74.3487	126.283	78.9041

5.2 Hasil Pengujian Kinerja Protokol AOMDV-Semut

5.2.1 Percobaan 50 *Node* AOMDV-Semut

a. Percobaan 1 AOMDV-Semut

Tabel 16 Hasil ujicoba percobaan 1, 50 *node* dengan algoritma Semut

Kecepatan (Km/Jam)	***Average Throughput* (Kbps)**	***Average End to end delay* (ms)**	***Packet delivery ratio* (%)**
30	79.0411	7.98426	79.0763
60	78.0988	12.3664	78.3843
90	77.4232	32.4245	77.2585
120	73.9793	33.2618	76.4304
150	70.7995	43.4727	74.6568

b. Percobaan 2 AOMDV-Semut

Tabel 17 Hasil ujicoba percobaan 2, 50 *node* dengan algoritma Semut

Kecepatan (Km/Jam)	***Average Throughput* (Kbps)**	***Average End to end delay* (ms)**	***Packet delivery ratio* (%)**
30	79.1493	11.8254	76.5174
60	78.6267	27.4897	76.0129
90	78.5106	33.7184	72.7636
120	77.7844	41.4532	71.4424
150	77.7187	48.7306	70.9998

c. Percobaan 3 AOMDV-Semut

Tabel 18 Hasil ujicoba percobaan 3, 50 *node* dengan algoritma Semut

Kecepatan (Km/Jam)	***Average Throughput* (Kbps)**	***Average End to end delay* (ms)**	***Packet delivery ratio* (%)**
30	80.2561	6.11734	80.9314
60	78.4725	10.1725	79.4733
90	78.2878	26.0124	76.7649
120	73.5535	32.5626	75.9742
150	73.1816	35.0368	75.3467

d. Percobaan 4 AOMDV-Semut

Tabel 19 Hasil ujicoba percobaan 4, 50 *node* dengan algoritma Semut

Kecepatan (Km/Jam)	***Average Throughput* (Kbps)**	***Average End to end delay* (ms)**	***Packet delivery ratio* (%)**
30	78.9106	10.4258	71.1262
60	73.1928	21.9557	70.6477
90	73.1528	28.3997	69.3695
120	71.8901	30.0151	69.1246
150	68.9419	36.2685	67.5881

e. Percobaan 5 AOMDV-Semut

Tabel 20 Hasil ujicoba percobaan 5, 50 *node* dengan algoritma Semut

Kecepatan (Km/Jam)	***Average Throughput* (Kbps)**	***Average End to end delay* (ms)**	***Packet delivery ratio* (%)**
30	79.7186	14.8606	77.1538
60	77.0315	23.9049	76.9262
90	76.9284	27.1928	75.8704
120	75.9425	29.8266	71.0938
150	75.4282	36.5818	70.2952

4.2.2 Percobaan 70 *Node* AOMDV-Semut

a. Percobaan 1 AOMDV-Semut

Tabel 21 Hasil ujicoba percobaan 1, 70 *node* dengan algoritma Semut

Kecepatan (Km/Jam)	***Average Throughput* (Kbps)**	***Average End to end delay* (ms)**	***Packet delivery ratio* (%)**
30	80.4064	22.0756	84.8768
60	79.9488	28.4647	82.8883
90	76.7277	38.9815	80.7727
120	74.8899	43.0905	77.1193
150	74.7277	47.9815	75.7727

b. Percobaan 2 AOMDV-Semut

Tabel 22 Hasil ujicoba percobaan 2, 70 *node* dengan algoritma Semut

Kecepatan (Km/Jam)	***Average Throughput* (Kbps)**	***Average End to end delay* (ms)**	***Packet delivery ratio* (%)**
30	83.5437	20.0807	85.5632
60	82.9705	31.3288	83.6775
90	79.8961	39.4193	80.1609
120	77.9006	52.0748	78.5054
150	77.7973	61.6772	77.7973

c. Percobaan 3 AOMDV-Semut

Tabel 23 Hasil ujicoba percobaan 3, 70 *node* dengan algoritma Semut

Kecepatan (Km/Jam)	***Average Throughput* (Kbps)**	***Average End to end delay* (ms)**	***Packet delivery ratio* (%)**
30	88.6867	21.3647	89.4178
60	86.2804	30.6043	87.3611
90	84.2496	38.0947	87.2247
120	79.8462	45.2455	84.1091
150	79.2816	47.9456	80.5853

d. Percobaan 4 AOMDV-Semut

Tabel 24 Hasil ujicoba percobaan 4, 70 *node* dengan algoritma Semut

Kecepatan (Km/Jam)	***Average Throughput* (Kbps)**	***Average End to end delay* (ms)**	***Packet delivery ratio* (%)**
30	87.6187	27.7586	83.2678
60	86.8351	34.8305	82.9186
90	81.8349	49.5062	80.3306
120	80.9316	54.8287	79.6906
150	76.8349	74.5062	77.3306

e. Percobaan 5 AOMDV-Semut

Tabel 25 Hasil ujicoba percobaan 5, 70 *node* dengan algoritma Semut

Kecepatan (Km/Jam)	***Average Throughput* (Kbps)**	***Average End to end delay* (ms)**	***Packet delivery ratio* (%)**
30	86.7873	29.0506	84.2041
60	82.1294	43.7257	81.9299
90	80.4554	48.7457	80.6601
120	75.8591	55.0261	79.6755
150	73.1034	73.8586	77.5014

4.2.3 Percobaan 100 *Node* AOMDV-Semut

a. Percobaan 1 AOMDV-Semut

Tabel 22 Hasil ujicoba percobaan 1, 100 *node* dengan algoritma Semut

Kecepatan (Km/Jam)	***Average Throughput* (Kbps)**	***Average End to end delay* (ms)**	***Packet delivery ratio* (%)**
30	87.4454	31.7833	89.6248
60	84.6504	38.6099	88.4843
90	82.4137	50.1663	84.2287
120	80.7603	67.3867	82.4847
150	80.5309	79.0473	81.2498

b. Percobaan 2 AOMDV-Semut

Tabel 24 Hasil ujicoba percobaan 2, 100 *node* dengan algoritma Semut

Kecepatan (Km/Jam)	***Average Throughput* (Kbps)**	***Average End to end delay* (ms)**	***Packet delivery ratio* (%)**
30	91.0914	36.5501	93.2831
60	90.7469	43.0844	92.4624
90	88.0217	59.9935	90.1146
120	87.5398	70.7905	89.9662
150	84.5809	99.7184	86.6369

c. Percobaan 3 AOMDV-Semut

Tabel 26 Hasil ujicoba percobaan 3, 100 *node* dengan algoritma Semut

Kecepatan (Km/Jam)	***Average Throughput* (Kbps)**	***Average End to end delay* (ms)**	***Packet delivery ratio* (%)**
30	91.0264	36.8841	95.9361
60	90.4486	42.8115	92.0125
90	87.5027	59.1674	89.4634
120	85.6233	68.9794	88.2673
150	83.8869	72.5449	87.3219

e. Percobaan 4 AOMDV-Semut

Tabel 28 Hasil ujicoba percobaan 4, 100 *node* dengan algoritma Semut

Kecepatan (Km/Jam)	***Average Throughput* (Kbps)**	***Average End to end delay* (ms)**	***Packet delivery ratio* (%)**
30	87.5318	42.0686	94.8098
60	85.2169	50.5497	90.6749
90	83.3866	55.5296	88.0302
120	82.9627	79.3909	85.5457
150	81.7958	95.9337	83.4703

e. Percobaan 5 AOMDV-Semut

Tabel 30 Hasil ujicoba percobaan 5, 100 *node* dengan algoritma Semut

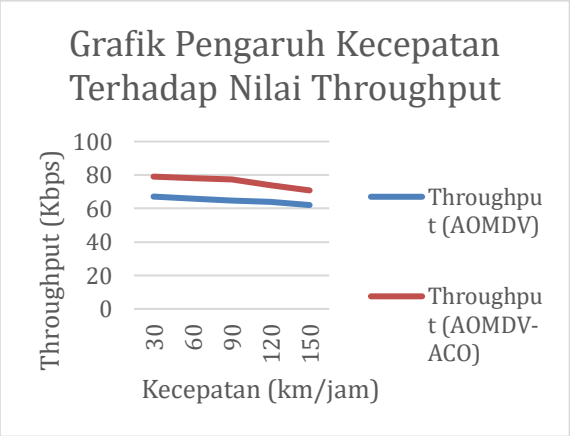
Kecepatan (Km/Jam)	***Average Throughput* (Kbps)**	***Average End to end delay* (ms)**	***Packet delivery ratio* (%)**
30	95.2705	26.4555	98.5002
60	93.2567	38.5872	96.3119
90	91.1559	52.7615	94.0121
120	90.1071	65.6145	90.4153
150	88.3469	76.9903	88.6366

LAMPIRAN 6

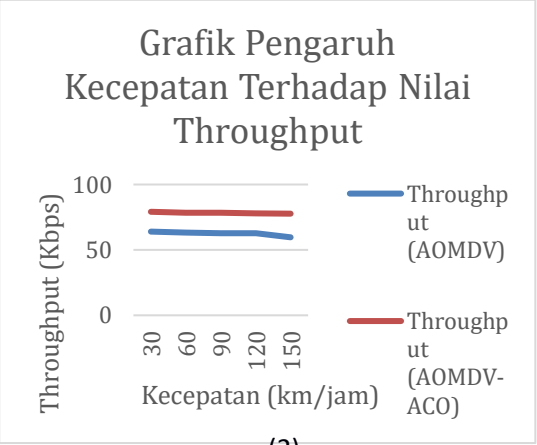
GRAFIK HASIL PERCOBAAN

6.1 Grafik Hasil Ujicoba *Throughput*

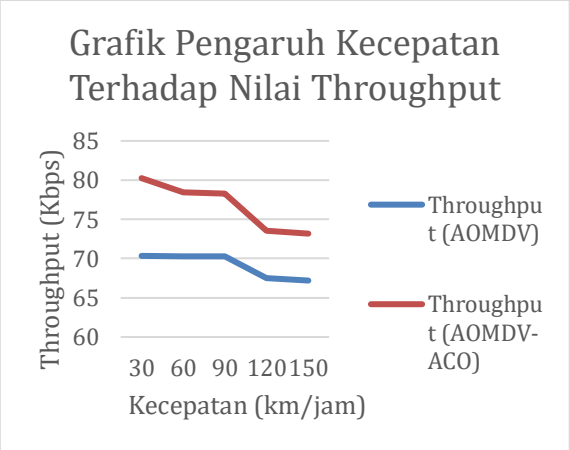
a. Grafik Hasil Ujicoba *Throughput* 50 *Node*



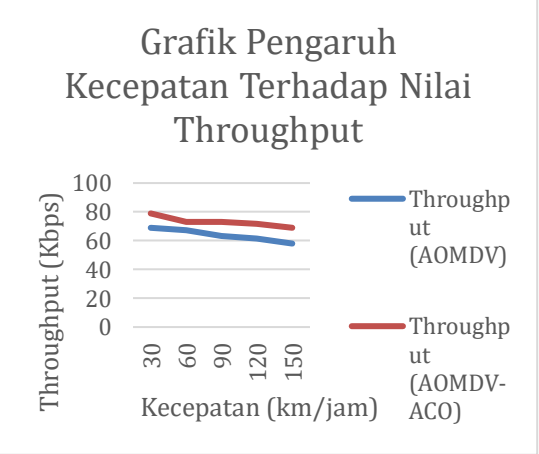
(1)



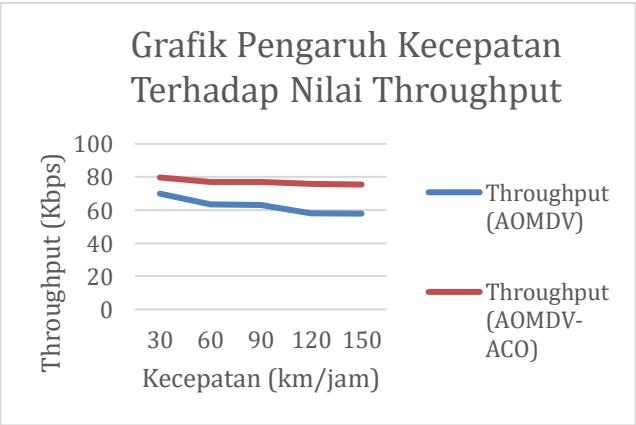
(2)



(3)

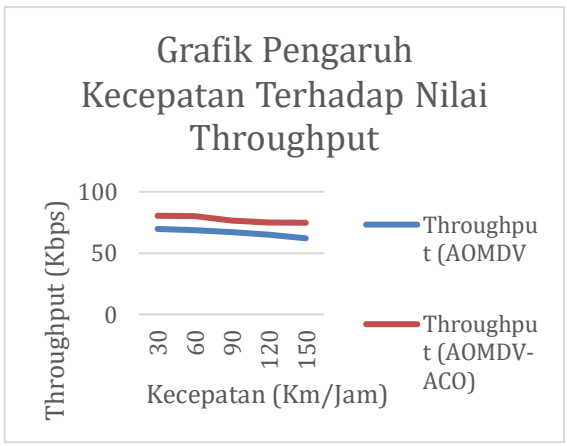


(4)

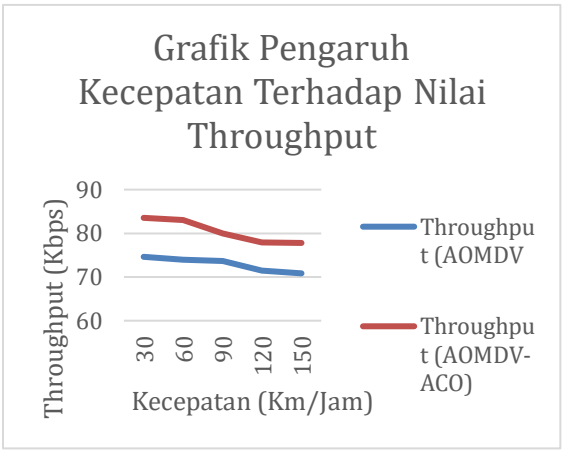


(5)

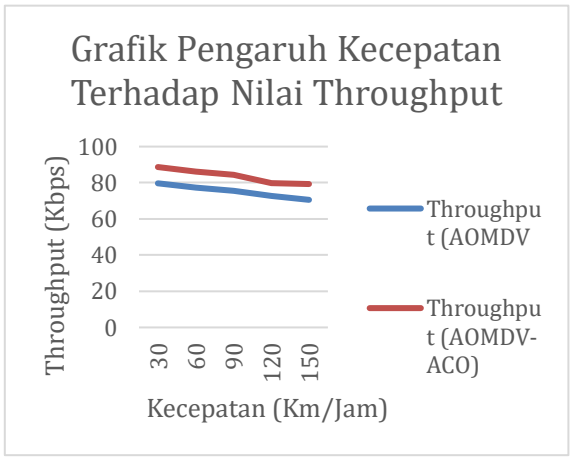
b. Grafik Hasil Ujicoba *Throughput* 70 *Node*



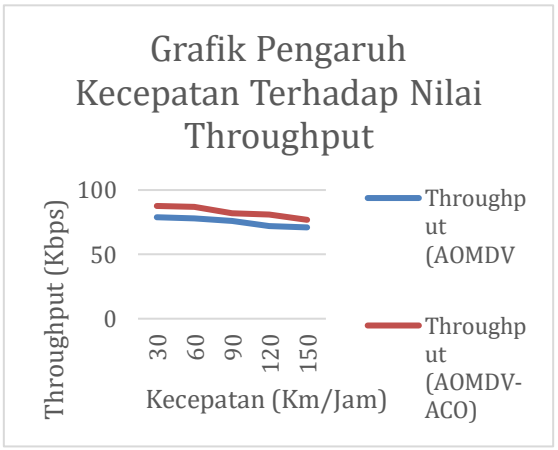
(1)



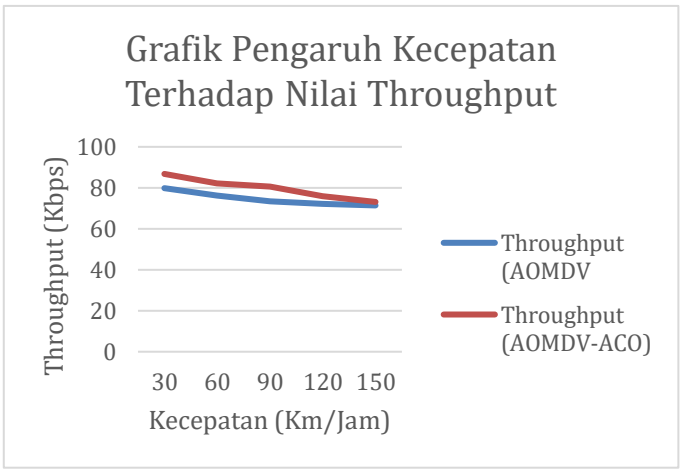
(2)



(3)

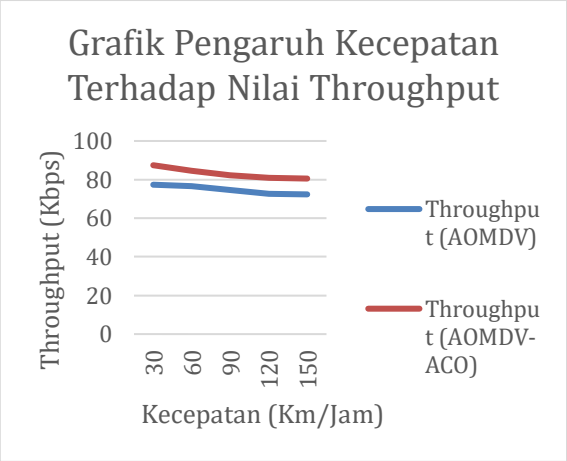


(4)

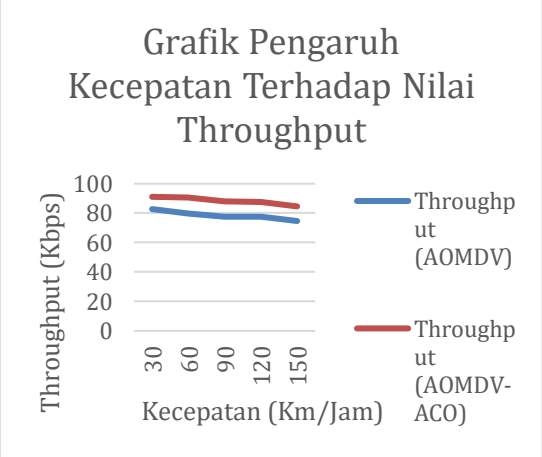


(5)

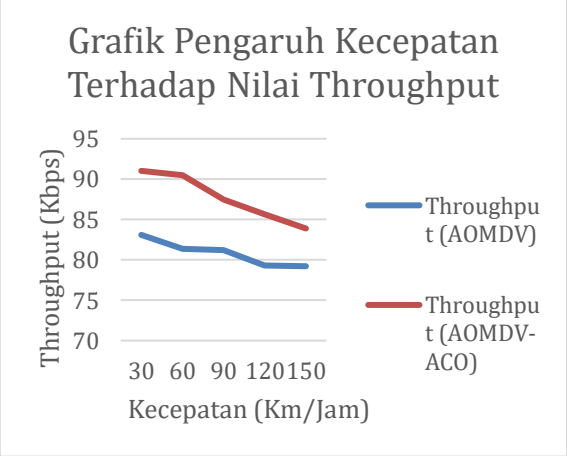
c. Grafik Hasil Ujicoba *Throughput* 100 *Node*



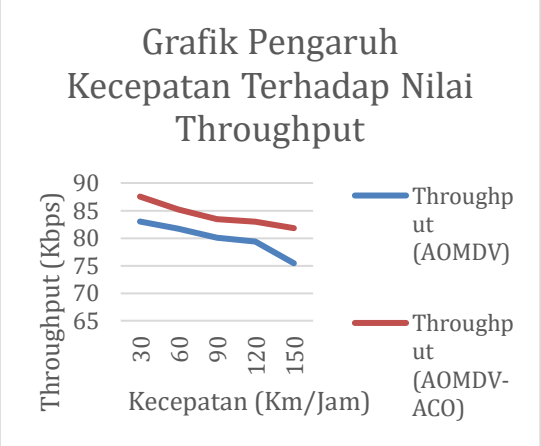
(1)



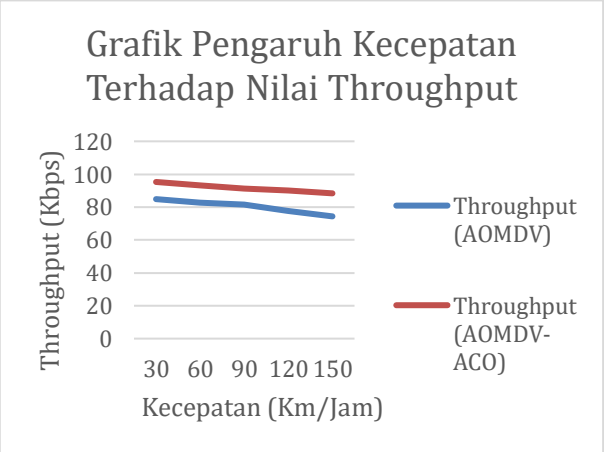
(2)



(3)



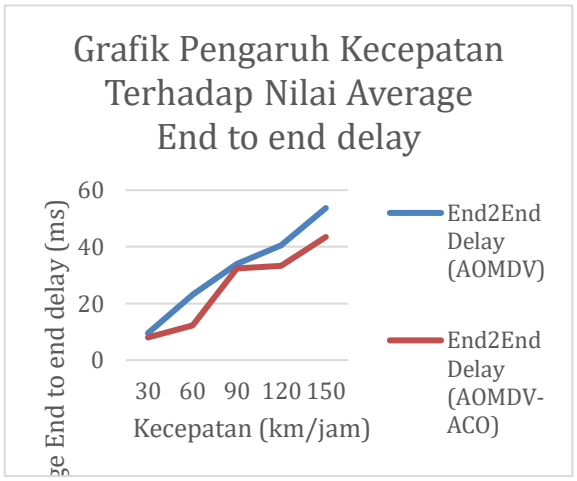
(4)



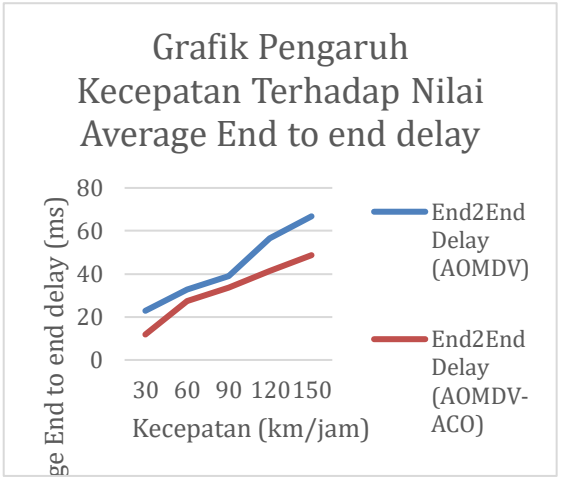
(5)

6.2 Grafik Hasil Ujicoba *Average End to end delay*

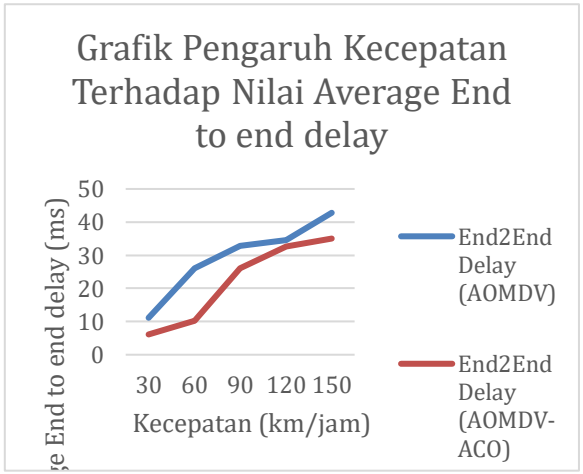
a. Grafik Hasil Ujicoba *Average End to end delay* 50 *Node*



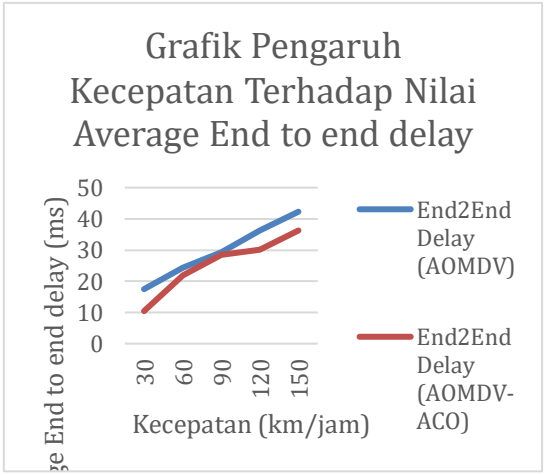
(1)



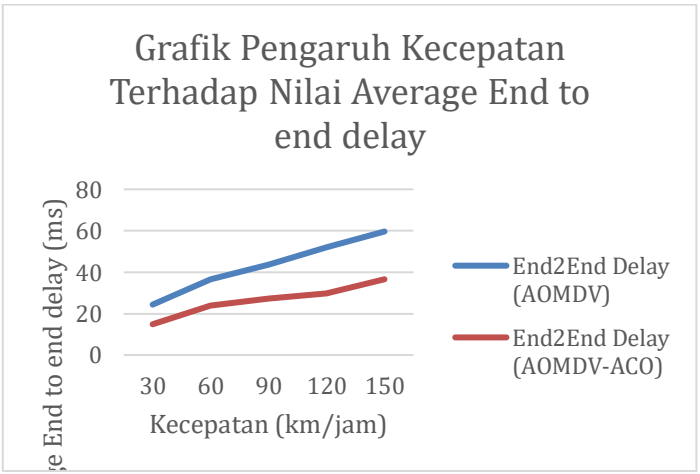
(2)



(3)

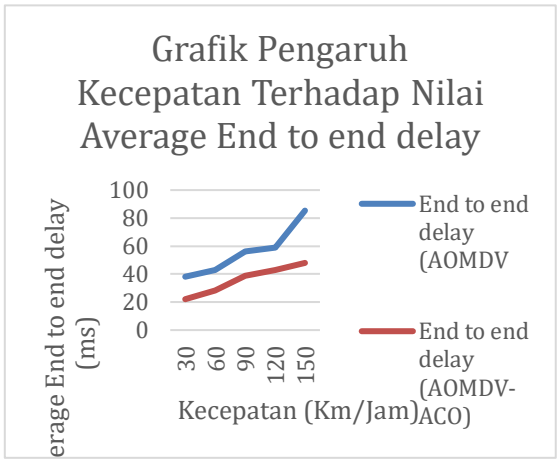


(4)

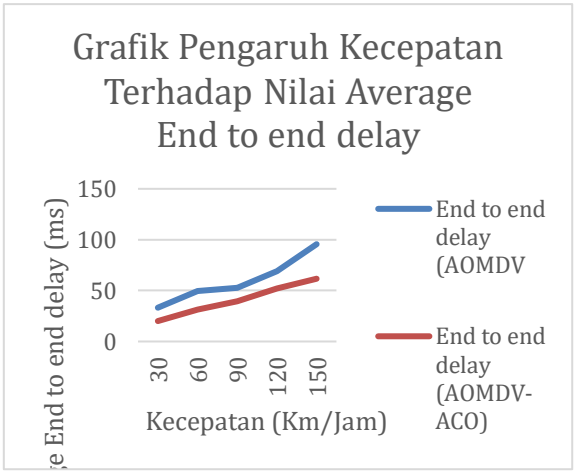


(5)

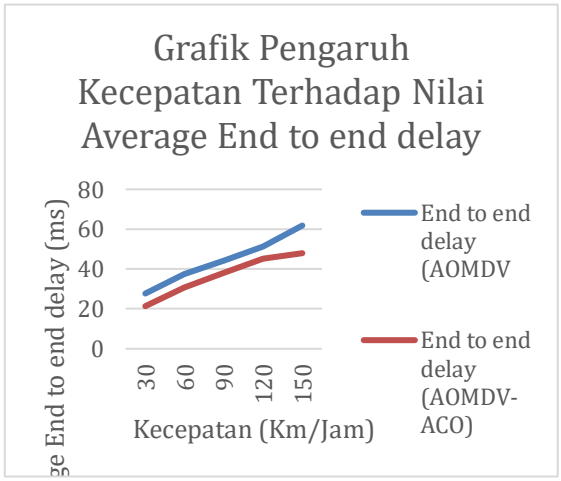
b. Grafik Hasil Ujicoba *Average End to end delay* 70 *Node*



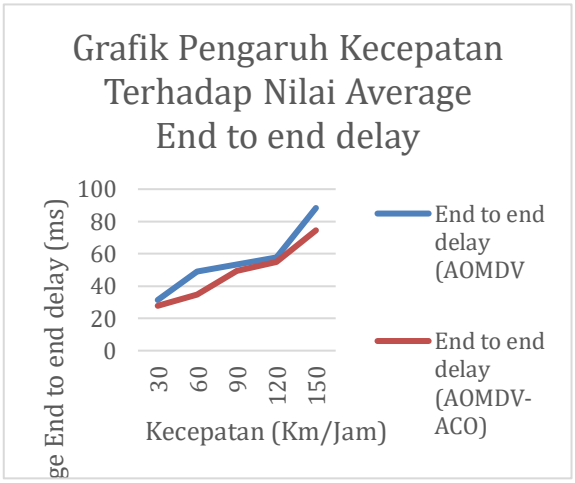
(1)



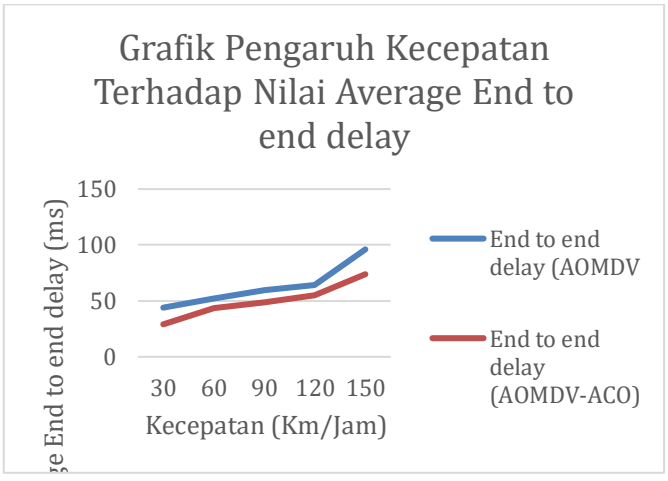
(2)



(2)

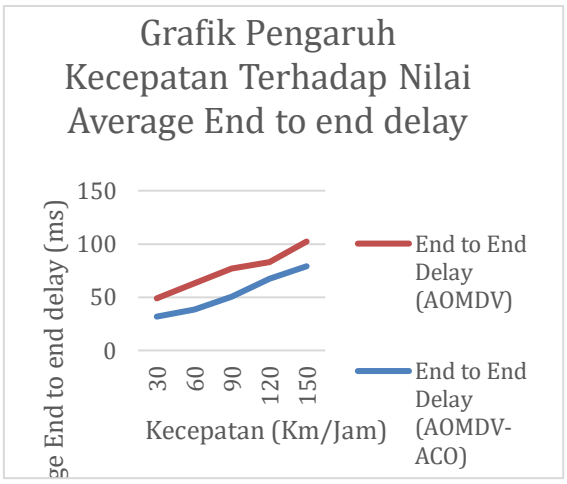


(4)

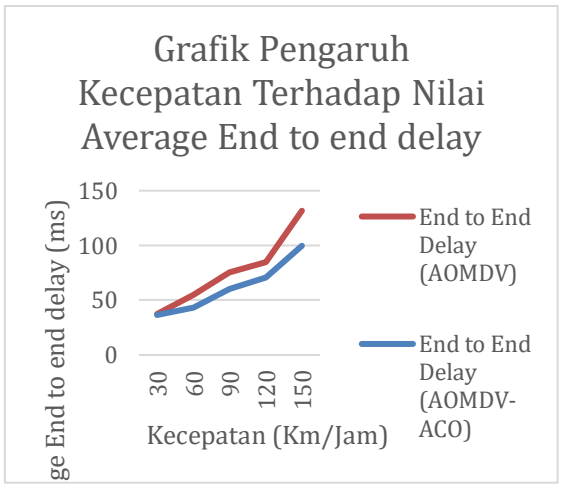


(5)

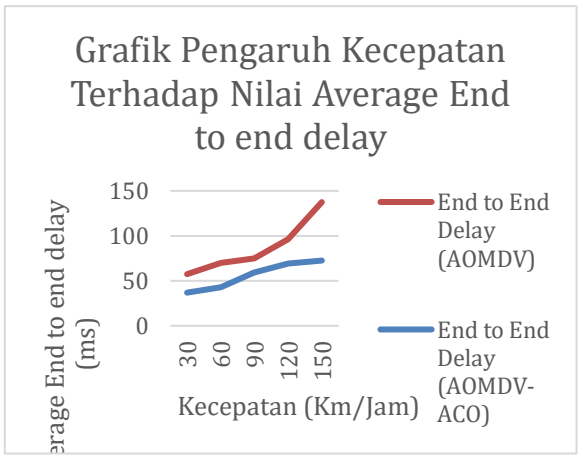
c. Grafik Hasil Ujicoba *Average End to end delay* 100 *Node*



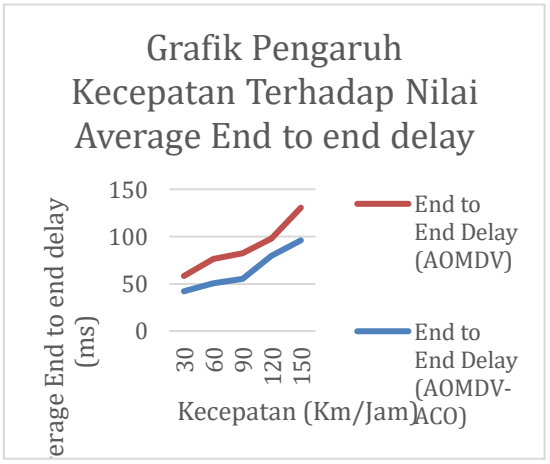
(1)



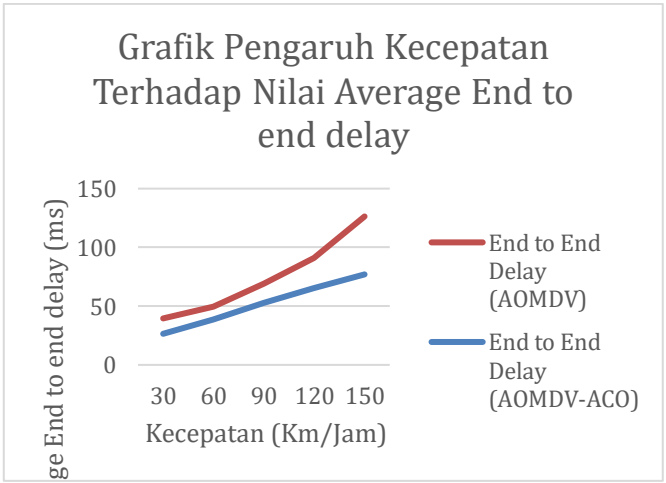
(2)



(3)



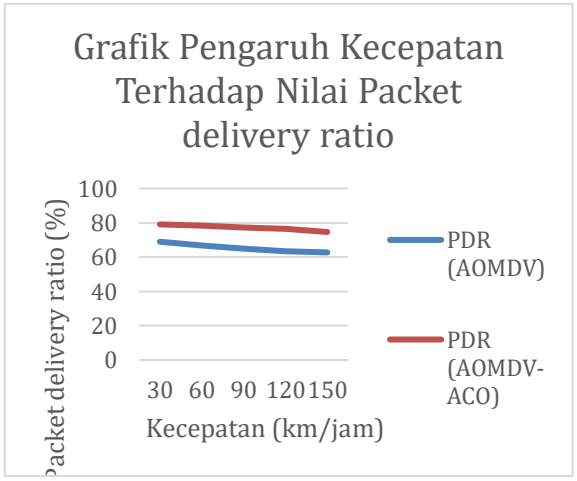
(4)



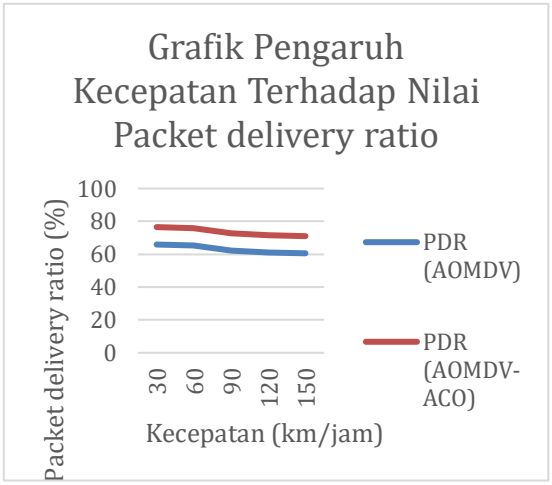
(5)

6.3 Grafik Hasil Ujicoba *Packet delivery ratio*

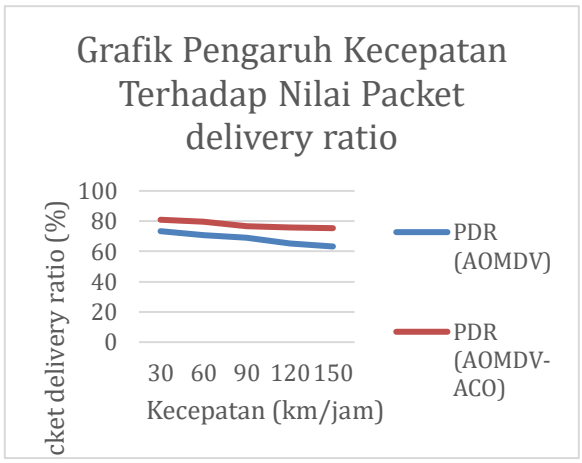
a. Grafik Hasil Ujicoba *Packet delivery ratio* 50 *Node*



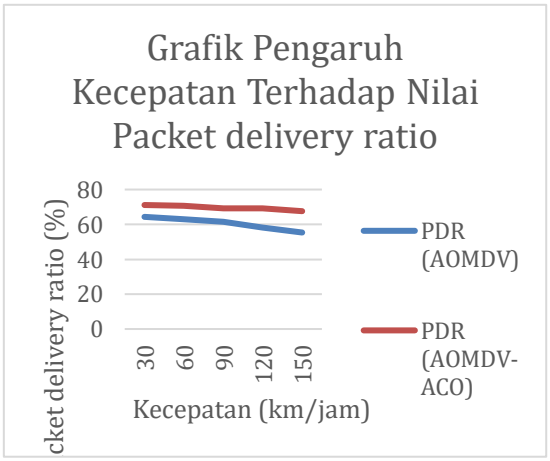
(1)



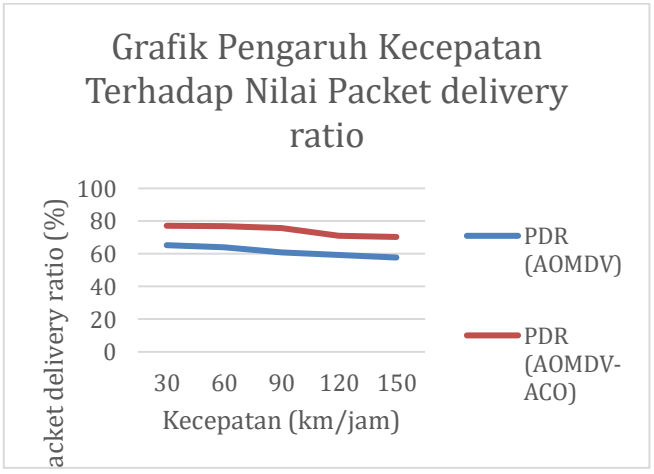
(2)



(3)

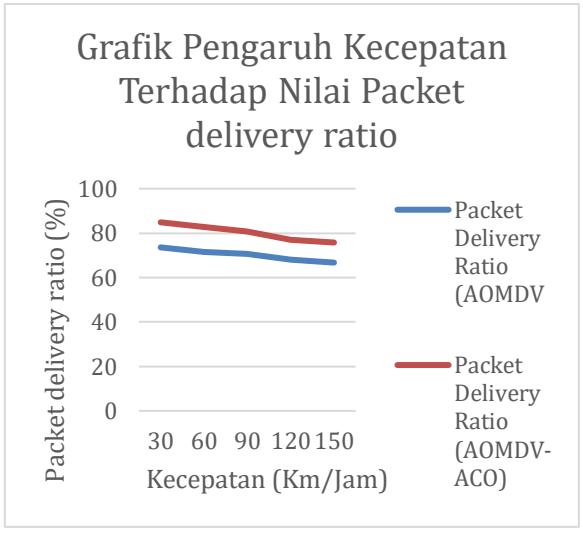


(4)

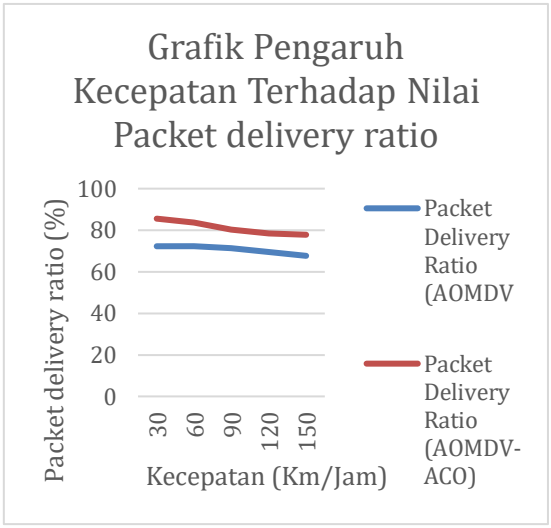


(5)

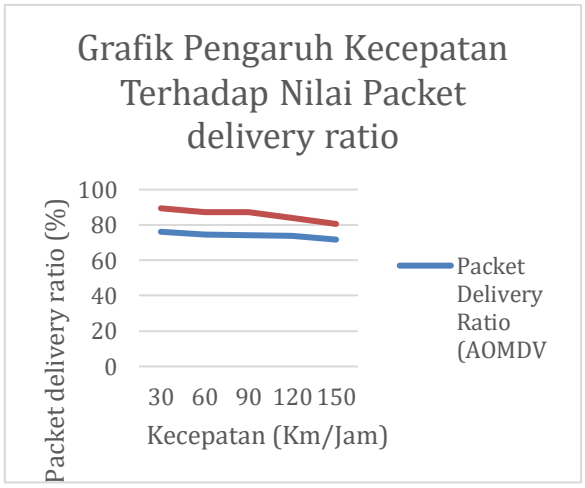
b. Grafik Hasil Ujicoba *Packet delivery ratio* 70 *Node*



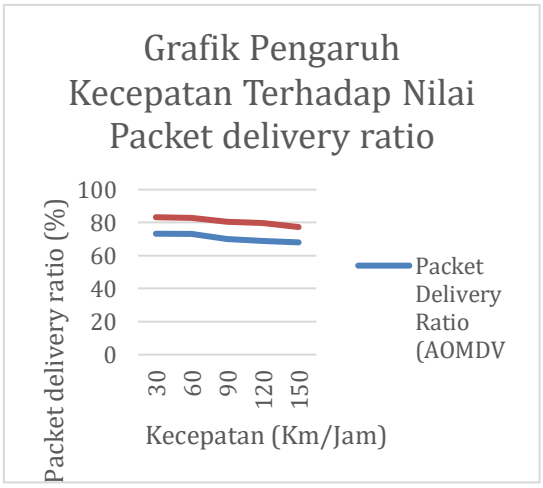
(1)



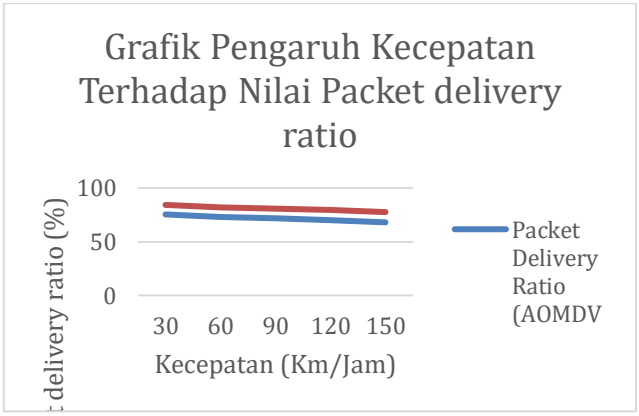
(2)



(3)

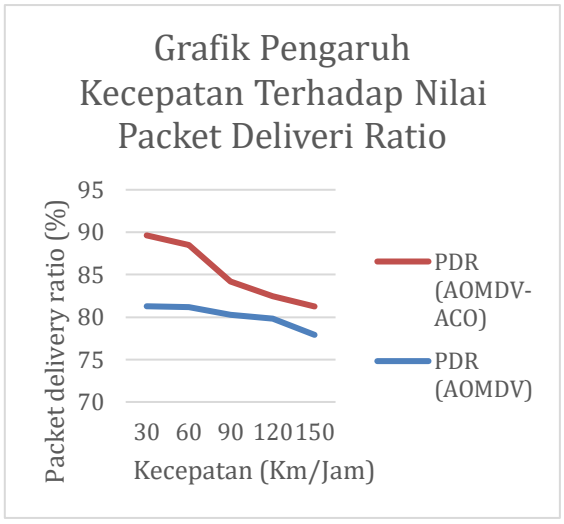


(4)

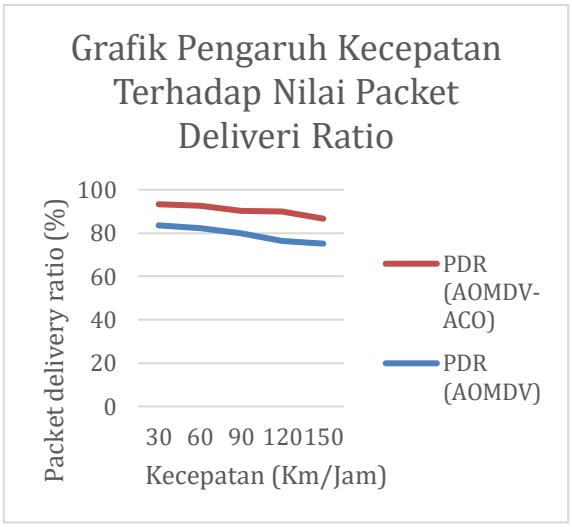


(5)

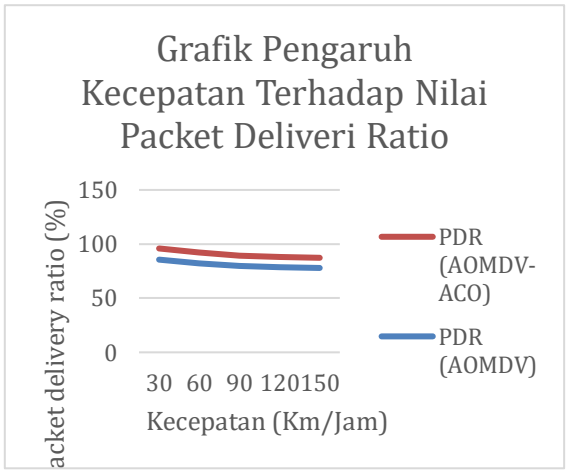
c. Grafik Hasil Ujicoba *Packet delivery ratio* 100 *Node*



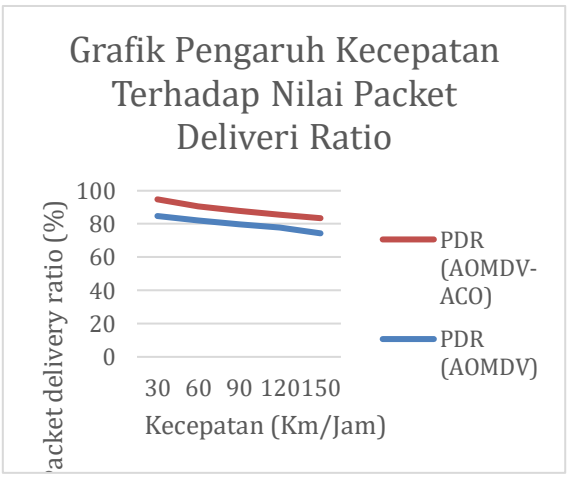
(1)



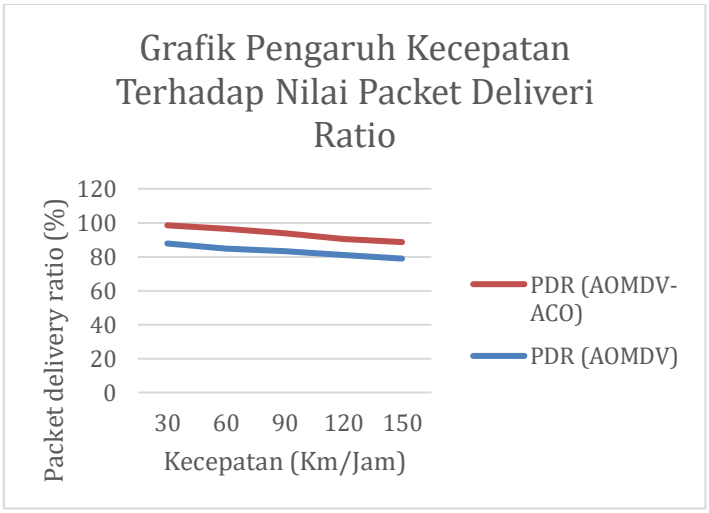
(2)



(3)



(4)



(5)