TUGAS AKHIR - KI141502

IMPLEMENTASI ELLIPTIC CURVE DIGITAL SIGNATURE ALGORITHM (ECDSA) UNTUK MENGATASI *BLACK HOLE* DAN *WORM HOLE ATTACK* PADA KOMUNIKASI V2V DI LINGKUNGAN VANETS

TITIES NOVANINDA OVARI
NRP 5113100083

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

Dosen Pembimbing II
Henning Titi Ciptaningtyas, S.Kom., M.Kom.

Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2017

TUGAS AKHIR – KI141502

IMPLEMENTASI ELLIPTIC CURVE DIGITAL SIGNATURE ALGORITHM (ECDSA) UNTUK MENGATASI *BLACK HOLE* DAN *WORM HOLE ATTACK* PADA KOMUNIKASI V2V DI LINGKUNGAN VANETS

TITIES NOVANINDA OVARI
NRP 511310003

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

Dosen Pembimbing II
Henning Titi Ciptaningtyas, S.Kom., M.Kom.

Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2017

(Halaman ini sengaja dikosongkan)

UNDERGRADUATE THESES – KI141502

ELLIPTIC CURVE DIGITAL SIGNATURE ALGORITHM (ECDSA) IMPLEMENTATION TO OVERCOME BLACK HOLE AND WORM HOLE ATTACK IN V2V COMMUNICATION OF VANETS

TITIES NOVANINDA OVARI
NRP 5113100083

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

Second Advisor
Henning Titi Ciptaningtyas, S.Kom., M.Kom.

Department of Informatics
Faculty of Information Technology
Sepuluh Nopember Institute of Technology
Surabaya 2017

(Halaman ini sengaja dikosongkan)

LEMBAR PENGESAHAN

IMPLEMENTASI ELLIPTIC CURVE DIGITAL SIGNATURE ALGORITHM UNTUK MENGATASI *BLACK HOLE* DAN *WORM HOLE ATTACK* PADA KOMUNIKASI V2V DI LINGKUNGAN VANETS

TUGAS AKHIR

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

Oleh:
TITIES NOVANINDA OVARI
NRP: 5113100083

Disetujui oleh Pembimbing Tugas Akhir:

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

2. Henning Titi Ciptaningtyas, S.Kom., M.Kom. .....
   (NIP. 198407082010122004) (Pembimbing 2)

SURABAYA
JUNI, 2017

[Halaman ini sengaja dikosongkan]

IMPLEMENTASI ELLIPTIC CURVE DIGITAL SIGNATURE ALGORITHM (ECDSA) UNTUK MENGATASI *BLACK HOLE* DAN *WORM HOLE ATTACK* PADA KOMUNIKASI V2V DI LINGKUNGAN VANETS

Nama Mahasiswa : TITIES NOVANINDA OVARI
NRP : 5113100083
Jurusan : Teknik Informatika FTIF-ITS
Dosen Pembimbing 1 : Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.
Dosen Pembimbing 2 : Henning Titi Ciptaningtyas, S.Kom., M.Kom.

Abstrak

Vehicular Ad-hoc Network (VANET) merupakan teknologi yang mendukung pertukaran data antar kendaraan di jalan raya. VANET menggunakan jaringan *wireless* berbasis *ad hoc* yang memungkinkan terjadi serangan, seperti *Black hole* dan *Worm hole attack*. *Black hole attack* adalah serangan yang dilakukan suatu *malicious node* dengan menjatuhkan paket data yang diterima, sedangkan *Worm hole attack* dilakukan dengan mengubah isi paket data. Kedua serangan ini dapat mempengaruhi performa *routing protocol* yang digunakan dalam komunikasi antar kendaraan.

Kedua serangan tersebut dapat diatasi jika tiap *node* dapat mendeteksi jika ada *malicious node* di sekitarnya, sehingga tiap *node* tersebut tidak meneruskan paket data ke *malicious node*. *Black hole attack* dapat dideteksi dengan melakukan pemantauan terhadap arus pertukaran data pada tiap *node*, sedangkan *Worm hole attack* dapat dideteksi dengan menggunakan *digital signature*. Tugas akhir ini mengimplementasikan Elliptic Curve Digital Signature Algorithm (ECDSA) sebagai algoritma *digital signature* yang dilampirkan pada tiap paket data.

Pada tugas akhir ini didapatkan *routing protocol* yang menerapkan pemantauan arus pertukaran data dan *digital signature* pada tiap paket data yang dikirimkan mempunyai performa yang lebih baik jika dihadapkan dengan *Black hole* dan *Worm hole attack* dibandingkan dengan *routing protocol* yang tidak menerapkannya, dibuktikan dengan peningkatan *packet delivery ratio* hingga mencapai 60.61% yang pada *routing protocol* tanpa modifikasi sebelumnya turun drastis.

Kata kunci: ECDSA, *Black hole attack*, *Worm hole attack*, VANET.

ELLIPTIC CURVE DIGITAL SIGNATURE ALGORITHM IMPLEMENTATION TO OVERCOME BLACK HOLE AND WORM HOLE ATTACK IN V2V COMMUNICATION OF VANETS

Student's Name	**: TITIES NOVANINDA OVARI**
Student's ID	**: 5113100083**
Department	**: Teknik Informatika FTIF-ITS**
First Advisor	**: Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.**
Second Advisor	**: Henning Titi Ciptaningtyas, S.Kom., M.Kom.**

Abstract

Vehicular Ad-hoc Network (VANET) is a technology enabling data exchange between vehicular on roads, in order to support Intelligent Transportation System. VANET use an ad-hoc base wireless network that can be attacked by several attacks, such as Black hole attack and Worm hole attack. Black hole attack is an attack conducted by dropping data packet that has been received, meanwhile Worm hole attack conducted by tampering the data packet. Both attacks can affect the performance of the routing protocol used in vehicular communication.

Both attacks can be overcome if each node can detect the malicious node around them, so they do not forward the data packet to malicious node. Black hole attack can be detected by recording the data packet exchange traffic in each node, meanwhile Worm hole attack can be detected by attaching digital signature on each data packets that will be sent. This thesis implements Elliptic Curve Digital Signature Algorithm (ECDSA) as the digital signature algorithm attached to each data packets that will be sent.

In this thesis, it is obtained that the routing protocol which applies data packet exchange traffic recording and the digital

signature on each packet that will be sent has better performance than the routing protocol which does not, proven by the increase of packet delivery ratio until 60.61%, which value has dropped dramatically when using the original routing protocol before.

Keywords: ECDSA, Black hole attack, Worm hole attack, VANET.

KATA PENGANTAR

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

Alhamdulillahirabbil'alamin, segala puji bagi Allah SWT, yang telah melimpahkan rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul "**IMPLEMENTASI ELLIPTIC CURVE DIGITAL SIGNATURE ALGORITHM (ECDSA) UNTUK MENGATASI *BLACK HOLE* DAN *WORM HOLE ATTACK* PADA KOMUNIKASI V2V DI LINGKUNGAN VANETS**". Tugas Akhir ini merupakan salah satu syarat dalam menempuh ujian sidang guna memperoleh gelar Sarjana Komputer. Dengan pengerjaan Tugas Akhir ini, penulis bisa belajar banyak untuk memperdalam dan meningkatkan apa yang telah didapatkan penulis selama menempuh perkuliahaan di Teknik Informatika ITS.

Selesainya Tugas Akhir ini tidak terlepas dari bantuan dan dukungan beberapa pihak, sehingga pada kesempatan ini penulis mengucapkan terima kasih kepada:

1. Allah SWT dan Nabi Muhammad SAW.
2. Keluarga penulis yang selalu memberikan dukungan doa, moral, dan material yang tak terhingga kepada penulis sehingga penulis dapat menyelesaikan Tugas Akhir ini.
3. Bapak Dr.Eng. Radityo Anggoro, S.Kom., M.Sc. dan Ibu Henning Titi Ciptaningtyas, S.Kom., M.Kom. selaku dosen pembimbing penulis yang telah membimbing dan memberikan motivasi, nasehat dan bimbingan dalam menyelesaikan tugas akhir ini.
4. Bapak Darlis Herumurti, S.Kom., M.Kom. selaku kepala jurusan Teknik Informatika ITS.

5. Seluruh dosen dan karyawan Teknik Informatika ITS yang telah memberikan ilmu dan pengalaman kepada penulis selama menjalani masa studi di ITS.
6. Ibu Eva Mursidah dan Ibu Sri Budiati yang selalu mempermudah penulis dalam peminjaman buku di RBTC.
7. Kakak Sari, Mama Risma, Nida, dan Mbak Yuyun yang telah menemani penulis selama berkuliah.
8. Muhammad yang telah banyak meluangkan waktu dan membantu penulis selama berkuliah.
9. Mas Randy, Mbak Nabila, dan Mas Raga yang telah meluangkan waktu berdiskusi dengan penulis selama mengerjakan Tugas Akhir.
10. Teman-teman seperjuangan RMK NCC/KBJ, yang telah menemani dan menyemangati penulis.
11. Teman-teman administrator NCC/KBJ, yang telah menemani dan menyemangati penulis selama penulis menjadi administrator, menjadi rumah kedua penulis selama penulis berkuliah.
12. Teman-teman angkatan 2013, yang sudah mendukung penulis selama perkuliahan.
13. Sahabat penulis yang tidak dapat disebutkan satu per satu yang selalu membantu, menghibur, menjadi tempat bertukar ilmu dan berjuang bersama-sama penulis.

Penulis menyadari bahwa Tugas Akhir ini masih memiliki banyak kekurangan sehingga dengan kerendahan hati penulis mengharapkan kritik dan saran dari pembaca untuk perbaikan ke depan.

Surabaya, Juni 2017

DAFTAR ISI

[Halaman ini sengaja dikosongkan]

DAFTAR GAMBAR

[Halaman ini sengaja dikosongkan]

DAFTAR TABEL

[Halaman ini sengaja dikosongkan]

BAB I
PENDAHULUAN

1.1 Latar Belakang

Vehicular Ad-hoc Network (VANET) merupakan salah satu teknologi yang saat ini banyak dikembangkan di berbagai negara. VANET menggunakan jaringan *wireless* berbasis *ad hoc* pada likungan yang dinamis, sehingga rentan terjadi serangan. Terdapat beberapa area riset dalam teknologi VANET, diantaranya adalah *routing protocol* dan sekuritas [1]. Fokus pada *routing protocol* adalah untuk memenuhi kebutuhan mengenai penentuan rute dari *source node* ke *destination node* dengan jangka waktu sesingkat mungkin dan akurasi rute yang optimal, sedangkan sekuritas berfokus pada ketercapaian dan autentikasi paket dalam pertukaran paket. Skenario serangan yang memungkinkan terjadi pada jaringan VANET adalah *Black hole* dan *Worm hole attack.*

Black hole attack merupakan serangan dimana *malicious node* selalu melakukan *drop* pada paket yang diterimanya dan membalas pada *source node* seolah paket sudah diterima oleh *destination node*. Sedangkan *Worm hole attack* memiliki banyak variasi serangan, salah satunya dengan mengubah isi dari paket yang diterimanya.

Dalam beberapa tahun terakhir terdapat penelitian terkait metode untuk mengatasi *Black hole* maupun *Worm hole attack* pada jaringan VANET, salah satunya adalah penerapan *evaluation system* dan *digital signature* pada *routing protocol* GPSR [2].

Pada penelitian kali ini akan dilakukan cara untuk mengatasi *Black hole* dan *Worm hole attack* dengan menggunakan *evaluation value* yang didapatkan melalui *forward-backward* evaluation dan implementasi *Elliptic Curve Digital Signature Algorithm* (ECDSA) sebagai algoritma digital signature pada *routing protocol* AODV.

Dikemudian hari dapat dihasilkan routing protocol yang cukup efektif dalam mengatasi serangan dalam jaringan VANET,

sehingga didapatkan *routing protocol* yang aman dan *reliable* untuk komunikasi antar kendaraan.

1.2 Rumusan Masalah

Tugas akhir ini mengangkat beberapa rumusan masalah sebagai berikut:

1. Bagaimana cara mendeteksi dan mengatasi *malicious node* yang melakukan *Black hole* dan *Worm hole attack*?
2. Bagaimana perbandingan performa *routing protocol* AODV ketika terdapat *malicious node* yang melakukan *Black hole* dan *Worm hole attack* sebelum dan sesudah ECDSA diterapkan pada skenario *grid*?
3. Bagaimana perbandingan performa *routing protocol* AODV ketika terdapat *malicious node* yang melakukan *Black hole* dan *Worm hole attack* sebelum dan sesudah ECDSA diterapkan pada skenario *real*?

1.3 Batasan Permasalahan

Permasalahan yang dibahas pada tugas akhir ini memiliki batasan sebagai berikut:

1. Simulator yang digunakan adalah simulator NS2.
2. *Routing protocol* yang digunakan adalah *routing protocol* AODV.
3. Algoritma *digital signature* yang digunakan adalah algoritma ECDSA.

1.4 Tujuan

Tujuan dari tugas akhir ini adalah sebagai berikut:

1. Dapat mendeteksi *malicious node* yang melakukan *Black hole* dan *Worm hole attack* pada lingkungan VANET dengan *routing protocol* AODV.

2. Dapat mengimplementasikan ECDSA untuk mengatasi *Black hole* dan *Worm hole attack* yang terjadi di lingkungan VANET dengan *routing protocol* AODV.
3. Menganalisa perbandingan performa *routing protocol* AODV ketika terdapat *malicious node* yang melakukan *Black hole* dan *Worm hole attack* sebelum dan sesudah ECDSA diterapkan.

1.5 Manfaat

Manfaat dari hasil pembuatan tugas akhir ini adalah sebagai berikut:

1. Memberikan manfaat di bidang VANET terutama pada area riset sekuritas dengan mendapatkan performa *routing protocol* yang relatif stabil meskipun terdapat *malicious node* yang melakukan *Black hole* dan *Worm hole attack* pada lingkungan VANET.
2. Menerapkan ilmu yang dipelajari selama kuliah di Teknik Informatika ITS agar dapat berguna bagi orang banyak.

1.6 Metodologi

Tahapan-tahapan yang dilakukan dalam pengerjaan tugas akhir ini adalah sebagi berikut:

1.6.1 Penyusunan Proposal Tugas Akhir

Tahap awal tugas akhir ini adalah menyusun proposal tugas akhir. Pada proposal, berisi tentang deskripsi pendahuluan dari tugas akhir yang akan dibuat. Proposal juga berisi tentang garis besar tugas akhir yang akan dikerjakan sehingga memberikan gambaran untuk dapat mengerjakan tugas akhir sesuai dengan *timeline* yang dibuat. Gagasan untuk mengatasi *Black Hole* dan *Worm Hole attack* pada komunikasi V2V di lingkungan VANETs.

1.6.2 Studi Literatur

Pada tahap ini dilakukan untuk mencari informasi dan studi literatur apa saja yang dapat dijadikan sebagai referensi untuk membantu pengerjaan tugas akhir ini. Tahap ini merupakan tahap untuk memahami semua metode yang akan dikerjakan, sehingga memberi gambaran selama pengerjaan tugas akhir. Informasi didapatkan dari buku dan literatur yang berhubungan dengan metode yang digunakan. Informasi yang dicari adalah *Black Hole* dan *Worm Hole attack*, dan ECDSA. Tugas akhir ini juga mengacu pada literatur jurnal dengan judul "*Secure and Efficient Protocol for Position-based Routing in VANET"* [2].

1.6.3 Implementasi Sistem

Implementasi merupakan tahap untuk mengimplementasikan metode-metode yang sudah diajukan pada proposal Tugas Akhir. Untuk membangun algoritma yang telah dirancang sebelumnya, implementasi dilakukan dengan menggunakan NS-2 sebagai *Network Simulator*, bahasa C/C++ sebagai bahasa pemrograman, dan OpenSSL sebagai *library* untuk mengimplementasikan ECDSA.

1.6.4 Pengujian dan Evaluasi

Pada tahap ini algoritma yang telah disusun diuji coba dengan melakukan *generate* skenario *grid* dan skenario *real* yang kemudian diberikan *malicious node* dalam jumlah tertentu dan melakukan serangan tertentu. Kemudian dari skenario tersebut akan didapatkan data uji *packet delivery ratio* (PDR), rata-rata *end to end delay* (E2E), dan *routing overhead* (RO). Dari data uji tersebut, dilakukan analisa untuk membandingkan performa *routing protocol* yang telah dimodifikasi dengan *routing protocol original*.

1.6.5 Penyusunan Buku

Pada tahap ini disusun buku sebagai dokumentasi dari pelaksanaan tugas akhir yang mencangkup seluruh konsep, teori, implementasi, serta hasil yang telah dikerjakan.

1.7 Sistematika Penulisan Laporan

Sistematika penulisan laporan tugas akhir adalah sebagai berikut:

1. Bab I. Pendahuluan
   Bab ini berisikan penjelasan mengenai latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, metodologi, dan sistematika penulisan dari pembuatan tugas akhir.
2. Bab II. Tinjauan Pustaka
   Bab ini berisi kajian teori dari metode dan algoritma yang digunakan dalam penyusunan tugas akhir ini. Secara garis besar, bab ini berisi tentang VANET, AODV, ECDSA, *Black Hole attack*, *Worm Hole attack*, OpenSSL, OpenStreetMap, SUMO, NS-2, dan AWK.
3. Bab III. Perancangan Perangkat Lunak
   Bab ini berisi pembahasan mengenai perancangan dari metode ECDSA untuk mengatasi *Black Hole* dan *Worm Hole attack* pada komunikasi V2V di lingkungan VANETs menggunakan NS-2 sebagai *simulator*.
4. Bab IV. Implementasi
   Bab ini menjelaskan implementasi ECDSA untuk mengatasi *Black Hole* dan *Worm Hole attack* pada komunikasi V2V di lingkungan VANETs menggunakan NS-2 sebagai *simulator*.
5. Bab V. Pengujian dan Evaluasi
   Bab ini berisikan hasil uji coba dari implementasi ECDSA untuk mengatasi *Black Hole* dan *Worm Hole attack* pada komunikasi V2V di lingkungan VANETs menggunakan NS-2 sebagai *simulator*. Pengujian dilakukan dengan skenario yang di*generate* oleh SUMO untuk mendapatkan data uji PDR, E2E,

dan RO yang nantinya akan dianalisa menggunakan skrip awk dan dilakukan perbandingan performa *routing protocol*.

6. Bab VI. Kesimpulan dan Saran
   Bab ini merupakan bab yang menyampaikan kesimpulan dari hasil uji coba yang dilakukan, masalah-masalah yang dialami pada proses pengerjaan tugas akhir, dan saran untuk pengembangan solusi ke depannya.
7. Daftar Pustaka
   Bab ini berisi daftar pustaka yang dijadikan literatur dalam tugas akhir.
8. Lampiran
   Dalam lampiran terdapat kode program secara keseluruhan.

BAB II
TINJAUAN PUSTAKA

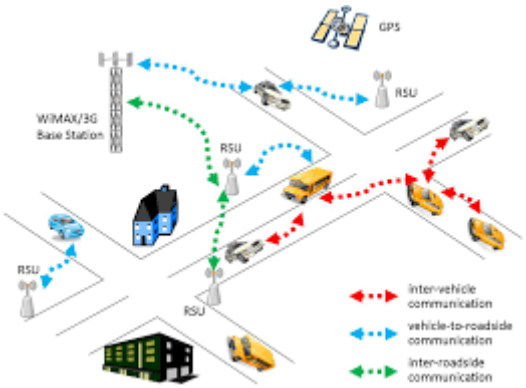
Bab ini berisi pembahasan mengenai teori-teori dasar yang digunakan dalam tugas akhir. Teori-teori tersebut diantaranya adalah *Elliptic Curve Digital Signature Algorithm*, dan beberapa teori lain yang mendukung pembuatan tugas akhir. Penjelesan ini bertujuan untuk memberikan gambaran secara umum terhadap program yang dibuat dan berguna sebagai penunjang dalam pengembangan riset yang berkaitan.

2.1 VANET (Vehicular Ad hoc Network)

VANET merupakan bentuk dari *wireless ad hoc network* untuk menyediakan komunikasi antar kendaraan dan *roadside equipment* terdekat. VANET muncul sebagai teknologi baru untuk mengintegrasikan kemampuan *wireless network* baru dengan kendaraan. Tujuan utama VANET adalah untuk menyediakan *ubiquitos connectivity* kepada *mobile user* ketika berada di jalan raya dan komunikasi antar kendaraan yang efisien menuju *Intelligent Transportation System* [3].

VANET merupakan pengembangan dari MANET (Mobile Ad hoc Network). Pada VANET maupun MANET, *node* yang bergerak bergantung pada *ad hoc routing protocol* untuk menentukan bagaimana mengirim pesan kepada tujuan. Perbedaan antara VANET dan MANET terletak pada kecepatan *node*. Pada VANET, *node* bergerak dengan kecepatan yang relatif tinggi. Hal tersebut merupakan sebuah tantangan dan memiliki peranan penting dalam menentukan desain jaringan seperti apakah yang akan dibuat.

Terdapat dua jenis *node* yang tergabung dalam VANET, yaitu RSU (*Road-side Unit*) dan OBU (*On-Board Unit*) [4]. Oleh karena itu terdapat dua tipe komunikasi dalam VANET, yaitu antara kendaraan dengan kendaraan, yang biasa disebut dengan komunikasi V2V dan antara kendaraan dengan RSU, yang biasa disebut dengan komunikasi V2I seperti pada Gambar 2.1.



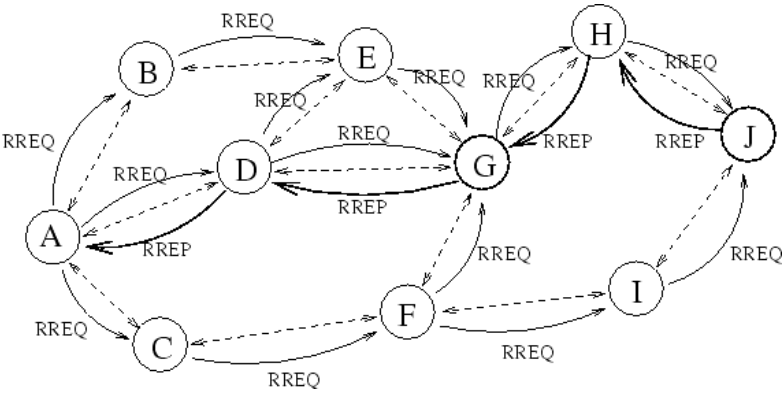
Gambar 2.1 Ilustrasi Komunikasi pada VANET [4]

Agar komunikasi antar *node* pada VANET dapat berjalan dengan baik, diperlukan sebuah *routing protocol* yang dapat beradaptasi dengan karakteristik VANET. Pada Tugas Akhir ini, penulis menggunakan *routing protocol* AODV yang dimodifikasi untuk menambahkan aspek sekuritas sehingga dapat mengatasi serangan yang memungkinkan terjadi pada VANET, yaitu *Black hole* dan *Worm hole attack*. Implementasi VANET dilakukan pada NS-2 dan dilakukan pengujian performa menggunakan skrip awk.

2.2 AODV (Ad hoc On-Demand Distance Vector)

AODV merupakan *routing protocol* yang umum digunakan di MANET dan telah diaplikasikan juga di VANET. Pada AODV, rute dibuat hanya ketika diminta oleh *node* asal (*source node*). Metode penemuan rute AODV berdasarkan pada *routing table* yang menyimpan rute terhadap beberapa *node*. *Source node* akan menyebarkan paket *route request* (RREQ) ke tiap *neighbor*-nya yang akan mengirimkan paket RREQ juga ke tiap *neighbor*-nya, begitu seterusnya hingga mencapai *node* tujuan (*destination node*).

Ketika paket RREQ mencapai *destination node* atau *node* yang mengetahui rute ke *destination node*, paket *route replay* (RREP) akan dikirimkan kembali ke *source node* melalui *reverse route*. AODV menggunakan sequence number untuk mendapatkan informasi rute yang paling update dan mencegah *routing loop* [5]. Ilustrasi pencarian rute *routing protocol* AODV dapat dilihat pada Gambar 2.2.



Gambar 2.2 Ilustrasi pencarian rute *routing protocol* AODV [6]

Sebagai contoh pencarian rute, ilustrasi pada Gambar 2.2 menggambarkan bagaimana *node* asal, yaitu *node* A mencari rute untuk menuju *node* destination yaitu *node* J. *Node* A mengirimkan RREQ kepada semua *neighbor*nya dan diteruskan oleh *node neighbor* hingga mencapai *node* J. Kemudian *node* J mengirimkan RREP melalui rute terdekat yang telah dibuat.

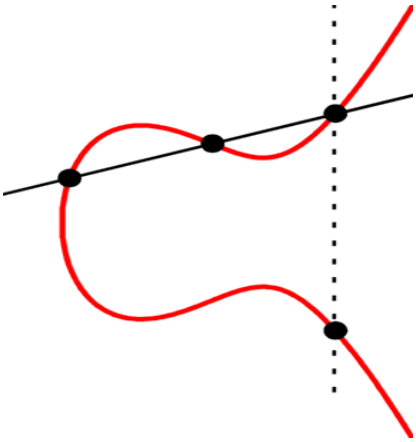
Routing protocol AODV termasuk ke dalam *routing protocol* yang bersifat reaktif, artinya *node* hanya akan mencari rute jika diminta (*on demand*). Sehingga *routing protocol* AODV dinilai memiliki performa yang relatif baik jika diterapkan pada lingkungan VANET yang memiliki karakteristik yaitu *node* bergerak dengan kecepatan tinggi.

Pada Tugas Akhir ini, penulis menggunakan *routing protocol* AODV sebagai algoritma *routing protocol* yang

digunakan untuk mengimplementasikan lingkungan VANET beserta skenario serangannya.

2.3 ECDSA (Elliptic Curve Digital Signature Algorithm)

Elliptic Curve Digital Signature Algorithm (ECDSA) merupakan varian dari *Digital Signature Algorithm* (DSA) yang beroperasi pada grup *elliptic curve*. Untuk mengirim pesan yang tertandatangani dari node asal ke node tujuan, keduanya sepakat pada parameter domain *elliptic curve*. Pengirim memiliki *key pair* yang terdiri dari *private key* d_A (sebuah *integer* kurang dari n yang dipilih secara acak, dimana n merupakan urutan dari kurva, sebuah parameter domain *elliptic curve*) dan *public key* $Q_A = d_A * G$ (G adalah *generator point*, sebuah parameter domain *elliptic curve*) [7]. Ilustrasi kurva dapat dilihat pada Gambar 2.3.



Gambar 2.3 Ilustrasi Kurva pada ECDSA [7]

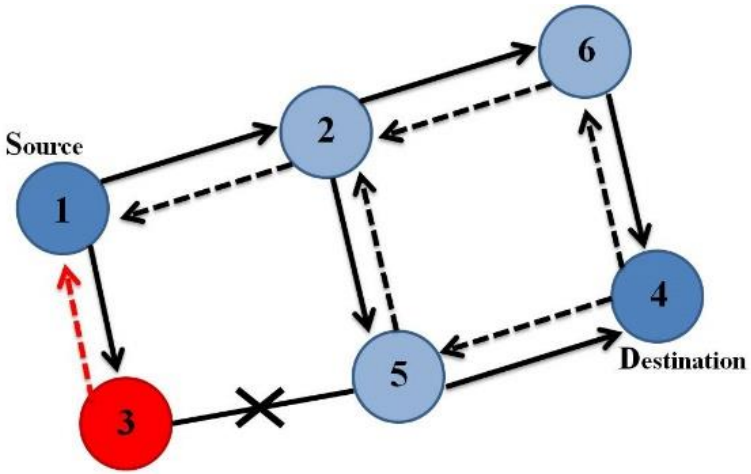
ECDSA merupakan salah satu algoritma *digital signature* yang umum digunakan selain RSA. Tingkat kemanan ECDSA dinilai cukup baik digunakan dalam jaringan. Di samping itu, ECDSA membutuhkan tempat yang lebih sedikit dibandingkan dengan RSA. Untuk tingkat keamanan yang sama, ECDSA

memiliki ukuran *signature* dan *public key* yang lebih kecil [8]. Sehingga jika akan ditambahkan pada *packet header*, tidak akan terlalu membebani dengan menambah ukuran paket yang akan dikirimkan.

Pada Tugas Akhir ini, penulis menggunakan ECDSA sebagai algoritma *digital signature* yang diterapkan untuk menjaga keaslian paket yang dikirimkan. Dengan menjaga keaslian paket, *Worm hole attack* yang melakukan penyerangan dengan memodifikasi paket dapat diatasi.

2.4 *Black hole Attack*

Black hole attack terjadi saat terdapat *malicious node* yang menunggu mendapatkan paket RREQ dari *neighbor*-nya. Namun ketika sudah mendapatkan paket RREQ, *malicious node* tersebut membalas seolah-olah dia memiliki rute terpendek menuju *destination node*. Sehingga *source node* akan meneruskan dengan mengirim paket data ke *destination node* melalui *malicious node* tersebut. Ketika mendapatkan paket data dari *source node*, *malicious node* tersebut tidak meneruskan paket data ke *destination node*, tetapi melakukan *drop* terhadap paket data yang diterima. Sehingga paket data tersebut tidak mencapai *destination source*-nya dan *malicious node* tidak memberi tahu *source node* jika paket datanya tidak mencapai *destination node* [2]. Ilustrasi *Black hole attack* dapat dilihat pada Gambar 2.4.



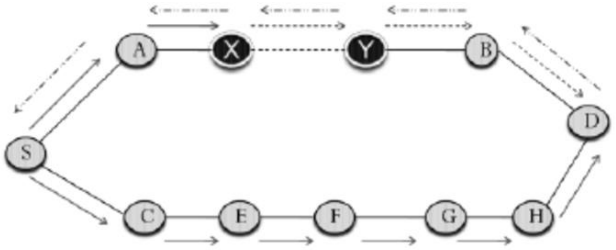
Gambar 2.4 Ilustrasi *Black Hole Attack* [9]

Pada Tugas Akhir ini, *Black hole attack* dilakukan sebagai skenario untuk menilai performa *routing protocol* dalam mengatasi serangan yang memungkinkan terjadi pada komunikasi V2V di lingkungan VANET.

2.5 *Worm hole Attack*

Worm hole attack terdiri dari 2 fase yang dilancarkan oleh satu atau lebih *malicious node*. Pada fase pertama, *malicious node* mencoba mengumpan *node neighbor*-nya agar mengirimkan paket melalui dirinya. Pada fase kedua, *malicious node* tersebut memanfaatkan paket data yang diterimanya sesuai keinginannya,

salah satunya dengan mengubah isi paket data [2]. Worm hole attack dapat diilustrasikan seperti pada Gambar 2.5.



Gambar 2.5 Ilustrasi *Worm Hole Attack* [10]

Pada Tugas Akhir ini, *Worm hole attack* dilakukan sebagai skenario untuk menilai performa *routing protocol* dalam mengatasi serangan yang memungkinkan terjadi pada komunikasi V2V di lingkungan VANET.

2.6 NS-2 (Network Simulator-2)

NS-2 merupakan sebuah *discrete event simulator* yang didesain untuk membantu penelitian pada bidang jaringan komputer. Pengembangan NS dimulai pada tahun 1989 sebagai sebuah varian dari REAL *network simulator*, pada tahun 1995 pengembangan NS didukung oleh DARPA melalui VINT project di LBL, Xerox PARC, UCB dan USC/ISI. NS kemudian memasuki versi dua pada tanggal 31 Juli 1995. Saat ini pengembangan NS didukung oleh DARPA melalui SAMAN dan NSF melalui CONSER beserta peneliti lainnya termasuk ACIRI [11].

NS-2 dinilai lebih stabil untuk melakukan simulasi lingkungan VANET jika dibandingkan dengan versi terbarunya, yaitu NS-3.

Pada Tugas Akhir ini, NS-2 digunakan untuk melakukan simulasi lingkungan VANET beserta dengan skenario serangan yang memungkinkan terjadi, yaitu *Black hole* dan *Worm hole attack*. *Trace file* yang dihasilkan oleh NS-2 juga digunakan sebagai informasi untuk mengukur performa *routing protocol*.

2.6.1 Instalasi

NS-2 membutuhkan beberapa *package* yang harus sudah terinstall sebelum memulai instalasi NS-2. Untuk menginstall *dependency* yang dibutuhkan dapat dilakukan dengan *command* yang ditunjukkan pada Gambar 2.6.

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

Gambar 2.6 Perintah untuk menginstall *dependency* NS-2

Setelah menginstall *dependency* yang dibutuhkan, ekstrak *package* NS-2 dan ubah baris kode ke-137 pada *file* ls.h di folder linkstate menjadi seperti pada Gambar 2.7.

```
void eraseAll(){this->erase(baseMap::begin(),
baseMap::end()); }
```

Gambar 2.7 Baris kode yang diubah pada *file* ls.h

Install NS-2 dengan menjalankan perintah ./install pada folder NS-2.

2.6.2 *Trace File*

Trace file merupakan *file* hasil simulasi yang dilakukan oleh NS-2 dan berisikan informasi detail pengiriman paket data. *Trace file* digunakan untuk menganalisis performa *routing protocol* yang

disimulasikan. Detail penjelasan trace file ditunjukkan pada Tabel 2.1.

Tabel 2.1 Tabel Detail Penjelasan *Trace File* AODV

Kolom ke-	Penjelasan	Isi
1	*Event*	s : *sent* r : *received* f : *forwarded* D: *dropped*
2	*Time*	Waktu terjadinya *event*
3	ID *Node*	_x_ : dari 0 hingga banyak *node* pada topologi
4	*Layer*	AGT : *application* RTR : *routing* LL : *link layer* IFQ : *packet queue* MAC : MAC PHY : *physical*
5	*Flag*	--- : Tidak ada
6	*Sequence Number*	Nomor paket
7	Tipe Paket	AODV : paket *routing* AODV cbr : berkas paket CBR (*Constant Bit Rate*) RTS : *Request To Send* yang dihasilkan MAC 802.11 CTS : *Clear To Send* yang dihasilkan MAC 802.11 ACK : MAC ACK ARP : Paket *link layer address resolution protocol*
8	Ukuran	Ukuran paket pada *layer* saat itu
9	Detail MAC	[a b c d] a : perkiraan waktu paket

		b : alamat penerima c : alamat asal d : IP *header*
10	*Flag*	-------- : Tidak ada
11	Detail IP *source*, *destination*, dan *nexthop*	[a:b c:d e f] a : IP *source node* b : *port source node* c : IP *destination node* (jika -1 berarti *broadcast*) d : *port destination node* e : IP *header* ttl f : IP *nexthop* (jika 0 berarti *node* 0 atau *broadcast*)

2.7 OpenSSL

OpenSSL adalah sebuah proyek open source yang menyediakan toolkit protokol Transport Layer Security (TLS) dan Secure Socket Layer (SSL). OpenSSL juga merupakan library yang umum digunakan untuk keperluan kriptografi. OpenSSL toolkit terlisensi di bawah lisensi Apache, yang bisa didapatkan secara gratis untuk tujuan komersial maupun non-komersial [12]. OpenSSL memiliki logo seperti pada Gambar 2.8.

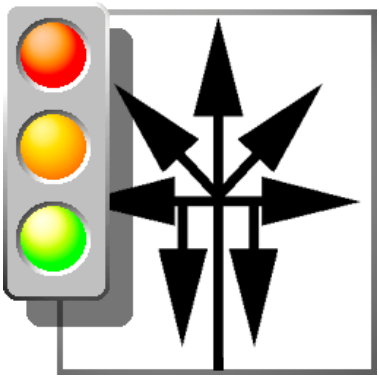
Pada tugas akhir ini, penulis menggunakan OpenSSL untuk mengimplementasikan ECDSA yang digunakan sebagai algoritma *digital signature* dalam mengatasi skenario serangan yang diberikan.

Gambar 2.8 Logo OpenSSL [12]

2.8 SUMO (Simulation of Urban Mobility)

Simulation of Urban Mobility atau disingkat SUMO merupakan simulasi lalu lintas jalan raya yang *open source*, *microscopic*, dan *multi-modal*. Sumo mensimulasikan bagaimana permintaan lalu lintas yang terdiri dari beberapa kendaraan berjalan pada jalan raya yang telah ditentukan. Simulasi SUMO dapat menunjukkan beberapa topik manajemen lalu lintas dalam skala besar. SUMO murni *microscopic*, yang artinya setiap kendaraan dimodelkan secara eksplisit, memiliki rutenya sendiri, dan bergerak secara individu dalam jaringan [13]. Logo SUMO ditunjukkan pada Gambar 2.9.

Pada Tugas Akhir ini, penulis menggunakan SUMO untuk men*generate* skenario VANET, peta area simulasi, dan pergerakan *node* sehingga menyerupai keadaan sebenarnya lalu lintas yang ingin disimulasikan.



Gambar 2.9 Logo SUMO [13]

2.9 OpenStreetMap

OpenStreetMap merupakan proyek kolaboratif untuk membuat sebuah peta dunia yang dapat dengan bebas disunting oleh siapapun. OpenStreetMap digunakan sebagai data peta pada

berbagai *website*, aplikasi *mobile*, dan *hardware device*. OpenStreetMap dibangun oleh komunitas pembuat peta yang berkontribusi dan mengelola data mengenai jalan raya, kafe, stasiun kereta api, dan sebagainya di seluruh dunia [14]. Logo OpenStreetMap ditunjukkan pada Gambar 2.10.

Gambar 2.10 Logo OpenStreetMap [14]

Pada Tugas Akhir ini, penulis menggunakan OpenStreetMap untuk mendapatkan peta sebenarnya guna melakukan simulasi dengan skenario *real*.

2.10 AWK

AWK merupakan sebuah bahasa pemrograman yag didesain untuk *text-processing* dan biasanya digunakan sebagai alat ekstrasi data dan pelaporan. AWK bersifat *data-driven* yang berisikan kumpulan perintah yang akan dijalankan pada data tekstural baik secara langsung pada *file* atau digunakan sebagai bagian dari *pipeline* [15].

Pada tugas akhir ini, penulis menggunakan AWK untuk memproses data yang diberikan oleh NS-2 agar mendapatkan analisis mengenai *packet delivery ratio*, *end to end delay*, dan *routing overhead*.

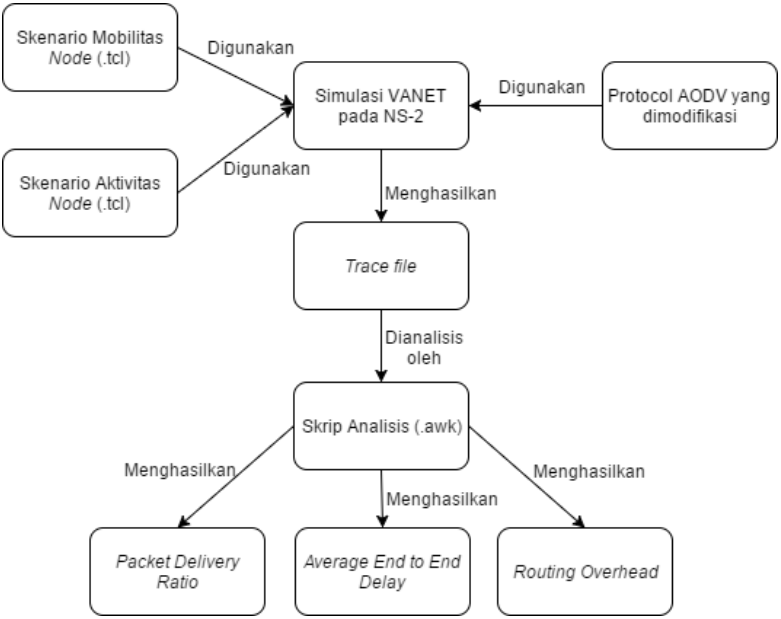
BAB III
PERANCANGAN

Bab ini membahas mengenai perancangan implemetasi sistem yang dibuat pada Tugas Akhir. Bagian yang akan dijelaskan pada bab ini berawal dari deskripsi umum, perancangan skenario, hingga alur dan implementasinya.

3.1 Deskripi Umum

Pada Tugas Akhir ini penulis akan mengimplementasikan *routing protocol* AODV yang dimodifikasi dengan menambahkan proses evaluasi untuk mendeteksi dan mengatasi *Black hole* dan *Worm hole attack*. Proses evaluasi pertama dilakukan dengan cara merekam keluar masuknya paket dalam suatu *node*. Proses evaluasi ini digunakan untuk dapat mendeteksi apakah *node* tetangga dari suatu *node* berpotensi melakukan *Black Hole attack*. Sedangkan proses evaluasi kedua dilakukan dengan mengimplementasikan ECDSA sebagai algoritma *digital signature* untuk memvalidasi keaslian paket yang diterima. Proses evaluasi ini digunakan untuk dapat mendeteksi apakah *node* tetangga dari suatu *node* berpotensi melakukan *Worm Hole attack*. Analisis dan perancangan modifikasi *routing protocol* AODV guna mendeteksi dan mengatasi *Black Hole* dan *Worm Hole attack* ini dijelaskan pada subbab 3.4.

Modifikasi *routing protocol* AODV ini akan disimulasikan menggunakan NS-2 dengan skenario pergerakan *node* yang dibuat menggunakan tools SUMO. Simulasi yang dilakukan akan menghasilkan *trace file*, yang kemudian dianalisis menggunakan AWK untuk mendapatkan *packet delivery ratio*, *average end to end delay*, dan *routing overhead.* Analisis tersebut dapat mengukur performa *routing protocol* AODV yang telah dimodifikasi dibandingkan dengan *routing protocol* AODV sebelum dimodifikasi. Diagram rancangan simulasi dapat dilihat pada Gambar 3.1.



Gambar 3.1 Diagram Rancangan Simulasi

Daftar istilah yang sering digunakan pada buku Tugas Akhir ini dapat dilihat pada Tabel 3.1.

Tabel 3.1 Daftar Istilah

No.	Istilah	Penjelasan
1.	AODV	*Ad Hoc On-Demand Distance Vector*
2.	ECDSA	*Elliptic Curve Digital Signature Algorithm*
3.	PDR	*Packet Delivery Ratio*
4.	E2E	*Average End to End Delay*
5.	RO	*Routing Overhead*
6.	RREQ	Paket *request* AODV
7.	RREP	Paket *reply* AODV

3.2 Perancangan Skenario Mobilitas

Perancangan skenario mobilitas dimulai dengan *generate* area simulasi, pergerakan *node*, implementasi pergerakan *node*, dan memilih *node* yang akan dijadikan *attacker*. Dalam Tugas Akhir ini, terdapat dua macam area simulasi yang akan digunakan. Diantaranya adalah peta *grid* dan peta *real*. Peta *grid* yang dimaksud adalah bentuk jalan berpetak-petak sebagai contoh jalan berpotongan yang sederhana. Peta *grid* digunakan sebagai simulasi awal VANET karena lebih seimbang dan stabil. Peta *grid* didapatkan dengan menentukan panjang dan jumlah petak area menggunakan SUMO. Sedangkan yang dimaksud peta *real* adalah peta asli yang digunakan sebagai area simulasi. Peta *real* didapatkan dengan mengambil area yang dimaksudkan sebagai area simulasi dari OpenStreetMap. Pada Tugas Akhir ini, peta *real* yang diambil penulis adalah salah satu area di Surabaya.

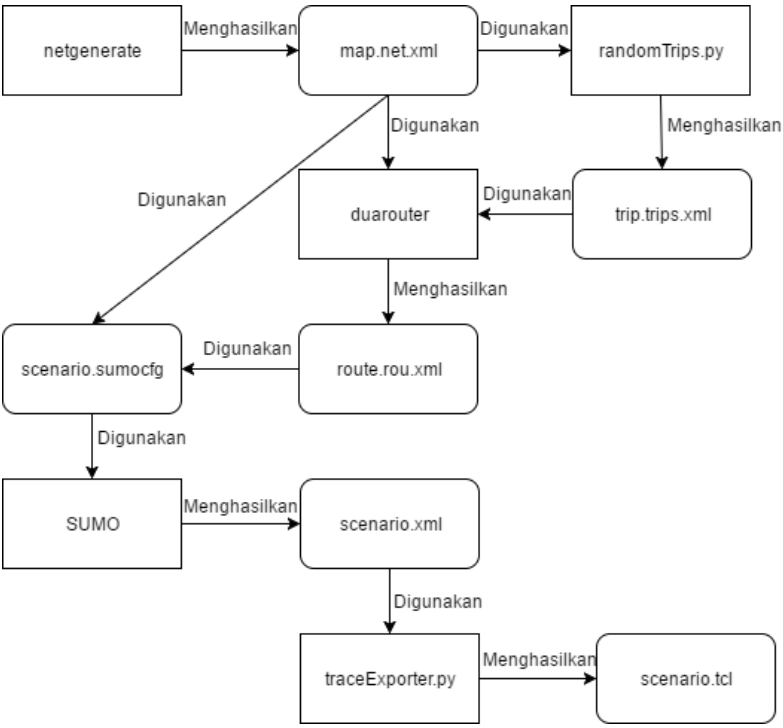
3.2.1 Perancangan Skenario Mobilitas *Grid*

Perancangan skenario mobilitas *grid* diawali dengan merancang luas area peta *grid* yang dibutuhkan. Luas area tersebut bisa didapatkan dengan cara menentukan terlebih dahulu jumlah titik persimpangan yang diinginkan pada peta *grid*, sehingga dari jumlah persimpangan tersebut dapat diketahui berapa banyak petak yang dibutuhkan. Dengan mengetahui jumlah petak yang dibutuhkan, dapat digunakan untuk menentukan panjang tiap petak sehingga mendapatkan luas area seperti yang diinginkan. Misalkan luas area yang dibutuhkan adalah 1000 m x 1000 m. Dengan 11 titik persimpangan, maka akan didapatkan 10 petak. Dengan begitu panjang tiap petak untuk mendapatkan luas area 1000 m x 1000 m adalah 100 m.

Peta *grid* yang telah ditentukan luas areanya tersebut kemudian dibuat dengan menggunakan tools SUMO yaitu netgenerate. Selain titik persimpangan dan panjang tiap petak *grid*,

dibutuhkan juga pengaturan kecepatan kendaraan dalam pembuatan peta *grid* menggunakan netgenerate ini. Peta *grid* yang dihasilkan oleh netgenerate akan memiliki ekstensi .net.xml. Peta *grid* ini kemudian digunakan untuk membuat pergerakan *node* dengan *tools* SUMO yaitu randomTrips dan duarouter.

Skenario mobilitas *grid* dihasilkan dengan menggabungkan *file* peta *grid* dan *file* pergerakan *node* yang telah di*generate*, yang akan menghasilkan *file* dengan ekstensi .xml. Selanjutnya, untuk dapat menerapkannya pada NS-2 file skenario mobilitas *grid* yang berekstensi .xml dikonversi ke dalam bentuk file .tcl. Konversi ini dilakukan menggunakan *tools* traceExporter. Alur pembuatan skenario *grid* dapat dilihat pada Gambar 3.2.

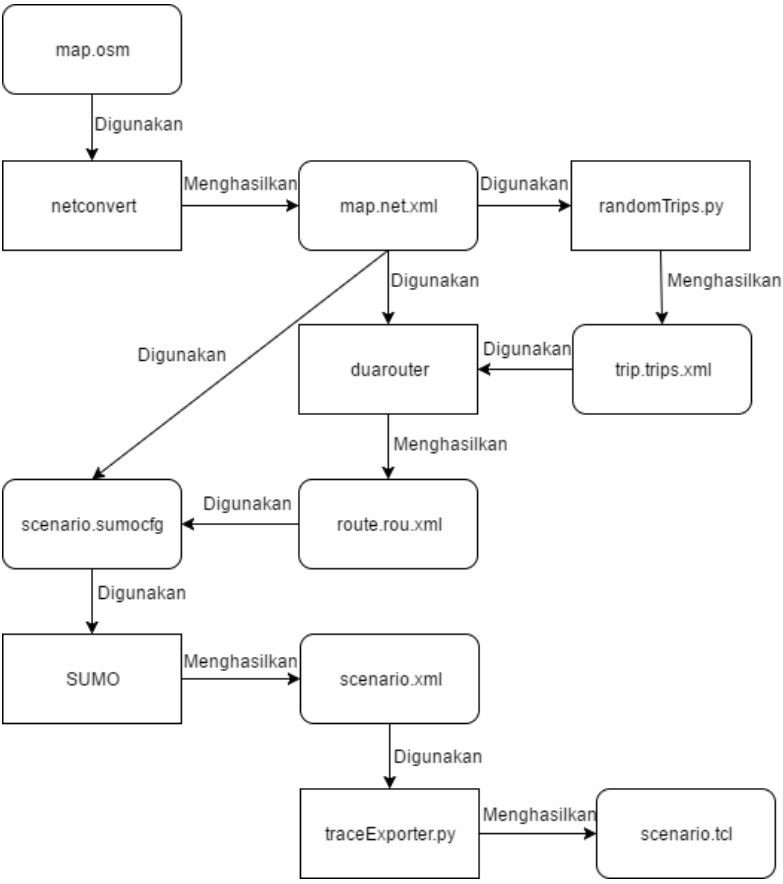


Gambar 3.2 Alur Pembuatan Skenario Mobilitas *Grid*

3.2.2 Perancangan Skenario Mobilitas *Real*

Perancangan skenario mobilitas *real* diawali dengan memilih area yang akan dijadikan simulasi. Pada Tugas Akhir ini, penulis menggunakan peta dari OpenStreetMap untuk mengambil area yang dijadikan model simulasi. Setelah memilih area, unduh dengan menggunakan fitur *export* dari OpenStreetMap. Peta hasil export dari OpenStreetMap ini memiliki ekstensi .osm.

Setelah mendapatkan peta area yang dijadikan simulasi, peta tersebut dikonversi ke dalam bentuk *file* dengan ekstensi .net.xml menggunakan tools SUMO yaitu netconvert. Tahap berikutnya memiliki tahapan yang sama seperti tahapan ketika merancang skenario mobilitas *grid*, yaitu membuat pergerakan *node* menggunakan randomTrips dan duarouter. Kemudian gabungkan *file* peta *real* yang sudah dikonversi ke dalam *file* dengan ekstensi .net.xml dan *file* pergerakan *node* yang sudah di*generate*. Hasil dari penggabungan tersebut merupakan *file* skenario yang berekstensi .xml. *File* yang dihasilkan tersebut dikonversi ke dalam bentuk *file* .tcl agar dapat diterapkan pada NS-2. Alur pembuatan skenario *real* dapat dilihat pada Gambar 3.3.



Gambar 3.3 Alur Pembuatan Skenario Mobilitas *Real*

3.3 Analisis dan Perancangan Modifikasi *Routing Protocol* AODV untuk Mengatasi *Black Hole* dan *Worm Hole Attack*

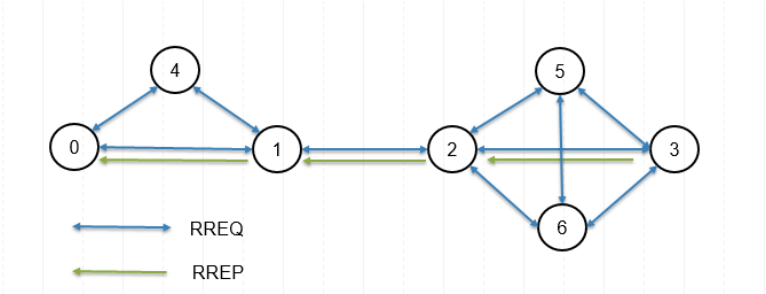
Routing Protocol AODV tidak memiliki benteng sekuritas sehingga rentan terjadi serangan, dalam hal ini adalah *Black hole* dan *Worm hole attack*. Modifikasi pada *Routing Protocol* AODV

dilakukan dengan adanya penambahan aspek sekuritas untuk dapat mendeteksi kemungkinan terdapat *malicious node* guna mengatasi *Black hole* dan *Worm hole attack*.

3.3.1 Analisis dan Perancangan Deteksi *Black Hole Attack*

Black Hole attack terjadi ketika suatu *node* selalu melakukan *drop* pada tiap paket yang diterima, yang seharusnya diteruskan ke *node* tetangganya. Untuk dapat mengatasi *Black Hole attack*, suatu *node* harus dapat mengetahui apakah *node* tetangganya adalah *malicious node*, sehingga *node* tersebut tidak dijadikan *nexthop node*.

Salah satu cara untuk dapat mendeteksi apakah *node* tetangganya berpotensi melakukan *Black Hole attack* adalah dengan melihat perbandingan antara jumlah paket yang dikirimkan ke *node* tetangga tersebut dan jumlah paket yang diterima dari *node* tetangga tersebut.



Gambar 3.4 Ilustrasi *Routing Protocol* AODV

Sebagai ilustrasi, pada Gambar 3.4 *node* 0 sebagai *source node* ingin mencari rute ke *node* 3 sebagai *destination node*, maka *node* 0 akan mem*broadcast* paket *request* (RREQ) pada semua tetangganya, *node* tetangganya akan mem*broadcast* ke semua tetangganya, begitu seterusnya hingga menemukan *node* 3. Jika *node* 5 adalah *malicious node* yang melakukan *Black Hole attack*,

maka *node* 2, *node* 6, dan *node* 3 harus dapat mendeteksi bahwa *node* 5 adalah *malicious node*.

Oleh karena itu, pada tiap *node* dilakukan evaluasi terhadap tiap *node* tetangganya. Sebagai contoh pada *node* 2, ia akan menghitung jumlah paket *request* yang ia kirimkan pada tiap *node* 5, *node* 6, dan *node* 3. Ia juga akan menghitung jumlah paket *request* yang ia terima dari tiap *node* 5, *node* 6, dan *node* 3. Kemudian *node* 2 membandingkan jumlah paket yang ia terima dengan jumlah paket yang ia kirimkan. Jika *node* 5 adalah *malicious node*, ia tidak akan mem*broadcast* paket *request* pada *node* tetangganya. Hal tersebut akan menyebabkan nilai perbandingan paket pada *node* 2 tidak seimbang. Dengan begitu *node* 2 dapat memperkirakan bahwa *node* 5 berpotensi melakukan *Black Hole attack*. Nilai perbandingan ini dinamakan nilai evaluasi *forward*.

3.3.2 Analisis dan Perancangan Deteksi *Worm Hole Attack*

Worm Hole attack terjadi ketika suatu *node* mengubah isi paket yang diterimanya, sehingga *node* tetangganya akan menerima informasi yang tidak valid. Sama halnya dengan *Black Hole attack*, untuk dapat mengatasi *Worm Hole attack*, suatu *node* harus dapat mengetahui apakah *node* tetangganya adalah *malicious node* agar *node* tersebut tidak dijadikan sebagai *nexthop node*.

Langkah awal yang dapat dilakukan agar dapat mengetahui apakah suatu *node* berpotensi melakukan *Worm Hole attack* adalah dengan melakukan verifikasi terhadap paket yang diterima dari *node* tersebut. Verifikasi ini dilakukan untuk dapat mengetahui apakah paket yang diterima masih terjamin keasliannya atau sudah diubah di tengah jalan. Untuk melakukan verifikasi keaslian paket tersebut dibutuhkan *digital signature*. ECDSA berperan sebagai algoritma untuk men*generate digital signature* dan memverifikasinya.

Agar dapat mengetahui apakah terdapat *malicious node*, pada tiap *node* dilakukan evaluasi terhadap tiap *node* tetangganya

dengan melihat nilai perbandingan antara jumlah paket yang dinyatakan lulus verifikasi dengan jumlah semua paket yang diterima dari *node* tetangga tersebut.

Dengan melihat ilustrasi pada Gambar 3.4, misalkan *node* 0 adalah *source node*, dan *node* 3 adalah *destination node*, maka sebelum mem*broadcast* paket *request*, node 0 akan men*generate digital signature* dari paket tersebut dengan ECDSA. Jika *node* 5 adalah *malicious node* yang melakukan *Worm Hole attack*, maka *node-node* tetangga dari *node* 5 harus dapat mengetahui bahwa *node* 5 berpotensi melakukan *Worm Hole attack* dengan selalu melakukan evaluasi terhadap *node* tetangganya.

Sebagai contoh, *node* 2 akan menghitung semua jumlah paket *request* yang ia terima dari *node* 5, *node* 6, dan *node* 3, juga melakukan verifikasi keaslian paket tersebut menggunakan ECDSA, kemudian ia akan menghitung jumlah paket *request* yang lulus verifikasi. Setelah itu dilakukan perbandingan antara jumlah paket yang lulus verifikasi dengan semua jumlah paket yang diterima. Jika *node* 5 melakukan *Worm Hole attack*, maka paket *request* yang diterima dari *node* 5 tidak lulus verifikasi yang akan menyebabkan nilai perbandingan paket tidak seimbang, sehingga dapat dikatakan bahwa *node* 5 berpotensi melakukan *Worm Hole attack*. Nilai perbandingan ini dinamakan dengan nilai evaluasi *backward*.

3.3.3 Perancangan Pemilihan *Nexthop Node*

Setelah dapat mendeteksi jika suatu *node* berpotensi melakukan *Black Hole* atau *Worm Hole attack* dengan melihat nilai evaluasi *forward* dan *backward*, langkah selanjutnya adalah dengan menentukan apakah suatu *node* layak dijadikan *nexthop node*.

Oleh karena itu, ditentukan suatu batas nilai yang dinamakan *threshold* untuk menyaring *node* mana yang layak dijadikan *nexthop*. Nilai yang akan dibandingkan dengan *threshold* tersebut adalah nilai *evaluasi total*, yang didapat dari penjumlahan

antara nilai evaluasi *forward* yang telah dikalikan dengan bilangan random dan nilai evaluasi *backward* yang telah dikalikan dengan bilangan random. Jika nilai evaluasi total dari *node* tersebut lebih besar dari nilai *threshold*, maka *node* tersebut dapat dikatakan layak dijadikan sebagai *nexthop node*, yang selanjutnya pemilihan *nexthop node* dilakukan sesuai dengan algoritma *routing protocol* AODV. Namun jika nilai evaluasi total dari *node* tersebut lebih kecil dari nilai *threshold*, maka *node* tersebut dideteksi sebagai *malicious node* dan secara otomatis dieliminasi dari pemilihan *nexthop node*, sehingga *malicious node* tersebut tidak akan menjadi *nexthop node*.

Ilustrasi digunakan skenario yang sama pada subbab 3.3.1 dan subbab 3.3.2 dimana *node* 0 sebagai *source node*, *node* 3 sebagai *destination node*, dan *node* 5 sebagai *malicious node*. Sebagai contoh, pada *node* 2 dilakukan evaluasi terhadap *node* 5, *node* 6, dan *node* 3 sehingga tiap *node* 5, *node* 6, dan *node* 3 tersebut memiliki nilai evaluasi total masing-masing. Kemudian masing-masing nilai evaluasi total akan dibandingkan dengan nilai *threshold*, dimana nilai evaluasi total dari *node* 5 akan menghasilkan nilai yang lebih kecil dari *threshold* karena ia melakukan *Black Hole attack* atau *Worm Hole attack*. Dengan demikian, *node* 5 dideteksi sebagai *malicious node* dan dieliminasi dari pemilihan *nexthop*. Sedangkan *node* 6 dan *node* 3 yang memiliki nilai evaluasi total lebih dari nilai *threshold* mempunyai kemungkinan untuk dijadikan *nexthop node*. Selanjutnya apakah *node* 6 atau *node* 3 yang dijadikan *nexthop node* dipilih menggunakan algoritma dari *routing protocol* AODV itu sendiri.

Pseudocode modifikasi *routing protocol* AODV ini, dari deteksi *malicious node* yang melakukan *Black Hole attack* atau *Worm Hole attack* hingga dilakukan pemilihan *nexthop node* dapat dilihat pada Gambar 3.5.

```
Part I
if source IP of packet != IP of this node
then
   counter array of packet received from node
n ++
   if verify of the signature successful then
      counter array of verified packet from
node n ++
else
   drop packet and return
end if

Part II
if neighbor IP != destination IP of packet
   counter array of packet sent to node n ++
   if eval value of node n > threshold
      record node n as nexthop
   forward packet
else
   forward packet
end if

Part III
for i = 1 to the number of neighbors
   forward eval [i] = counter array of packet
received from node i / counter array of
packet send to node i
   backward eval [i]= counter array of
verified packet from node i / counter array
of packet received from node i
   eval value [i] = random number * forward
eval [i] + random number * backward eval [i]
```

Gambar 3.5 Pseudocode Modifikasi Routing Protocol AODV

3.4 Perancangan Simulasi pada NS-2

Simulasi VANET pada NS-2 dilakukan dengan menggabungkan *file* skenario yang telah dibuat menggunakan SUMO dan *file* skrip tcl yang berisikan konfigurasi lingkungan simulasi. Konfigurasi lingkungan simulasi VANET pada NS-2 dapat dilihat pada Tabel 3.2.

Tabel 3.2 Parameter Lingkungan Simulasi dengan Skenario

No.	**Parameter**	**Spesifikasi**
1.	*Network simulator*	NS-2, 2.35
2.	*Routing protocol*	AODV
3.	Waktu simulasi	200 detik
4.	Area simulasi	1700 m x 1700 m
5.	Banyak kendaraan	30, 40, 50
6.	Agen pengirim	*Constant Bit Rate* (CBR)
7.	*Protocol* MAC	IEEE 802.11
8.	Propagasi sinyal	*Two-ray ground*
9.	*Source/Destination*	Dinamis
10.	Tipe Kanal	*Wireless channel*

Selain konfigurasi lingkungan simulasi VANET, pada skrip tcl tersebut juga ditambahkan perintah untuk menugaskan *node* yang akan menjadi *attacker* untuk *Black hole attack* maupun *Worm hole attack*.

Agar *node attacker* dapat menjalankan tugasnya melakukan *Black hole attack* atau *Worm hole attack*, ditambahkan beberapa kode untuk melakukan *Black hole attack* dan *Worm hole attack* pada kode routing protocol AODV di NS-2.

Kode yang ditambahkan diantaranya adalah deklarasi variabel pada *file* aodv.h. Kemudian pada *file* aodv.cc ditambahkan kode untuk mengenali *node attacker* dan alur penyerangan. Ketika simulasi NS-2 dijalankan *routing protocol* AODV akan mengenali apakah suatu *node* adalah *Black hole attacker* atau *Worm hole attacker*. Jika *node* tersebut adalah *Black hole attacker*, maka *node*

tersebut akan melakukan *drop* pada setiap paket yang diterimanya, sedangkan jika *node* tersebut adalah *Worm hole attacker*, maka *node* tersebut akan mengubah *node* tujuan yang ada pada paket *header*.

3.5 Perancangan Metrik Analisis

Berikut ini merupakan parameter-parameter yang akan dianalisis pada Tugas Akhir ini untuk dapat membandingkan performa dari *routing protocol* AODV yang telah dilakukan modifikasi dan *routing protocol* AODV yang asli:

3.5.1 *Packet Delivery Ratio* (PDR)

Packet Delivery Ratio merupakan perbandingan antara jumlah paket yang diterima dengan jumlah paket yang dikirimkan. *Packet Delivery Ratio* dihitung dengan persamaan 3.1.

$$PDR = \frac{packet\ received}{packet\ sent} \tag{3.1}$$

Packet Delivery Ratio dapat menunjukkan keberhasilan paket yang dikirimkan. Semakin tinggi *Packet Delivery Ratio*, artinya semakin berhasil pengiriman paket yang dilakukan.

3.5.2 Rata-rata *End to End Delay* (E2E)

Rata-rata *End to End Delay* merupakan rata-rata dari *delay* atau waktu yang dibutuhkan tiap paket untuk sampai ke *node* tujuan dalam satuan detik. *Delay* tiap paket didapatkan dari rentang waktu antara *node* asal mengirimkan paket (`CBRSentTime`) dan *node* tujuan menerima paket (`CBRRecvTime`). Dari *delay* tiap paket tersebut semua dijumlahkan dan dibagi dengan jumlah paket yang berhasil diterima (`recvnum`), maka akan didapatkan rata-rata *end to end delay*, yang dapat dihitung dengan persamaan 3.2.

$$E2E = \frac{\sum_{m=1}^{recvnum} CBRRecvTime \ - CBRSentTime}{recvnum} \quad (3.2)$$

3.5.3 *Routing Overhead* (RO)

Routing Overhead adalah jumlah paket kontrol *routing* yang ditransmisikan per data paket ke *node* tujuan selama simulasi terjadi. *Routing Overhead* didapatkan dengan menjumlahkan semua paket kontrol *routing* yang ditransmisikan, baik itu paket *route request* (RREQ), *route reply* (RREP), maupun *route error* (RERR). Perhitungan *Routing Overhead* dapat dilihat dengan persamaan 3.3.

$$RO = \sum_{m=1}^{sentnum} packet\ sent \quad (3.3)$$

BAB IV
IMPLEMENTASI

Bab ini membahas mengenai implementasi dari perancangan sistem yang telah dijabarkan pada bab sebelumnya.

4.1 Implementasi Skenario Mobilitas

Implementasi skenario mobilitas VANET dibagi menjadi dua, yaitu skenario *grid* yang menggunakan peta jalan berpetak dan skenario *real* yang menggunakan peta hasil pengambilan suatu area di kota Surabaya.

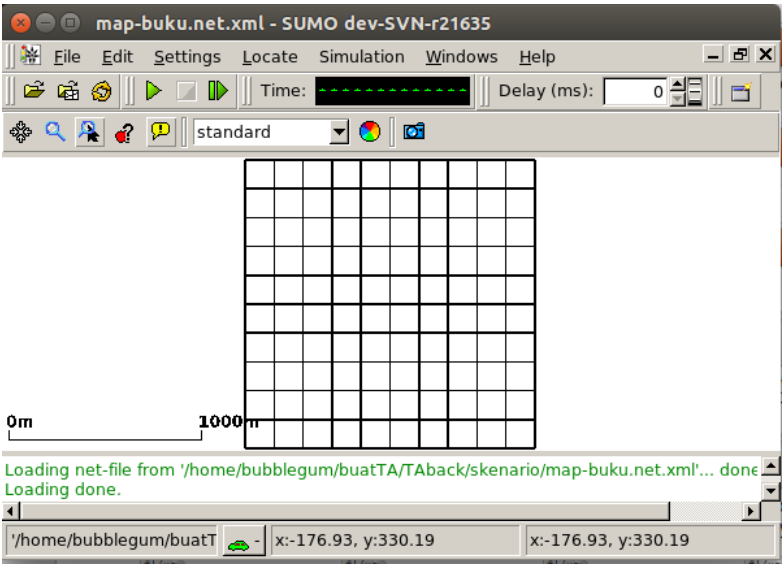
4.1.1 Skenario *Grid*

Dalam mengimplementasikan skenario *grid*, SUMO menyediakan *tools* untuk membuat peta *grid* yaitu netgenerate. Pada Tugas Akhir ini, penulis membuat peta *grid* dengan luas 1500 m x 1500 m yang terdiri dari titik persimpangan antara jalan vertikal dan jalan horizontal sebanyak 11 titik x 11 titik. Dengan jumlah titik persimpangan sebanyak 11 titik tersebut, maka terbentuk 10 buah petak. Sehingga untuk mencapai luas area sebesar 1500 m x 1500 m dibutuhkan luas per petak sebesar 150 m x 150 m. Berikut perintah netgenerate untuk membuat peta tersebut dengan kecepatan *default* kendaraan sebesar 20 m/s dapat dilihat pada Gambar 4.1.

```
netgenerate --grid --grid.number=11 --
grid.length=150 --default.speed=20 --
tls.guess=1 --output-file=map.net.xml
```

Gambar 4.1 Perintah netgenerate

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



Gambar 4.2 Hasil *Generate* Peta *Grid*

Setelah peta terbentuk, maka dilakukan pembuatan *node* dan pergerakan *node* dengan menentukan titik awal dan titik akhir setiap *node* secara random menggunakan *tools* randomTrips. Perintah penggunaan *tools* randomTrips untuk membuat *node* sebanyak n *node* dengan pergerakannya dapat dilihat pada Gambar 4.3.

```
python sumo-0.27.1/tools/randomTrips.py -n
map.net.xml -e 30 -l --trip-
attributes=”departLane=\”best”\
departSpeed=\”max\” departPos=\”random_free\””
-o trip.trips.xml
```

Gambar 4.3 Perintah randomTrips

Selanjutnya dibuatkan rute yang digunakan kendaraan untuk mencapai tujuan dari *file* hasil sebelumnya menggunakan *tools* duarouter. Secara *default* algoritma yang digunakan untuk

membuat rute ini adalah algoritma dijkstra. Perintah penggunaan *tools* duarouter dapat dilihat pada Gambar 4.4.

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

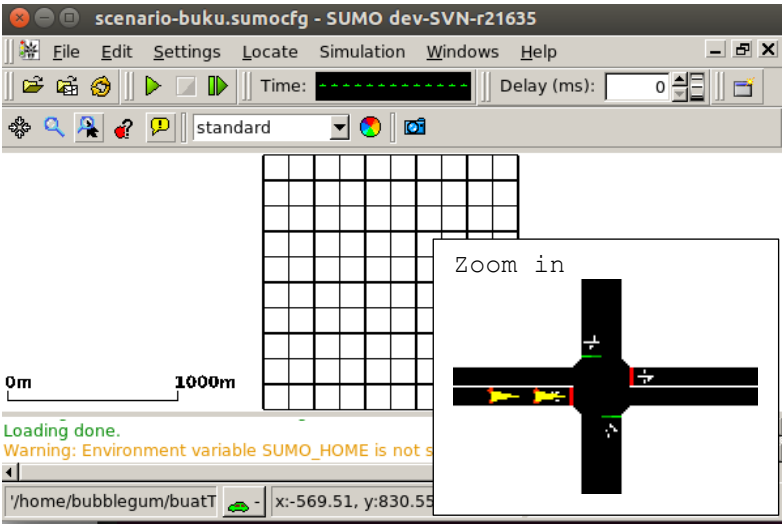
Gambar 4.4 Perintah duarouter

Untuk menjadikan peta dan pergerakan *node* yang telah di*generate* menjadi sebuah skenario dalam bentuk *file* berekstensi .xml, dibutuhkan sebuah *file* skrip dengan ekstensi .sumocfg guna menggabungkan *file* peta dan rute pergerakan *node*. Isi dari *file* skrip tersebut dapat dilihat pada Gambar 4.5.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:noNamespaceSchemaLocation="http://sumo.dlr
.de/xsd/sumoConfiguration.xsd">
   <input>
      <net-file value="map.net.xml"/>
      <route-files value="routes.rou.xml"/>
   </input>
   <time>
      <begin value="0"/>
      <end value="200"/>
   </time>
</configuration>
```

Gambar 4.5 File Skrip sumocfg

File .sumocfg disimpan dalam direktori yang sama dengan *file* peta dan *file* rute pergerakan *node*. Untuk percobaan sebelum dikonversi, *file* .sumocfg dapat dibuka dengan menggunakan *tools* sumo-gui. Cuplikan pergerakan kendaraan dapat dilihat pada Gambar 4.6.



Gambar 4.6 Simulasi Pergerakan Kendaraan

Kemudian buat *file* skenario dalam bentuk *file* .xml dari sebuah *file* skrip berekstensi .sumocfg menggunakan *tools* SUMO. Perintah untuk menggunakan *tools* SUMO dapat dilihat pada Gambar 4.7.

```
sumo -c scenario.sumocfg --fcd-output
scenario.xml
```

Gambar 4.7 Perintah SUMO

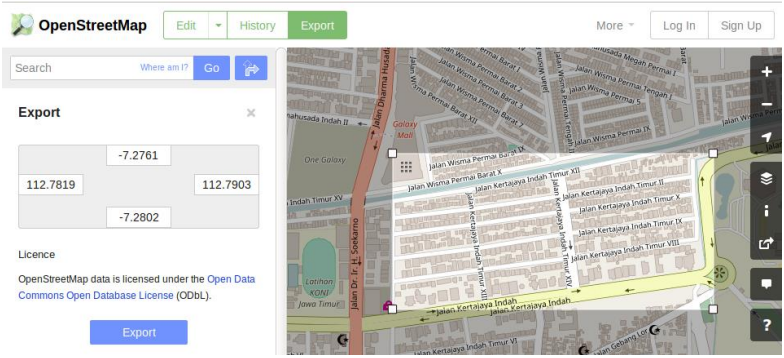
File skenario berekstensi .xml selanjutnya dikonversi ke dalam bentuk *file* berekstensi .tcl agar dapat disimulasikan menggunakan NS-2. *Tools* yang digunakan untuk melakukan konversi ini adalah *traceExporter*. Perintah untuk menggunakan *traceExporter* dapat dilihat pada Gambar 4.8.

```
python sumo-0.27.1/tools/traceExporter.py --
fcd-input=scenario.xml --ns2-mobility-
output=scenario.tcl
```

Gambar 4.8 Perintah traceExporter

4.1.2 Skenario *Real*

Dalam mengimplementasikan skenario *real*, langkah pertama adalah dengan menentukan area yang akan dijadikan area simulasi. Pada Tugas Akhir ini penulis mengambil area jalan sekitar Kertajaya Surabaya. Setelah menentukan area simulasi, ekspor data peta tersebut dari OpenStreetMap seperti yang ditunjukkan pada Gambar 4.9.



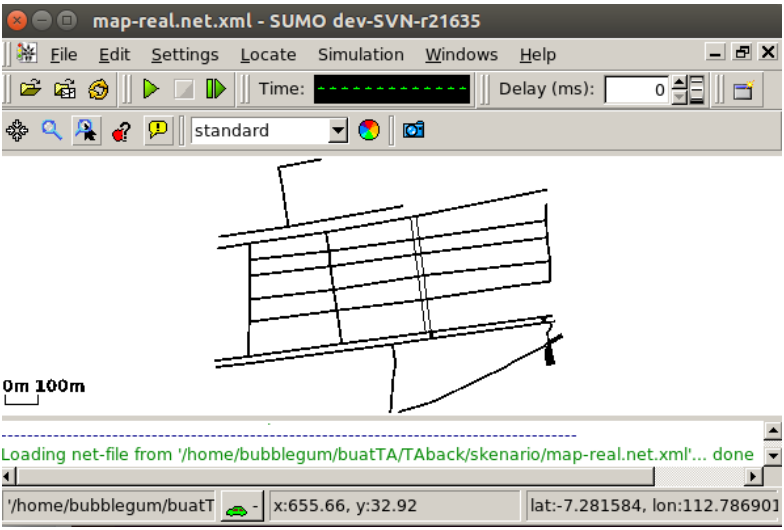
Gambar 4.9 Ekspor peta dari OpenStreetMap

File hasil ekspor dari OpenStreetMap tersebut adalah *file* peta dengan ekstensi .osm. Kemudian konversi *file* .osm tersebut menjadi peta dalam bentuk *file* berekstensi .xml menggunakan *tools* netconvert. Perintah untuk menggunakan netconvert dapat dilihat pada Gambar 4.10.

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

Gambar 4.10 Perintah netconvert

Hasil konversi peta dari *file* berekstensi .osm menjadi *file* berekstensi .xml dapat dilihat menggunakan *tools* sumo-gui seperti yang ditunjukkan pada Gambar 4.11.



Gambar 4.11 Hasil Konversi Peta *Real*

Langkah selanjutnya sama dengan ketika membuat skenario mobilitas *grid*, yaitu membuat *node*, asal, dan tujuan *node* menggunakan *tools* randomTrips. Lalu membuat rute *node* untuk sampai ke tujuan menggunakan *tools* duarouter. Kemudian membuat *file* skenario berekstensi .xml menggunakan *tools* SUMO dengan bantuan *file* skrip berekstensi .sumocfg. Selanjutnya konversikan *file* skenario berekstensi .xml tersebut menjadi *file* skenario berekstensi .tcl untuk dapat disimulasikan pada NS-2 menggunakan *tools* traceExporter. Perintah untuk menggunakan *tools* tersebut sama dengan ketika membuat skenario *grid* di atas.

4.2 Implementasi Modifikasi pada *Routing Protocol* AODV untuk Mengatasi *Black Hole* dan *Worm Hole Attack*

Routing protocol AODV merupakan salah satu *routing protocol* yang umum digunakan dalam simulasi VANET. Namun, *routing protocol* ini belum dilengkapi dengan aspek sekuritas sehingga memungkinkan penurunan performa yang cukup drastis ketika terjadi serangan, dalam hal ini adalah *Black hole* dan *Worm hole attack*.

Pada Tugas Akhir ini dilakukan pada *routing protocol* AODV untuk menambahkan aspek sekuritas agar dapat mendeteksi adanya *malicious node* guna mengatasi *Black Hole* dan *Worm Hole attack*. Sehingga ketika terjadi serangan tersebut, performa *routing protocol* AODV tetap terjaga.

Implementasi modifikasi *routing protocol* AODV ini dibagi menjadi tiga bagian, yaitu:

- Implementasi Deteksi *Black Hole Attack*
- Implementasi Deteksi *Worm Hole Attack*
- Implementasi Pemilihan *Nexthop Node*

Kode implementasi dari *routing protocol* AODV pada NS-2 versi 2.35 berada pada direktori ns-2.35/aodv. Di dalam diretori tersebut terdapat beberapa *file*, diantaranya aodv.cc, aodv.h, dan aodv_packet.h. Ketiga *file* tersebut akan dimodifikasi untuk dapat mengatasi *Black hole* dan *Worm hole attack* yang telah disebutkan sebelumnya.

Pada bagian ini penulis akan menjelaskan langkah-langkah dalam mengimplementasikan modifikasi *routing protocol* AODV untuk mengatasi *Black hole* dan *Worm hole attack*.

4.2.1 Implementasi Deteksi *Black Hole Attack*

Langkah awal yang dilakukan untuk mengimplementasikan pendeteksian *Black Hole attack* seperti yang telah dirancang pada subbab 3.3.1 adalah dengan menghitung jumlah paket RREQ yang diterima dari tiap *node* tetangga dan jumlah paket RREQ yang

dikirimkan kepada tiap *node* tetangga. Namun, secara *default* pada *routing protocol* AODV tidak dapat mengetahui siapa saja tetangga dari suatu *node*. Sehingga diperlukan pengiriman *hello packet* terlebih dahulu untuk dapat mengetahui siapa saja tetangga dari suatu *node*. *Hello packet* dapat diaktifkan dengan mengubah beberapa baris kode pada fungsi `AODV::command` menjadi seperti pada Gambar 4.12.

```
//#ifndef AODV_LINK_LAYER_DETECTION
htimer.handle((Event*) 0);
ntimer.handle((Event*) 0);
//#endif // LINK LAYER DETECTION
```

Gambar 4.12 Pengaktifan *Hello Packet*

Selanjutnya *node* tetangga dari suatu *node* disimpan dalam variabel `std::vector<std::vector<int> > myneigh(50)` dengan menambahkan kode pada fungsi `AODV::recvHello` seperti pada Gambar 4.13. Kode fungsi `AODV::recvHello` dapat dilihat pada Lampiran A.2. Karena *node* bersifat dinamis, maka *node* tetangganya bisa berubah-ubah setiap saat, sehingga daftar *node* tetangga dari suatu *node* harus selalu diperbarui. Memperbarui daftar *node* tetangga dari suatu *node* ini dapat dilakukan dengan mengosongkan variabel `myneigh(50)` setiap akan mengirimkan *hello packet*.

```
myneigh[(int)rp->rp_dst].push_back(index);
```

Gambar 4.13 Potongan Kode untuk Menyimpan *Node* Tetangga

Untuk dapat menghitung jumlah paket RREQ yang diterima dari tiap node tetangga, ditambahkan variabel `int record` pada header packet RREQ agar node yang menerima dapat mengetahui siapa yang mengirimkan paket tersebut. Kemudian diperlukan beberapa variabel untuk menjumlahkan paket RREQ yang diterima oleh suatu node dari tiap *node* tetangganya dan paket RREQ yang dikirim dari suatu *node* ke tiap *node* tetangganya, serta

variabel untuk membandingkan nilai keduanya. Variabel-variabel tersebut diantaranya `int kirimdari[50][50]`, `int kirimke[50][50]`, `double forward_eval[50][50]`. Gambaran inisialisasi variabel dapat dilihat pada Gambar 4.14.

```
std::vector<std::vector<int> > myneigh(50);
int kirimdari[50][50] = {0};
int kirimke[50][50] = {0};
double forward_eval[50][50] = {0.0};
```

Gambar 4.14 Inisialisasi Variabel untuk Melakukan Deteksi *Black Hole Attack*

Selanjutnya setiap *node* akan menjumlahkan paket RREQ yang diterima dari tiap *node* tetangganya setiap kali menerima paket dan menyimpan hasil jumlah pada variabel `kirimdari[50][50]`. *Node* tersebut juga akan menjumlahkan paket yang dikirimkan kepada *node* tetangganya setiap akan melakukan *forward* dan menyimpan hasil jumlah pada variabel `kirimke[50][50]`. Kemudian *node* tersebut membandingkan jumlah paket RREQ yang diterima dengan jumlah paket RREQ yang dikirimkan dan menyimpan nilainya pada variabel `forward_eval[50][50]`. Nilai dari variabel tersebut akan dijadikan bahan evaluasi untuk menentukan apakah *node* tersebut layak dijadikan *nexthop node*. Kode implementasi evaluasi ini diletakkan pada fungsi `AODV::recvRequest` dan dapat dilihat pada lampiran A.3.

4.2.2 Implementasi Deteksi *Worm Hole Attack*

Worm Hole attack dapat dideteksi dengan memperhatikan keaslian paket RREQ yang diterima dari *node* tetangga. Untuk dapat mengetahui keaslian paket RREQ yang diterima, dilakukan implementasi ECDSA sebagai algoritma *digital signature* yang digunakan.

Pada Tugas Akhir ini, ECDSA diimplementasikan menggunakan *library* OpenSSL. Namun *library* ini tidak termasuk ke dalam *package* NS-2, sehingga diperlukan modifikasi pada Makefile yang terdapat pada direktori ns2.35. Perubahan yang dilakukan pada Makefile dapat dilihat pada Gambar 4.15.

```
LIB   = \
      -L/home/tities/Documents/TA/ns-allinone-
2.35/tclcl-1.20 -ltclcl -
L/home/tities/Documents/TA/ns-allinone-
2.35/otcl-1.14 -lotcl -
L/home/tities/Documents/TA/ns-allinone-
2.35/lib -ltk8.5 -
L/home/tities/Documents/TA/ns-allinone-
2.35/lib -ltcl8.5 \
      -L/usr/include/openssl -lssl -lcrypto\
      -lXext -lX11 \
       -lnsl -ldl \
      -lm -lm
```

Gambar 4.15 Modifikasi Makefile

Setelah melakukan modifikasi Makefile, *library* OpenSSL dapat digunakan pada NS-2 dengan memanggil *library* yang dibutuhkan. Daftar *library* OpenSSL yang digunakan untuk mengimplementasikan ECDSA dapat dilihat pada Gambar 4.16.

```
#include <openssl/sha.h>
#include <openssl/ec.h>
#include <openssl/obj_mac.h>
#include <openssl/ecdsa.h>
```

Gambar 4.16 Daftar *Library* OpenSSL yang Digunakan

Untuk menjaga keaslian paket RREQ yang dikirimkan, maka sebelum suatu *node* mengirimkan paket RREQ, akan di*generate signature* dari paket RREQ tersebut. Kode untuk men*generate signature* ini ditambahkan dalam fungsi

`AODV::sendRequest` dan dapat dilihat pada lampiran A.4. Kemudian signature dari paket RREQ tersebut dicantumkan pada *header packet*. Oleh karena itu diperlukan beberapa variabel, diantaranya adalah `char mdString[SHA256_DIGEST_LENGTH*2+1]`, `EC_KEY *eckey`, dan `ECDSA_SIG *signature`. Variabel `mdString` digunakan untuk menyimpan nilai *hash* dari isi paket. Variabel `eckey` digunakan untuk menyimpan *public key*, sedangkan variabel `signature` digunakan untuk menyimpan *digital signature*. Kode implementasi perubahan struktur *header packet* dapat dilihat pada Gambar 4.17.

```
struct hdr_aodv_request {
        ...
        int record;
        char mdString[SHA256_DIGEST_LENGTH*2+1];
        EC_KEY *eckey;
        ECDSA_SIG *signature;
        ...
}
```

Gambar 4.17 Perubahan Struktur *Header Packet*

Selanjutnya, pada tiap node yang menerima paket RREQ dari tiap node tetangganya dilakukan verifikasi signature paket. Beberapa variabel yang dibutuhkan yaitu `int verified[50][50]`, dan `double backward_eval[50][50]`. Selain menghitung jumlah paket RREQ yang diterima, node yang menerima paket juga akan menghitung jumlah paket RREQ yang terverifikasi keasliannya dan menyimpan nilainya pada variabel `verified[50][50]`. Kemudian jumlah paket RREQ yang terverifikasi tersebut dibandingkan dengan jumlah paket RREQ yang diterima dari tiap node tetangganya. Nilai perbandingan ini disimpan dalam variabel `backward_eval[50][50]`. Nilai dari variabel tersebut akan dijadikan bahan evaluasi untuk menentukan apakah *node* tersebut layak dijadikan *nexthop node*. Kode implementasi evaluasi ini diletakkan

pada fungsi `AODV::recvRequest` dan dapat dilihat pada lampiran A.3.

4.2.3 Implementasi Pemilihan *Nexthop Node*

Setelah mendapatkan nilai perbandingan `forward_eval` sebagai hasil deteksi *Black Hole attack* dan `backward_eval` sebagai hasil deteksi *Worm Hole attack*, langkah selanjutnya adalah melakukan evaluasi dari kedua nilai perbandingan tersebut untuk menentukan apakah suatu *node* layak dijadikan sebagai *nexthop node*. Oleh karena itu, dibutuhkan batas nilai sebagai pembanding dalam menentukan hal tersebut yang dinamakan dengan threshold. Pada Tugas Akhir ini, nilai threshold yang digunakan adalah 0.7.

Nilai evaluasi yang akan dibandingkan dengan nilai threshold didapatkan dengan menjumlahkan nilai `forward_eval` yang telah dikalikan dengan bilangan desimal random antara 0 sampai 1 dan nilai `backward_eval` yang juga telah dikalikan dengan bilangan desimal random antara 0 sampai 1. Kemudian nilai evaluasi ini disimpan ke dalam variabel `double eval_value[50][50]`.

Sebelum memilih *nexthop node*, nilai `eval_value` dibandingkan dengan nilai threshold terlebih dahulu. Jika nilai `eval_value` lebih besar dari nilai threshold, maka *node* tersebut layak untuk dijadikan *nexthop node*. Kemudian *nexthop node* akan dipilih sesuai dengan algoritma pada *routing protocol* AODV. Jika nilai `eval_value` lebih kecil atau sama dengan nilai threshold, maka *node* tersebut dideteksi sebagai *malicious node* yang berpotensi melakukan *Black Hole* atau *Worm Hole attack* sehingga secara otomatis tidak akan mendapat kesempatan untuk dipilih sebagai *nexthop node*. Potongan kode untuk pemilihan *nexthop node* dapat dilihat pada Gambar 4.18. Kode tersebut diletakkan pada fungsi `AODV::recvRequest` dan dapat dilihat pada lampiran A.3.

```
if(eval_value[(int)myneigh[index][i]][index] >
0.7){
       if (rt) rq->rq_dst_seqno = max(rt-
>rt_seqno, rq->rq_dst_seqno);
    }
```

Gambar 4.18 Potongan Kode Pemilihan *Nexthop Node*

4.3 Implementasi Simulasi pada NS-2

Implementasi simulasi VANET diawali dengan pendeskripsian lingkungan simulasi pada sebuah *file* tcl. *File* ini berisikan konfigurasi setiap *node* dan langkah-langkah yang dilakukan selama simulasi. Potongan konfigurasi lingkungan simulasi dapat dilihat pada Gambar 4.19.

```
set val(chan)      Channel/WirelessChannel
set val(prop)      Propagation/TwoRayGround
set val(netif)     Phy/WirelessPhy
set val(mac)       Mac/802_11
set val(ifq)       Queue/DropTail/PriQueue
set val(ll)        LL
set val(ant)       Antenna/OmniAntenna
set val(ifqlen)    50
set val(nn)        30
set val(rp)        AODV
set val(energymodel)    EnergyModel
set val(initialenergy)  100
set val(lm)        "off"
set val(x)         1700
set val(y)         1700
set val(stop)      200
set val(cp)        "traffic"
set val(sc)        "scenario.tcl"
```

Gambar 4.19 Konfigurasi Lingkungan Simulasi

Pada konfigurasi dilakukan pemanggilan terhadap *file* traffic yang berisikan konfigurasi *node* asal dan *node* tujuan, serta *file* skenario yang berisi pergerakan node yang telah di*generate* oleh SUMO. Kode implementasi pada NS-2 dapat dilihat pada lampiran A.1.

4.3.1 Implementasi Black Hole Attack

Selain konfigurasi lingkungan simulasi dan pergerakan *node*, dilakukan juga konfigurasi untuk menugaskan *node* yang akan melakukan *Black hole attack* dan bagaimana *node* tersebut melakukan tugasnya sebagai *attacker*. Konfigurasi dilakukan di beberapa *file*, diantaranya *file* tcl, aodv.h, dan aodv.cc.

Pada *file* tcl, suatu *node* ditugaskan menjadi *Black hole attacker* dengan memberikan tambahan *kode* seperti yang ditunjukkan oleh Gambar 4.20.

```
$ns_ at 0.0 "[$node_(4) set ragent_] hacker"
```

Gambar 4.20 Potongan kode untuk menugaskan *node* sebagai *Black hole attacker*

Setelah itu, pada *file* aodv.h dilakukan modifikasi dengan menambahkan variabel `bool malicious` yang digunakan untuk menandai jika suatu *node* adalah *malicious node* yang melakukan *Black hole attack*.

Agar *node* yang ditugaskan menjadi *Black hole attacker* dapat melakukan *Black hole attack*, dilakukan modifikasi pada *file* aodv.cc. Modifikasi pertama yang dilakukan adalah mengenali *attacker* yang telah dideklarasikan di file tcl. Modifikasi ini diletakkan pada fungsi `AODV::command`, kode untuk mengenali *attacker* dapat dilihat pada Gambar 4.21.

```
if(strncasecmp(argv[1], "hacker", 6) == 0) {
      malicious = true;
      return TCL_OK;
}
```

Gambar 4.21 Potongan Kode untuk Mengenali *Black Hole Attacker*

Selanjutnya menginisialisasi variabel `malicious` diletakkan pada fungsi `AODV::AODV`, inisialisasi dapat dilihat pada Gambar 4.22.

```
malicious = false;
```

Gambar 4.22 Inisialisasi Variabel Malicious

Black hole attack dilakukan dengan melakukan *drop* pada setiap paket yang diterima. Untuk dapat melakukan itu, dilakukan modifikasi pada fungsi `AODV::rt_resolve` dengan menambahkan kode seperti pada Gambar 4.23.

```
if (malicious == true ) {
      drop(p, DROP_RTR_ROUTE_LOOP);
      return;
}
```

Gambar 4.23 Potongan Kode untuk Melakukan *Drop* paket

Selain melakukan *drop* pada paket, *attacker* juga mengirimkan respon kepada *node* asal sehingga *node* asal mengira bahwa paket yang ia kirimkan telah diterima. Untuk dapat melakukan hal tersebut, dilakukan modifikasi pada fungsi `AODV::recvRequest` dengan menambahkan kode seperti pada Gambar 4.24.

```
else if (malicious == true) {
      seqno = max(seqno, rq->rq_dst_seqno)+1;
      if (seqno%2) seqno++;
      sendReply(rq->rq_src,
            1,
            rq->rq_dst,
            seqno,
            MY_ROUTE_TIMEOUT,
            rq->rq_timestamp);
      Packet::free(p);
}
```

Gambar 4.24 Potongan Kode *Black Hole Attacker* Mengirim Paket *Reply*

4.3.2 Implementasi Worm Hole Attack

Sama seperti implementasi *Black hole attack*, *file* yang diperlukan untuk konfigurasi antara lain *file* tcl, aodv.h dan aodv.cc. Konfigurasi pertama yang dilakukan adalah dengan menugaskan suatu *node* untuk menjadi *Worm hole attacker* dengan menambahkan kode pada *file* tcl seperti pada Gambar 4.25.

```
$ns_ at 0.0 "[$node_(4) set ragent_]
wormhole"
```

Gambar 4.25 Potongan Kode untuk Menugaskan *Node* sebagai *Worm Hole Attacker*

Kemudian pada *file* aodv.h dilakukan modifikasi dengan menambahkan variabel `bool worm` yang digunakan untuk menandai apakah suatu *node* adalah *malicious node* yang melakukan *Worm hole attack*.

Selanjutnya dilakukan modifikasi pada *file* aodv.cc agar *node* dapat melakukan *Worm hole attack*. Sama seperti implementasi *Black hole attack*, hal pertama yang dilakukan adalah mengenali jika suatu *node* ditugaskan melakukan *Worm hole*

attack dengan menambahkan kode pada fungsi `AODV::command` seperti pada Gambar 4.26.

```
if(strncasecmp(argv[1], "wormhole", 8) == 0){
      worm = true;
      return TCL_OK;
}
```

Gambar 4.26 Potongan Kode untuk Mengenali *Worm Hole Attacker*

Variabel `worm` diinisialisasi terlebih dahulu nilainya dengan melakukan modifikasi pada fungsi `AODV::AODV` seperti yang ditunjukkan pada Gambar 4.27.

```
worm = false;
```

Gambar 4.27 Inisialisasi Variabel Worm

Worm hole attack dilakukan dengan memodifikasi isi paket oleh *node* yang berada di antara *node* asal dan *node* tujuan. Dalam Tugas Akhir ini, isi paket yang diubah adalah *node* tujuan. Selain mengubah isi paket, *Worm hole attacker* mengirimkan respon kepada *node* asal sehingga *node* asal mengira bahwa paketnya telah tersampaikan dengan aman. Modifikasi untuk mengimplementasikan hal tersebut diletakkan pada fungsi `AODV::recvRequest` dengan menambahkan kode seperti yang ditunjukkan pada Gambar 4.28.

```
if(worm == true){
      seqno = max(seqno, rq->rq_dst_seqno)+1;
      if (seqno%2) seqno++;
      sendReply(rq->rq_src,
            1,
            rq->rq_dst,
            seqno,
            MY_ROUTE_TIMEOUT,
            rq->rq_timestamp);
      rq->rq_dst = 3;
}
```

Gambar 4.28 Potongan Kode untuk Mengirim Paket *Reply* dan Memodifikasi Paket yang Diterima

4.4 Implementasi Metrik Analisis

Simulasi yang telah dijalankan oleh NS-2 menghasilkan keluaran sebuah *trace file* yang berisikan data mengenai apa saja yang terjadi selama simulasi dalam bentuk *plain text*. Dari data *trace file*, dapat dilakukan analisis performa *routing protocol* dengan mengukur beberapa metrik. Pada Tugas Akhir ini, metrik yang akan dianalisis adalah *Packet Delivery Ratio* (PDR), Rata-rata *End to End Delay* (E2E), dan *Routing Overhead* (RO).

4.4.1 Implementasi Packet Delivery Ratio (PDR)

Pada bagian telah ditunjukkan contoh struktur data *event* yang dicatat dalam *trace file* oleh NS-2. Kemudian, pada persamaan telah dijelaskan bagaimana menghitung PDR. Dengan begitu, dilakukan penghitungan PDR melalui bantuan skrip awk. Skrip awk untuk menghitung PDR berdasarkan kedua informasi tersebut dapat dilihat pada lampiran A.5 Kode Skrip AWK *Packet Delivery Ratio*.

Dalam perhitungan PDR, kata kunci yang perlu diperhatikan dari *trace file* adalah *layer* AGT, karena kata kunci tersebut menunjukkan *event* yang bersangkutan dengan paket

komunikasi data. Kemudian hitung jumlah paket yang dikirimkan dan paket yang diterima dengan menggunakan karakter pada kolom pertama sebagai *filter*, karena kolom pertama yang menunjukkan *event* yang terjadi dari sebuah paket. Setelah itu nilai PDR dapat dihitung dengan persamaan yang telah dijelaskan. *Pseudocode* untuk menghitung PDR dapat dilihat pada Gambar 4.29.

```
sent = 0
received = 0
for i = 1 to the number of rows
     if in a row contains "s" and AGT then
          sent++
     else if in a row contains "r" and AGT
then
          received++
     end if
pdr = received / sent
```

Gambar 4.29 Pseudocode untuk Menghitung PDR

Contoh perintah pengeksekusian skrip awk untuk menganalisis *trace file* beserta *output*nya dapat dilihat masing-masing pada Gambar 4.30 dan Gambar 4.31.

```
awk –f pdr.awk s-aodv2.tr
```

Gambar 4.30 Perintah Pengeksekusian Skrip awk PDR

```
pdr: xx.xx (float number)
```

Gambar 4.31 *Output* Skrip awk PDR

4.4.2 Implementasi Rata-rata *End to End Delay* (E2E)

Perhitungan E2E telah dijelaskan melalui persamaan. Skrip awk untuk menghitung E2E dapat dilihat pada lampiran A.6 Kode Skrip AWK Rata-rata *End to End Delay*.

Dalam perhitungan E2E, langkah yang digunakan untuk mendapatkan E2E hampir sama dengan ketika mencari PDR, hanya saja yang perlu diperhatikan adalah waktu dari sebuah *event* yang tercatat pada kolom ke-2 dengan *filter event* pada kolom ke-4 adalah layer AGT dan *event* pada kolom pertama guna membedakan paket dikirim atau diterima. Setelah seluruh baris yang memenuhi didapatkan, akan dihitung *delay* dari paket dengan mengurangi waktu dari paket diterima dengan waktu dari paket dikirim dengan sayarat memiliki *id* paket yang sama.

Setelah mendapatkan *delay* paket, langkah selanjutnya adalah dengan mencari rata-rata dari *delay* tersebut dengan menjumlahkan semua *delay* paket dan membaginya dengan jumlah paket. *Pseudocode* untuk menghitung rata-rata *end to end delay* dapat dilihat pada Gambar 4.32.

```
sum_delay = 0
counter = 0
for i = 1 to the number of rows
     counter++
     if layer == AGT and event == s then
      start_time[packet_id] = time
     else if layer == AGT and event == r then
      end_time[packet_id] = time
     end if
     delay[packet_id] = end_time[packet_id] -
start_time[packet_id]
     sum_delay += delay[packet_id]
e2e = sum_delay / counter
```

Gambar 4.32 Pseudocode untuk Menghitung E2E

Contoh perintah pengeksekusian skrip awk untuk menganalisis *trace file* beserta *output*nya dapat dilihat masing-masing pada Gambar 4.33 dan Gambar 4.34.

```
awk –f e2e.awk s-aodv2.tr
```

Gambar 4.33 Perintah Pengeksekusian Skrip awk E2E

```
e2e: xx.xx (float number)
```

Gambar 4.34 *Output* Skrip awk E2E

4.4.3 Implementasi Routing Overhead (RO)

Perhitungan RO telah dijelaskan pada persamaan. Skrip awk untuk menghitung RO dapat dilihat pada lampiran A.7 Kode Skrip AWK *Routing Overhead*.

Seperti yang telah dijelaskan sebelumnya, *routing overhead* merupakan jumlah dari paket kontrol *routing* baik itu RREQ, RREP, maupun RERR. Dengan begitu, untuk mendapatkan RO yang perlu dilakukan adalah menjumlahkan tiap paket dengan *filter event sent* pada kolom pertama dan *event layer* RTR pada kolom ke-4. *Pseudocode* untuk menghitung RO dapat dilihat pada Gambar 4.35.

```
ro = 0
for i = 1 to the number of rows
     if in a row contains "s" and RTR and is
not hello packet then
          ro++
     end if
```

Gambar 4.35 Pseudocode untuk Menghitung RO

Contoh perintah pengeksekusian skrip awk untuk menganalisis *trace file* beserta outputnya dapat dilihat masing-masing pada Gambar 4.36 dan Gambar 4.37.

```
awk -f ro.awk s-aodv2.tr
```

Gambar 4.36 Perintah Pengeksekusian Skrip awk RO

```
ro: xxx (integer number)
```

Gambar 4.37 *Output* Skrip awk RO

BAB V
HASIL UJI COBA DAN EVALUASI

Bab ini membahas mengenai uji coba dan evaluasi dari skenario yang telah disimulasikan di NS-2.

5.1 Lingkungan Uji Coba

Uji coba dilakukan pada perangkat dengan spesifikasi seperti yang tertera pada Tabel 5.1.

Tabel 5.1 Spesifikasi Perangkat yang Digunakan

Komponen	**Spesifikasi**
CPU	Intel(R) Core(TM) i3-3217U CPU @ 1.80 GHz x 2
Sistem Operasi	Linux Mint 17.3 Cinnamon 64-bit
Linux Kernel	3.19.0-32-generic
Memori	1.8 GiB
Penyimpanan	26.5 GB

Pengujian dilakukan dengan menjalankan skenario yang disimulasikan pada NS-2. Dari simulasi tersebut dihasilkan sebuah trace file dengan ekstensi.tr yang akan dianalisis dengan bantuan skrip awk untuk mendapatkan packet delivery ratio, rata-rata end to end delay, dan routing overhead menggunakan kode pada lampiran masing-masing A.5 Kode Skrip AWK *Packet Delivery Ratio*, A.6 Kode Skrip AWK Rata-rata *End to End Delay*, dan A.7 Kode Skrip AWK *Routing Overhead*.

5.2 Hasil Uji Coba

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

5.2.1 Hasil Uji Coba Skenario *Grid*

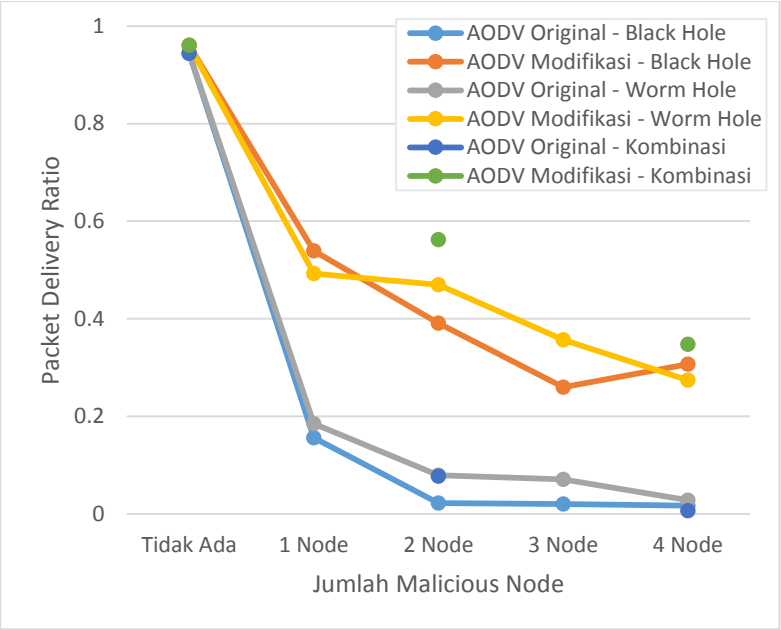
Pengujian pada skenario *grid* digunakan untuk melihat perbandingan *packet delivery ratio*, rata-rata *end to end delay*, dan *routing overhead* antara *routing protocol* AODV *original* dan *routing protocol* AODV yang telah dimodifikasi ketika terjadi serangan *Blackhole, Wormhole*, maupun keduanya di satu waktu, dengan kondisi peta yang lebih stabil dan mempunyai kecenderungan persebaran *node* yang merata.

Pengujian dilakukan sesuai dengan perancangan parameter lingkungan simulasi yang dapat dilihat pada tabel Tabel 3.2. Selain itu, ditambahkan beberapa *malicious node* dengan beberapa variasi serangan dalam satu skenario.

Pengambilan data uji PDR, rata-rata *end to end delay*, dan *routing overhead* dengan luas area 1700 m x 1700 m dan *node* sebanyak 30, 40, dan 50 dilakukan pada kecepatan standar yaitu 20 m/s. Pada skenario 30 *node*, dilakukan variasi serangan dengan jumlah malicious *node* yang bertambah dari 1 *node* hingga 2 *node*. Sedangkan pada skenario 40 dan 50 *node*, jumlah malicious *node* bertambah dari 1 *node* hingga 4 *node*. Pada tiap *malicious node*, ditugaskan untuk menjadi *Blackhole attacker* dan *Wormhole attacker*. Untuk jumlah lebih dari 1 *malicious node*, dilakukan beberapa variasi serangan, yaitu semua *malicious node* melakukan *Blackhole attack*, semua *malicious node* melakukan *Wormhole attack*, dan sebagian melakukan *Blackhole attack*, sedangkan sebagian yang lain melakukan *Wormhole attack*.

Pengujian dilakukan dengan beberapa variasi serangan yang berbeda untuk melihat pengaruh serangan pada performa routing protocol AODV dan seberapa banyak *malicious node* beserta variasi serangannya yang dapat diatasi oleh *routing protocol* AODV yang telah dimodifikasi. Untuk tiap skenario penyerangan, pengujian dilakukan sebanyak 3 kali.

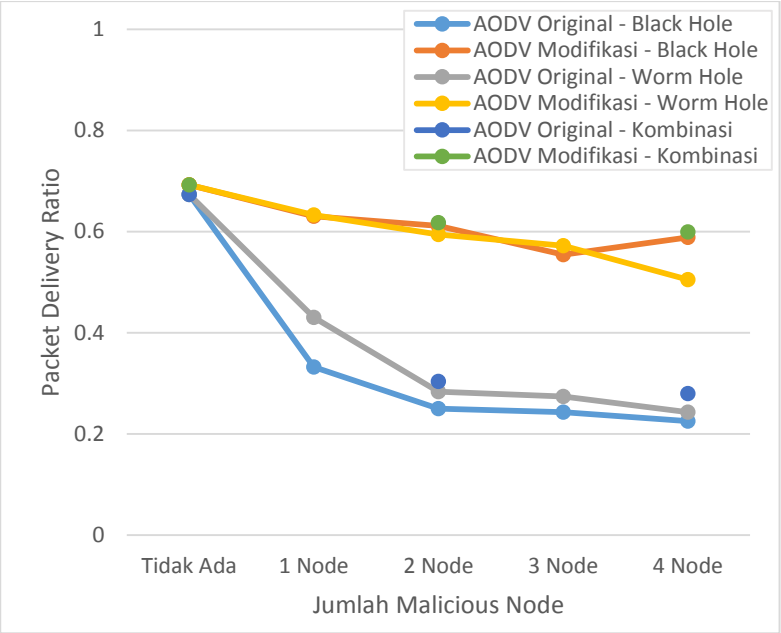
Hasil pengambilan data *packet delivery ratio* (PDR) pada skenario *grid* 30, 40, dan 50 *node* masing-masing dapat dilihat pada Gambar 5.1, Gambar 5.2, dan Gambar 5.3.



Gambar 5.1 Grafik PDR terhadap Banyak *Malicious Node* dan Variasi Serangannya pada Skenario *Grid* 30 *Node*

Berdasarkan grafik pada Gambar 5.1, Gambar 5.2, dan Gambar 5.3 dapat dilihat bahwa *routing protocol* AODV yang telah dimodifikasi mampu mengatasi *Black hole*, dan *Worm hole attack* maupun kombinasi keduanya, yang ditunjukkan oleh mampunya *protocol* melakukan *recovery* paket yang sempat turun drastis akibat terjadinya serangan, yang dibuktikan oleh nilai PDR.

Dengan mengimplementasikan beberapa metode yang telah dipaparkan pada bab sebelumnya, *routing protocol* ini mampu mengatasi beberapa *malicious node* dengan variasinya dari 1 *node attacker* hingga 4 *node attacker* dalam satu topologi.

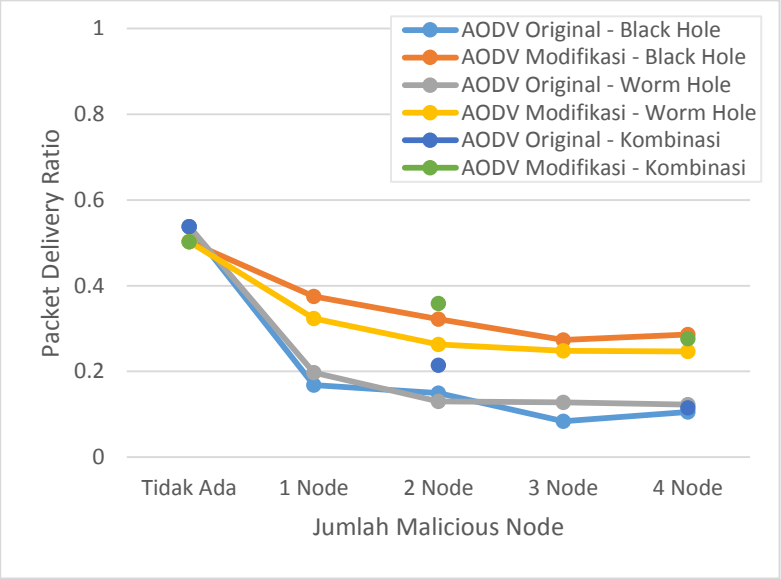


Gambar 5.2 Grafik PDR terhadap Banyak *Malicious Node* dan Variasi Serangannya pada Skenario *Grid* 40 *Node*

Pada uji coba skenario dengan *malicious node*, *routing protocol* yang telah dimodifikasi mengalami peningkatan PDR jika dibandingkan dengan *routing protocol original*. Peningkatan PDR rata-rata pada skenario 30 *node*, 40 *node*, dan 50 *node* sebesar 39.42% ketika terjadi *Black Hole attack*, 31.93% ketika terjadi *Worm Hole attack*, dan 39.73% ketika terjadi kombinasi serangan keduanya.

Pada skenario dimana *malicious node* melakukan *Black Hole attack*, peningkatan PDR tertinggi terjadi saat terdapat 4 *node Black Hole* pada uji coba 40 *node*. Nilai PDR yang awalnya pada skenario normal adalah 0.674 turun drastis saat terjadi *Black Hole attack* menjadi 0.221 ketika menggunakan *routing protocol original*. Namun dengan *routing protocol* yang telah dimodifikasi, nilai PDR menjadi 0.588 atau dapat dikatakan mengalami

peningkatan sebesar 53.94%. Sedangkan peningkatan terendah terjadi saat terdapat 3 *node Black Hole* pada uji coba 30 *node*, yaitu hanya sebesar 25.34%.



Gambar 5.3 Grafik PDR terhadap Banyak *Malicious Node* dan Variasi Serangannya pada Skenario *Grid* 50 *Node*

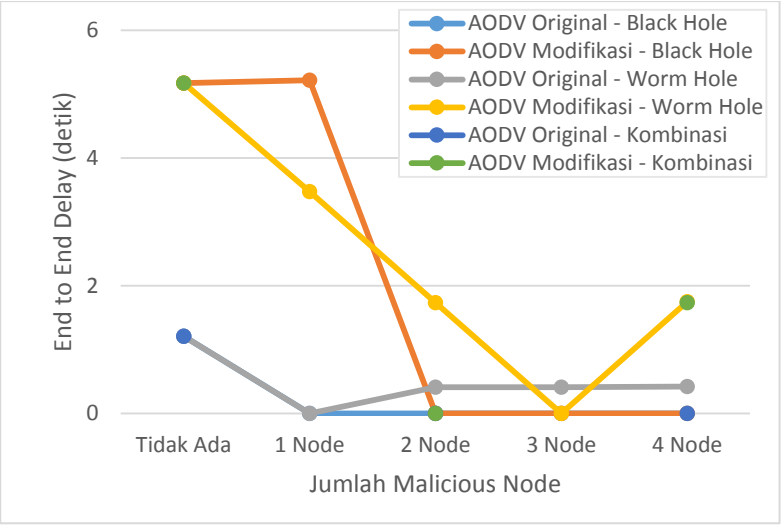
Pada skenario dimana *malicious node* melakukan *Worm Hole attack*, peningkatan PDR tertinggi terjadi saat terdapat 2 *node Worm Hole* pada uji coba 40 *node*. Nilai PDR yang awalnya pada skenario normal adalah 0.674 turun drastis saat terjadi *Worm Hole attack* menjadi 0.284 ketika menggunakan *routing protocol original*. Namun dengan *routing protocol* yang telah dimodifikasi, nilai PDR menjadi 0.572 atau dapat dikatakan mengalami peningkatan sebesar 46.07%. Sedangkan peningkatan terendah terjadi saat terdapat 3 *node Worm Hole* pada uji coba 50 *node*, yaitu hanya sebesar 22.29%.

Untuk uji coba dimana *malicious node* melakukan kombinasi antara *Black Hole* dan *Worm Hole attack*, hanya

dilakukan ketika terdapat 2 *malicious node* dan 4 *malicious node* dengan perbandingan jumlah *malicious node* yang melakukan *Black Hole attack* sama dengan jumlah *malicious node* yang melakukan *Worm Hole attack*. Pada skenario ini, peningkatan PDR tertinggi terjadi saat terdapat 2 *malicious node* dengan rincian 1 *node* melakukan *Black Hole attack* dan 1 *node* yang lain melakukan *Worm Hole attack* pada uji coba 30 *node*. Nilai PDR yang awalnya pada skenario normal adalah 0.944 turun drastis saat terjadi kombinasi kedua serangan tersebut menjadi 0.078 ketika menggunakan *routing protocol original*. Namun dengan *routing protocol* yang telah dimodifikasi, nilai PDR menjadi 0.562 atau dapat dikatakan mengalami peningkatan sebesar 51.28%. Sedangkan peningkatan terendah terjadi saat terdapat 2 *node attacker* pada uji coba 50 *node*, yaitu hanya sebesar 26.81%.

Jika ketiga grafik tersebut dibandingkan, dapat dilihat bahwa dengan *node* yang lebih banyak, selisih antara paket data yang hilang dan paket data yang di*recovery* menjadi kurang signifikan, namun *recovery* yang dilakukan terlihat lebih stabil.

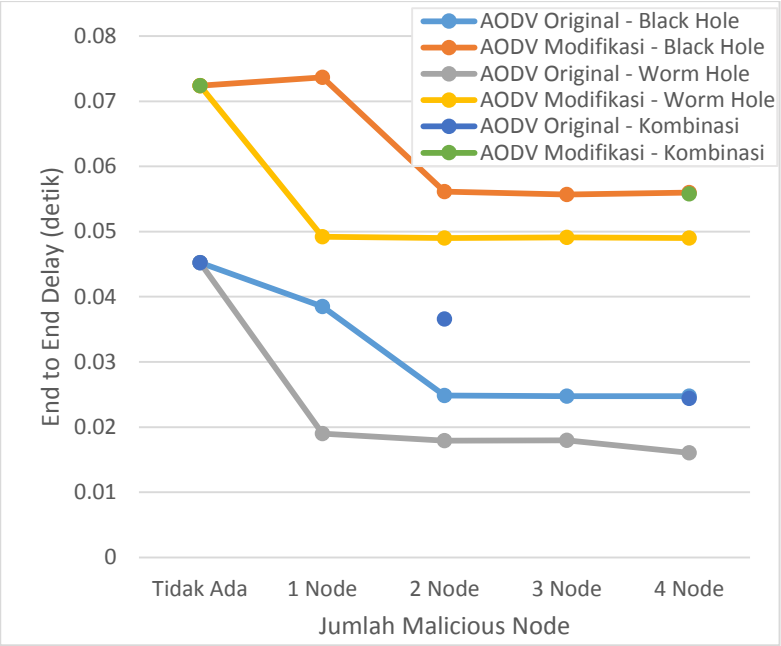
Hasil pengambilan data rata-rata *end to end delay* (E2E) pada skenario *grid* 30, 40, dan 50 *node* dapat dilihat masing-masing pada Gambar 5.4, Gambar 5.5, dan Gambar 5.6.



Gambar 5.4 Grafik E2E terhadap Banyak *Malicious Node* dan Variasi Serangannya pada Skenario *grid* 30 *node*

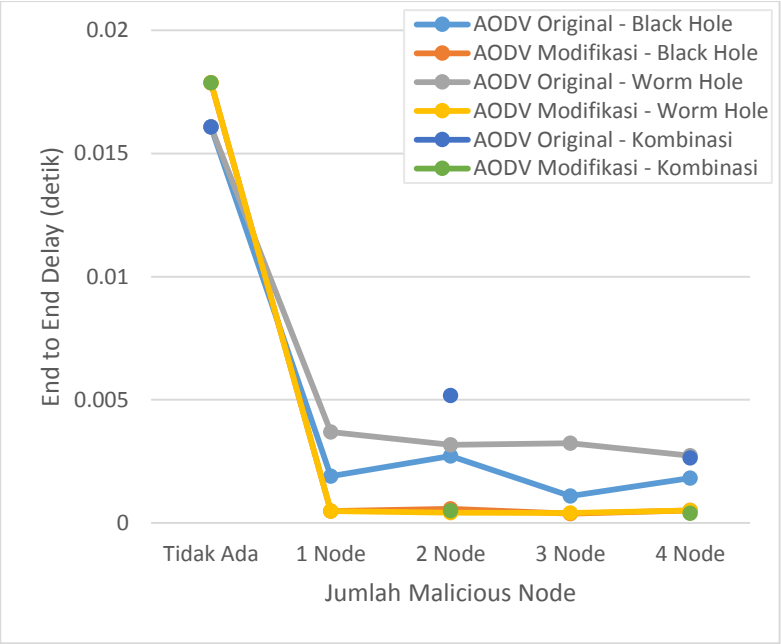
Berdasarkan grafik pada Gambar 5.4, Gambar 5.5, dan Gambar 5.6 dapat dilihat bahwa rata-rata *end to end delay* antara *routing protocol* AODV *original* dan AODV yang telah dimodifkasi tidak selalu lebih besar dan tidak selalu lebih kecil. Pada grafik-grafik diatas, *delay* menjadi lebih besar ketika paket yang sampai ke tujuan lebih banyak, begitu pula sebaliknya.

Pada skenario 30 *node* saat terdapat 1 *malicious node* yang melakukan *Black Hole attack*, paket yang sampai ke tujuan jauh lebih sedikit ketika menggunakan *routing protocol original* dibandingkan ketika menggunakan *routing protocol* yang telah dimodifikasi. Sehingga nilai *end to end delay* meningkat drastis ketika menggunakan *routing protocol* yang telah dimodifikasi. Namun pada skenario 30 *node* ini, nilai *end to end delay* terlihat tidak stabil. Hal ini dapat dilihat pada Gambar 5.4.



Gambar 5.5 Grafik E2E terhadap Banyak *Malicious Node* dan Variasi Serangannya pada Skenario *Grid* 40 *Node*

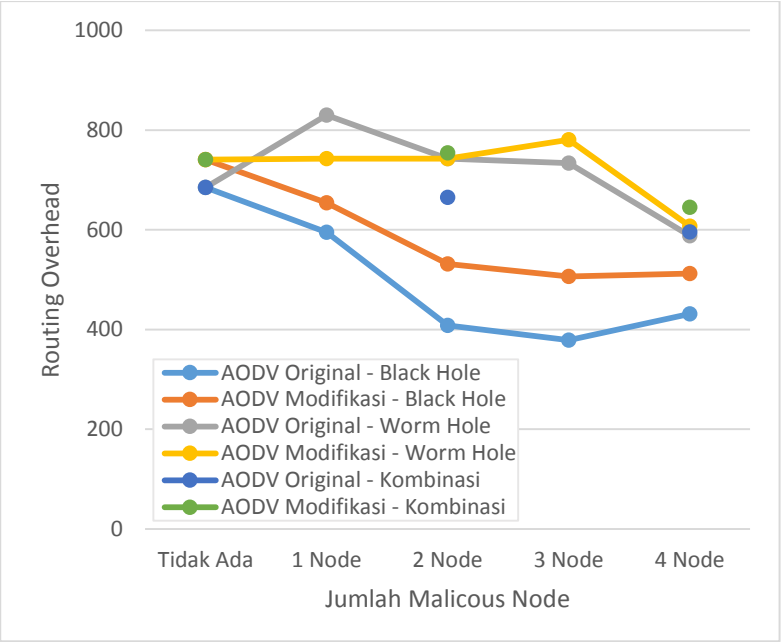
Sedangkan pada skenario 40 *node* dan 50 *node*, nilai *end to end delay* terlihat relatif lebih stabil jika dibandingkan dengan skenario 30 *node*. Dapat dibandingkan antara nilai *end to end delay* ketika menggunakan *routing protocol original* dengan *routing protocol* yang telah dimodifikasi, grafik keduanya terlihat identik. Hanya saja nilai *end to end delay* ketika menggunakan *routing protocol original* lebih rendah. Hal ini disebabkan jumlah paket yang sampai ke tujuan lebih sedikit dibandingkan dengan ketika menggunakan *routing protocol* yang telah dimodifikasi. Grafik dapat dilihat pada Gambar 5.5 dan Gambar 5.6.



Gambar 5.6 Grafik E2E terhadap Banyak *Malicious Node* dan Variasi Serangannya pada Skenario *Grid* 50 *Node*

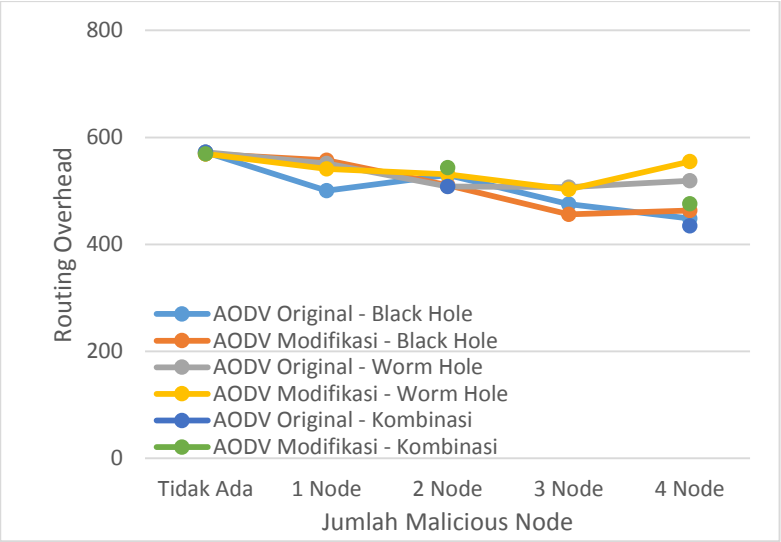
Hasil pengambilan data *routing overhead* (RO) pada skenario *grid* 30, 40, dan 50 *node* dapat dilihat masing-masing pada Gambar 5.7, Gambar 5.8, dan Gambar 5.9.

Berdasarkan grafik pada Gambar 5.7, Gambar 5.8, dan Gambar 5.9 dapat dilihat bahwa *routing protocol* AODV yang telah dimodifikasi tidak selalu dapat mengatasi *routing overhead* yang disebabkan oleh serangan. Ada di beberapa skenario uji coba saat *routing protocol* ini dapat meredakan *routing overhead*, terutama saat terjadi *Worm Hole attack*, namun hal tersebut tidak selalu terjadi.

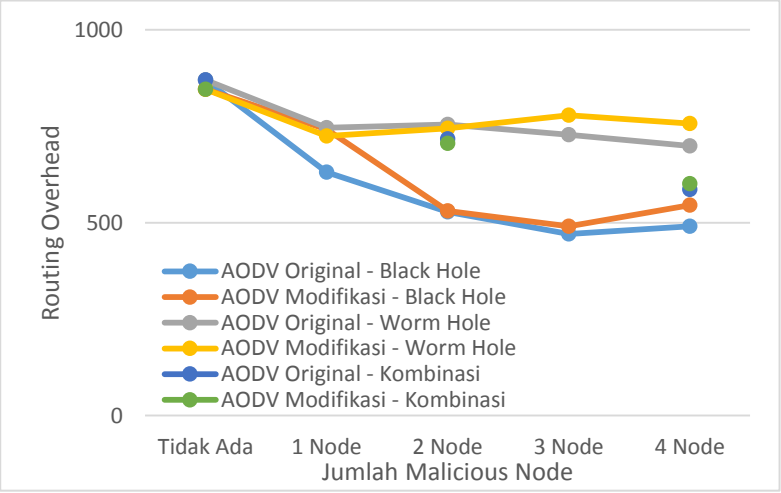


Gambar 5.7 Grafik RO terhadap Banyak *Malicious Node* dan Variasi Serangannya pada Skenario *Grid* 30 *Node*

Salah satu yang merepresentasikan hal tersebut adalah pada skenario 30 *node* ketika terdapat 1 *malicious node* yang melakukan *Worm Hole attack*, nilai *routing overhead* meningkat saat menggunakan *routing protocol original*. Namun saat menggunakan *routing protocol* yang telah dimodifikasi, peningkatan *routing overhead* tersebut dapat diredakan. Hal ini dapat dilihat pada Gambar 5.7.



Gambar 5.8 Grafik RO terhadap Banyak *Malicious Node* dan Variasi Serangannya pada Skenario *Grid* 40 *Node*



Gambar 5.9 Grafik RO terhadap Banyak *Malicious Node* dan Variasi Serangannya pada Skenario *Grid* 50 *Node*

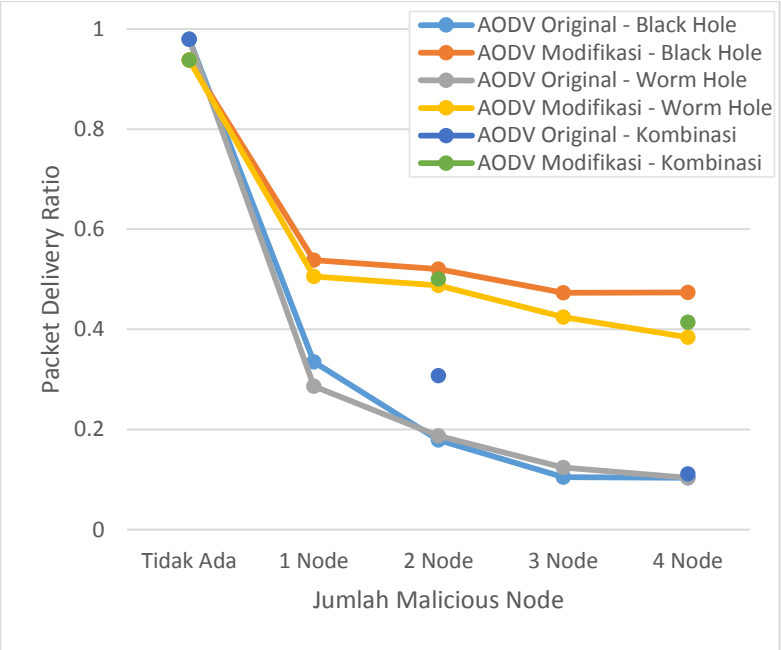
5.2.2 Hasil Uji Coba Skenario *Real*

Pengujian pada skenario *real* digunakan untuk melihat perbandingan performa *routing protocol* AODV *original* dengan *routing protocol* AODV yang telah dimodifikasi ketika terjadi serangan, dalam hal ini adalah *Black Hole attack* dan *Worm Hole attack*, maupun kombinasi keduanya jika diimplementasikan menggunakan peta nyata dimana persebaran kendaraannya tidak selalu merata. Pengambilan data uji *packet delivery ratio* (PDR), rata-rata *end to end delay* (E2E), dan *routing overhead* (RO) dilakukan pada peta *real* dengan luas area 1700 m x 1700 m pada wilayah perumahan Kertajaya Surabaya.

Sama seperti uji coba skenario *grid*, pada uji coba skenario *real* ini dilakukan dengan 30 *node*, 40 *node*, dan 50 *node*. Selain itu, ditambahkan *malicious node* sebagai *Black Hole* dan *Worm Hole attacker*, serta kombinasi keduanya. Variasi skenario serangan yang dilakukan sama seperti uji coba skenario grid, yaitu dari 1 hingga 4 *node attacker* untuk *Black Hole* dan *Worm Hole attack*, serta 2 dan 4 *node* untuk kombinasi *Black Hole* dan *Worm Hole attack*.

Hasil pengambilan data PDR pada skenario *real* 30, 40, dan 50 *node* dapat dilihat pada Gambar 5.10, Gambar 5.11, dan Gambar 5.12.

Berdasarkan grafik pada Gambar 5.10, Gambar 5.11, dan Gambar 5.12 *routing protocol* AODV yang telah dimodifikasi mampu mengatasi *Black Hole*, dan *Worm Hole attack* maupun kombinasi keduanya dengan ditunjukkan oleh paket data yang dapat di*recovery* selama serangan terjadi.

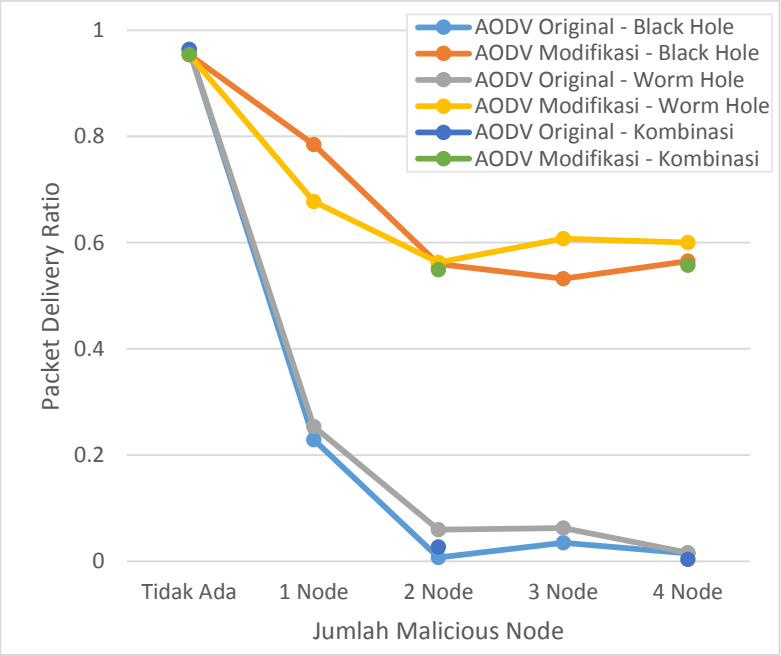


Gambar 5.10 Grafik PDR terhadap Banyak *Malicious Node* dan Variasi Serangannya pada Skenario *Real* 30 *Node*

Pada uji coba skenario dengan *malicious node*, *routing protocol* yang telah dimodifikasi mengalami peningkatan PDR jika dibandingkan dengan *routing protocol original*. Peningkatan PDR rata-rata pada skenario 30 *node*, 40 *node*, dan 50 *node* sebesar 46.04% ketika terjadi *Black Hole attack*, 41.91% ketika terjadi *Worm Hole attack*, dan 44.25% ketika terjadi kombinasi serangan keduanya.

Pada skenario dimana *malicious node* melakukan *Black Hole attack*, peningkatan PDR tertinggi terjadi saat terdapat 1 *node Black Hole* pada uji coba 40 *node*. Nilai PDR yang awalnya pada skenario normal adalah 0.964 turun drastis saat terjadi *Black Hole attack* menjadi 0.229 ketika menggunakan *routing protocol original*. Namun dengan *routing protocol* yang telah dimodifikasi,

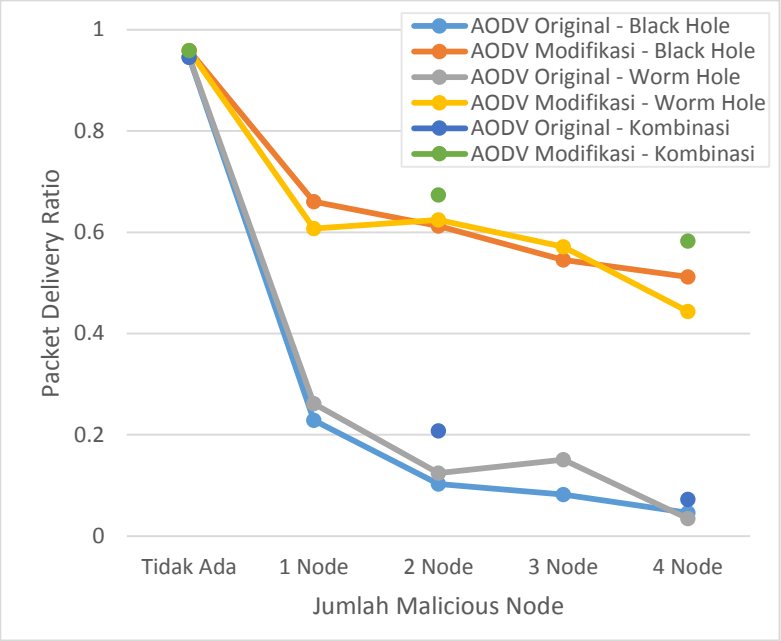
nilai PDR menjadi 0.784 atau dapat dikatakan mengalami peningkatan sebesar 57.62%. Sedangkan peningkatan terendah terjadi saat terdapat 1 *node Black Hole* pada uji coba 30 *node*, yaitu hanya sebesar 20.74%.



Gambar 5.11 Grafik PDR terhadap Banyak *Malicious Node* dan Variasi Serangannya pada Skenario *Real* 40 *Node*

Pada skenario dimana *malicious node* melakukan *Worm Hole attack*, peningkatan PDR tertinggi terjadi saat terdapat 4 *node Worm Hole* pada uji coba 40 *node*. Nilai PDR yang awalnya pada skenario normal adalah 0.964 turun drastis saat terjadi *Worm Hole attack* menjadi 0.016 ketika menggunakan *routing protocol original*. Namun dengan *routing protocol* yang telah dimodifikasi, nilai PDR menjadi 0.6 atau dapat dikatakan mengalami peningkatan sebesar 60.01%. Sedangkan peningkatan terendah

terjadi saat terdapat 1 *node Worm Hole* pada uji coba 30 *node*, yaitu hanya sebesar 22.42%.

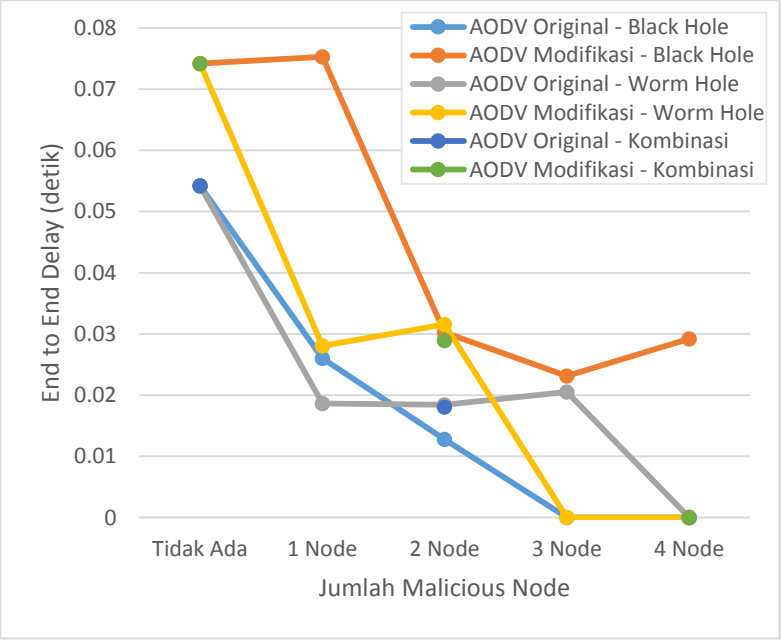


Gambar 5.12 Grafik PDR terhadap Banyak *Malicious Node* dan Variasi Serangannya pada Skenario *Real* 50 *Node*

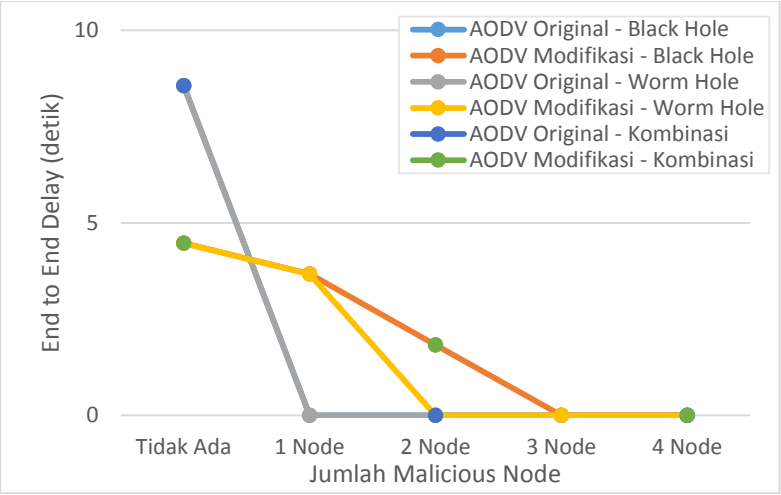
Untuk uji coba dimana *malicious node* melakukan kombinasi antara *Black Hole* dan *Worm Hole attack*, sama seperti skenario *grid*, yaitu hanya dilakukan ketika terdapat 2 *malicious node* dan 4 *malicious node* dengan perbandingan jumlah *malicious node* yang melakukan *Black Hole attack* sama dengan jumlah *malicious node* yang melakukan *Worm Hole attack*. Pada skenario ini, peningkatan PDR tertinggi terjadi saat terdapat 4 *malicious node* dengan rincian 2 *node* melakukan *Black Hole attack* dan 2 *node* yang lain melakukan *Worm Hole attack* pada uji coba 40 *node*. Nilai PDR yang awalnya pada skenario normal adalah 0.964

turun drastis saat terjadi kombinasi kedua serangan tersebut menjadi 0.003 ketika menggunakan *routing protocol original*. Namun dengan *routing protocol* yang telah dimodifikasi, nilai PDR menjadi 0.557 atau dapat dikatakan mengalami peningkatan sebesar 57.49%. Sedangkan peningkatan terendah terjadi saat terdapat 2 *node attacker* pada uji coba 30 *node*, yaitu hanya sebesar 19.73%.

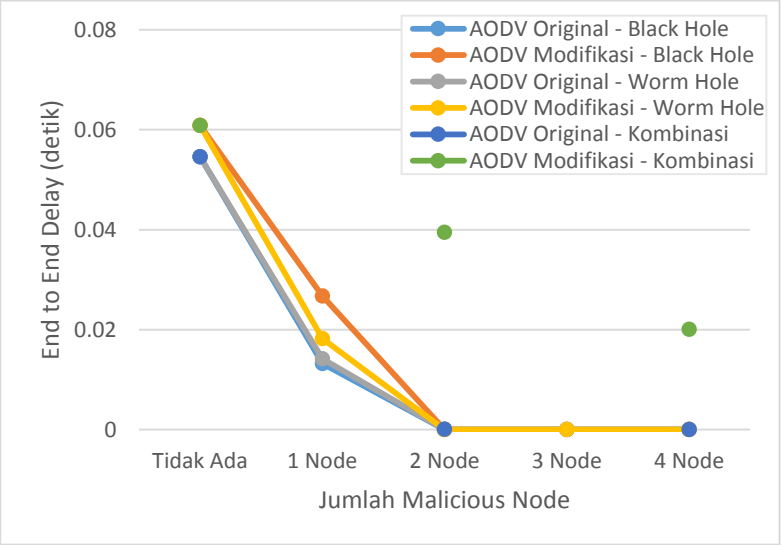
Hasil pengambilan data E2E pada skenario *real* 30, 40, dan 50 *node* dapat dilihat pada Gambar 5.13, Gambar 5.14, dan Gambar 5.15.



Gambar 5.13 Grafik E2E terhadap Banyak *Malicious Node* dan Variasi Serangannya pada Skenario *Real* 30 *Node*



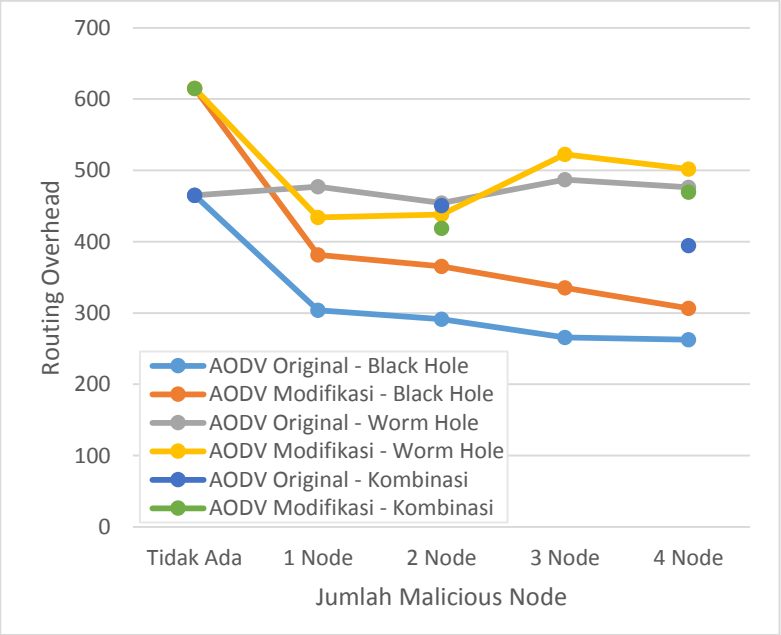
Gambar 5.14 Grafik E2E terhadap Banyak *Malicious Node* dan Variasi Serangannya pada Skenario *Real* 40 *Node*



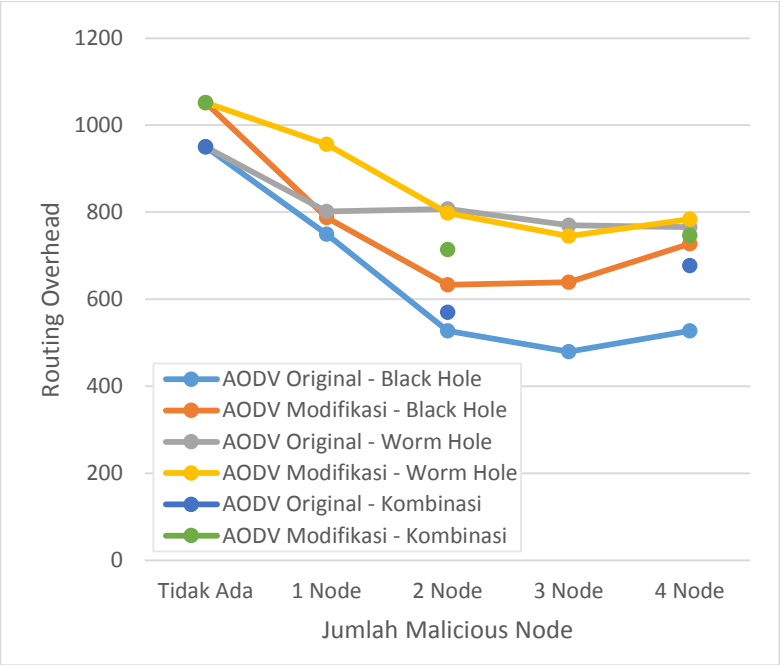
Gambar 5.15 Grafik E2E terhadap Banyak *Malicious Node* dan Variasi Serangannya pada Skenario *Real* 50 *Node*

Berdasarkan grafik pada gambar Gambar 5.13, Gambar 5.14, dan Gambar 5.15 perbandingan rata-rata *end to end delay* antara *routing protocol* AODV yang telah dimodifikasi dengan *routing protocol* AODV original hampir sama dengan ketika diimplementasikan pada skenario *grid*. Namun pada skenario *real* 50 *node*, grafik *rata-rata end to end delay* ketika terjadi serangan terlihat relatif lebih stabil dibandingkan dengan skenario *real* 30 dan 40 *node*.

Hasil pengambilan data RO pada skenario *real* 30, 40, dan 50 *node* dapat dilihat pada Gambar 5.16, Gambar 5.17, dan Gambar 5.18.

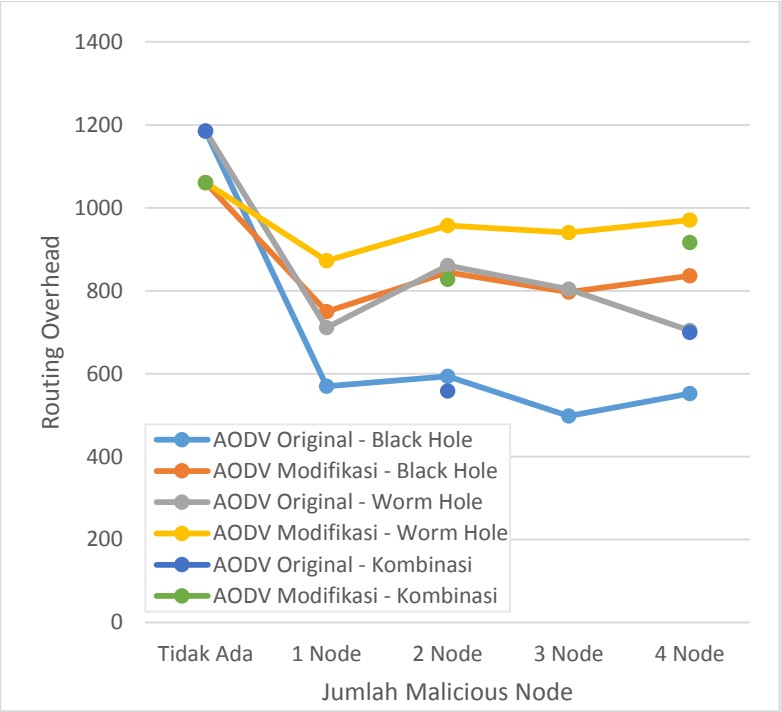


Gambar 5.16 Grafik RO terhadap Banyak *Malicious Node* dan Variasi Serangannya pada Skenario *Real* 30 *Node*



Gambar 5.17 Grafik RO terhadap Banyak *Malicious Node* dan Variasi Serangannya pada Skenario *Real* 40 *Node*

Berdasarkan grafik pada Gambar 5.16, Gambar 5.17, dan Gambar 5.18 perbandingan *routing overhead* antara *routing protocol* AODV yang telah dimodifikasi dengan *routing protocol* AODV *original* hampir sama dengan ketika diimplementasikan pada skenario *grid*. Hal ini menunjukkan bahwa *routing protocol* AODV yang telah dimodifikasi belum mampu untuk dapat mengatasi *routing overhead* ketika terjadi serangan.



Gambar 5.18 Grafik RO terhadap Banyak *Malicious Node* dan Variasi Serangannya pada Skenario *Real* 50 *Node*

Namun seperti pada skenario *grid*, ada saat dimana *routing protocol* dapat meredakan *routing overhead*. Hal ini terjadi di beberapa skenario uji coba dimana *malicious node* melakukan *Worm Hole attack*. Seperti yang ditunjukkan oleh grafik pada Gambar 5.16, ketika terdapat 1 *malicious node* yang melakukan *Worm Hole attack*, nilai *routing overhead* dari *routing protocol original* lebih tinggi dibandingkan dengan nilai *routing overhead* dari *routing protocol* yang telah dimodifikasi.

BAB VI
KESIMPULAN DAN SARAN

Pada bab ini akan diberikan kesimpulan yang diperoleh dari Tugas Akhir yang telah dikerjakan dan saran tentang pengembangan dari Tugas Akhir ini yang dapat dilakukan di masa yang akan datang.

6.1 Kesimpulan

Kesimpulan yang diperoleh dari hasil uji coba dan evaluasi Tugas Akhir ini adalah sebagai berikut:

1. Dengan menambahkan ECDSA pada *packet header* dan melakukan verifikasi serta evaluasi pada tiap *node*, dapat mendeteksi adanya *malicious node* di sekitar *node* tersebut sehingga mampu mengatasi *Black Hole* dan *Worm Hole attack.*
2. Performa *routing protocol* dalam mengatasi serangan ditunjukkan dengan peningkatan PDR rata-rata pada skenario *grid* 30 node, 40 node, dan 50 node sebesar 39.42% untuk *Black Hole attack*, 31.93% untuk *Worm Hole attack*, dan 39.73% untuk kombinasi keduanya.
3. Performa *routing protocol* dalam mengatasi serangan ditunjukkan dengan peningkatan PDR rata-rata pada skenario *real* 30 node, 40 node, dan 50 node sebesar 46.04% untuk *Black Hole attack*, 41.91% untuk *Worm Hole attack*, dan 44.25% untuk kombinasi keduanya.

6.2 Saran

Saran yang diberikan dari hasil uji coba dan evaluasi Tugas Akhir ini adalah sebagai berikut:

1. Untuk mendapatkan hasil uji coba yang lebih *smooth*, dapat dilakukan uji coba lebih banyak, misal 10 kali untuk tiap skenario.

2. Implementasi Elliptic Curve Digital Signature Algorithm dan evaluasi pada tiap *node* dapat dilakukan pada *routing protocol* lain, seperti GPSR, DSR, dsb.
3. Implementasi *digital signature* untuk memvalidasi keaslian paket dapat dilakukan dengan menggunakan algoritma *digital signature* yang lain, seperti DSA atau RSA.
4. Dapat dilakukan penambahan metode guna meningkatkan aspek sekuritas pada *routing protocol* untuk mengatasi variasi *Wormhole attack* yang lain, seperti *Out of Band Wormhole attack*, maupun variasi lain yang dilakukan oleh dua *malicious node* yang bekerja sama.

DAFTAR PUSTAKA

[1] Y. Liu, J. Bi, and J. Yang, "Research on Vehicular Ad Hoc Networks," in *2009 Chinese Control and Decision Conference*, 2009, pp. 4430–4435.

[2] J. Hou, J. Liu, L. Han, and J. Zhao, "Secure and efficient protocol for position-based routing in VANETs," in *2012 IEEE International Conference on Intelligent Control, Automatic Detection and High-End Equipment*, 2012, pp. 142–148.

[3] Y. Wang and F. Li, "Vehicular Ad Hoc Networks," in *Guide to Wireless Ad Hoc Networks*, S. Misra, I. Woungang, and S. C. Misra, Eds. Springer London, 2009, pp. 503–525.

[4] C. Sommer and F. Dressler, *Vehicular Networking*. Cambridge University Press, 2014.

[5] S. Bitam, A. Mellouk, and S. Zeadally, "HyBR: A Hybrid Bio-inspired Bee swarm Routing protocol for safety applications in Vehicular Ad hoc NETworks (VANETs)," *J. Syst. Archit.*, vol. 59, no. 10, Part B, pp. 953–967, Nov. 2013.

[6] J. Jamal, "Routing Protocols in MANETs," *Eexploria*, 18-Sep-2012. .

[7] S. S. Manvi, M. S. Kakkasageri, and D. G. Adiga, "Message Authentication in Vehicular Ad Hoc Networks: ECDSA Based Approach," in *2009 International Conference on Future Computer and Communication*, 2009, pp. 16–20.

[8] Acevpn, "Elliptic Curve IPSEC IKEv2 VPN Service - Next Gen Security," *Acevpn.com*, 03-Jun-2014.

[9] F.-H. Tseng, L.-D. Chou, and H.-C. Chao, "A survey of black hole attacks in wireless mobile ad hoc networks," *Hum.-Centric Comput. Inf. Sci.*, vol. 1, no. 1, p. 4, Nov. 2011.

[10] "ResearchGate," *ResearchGate*. [Online]. Available: https://www.researchgate.net/publication/51956520_Different_types_of_attacks_in_Mobile_ADHOC_Network. [Accessed: 23-May-2017].

[11]"The Network Simulator - ns-2." [Online]. Available: http://www.isi.edu/nsnam/ns/. [Accessed: 13-Apr-2017].
[12]"Sumo_at_a_Glance - SUMO - Simulation of Urban Mobility." [Online]. Available: http://www.sumo.dlr.de/userdoc/Sumo_at_a_Glance.html. [Accessed: 13-Apr-2017].
[13]"OpenStreetMap," *OpenStreetMap*. [Online]. Available: https://www.openstreetmap.org/about. [Accessed: 13-Apr-2017].
[14]"AWK," *Wikipedia*. 29-Apr-2017.
[15]"/index.html." [Online]. Available: https://www.openssl.org/. [Accessed: 13-Apr-2017].
[16]A. Robbins, *Effective Awk Programming: Text Processing and Pattern Matching*. O'Reilly Media, Inc., 2001.

LAMPIRAN

A.1 Kode Skenario NS-2

```
set val(chan)        Channel/WirelessChannel;
set val(prop)        Propagation/TwoRayGround;
set val(netif)       Phy/WirelessPhy;
set val(mac)         Mac/802_11;
set val(ifq)         Queue/DropTail/PriQueue;
set val(ll)          LL;
set val(ant)         Antenna/OmniAntenna;
set val(ifqlen)      50;
set val(nn)          30;
set val(rp)          AODV;
set val(energymodel)    EnergyModel;
set val(initialenergy)  100;
set val(lm)          "off";
set val(x)           1700;
set val(y)           1700;
set val(stop)        200;
set val(cp)          "traffic2";
set val(sc)          "scenario30.tcl";

set ns_ [new Simulator]
set tracefd [open s-aodv230.tr w]
set namtrace [open s-aodv230.nam w]

$ns_ trace-all $tracefd

$ns_ namtrace-all-wireless $namtrace $val(x)
$val(y)

Agent/AODV set num_nodes $val(nn)

set topo [new Topography]
$topo load_flatgrid $val(x) $val(y)
create-god $val(nn)
set chan_1_ [new $val(chan)]
```

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

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

source $val(cp)
source $val(sc)
for {set i 0} {$i < $val(nn)} {incr i} {
$ns_ initial_node_pos $node_($i) 100
}

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

$ns_ at 200.01 "puts \"end simulation\" ;
$ns_ halt"
proc stop {} {
global ns_ tracefd namtrace
$ns_ flush-trace
close $tracefd
close $namtrace
}

$ns_ run
```

A.2 Kode Fungsi AODV::recvHello

```
void AODV::recvHello(Packet *p) {

 struct hdr_aodv_reply *rp=HDR_AODV_REPLY(p);
 AODV_Neighbor *nb;

 nb = nb_lookup(rp->rp_dst);
 if(nb == 0) {
   nb_insert(rp->rp_dst);
 }
 else {
   nb->nb_expire = CURRENT_TIME + (1.5 *
ALLOWED_HELLO_LOSS * HELLO_INTERVAL);
 }

 myneigh[(int)rp->rp_dst].push_back(index);

 Packet::free(p);
}
```

A.3 Kode Fungsi AODV::recvRequest

```
Void AODV::recvRequest(Packet *p) {

 struct hdr_ip *ih = HDR_IP(p);
 struct hdr_aodv_request *rq =
HDR_AODV_REQUEST(p);
 aodv_rt_entry *rt;

 int function_status = -1;
    char string[SHA256_DIGEST_LENGTH];
    strcpy(string,(char *)&rq->rq_type);
    strcat(string,(char *)&rq->rq_bcast_id);
    strcat(string,(char *)&rq->rq_dst);
    strcat(string,(char *)&rq->rq_dst_seqno);
    strcat(string,(char *)&rq->rq_src);
```

```
 strcat(string,(char *)&rq->rq_src_seqno);
 SHA256((unsigned char*)&string,
strlen(string), (unsigned char*)&hash);
 char mdString[SHA256_DIGEST_LENGTH*2+1];

 for(int i = 0; i < SHA256_DIGEST_LENGTH;
i++)
 sprintf(&mdString[i*2], "%02x", (unsigned
int)hash[i]);
 int verify_status =
ECDSA_do_verify(ucstr(mdString),
strlen(mdString), rq->signature, rq->eckey);
 const int verify_success = 1;

 if (verify_success != verify_status){
   function_status = -1;
 }
 else{
   verified[rq->record][index]+=1;
   function_status = 1;
 }

 if(rq->rq_src == index) {
   fprintf(stderr, "%s: got my own
REQUEST\n", __FUNCTION__);
   Packet::free(p);
   return;
 }

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

#ifdef DEBUG
   fprintf(stderr, "%s: discarding
request\n", __FUNCTION__);
#endif // DEBUG
   Packet::free(p);
   return;
 }
```

```
 id_insert(rq->rq_src, rq->rq_bcast_id);
 aodv_rt_entry *rt0;
 rt0 = rtable.rt_lookup(rq->rq_src);
 if(rt0 == 0) {
   rt0 = rtable.rt_add(rq->rq_src);
 }

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

 if ( (rq->rq_src_seqno > rt0->rt_seqno ) ||
      ((rq->rq_src_seqno == rt0->rt_seqno) &&
       (rq->rq_hop_count < rt0->rt_hops)) ) {
   rt_update(rt0, rq->rq_src_seqno, rq-
>rq_hop_count, ih->saddr(),max(rt0-
>rt_expire, (CURRENT_TIME + REV_ROUTE_LIFE))
);
   if (rt0->rt_req_timeout > 0.0) {
     rt0->rt_req_cnt = 0;
     rt0->rt_req_timeout = 0.0;
     rt0->rt_req_last_ttl = rq->rq_hop_count;
     rt0->rt_expire = CURRENT_TIME +
ACTIVE_ROUTE_TIMEOUT;
   }

   assert (rt0->rt_flags == RTF_UP);
   Packet *buffered_pkt;
   while ((buffered_pkt = rqueue.deque(rt0-
>rt_dst))) {
     if (rt0 && (rt0->rt_flags == RTF_UP)) {
       assert(rt0->rt_hops != INFINITY2);
       forward(rt0, buffered_pkt, NO_DELAY);
     }
   }
 }

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

```
 if(rq->rq_dst == index) {
   #ifdef DEBUG
     fprintf(stderr, "%d - %s: destination
sending reply\n",index, __FUNCTION__);
   #endif // DEBUG

   seqno = max(seqno, rq->rq_dst_seqno)+1;
   if (seqno%2) seqno++;
   sendReply(rq->rq_src, 1, index,
seqno, MY_ROUTE_TIMEOUT, rq->rq_timestamp);
   Packet::free(p);
 }

 else if (malicious == true) {
   seqno = max(seqno, rq->rq_dst_seqno)+1;
   if (seqno%2) seqno++;
    sendReply(rq->rq_src, 1, rq->rq_dst,
seqno, MY_ROUTE_TIMEOUT, rq->rq_timestamp);
   Packet::free(p);
 }

else if (rt && (rt->rt_hops != INFINITY2) &&
(rt->rt_seqno >= rq->rq_dst_seqno) ) {
   assert(rq->rq_dst == rt->rt_dst);
   sendReply(rq->rq_src, rt->rt_hops + 1, rq-
>rq_dst, rt->rt_seqno,(u_int32_t) (rt-
>rt_expire - CURRENT_TIME), rq-
>rq_timestamp);
   rt->pc_insert(rt0->rt_nexthop);
   rt0->pc_insert(rt->rt_nexthop);

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

   Packet::free(p);
 }
```

```
 else {
   ih->saddr() = index;
   ih->daddr() = IP_BROADCAST;
   rq->rq_hop_count += 1;

   if(worm == true){
     seqno = max(seqno, rq->rq_dst_seqno)+1;
     if (seqno%2) seqno++;
     sendReply(rq->rq_src, 1, rq->rq_dst,
seqno, MY_ROUTE_TIMEOUT, rq->rq_timestamp);
     rq->rq_dst = 4;
    }

 backward_eval[rq->record][index] =
(double)verified[rq-
>record][index]/(double)kirimdari[rq-
>record][index];
 rq->record = index;
 for(int i=0; i < (int)myneigh[index].size();
i++){
   if((int)rq->rq_dst !=
(int)myneigh[index][i]){
     kirimke[(int)myneigh[index][i]][index]
+= 1;
  forward_eval[(int)myneigh[index][i]][index]
=(double)kirimdari[(int)myneigh[index][i]][in
dex]/(double)kirimke[(int)myneigh[index][i]][
index];
   }

   eval_value[(int)myneigh[index][i]][index]
= (double) rand()/(double) RAND_MAX *
forward_eval[(int)myneigh[index][i]][index] +
(double) rand()/(double) RAND_MAX *
backward_eval[(int)myneigh[index][i]][index];
```

```
if(eval_value[(int)myneigh[index][i]][index]
> 0.7){
  if (rt) rq->rq_dst_seqno = max(rt-
>rt_seqno, rq->rq_dst_seqno);
      }
     }
    }
    forward((aodv_rt_entry*) 0, p, DELAY);
 }
}
```

A.4 Kode Fungsi AODV::sendRequest

```
void
AODV::sendRequest(nsaddr_t dst) {

  Packet *p = Packet::alloc();
  struct hdr_cmn *ch = HDR_CMN(p);
  struct hdr_ip *ih = HDR_IP(p);
  struct hdr_aodv_request *rq =
HDR_AODV_REQUEST(p);
  aodv_rt_entry *rt = rtable.rt_lookup(dst);

  assert(rt);

  if (rt->rt_flags == RTF_UP) {
    assert(rt->rt_hops != INFINITY2);
    Packet::free((Packet *)p);
    return;
  }

  if (rt->rt_req_timeout > CURRENT_TIME) {
    Packet::free((Packet *)p);
    return;
  }
```

```
 if (rt->rt_req_cnt > RREQ_RETRIES) {
   rt->rt_req_timeout = CURRENT_TIME +
MAX_RREQ_TIMEOUT;
   rt->rt_req_cnt = 0;
   Packet *buf_pkt;
   while ((buf_pkt = rqueue.deque(rt-
>rt_dst))) {
     drop(buf_pkt, DROP_RTR_NO_ROUTE);
   }
   Packet::free((Packet *)p);
   return;
 }

 #ifdef DEBUG
   fprintf(stderr, "(%2d) - %2d sending Route
Request, dst: %d\n", ++route_request, index,
rt->rt_dst);
 #endif // DEBUG

 rt->rt_req_last_ttl = max(rt-
>rt_req_last_ttl,rt->rt_last_hop_count);

 if (0 == rt->rt_req_last_ttl) {
   ih->ttl_ = TTL_START;
 }
 else {
   if (rt->rt_req_last_ttl < TTL_THRESHOLD){
     ih->ttl_ = rt->rt_req_last_ttl +
TTL_INCREMENT;
   }
   else {
     ih->ttl_ = NETWORK_DIAMETER;
     rt->rt_req_cnt += 1;
   }
 }
```

```
 rt->rt_req_last_ttl = ih->ttl_;
 rt->rt_req_timeout = 2.0 * (double) ih-
>ttl_ * PerHopTime(rt);
 if (rt->rt_req_cnt > 0)
   rt->rt_req_timeout *= rt->rt_req_cnt;
 rt->rt_req_timeout += CURRENT_TIME;

 if (rt->rt_req_timeout > CURRENT_TIME +
MAX_RREQ_TIMEOUT)
   rt->rt_req_timeout = CURRENT_TIME +
MAX_RREQ_TIMEOUT;
   rt->rt_expire = 0;

#ifdef DEBUG
 fprintf(stderr, "(%2d) - %2d sending Route
Request, dst: %d, tout %f ms\n",
              ++route_request,
             index, rt->rt_dst,
             rt->rt_req_timeout -
CURRENT_TIME);
#endif

 ch->ptype() = PT_AODV;
 ch->size() = IP_HDR_LEN + rq->size();
 ch->iface() = -2;
 ch->error() = 0;
 ch->addr_type() = NS_AF_NONE;
 ch->prev_hop_ = index;

 ih->saddr() = index;
 ih->daddr() = IP_BROADCAST;
 ih->sport() = RT_PORT;
 ih->dport() = RT_PORT;

 rq->rq_type = AODVTYPE_RREQ;
 rq->rq_hop_count = 1;
 rq->rq_bcast_id = bid++;
```

```
 rq->rq_dst = dst;
 rq->rq_dst_seqno = (rt ? rt->rt_seqno : 0);
 rq->rq_src = index;
 seqno += 2;
 assert ((seqno%2) == 0);
 rq->rq_src_seqno = seqno;
 rq->rq_timestamp = CURRENT_TIME;
 rq->record = index;

 char string[SHA256_DIGEST_LENGTH];
 strcpy(string,(char *)&rq->rq_type);
 strcat(string,(char *)&rq->rq_bcast_id);
 strcat(string,(char *)&rq->rq_dst);
 strcat(string,(char *)&rq->rq_dst_seqno);
 strcat(string,(char *)&rq->rq_src);
 strcat(string,(char *)&rq->rq_src_seqno);

 SHA256((unsigned char*)&string,
strlen(string), (unsigned char*)&hash);
 char mdString[SHA256_DIGEST_LENGTH*2+1];
 for(int i = 0; i < SHA256_DIGEST_LENGTH;
i++)
   sprintf(&mdString[i*2], "%02x", (unsigned
int)hash[i]);

 strcpy(rq->mdString,mdString);
 int function_status = -1;
 rq->eckey = EC_KEY_new();
 if (NULL == rq->eckey){
   function_status = -1;
 }
 else{
   EC_GROUP *ecgroup=
EC_GROUP_new_by_curve_name(NID_secp192k1);
   if (NULL == ecgroup){
     function_status = -1;
   }
```

```
 else{
   int set_group_status =
EC_KEY_set_group(rq->eckey,ecgroup);
   const int set_group_success = 1;
   if (set_group_success != set_group_status)
   {
      function_status = -1;
   }
   else{
      const int gen_success = 1;
      int gen_status =
EC_KEY_generate_key(rq->eckey);
      if (gen_success != gen_status){
        function_status = -1;
      }
      else{
        rq->signature =
ECDSA_do_sign(ucstr(mdString),
strlen(mdString), rq->eckey);
        if (NULL == rq->signature){
          function_status = -1;
        }
        else{
          function_status = 1;
        }
       }
      }
      EC_GROUP_free(ecgroup);
      }}
for(int i=0; i < (int)myneigh[index].size();
i++){
    if((int)rq->rq_dst !=
(int)myneigh[index][i]){
      kirimke[(int)myneigh[index][i]][index]
+= 1;
    }}
 Scheduler::instance().schedule(target_, p,
0.);
}
```

A.5 Kode Skrip AWK *Packet Delivery Ratio*

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

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

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

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

END {
        printf "cbr s:%d r:%d, r/s
Ratio:%.4f, f:%d \n", sendLine, recvLine,
(recvLine/sendLine),fowardLine;
}
```

A.6 Kode Skrip AWK Rata-rata *End to End Delay*

```
BEGIN{
      sum_delay = 0;
      count = 0;
}
{
      event = $1;
      type = $4;
      packet_id = $6;
      time = $2;
```

```
count++;

      if(type == "AGT" && event == "s"){
            start[packet_id] = time;
      }
      else if(type == "AGT" && event == "r"){
            end[packet_id] = time;
      }

      if(end[packet_id] != 0){
            delay = end[packet_id] -
start[packet_id];
            sum_delay += delay;
      }

}
END{
      printf("Average end to end delay : %lf
\n",sum_delay/count);
}
```

A.7 Kode Skrip AWK *Routing Overhead*

```
BEGIN {
        ro = 0;
}

$0 ~/^s.* RTR/ {
      if($23 != "(HELLO)"){
            ro++ ;
      }

}

END {
        printf("Routing Overhead : %d\n",ro);
}
```

BIODATA PENULIS

Tities Novaninda Ovari lahir di Surabaya pada 01 November 1995. Penulis menempuh pendidikan formal dimulai dari TK Harapan (2000-2001), SDN Sidotopo Wetan 1 No. 255 Surabaya (2001-2007), SMPN 6 Surabaya (2007-2010), SMAN 2 Surabaya (2010-2013) dan S1 Teknik Informatika ITS (2013-2017). Bidang studi yang diambil oleh penulis pada saat berkuliah di Teknik Informatika ITS adalah Komputasi Berbasis Jaringan (KBJ). Penulis aktif dalam organisasi seperti Himpunan Mahasiswa Teknik Computer-Informatika (2014-2015) dan Keluarga Muslim Informatika (2014-2016). Penulis juga aktif dalam kegiatan kepanitiaan seperti SCHEMATICS 2014 dan SCHEMATICS 2015 divisi Hubungan Masyarakat. Penulis pernah kerja praktik di PT. Yolo periode Mei – Juli 2016. Selama berkuliah, penulis juga bekerja magang di Direktorat Pengembangan Teknologi dan Sistem Informasi ITS periode Agustus 2016 – Agustus 2017 dan menjadi administrator Laboratorium Komputasi Berbasis Jaringan. Penulis dapat dihubungi melalui email: titiesovari@gmail.com.