TUGAS AKHIR – KI141502

STUDI KINERJA PEMILIHAN RUTE AODV BERDASARKAN TINGKAT ENERGI PADA MANETs

AHMAD BILAL
NRP 05111440000121

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

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

Departemen Informatika
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya 2018

(Halaman ini sengaja dikosongkan)

TUGAS AKHIR – KI141502

STUDI KINERJA PEMILIHAN RUTE AODV BERDASARKAN TINGKAT ENERGI PADA MANETs

AHMAD BILAL
NRP 05111440000121

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

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

Departemen Informatika
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya 2018

(Halaman ini sengaja dikosongkan)

UNDERGRADUATE THESES – KI141502

STUDY ON PERFORMANCE OF AODV ROUTE SELECTION BASED ON ENERGY LEVEL IN MANETs

AHMAD BILAL
NRP 05111440000121

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

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

Department of Informatics
Faculty of Information and Communication Technology
Sepuluh Nopember Institute of Technology
Surabaya 2018

(Halaman ini sengaja dikosongkan)

LEMBAR PENGESAHAN

STUDI KINERJA PEMILIHAN RUTE AODV BERDASARKAN TINGKAT ENERGI PADA MANETs

TUGAS AKHIR

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Arsitektur Jaringan Komputer
Program Studi S-1 Departemen Informatika
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember

Oleh:
AHMAD BILAL
NRP 05111440000121

Disetujui oleh Pembimbing Tugas Akhir:

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

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

SURABAYA
JULI, 2018

(Halaman ini sengaja dikosongkan)

STUDI KINERJA PEMILIHAN RUTE AODV BERDASARKAN TINGKAT ENERGI PADA MANETs

Nama Mahasiswa	**: AHMAD BILAL**
NRP	**: 05111440000121**
Departemen	**: Informatika FTIK-ITS**
Dosen Pembimbing 1	**: Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.**
Dosen Pembimbing 2	**: Ir. F.X. Arunanto, M.Sc.**

Abstrak

Mobile Ad Hoc Network (MANET) merupakan sebuah jaringan yang terdiri dari perangkat-perangkat bergerak dan terhubung secara nirkabel yang dapat secara terus-menerus melakukan konfigurasi diri tanpa memiliki infrastruktur jaringan. *Node-node* pada MANET memiliki karakteristik dengan mobilitas tinggi. Terdapat beberapa protokol routing yang dapat diimplementasikan pada MANET, salah satunya adalah *Adhoc On-Demand Distance Vector* (AODV).

Dalam MANET, setiap *node* yang termasuk dalam jaringan dapat berperan sebagai *router*. Dikarenakan *node-node* yang terlibat seringkali bergerak (*mobile*), maka baterai digunakan sebagai sumber daya listrik. Apabila terdapat salah satu *node* yang keluar dari jaringan, dimana *node* tersebut merupakan rute transmisi, maka transmisi data gagal dan membutuhkan pembuatan rute baru. Salah satu penyebab keluarnya *node* dari suatu jaringan adalah karena daya listrik baterai pada *node* tersebut habis.

Untuk mengurangi kemungkinan gagalnya transmisi akibat energi pada *node* habis, maka diperlukan sebuah metode untuk memilih rute transmisi berdasarkan tingkat energinya. Salah satu metode yang dapat digunakan yaitu dengan menghitung total energi *node-node* untuk setiap rute yang dilalui paket *request* dan memilih rute dengan total energi paling tinggi.

Berdasarkan hasil uji coba modifikasi menggunakan metode yang telah disebutkan, *packet delivery ratio* algoritma modifikasi unggul untuk jumlah *node* 10 dan 20. Untuk *routing overhead*, algoritma AODV modifikasi selalu lebih unggul dari AODV asli, yang ditandai dengan jumlah *routing overhead* yang selalu lebih kecil. Namun, untuk rata-rata *end-to-end delay* AODV masih mengungguli.

Kata kunci: AODV, *Energy,* MANETs

STUDY ON PERFORMANCE OF AODV ROUTE SELECTION BASED ON ENERGY LEVEL IN MANETS

Nama Mahasiswa	**: AHMAD BILAL**
NRP	**: 05111440000121**
Departemen	**: Informatika FTIK-ITS**
Dosen Pembimbing 1	**: Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.**
Dosen Pembimbing 2	**: Ir. F.X. Arunanto, M.Sc.**

Abstract

Mobile Ad Hoc Network (MANET) is a network of wirelessly connected devices that can constantly configure themselves without having a network infrastructure. MANET nodes have a certain characteristic which is high mobility. There are several routing protocols that can be implemented on MANET, one of which is Adhoc On-Demand Distance Vector (AODV).

In MANET, any node included in the network can act as a router. Because the nodes involved are often mobile, the battery is used as a power source. If there is one node coming out of the network, where the node is the transmission route, the data transmission fails. One of the causes that a node of a network is out of the network is because the node has no battery power left.

To reduce the possibility of transmission failure because of insufficient energy of node, a method is needed to select the transmission route based on its energy level. One method that can be used is to calculate the total energy of the nodes for each route through which the request packet and choose the route with the highest total energy.

Based on the modification test results using the mentioned methods, the packet delivery ratio of modified algorithm is superior to the number of nodes 10 and 20. For routing overheads, the AODV algorithm modifications are always superior to the

original AODV, which is indicated by the number of routing overheads which is always smaller. However, for the average end-to-end delay AODV still outperforms the modified algorithm.

Keyword: **AODV,** ***Energy,*** **MANETs**

KATA PENGANTAR

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

Puji syukur kepada Allah Yang Maha Esa atas segala karunia dan rahmat-Nya penulis dapat menyelesaikan tugas akhir yang berjudul

"STUDI KINERJA PEMILIHAN RUTE AODV BERDASARKAN TINGKAT ENERGI PADA MANETs".

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

1. Allah SWT.
2. Keluarga penulis yang selalu memberikan dukungan baik berupa doa, moral, dan material kepada penulis, sehingga penulis dapat menyelesaikan Tugas Akhir ini.
3. Bapak Dr.Eng. Radityo Anggoro, S.Kom., M.Sc. dan Bapak Ir. F.X. Arunanto, M.Sc.. selaku dosen pembimbing penulis yang telah membimbing, memberikan nasihat, dan memotivasi penulis sehingga penulis dapat menyelesaikan Tugas Akhir ini.
4. Bapak Dr.Eng. Darlis Herumurti, S.Kom., M.Kom. selaku kepala Departemen Informatika ITS.
5. Teman-teman angkatan 2014.
6. Teman-teman kontrakan BME F-119 (Zainul Rahmawan, Akbar Nadzif, Angga Putra Pratama, Syamsul Bahri, Darari Nur Amali, M. Arif Fachruddin, Shofi Al Ghozi R, Firsta Agung S, Rama Kurniawan, Abyan Faris Putranto dan Dhimas Abdi Prayogo).

Penulis telah berusaha sebaik-baiknya dalam menyusun tugas akhir ini, namun penulis mohon maaf apabila terdapat kekurangan, kesalahan maupun kelalaian yang telah penulis

lakukan. Kritik dan saran yang membangun dapat disampaikan sebagai bahan perbaikan selanjutnya.

Surabaya, 5 Juni 2018

Ahmad Bilal

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I
PENDAHULUAN

1.1 Latar Belakang

MANET merupakan sebuah teknologi yang memungkinkan sebuah jaringan dapat terbentuk tanpa adanya infrastruktur sentral. Dalam MANET, setiap *node* yang termasuk dalam jaringan dapat berperan sebagai *router*. Dikarenakan *node-node* yang terlibat seringkali bergerak (*mobile*), maka baterai digunakan sebagai sumber daya listrik. Apabila terdapat salah satu *node* yang keluar dari jaringan, dimana *node* tersebut merupakan rute transmisi, maka transmisi data gagal. Salah satu penyebab keluarnya *node* dari suatu jaringan adalah karena daya listrik baterai pada *node* tersebut habis.

Untuk mengurangi kemungkinan gagalnya transmisi akibat tingkat energi pada *node* habis, maka diperlukan sebuah metode untuk memilih rute transmisi yang tingkat energinya terbilang cukup. Salah satu metode yang dapat digunakan yaitu dengan menghitung total energi *node-node* untuk setiap rute yang dilalui paket *request* dan memilih rute dengan total energi paling tinggi.

1.2 Rumusan Masalah

Berikut adalah rumusan masalah yang akan diselesaikan pada tugas akhir ini:

1. Bagaimana pengaruh pemilihan rute berdasarkan tingkat energi terhadap *packet delivery ratio* ?
2. Bagaimana pengaruh pemilihan rute berdasarkan tingkat energi terhadap *routing overhead* ?
3. Bagaimana pengaruh pemilihan rute berdasarkan tingkat energi terhadap *end-to-end delay* ?

1.3 Batasan Permasalahan

Batasan masalah dari tugas akhir adalah sebagai berikut:

- Jaringan yang akan digunakan merupakan jaringan *Mobile Ad hoc Network* (MANET)
- Algoritma routing yang digunakan berbasis dari algoritma AODV
- Simulasi dari algoritma akan dilakukan pada aplikasi NS-2

1.4 Tujuan

Tujuan pembuatan Tugas Akhir ini adalah untuk mengetahui kinerja dari algoritma AODV modifikasi pada MANET dengan memilih rute dengan tingkat energi paling tinggi.

1.5 Manfaat

Dengan dibuatnya Tugas Akhir ini dapat memberikan informasi tentang kinerja pemilihan rute berdasarkan tingkat energi pada *node*.

1.6 Metodologi

Pembuatan Tugas Akhir ini dilakukan dengan menggunakan metodologi sebagai berikut:

1.6.1 Penyusunan Proposal Tugas Akhir

Proposal tugas akhir ini berisi tentang sebuah metode yang yang akan digunakan pada algoritma *routing Mobile Ad Hoc Network* (MANET) dengan memodifikasi algoritma *routing* AODV berdasarkan tingkat energi pada *node*.

1.6.2 Studi Literatur

Untuk menunjang pemahaman tentang metode yang diajukan dan bagaimana implementasinya, dilakukan studi dari beberapa literatur berikut:

- *Compromising AODV for Better Performance*, Praneeth Paranavithana *and* Anuradha Jayakody
- Panduan Penggunaan *Network Simulator* 2.
- Jaringan *ad hoc*
- MANET
- *Trace file*
- AWK

1.6.3 Implementasi Sistem

Untuk implementasi metode, berikut adalah kakas bantu yang akan digunakan:

• *Network Simulator* 2

Network Simulator 2 akan digunakan untuk melakukan simulasi algoritma routing AODV yang telah dimodifikasi.

• TCL Script

TCL *Script* merupakan Bahasa Skrip yang akan digunakan pada NS-2.

• C++

C++ merupakan Bahasa pemrograman yang digunakan dalam implementasi pada NS-2.

1.6.4 Pengujian dan Evaluasi

Untuk simulasi *node-node* pada MANET, akan digunakan *Network Simulator* 2 (NS-2). Kemudian akan dilakukan 2 pengujian, pertama menggunakan algoritma AODV dan yang kedua adalah menggunakan algoritma AODV yang telah

dimodifikasi. Metrik yang akan diuji adalah *packet delivery ratio, routing overhead* dan rata-rata e*nd-to-end delay*.

1.6.5 Penyusunan Buku

Pada tahap ini dilakukan penyusunan laporan yang menjelaskan dasar teori dan metode yang digunakan dalam tugas akhir ini serta hasil dari implementasi aplikasi perangkat lunak yang telah dibuat.

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 atau penjelasan dari metode, algoritma dan *tools* yang digunakan dalam penyusunan Tugas Akhir ini. Bab ini berisi tentang penjelasan singkat mengenai Jaringan *Ad Hoc*, MANET, AODV, NS-2 *Trace File*, dan AWK.
3. Bab III. Perancangan
   Bab ini berisi pembahasan mengenai perancangan skenario simulasi pada NS-2, perancangan modifikasi AODV, serta perancangan metrik analisis (*packet delivery ratio, end-to-end delay, routing overhead*)
4. Bab IV. Implementasi
   Bab ini menjelaskan implementasi yang berbentuk kode sumber dari proses modifikasi protokol AODV, melakukan simulasi menggunakan NS-2 dan perhitungan metrik analisis.
5. Bab V. Pengujian dan Evaluasi

Bab ini berisikan hasil uji coba dan evaluasi dari implementasi modifikasi pada *routing protocol* AODV yang telah dilakukan untuk menyelesaikan masalah yang dibahas pada Tugas Akhir.

6. Bab VI. Kesimpulan dan Saran
   Bab ini merupakan bab yang menyampaikan kesimpulan dari hasil uji coba yang dilakukan 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 sumber program secara keseluruhan.

(Halaman ini sengaja dikosongkan

BAB II
TINJAUAN PUSTAKA

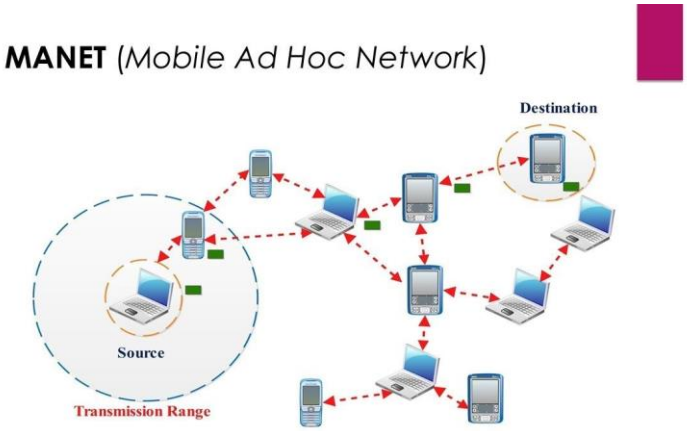
Bab ini berisi pembahasan mengenai teori-teori dasar atau penjelasan dari metode dan *tools* yang digunakan dalam Tugas Akhir. Penjelasan ini bertujuan untuk memberikan gambaran secara umum sebagai penunjang dalam pengembangan riset yang berkaitan.

2.1 Jaringan *Ad Hoc*

Jaringan *ad hoc* adalah sebuah jaringan yang terdiri dari perangkat-perangkat individu yang berkomunikasi satu sama lain secara langsung. Banyak dari jaringan *ad hoc* merupakan jaringan area lokal (*Local Area Network*) dimana komputer-komputer atau perangkat lainnya dapat saling melakukan pengiriman data secara langsung tanpa melalui titik akses yang tersentralisasi [1].

2.2 MANET

Mobile Ad Hoc Network (MANET) merupakan sebuah jaringan teknologi komunikasi dimana tidak ada infrastruktur sentral pada jaringan tersebut. Tidak adanya topologi yang tetap dalam operasi dalam MANET artinya jaringan tersebut merupakan jaringan yang dapat mengatur diri sendiri dan mengatasi masalah-masalah sendiri. Pada lingkungan MANET, setiap *node* dapat beroperasi sebagai *router* atau *node* terminal. Oleh karena itu, matinya suatu *node* dapat mengganggu performa jaringan secara keseluruhan [2]. Ilustrasi jaringan MANET dapat dilihat pada gambar 2.2.



Gambar 2.1 Ilustrasi MANET
Sumber: Zakaria, Aznida & Yazid Mohd Saman, Md & Mohd Noor, Ahmad Shukri & Hassan, Hasni. (2015). Finding shortest routing solution in mobile AD HOC networks using firefly algorithm and queuing network analysis. Jurnal Teknologi. 77. 10.11113/jt.v77.6485.

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

Ad hoc On-Demand Distance Vector (AODV) merupakan sebuah algoritma routing yang bertujuan untuk digunakan pada *node-node* bergerak dalam sebuah jaringan *ad hoc*. AODV menawarkan adaptasi yang cepat dalam kondisi *link* yang dinamis, *overhead* proses dan memori yang kecil, tingkat penggunaan jaringan yang kecil dan dapat menentukan rute-rute *unicast* menuju *node* tujuan dalam sebuah jaringan *ad hoc* [3].

Pencarian rute (*route discovery process*) dimulai ketika sebuah *node* sumber membutuhkan untuk berkomunikasi dengan *node* lain dimana *node* sumber tersebut tidak memiliki informasi *routing* menuju *node* tujuan pada tabel routingnya. Setiap *node* memiliki dua *counter*, yaitu *node sequence number* dan *broadcast_id*. *Node* sumber menginisiasi pencarian rute dengan mengirimkan paket *route request* (RREQ) menuju *node* tetangga. Isi dari RREQ diantaranya adalah alamat pengirim dan *broadcast_id*. Kedua kombinasi tersebut

membuat setiap paket *route request* menjadi unik. *Broadcast_id* akan ditambah ketika akan dilakukan pengiriman RREQ baru. Ketika menerima paket RREQ, *node* tetangga dapat mengirimkan paket *route reply* atau melanjutkan pengiriman RREQ ke tetangga *node* tersebut (*rebroadcast*). Sebuah *node* dapat menerima RREQ yang sama dari tentagga yang berbeda. Ketika hal tersebut terjadi, maka paket RREQ yang diterima tadi akan di-*drop* oleh *node* penerima dan tidak me-*broadcast* ulang paket RREQ tersebut. Ketika sebuah *node* bukan merupakan tujuan dan tidak tahu rute menuju suatu *node*, *node* akan menyimpan informasi *reverse path* yang nantinya akan digunakan oleh RREP untuk menuju *node* pengirim.

Pada saat tertentu, sebuah paket RREQ tentunya akan sampai di *node* tujuan atau *node* yang memiliki rute menuju tujuan. *Node* yang memiliki rute menuju tujuan pertama akan membandingkan *sequence number* pada RREQ dengan yang ada pada tabel *routing*-nya. Jika *sequence number* pada RREQ lebih besar dari yang ada pada *routing table node* tersebut, maka *node* tersebut akan me-*broadcast* kembali paket RREQ tersebut. Namun jika hal tersebut tidak terjadi, maka *node* penerima akan mengirimkan paket RREP menuju *node* sumber menggunakan *reverse path* yang sudah disiapkan oleh setiap *node* yang dilalui oleh RREQ. Ketika RREP melewati *reverse path*, setiap *node* yang dilewati menyiapkan sebuah rute bernama *forward path* yang nantinya akan digunakan untuk transmisi data. Ilustrasi *reverse route* dan *forward route* dapat dilihat pada gambar 2.3.

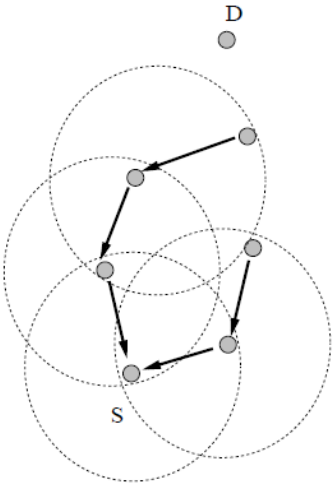


Figure 1. Reverse Path Formation

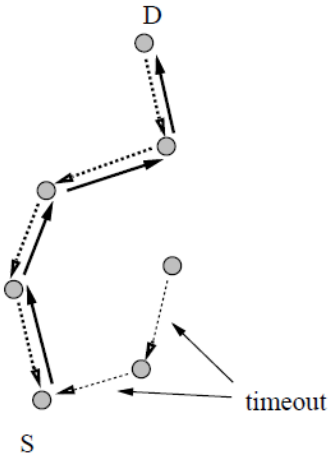


Figure 2. Forward Path Formation

Gambar 2.2 Ilustrasi reverse route dan forward route
Sumber: E. M. R. Charles E. Perkins, "Ad-hoc On-Demand Distance Vector Routing".

Pada AODV, ketika suatu *node* mendapatkan informasi baru tentang rute untuk menuju suatu *node*, maka *node* tersebut akan membandingkan informasi rute tersebut dengan rute yang ada pada tabel *routing*-nya. Jika sequence number dari rute baru lebih besar, maka rute lama pada tabel akan diganti dengan rute baru. Jika *sequence number* dari rute sama, maka akan dipilih rute dengan jumlah *hop* yang paling kecil [4].

2.4 *Network Simulator 2* (NS-2)

NS merupakan *simulator* yang berfokus pada penelitian tentang jaringan komputer. NS dapat melakukan simulasi TCP,

routing dan protokol-protokol *multicast* pada sebuah jaringan kabel dan nirkabel (lokal dan satelit) [5].

Dalam *Network Simulator* 2, terdapat banyak protokol *routing* yang telah disediakan. Diantara protokol-protokol tersebut adalah AODV, AOMDV, DSDV dan DSR. Dengan adanya protokol yang telah ada, maka untuk mengimplementasikan sebuah modifikasi dari sebuah protokol, maka yang perlu dilakukan adalah mengubah kode sumber dari protokol yang dimaksud sesuai dengan apa yang diinginkan. Agar suatu protokol yang telah diubah dapat digunakan untuk simulasi, diperlukan kompilasi ulang dari *Network Simulator* 2.

File-file yang dibutuhkan untuk simulasi adalah *file* TCL, dimana *file* tersebut berisi konfigurasi lingkungan beserta bagaimana *node-node* dalam sebuah jaringan diinisiasi posisinya maupun pergerakannya.

2.5 *Trace File*

Trace file merupakan *file* hasil simulasi yang dilakukan oleh NS-2 dan berisikan informasi detail pengiriman paket data. *Trace file* berisi data yang dapat digunakan dalam analisis performa *routing protocol* yang disimulasikan.

Berikut adalah contoh dari *trace file* untuk AODV :

s 4.500000000 _8_ RTR --- 0 AODV 52 [0 0 0 0] [energy 49.999419 ei 0.000 es 0.000 et 0.001 er 0.000] ------- [8:255 -1:255 30 0] [0x2 0 2 [9 0] [8 6]] (REQUEST)

Dan berikut adalah penjelasan untuk masing-masing *output*:

- *Node* "_8_" mengirim paket pada detik ke "4.500000000".
- *Level trace* berada pada layer "RTR".
- Paket memiliki ID "0" dan berisi tipe *payload* "AODV" dan memiliki ukuran sebesar 52 *byte*.
- Ketika pengiriman, *node* memiliki energi sebesar 49.999419 joule

2.6 AWK

AWK merupakan Bahasa yang terinterpretasi *(interpreted language*). Artinya, AWK membaca sebuah program dan memproses data berdasarkan instruksi pada program. Hal ini berbeda dengan Bahasa terkompilasi (*compiled language*) seperti C, dimana program terlebih dahulu di-*compile* menjadi kode mesin yang akan dieksekusi secara langsung oleh prosesor sistem [6]. Pada tugas akhir ini AWK akan digunakan untuk menganalisis *trace file* hasil simulasi.

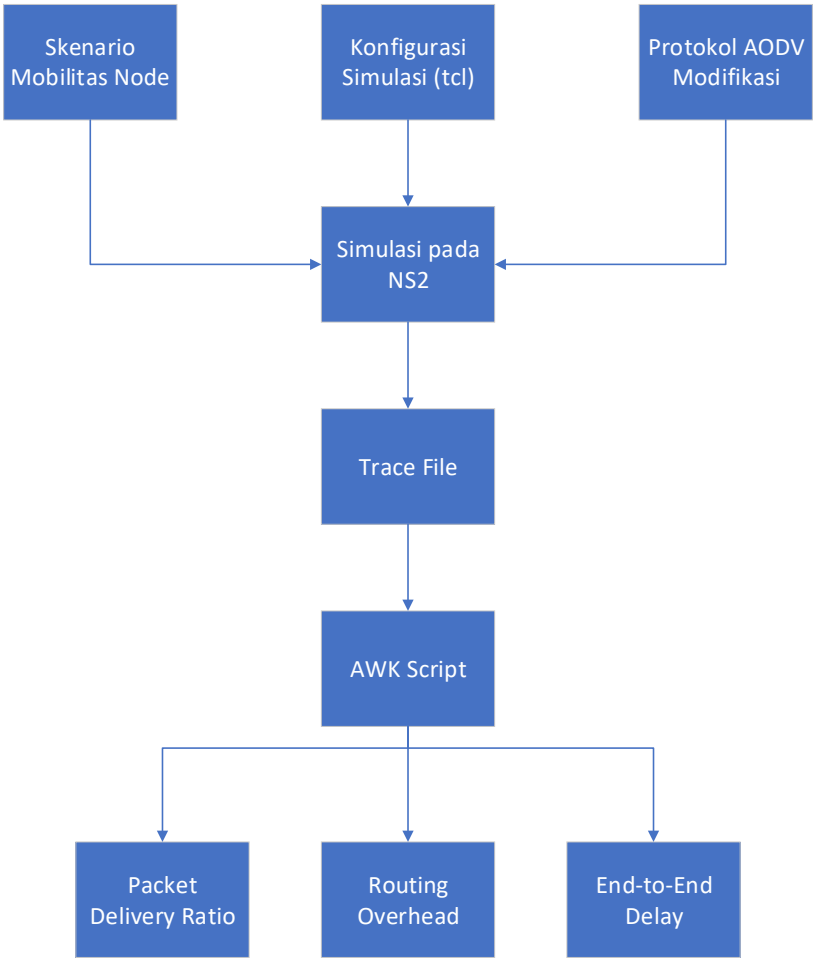
BAB III
PERANCANGAN

Bab ini membahas mengenai perancangan implementasi 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 Deskripsi Umum

Tugas akhir ini berisi tentang metode routing pada jaringan MANET dengan memodifikasi algoritma AODV. Metode yang diajukan berfokus kepada pemilihan rute transmisi dengan tingkat energi tertinggi.

Modifikasi protokol *routing* AODV pada tugas akhir ini akan disimulasikan menggunakan *Network Simulator* 2. Untuk mendapatkan mobilitas dari *node-node*, digunakan fitur *setdest* dari NS-2. Kemudian hasil dari simulasi berupa *trace file* akan dianalisis menggunakan AWK *script* untuk mendapatkan *packet delivery ratio*, *routing overhead* dan rata-rata *end-to-end delay*. Alur dari simulasi dapat dilihat pada gambar 3.1.



Gambar 3.1 Alur simulasi

Hal pertama yang perlu dilakukan adalah menyiapkan *file-file* yang dibutuhkan untuk simulasi. *File-file* tersebut adalah :

- Skenario mobilitas *node*

Untuk membuat skenario mobilitas *node* diperlukan fitur *setdest* pada NS-2. Fitur *setdest* pada NS-2 dapat digunakan untuk membuat pergerakan *node* secara *random*.

- Konfigurasi simulasi
  Konfigurasi simulasi merupakan *file* TCL yang berisi konfigurasi jaringan. Konfigurasi tersebut berisi diantaranya tipe kanal, protokol *routing* yang digunakan dan jumlah *node* yang terlibat.
- Protokol AODV modifikasi
  Protokol AODV modifikasi merupakan hasil pengubahan dari kode sumber yang telah disediakan NS-2. Kode sumber dimodifikasi sesuai dengan algoritma AODV yang diajukan dalam Tugas Akhir ini.

Setelah semua *file* sudah dipersiapkan, maka dilakukan simulasi menggunakan *Network Simulator* 2. Hasil dari simulasi adalah *trace file* yang berisi informasi-informasi yang terjadi pada saat simulasi seperti pengiriman paket dan waktu pengiriman paket tersebut. Dari *trace file*, penulis dapat menganalisis *packet delivery ratio, routing overhead* dan rata-rata *end-to-end delay* menggunakan AWK *script*.

3.2 Analisis dan Perancangan Modifikasi *Routing Protocol* AODV

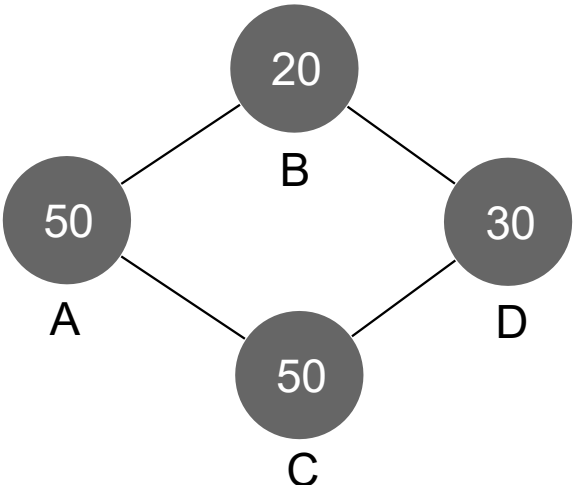
Pada tugas akhir ini, dilakukan modifikasi pada protokol AODV tepatnya pada pemilihan rute yang dilakukan oleh *node* sumber berdasarkan jumlah energi pada *node-node* di suatu jalur transmisi.

Untuk mewujudkan metode tersebut, diperlukan 3 perubahan pada algoritma AODV :

- Menambahkan *field* baru pada paket RREQ dan RREP untuk menyimpan nilai energi
- Mengimplementasikan sebuah fungsi untuk menghitung energi pada suatu *node*

- Menambahkan vektor energi untuk pemilihan jalur terbaik pada transmisi paket

Berikut adalah contoh dari bagaimana alur modifikasi bekerja.



Gambar 3.2 Ilustrasi node

Diasumsikan bahwa *node* A (gambar 3.2) ingin berkomunikasi dengan *node* D. Berikut adalah proses yang terjadi pada *node* A:

```
If node_A_want_to_communicate_with_D
{
 If A_has_a_path_to_D
           Start_sending_packets();
     else
     {
           Create RREQ_packet()
           Calculate_residual_energy();
           Insert_value_to_RREQ_packet();
           Forward_the_packet_to_neightbor
     s();
     }
}
```

Gambar 3.3 Pseudocode node pengirim

- Jika *node* asal A memiliki rute ke *node* tujuan D, maka *node* A dapat langsung mengirim paket ke *node* D.
- Jika tidak, maka *node* A membuat paket RREQ, menghitung energi di *node* tersebut lalu memasukkan nilai energi tersebut ke paket RREQ dan mengirimkannya ke *node* tetangga.

Node B dan C adalah tetangga (*neighbors*) yang akan menerima RREQ dari *node* A. *Node-node* tersebut akan meneruskan paket ke *node* D. Berikut adalah proses yang terjadi pada *node* B dan C:

```
If I_am_not_the_destination
{
      If has_a_path_to_reach_node_D
      {
      Calculate_residual_energy();
      Add_that_value_to_the_received_energy_
value_in_RREQ();
      Forward_the_path_to_the_next_hop_();
}
else
      send_RERR_to_source_node_A_();
}
```

Gambar 3.4 Pseudocode intermediate node

- Jika *node* yang menerima RREQ bukan tujuan dan memiliki rute menuju *node* D, maka *node* tersebut menghitung energinya dan menambahkannya ke nilai energi yang terdapat pada RREQ. Kemudian meneruskan paket RREQ ke *hop* selanjutnya.
- Jika tidak ada rute ke *node* tujuan (*node* D) maka *node* mengirim paket RERR kembali ke *node* sumber (*node* A).

Kemudian, paket RREQ akan tiba di *node* tujuan D. Berikut adalah proses yang akan terjadi di *node* D:

```
If I_am_the_destination
{
      Calculate_the_residual_energy();
      Add_that_value_from_RREQ_and_insert_to
_RREP();
      Create_the_RREP_packet();
      Copy_energy_value_from_RREQ_and_insert
_to_RREP();
      Send_the_RREP_back_to_source_node_A_()
;
}
```

Gambar 3.5 Pseudocode node tujuan

- Jika suatu *node* adalah *node* tujuan, maka *node* tersebut akan menghitung energinya dan menambahkan nilai energi tersebut dengan nilai energi pada paket RREQ. Kemudian *node* membuat paket RREP dan memasukkan nilai energi yang telah ditambahkan tadi ke RREP dan mengirim paket tersebut ke *node* asal (*node* A).

Sekarang *node* A akan menerima dua paket RREP dari *node* B, D dan C, D. Berikut adalah proses yang terjadi pada *node* A:

```
If multiple_paths_exist_to_reach_node_D
{
      Select_the_path_with_highest_residual_
energy_();
      Start_sending_packets();
}
Else {
      Select_the_only_path_that_exist();
      Start_sending_packets_();
}
```

Gambar 3.6 Pseudocode penerimaan RREP

- Jika terdapat lebih dari satu rute menuju tujuan, maka rute yang akan dipilih adalah rute dengan nilai energi tertinggi.

3.3 Perancangan Simulasi pada NS-2

Simulasi pada NS-2 dilakukan dengan menggabungkan *script* TCL *scenario* yang berisi konfigurasi *node* dan mobilitas *node*. Mobilitas *node* didapat dengan menggunakan fitur *setdest* pada NS-2. Konfigurasi lingkungan simulasi dapat dilihat pada table 3.1.

No.	**Parameter**	**Spesifikasi**
1.	*Network Simulator*	NS-2, 2.35
2.	*Routing Protocol*	AODV
3.	Waktu Simulasi	1000
4.	Area Simulasi	600 x 600 m
5.	Banyak *node*	10, 20, 30, 40, 50
6.	Energi awal	50 Joule
7.	Agen Pengirim	*Constant Bit Rate* (CBR)
8.	*Protocol* MAC	IEEE 802.11
9.	Propagasi sinyal	*Two-ray ground*
10.	*Source/Destination*	Statis
11.	Tipe Kanal	*Wireless Channel*

Tabel 3.3.1 Konfigurasi lingkungan simulasi

Berikut adalah skenario simulasi yang akan dilakukan :

- Simulasi dilakukan 5 kali untuk masing-masing menggunakan AODV asli dan AODV modifikasi
- Dari hasil simulasi dihasilkan 5 *trace file* yang nantinya akan dianalisis menggunakan AWK *script* dan dihitung rata-rata masing-masing metrik.

3.4 Perancangan Metrik Analisis

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

3.4.1 *Packet Delivery Ratio* (PDR)

Packet Delivery Ratio (PDR) merupakan perbandingan antara berapa jumlah paket yang diterima oleh *node* tujuan dengan jumlah paket yang dikirimkan oleh *node* sumber. PDR dihitung dengan persamaan 3.1.

$$PDR = \frac{received}{sent} x\ 100\ \% \qquad (3.1)$$

3.4.2 Rata-Rata *End-to-End Delay*

Rata-rata e*nd-to-end delay* merupakan rata-rata dari *delay* atau waktu yang dibutuhkan tiap paket untuk sampai ke *node* tujuan dalam satuan milidetik (ms). *Delay* tiap paket didapatkan dari selisih waktu pengiriman (`CBRSentTime`) dan waktu terima (`CBRRecvTime`). Kemudian dilakukan penjumlahan untuk semua selisih dan dibagi dengan jumlah paket yang berhasil diterima *node* tujuan (`recvnum`), maka akan didapatkan rata-rata *end-to-end delay*. Penghitungan rata-rata delay dapat dilihat pada persamaan 3.2.

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

3.4.3 *Routing Overhead* (RO)

Routing overhead adalah jumlah paket kontrol *routing* yang ditransmisikan selama simulasi terjadi. *Routing Overhead* didapatkan dengan cara menjumlahkan semua paket kontrol *routing* yang ditransmisikan. Paket kontrol tersebut diantaranya adalah *route request* (RREQ), *route reply* (RREP) dan *route error* (RERR).

BAB IV
IMPLEMENTASI

Pada bab ini akan dibahas mengenai implementasi dari perancangan yang sudah dilakukan pada bab sebelumnya. Implementasi berupa kode sumber untuk membangun program.

4.1 Implementasi Modifikasi pada *Routing Protocol* AODV

Protokol *routing* AODV pada dasarnya memilih rute berdasarkan rute tercepat menuju tujuan. Artinya, *node* sumber hanya akan menerima satu balasan RREP dari *node* tujuan dan rute itulah yang nantinya akan digunakan untuk transmisi data.

Modifikasi akan dilakukan pada AODV sehingga *node* tujuan dapat membalas RREQ yang diterima selama total energi yang ada pada RREQ melebihi energi yang ada pada tabel routing *node* tujuan sehingga nantinya *node* sumber dapat memperbarui rute pada *routing table* dengan rute yang baru, yaitu rute dengan jumlah energi tertinggi.

Untuk melakukan modifikasi tersebut, diperlukan modifikasi *node* sumber untuk *routing protocol* AODV. Berikut adalah perubahan yang dilakukan pada *source code*:

- Membuat fungsi untuk mendapatkan nilai energi pada suatu *node*

  Fungsi ini diperlukan untuk mendapatkan nilai energi pada suatu *node*. Pemanggilan fungsi akan menghasilkan variabel berupa tingkat energi pada *node* tersebut. Nilai energi tersebut nantinya dapat digunakan untuk ditambahkan pada RREQ maupun RREP untuk menambahkan total energi pada jalur tertentu. Penambahan fungsi dilakukan pada file aomdv.cc yang dapat dilihat pada lampiran A.7 serta penambahan pada file aomdv.h yang dapat dilihat pada lampiran A.4. Gambar 4.1 adalah kode fungsi yang ditambahkan pada *source code*.

```
double AOMDV::get_energy(nsaddr_t addr)
{
  Node* thisnode =
Node::get_node_by_address(addr);
  double energy ;
  energy = thisnode->energy_model()->energy();
  return energy;
}
```

Gambar 4.1 Fungsi energi pada node

- Menambahkan variabel energi total pada paket RREQ dan RREP

  Agar nilai energi total dapat terkirimkan, diperlukan variabel baru pada paket RREQ dan RREP. Penambahan variabel tersebut ditambahkan pada file aomdv_packet.h yang dapat dilihat pada lampiran A.6. Berikut adalah tambahan variabel pada paket :

```
struct hdr_aomdv {
      . . .
      double          rq_totalEnergy;
}
```

Gambar 4.2 Variabel energi total RREQ

```
struct hdr_aomdv_reply {
      . . .
      double          rp_totalEnergy;
}
```

Gambar 4.3 Variabel energi total RREP

Penambahan energi total pada RREQ dan RREP akan dilakukan ketika sebuah *node* akan me-*broadcast* sebuah RREQ ataupun RREP. Untuk mencapai hal tersebut, diperlukan

pengubahan pada *file* aomdv.cc yang dapat dilihat pada lampiran A.7.

- Memilih rute berdasarkan total energi

  Node sumber perlu untuk memilih rute berdasarkan total energi suatu jalur. Rute yang akan dipilih adalah rute yang memiliki total energi paling tinggi. Untuk itu, setiap kali *node* sumber menerima RREP dari *node* tujuan, *node* sumber akan memperbarui tabel *routing* dengan rute yang baru. Untuk mencapai hal tersebut diperlukan pengubahan pada pemilihan rute pada aomdv.cc yang dapat dilihat pada lampiran A.7. Selain itu, juga diperlukan adanya penambahan variabel yang menyimpan energi total pada tabel *routing* yang ditambahkan pada *file* aomdv_rtable.h yang dapat dilihat pada lampiran A.5. Perubahan yang dilakukan pada *source code* untuk memilih rute dengan total energi paling tinggi dapat dilihat pada gambar 4.4. Potongan penambahan variabel energi total pada tabel *routing* dapat dilihat pada gambar 4.5.

```
if ((rt->rt_seqno < rp->rp_dst_seqno) ||
((rt->rt_seqno == rp->rp_dst_seqno) && (rt-
>rt_totalEnergy < rp->rp_totalEnergy)))
```

Gambar 4.4 Pemilihan rute berdasarkan total energi

```
class aomdv_rt_entry {
      ...
      double           rt_totalEnergy;
}
```

Gambar 4.5 Variabel total energi pada tabel routing

4.2 Implementasi Simulasi pada NS-2

Implementasi simulasi pada NS-2 diawali dengan konfigurasi lingkungan simulasi menggunakan *file* TCL.

```
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)     10
set val(rp)     AODV
set val(x)      600
set val(y)      600
set val(stop)   1000.0
set val(cp)     "cbr-10-test-1"
set val(sc)     "scen-10-test-1"
```

Gambar 4.6 Konfigurasi lingkungan simulasi

Simulasi pada NS-2 akan dilakukan selama 1000 detik dengan luas lingkungan sebesar 600 x 600 meter. *File* yang dibutuhkan untuk menjalankan simulasi adalah *file traffic* dan *file* mobilitas *node* yang dapat dilihat pada gambar 4.6 dan pemanggilannya yang dapat dilihat pada gambar 4.7. Kode *traffic* dapat dilihat pada lampiran A.2.

```
#
# Define node movement model
#
puts "Loading connection pattern..."
source $val(cp)

#
# Define traffic model
#
puts "Loading scenario file..."
source $val(sc)
```

Gambar 4.7 Pemuatan file raffic dan mobilitas

```
$ns_ node-config -adhocRouting  $val(rp) \
                 -llType        $val(ll) \
                 -macType       $val(mac) \
                 -ifqType       $val(ifq) \
                 -ifqLen        $val(ifqlen)
\
                 -antType       $val(ant) \
                 -propType      $val(prop) \
                 -phyType       $val(netif)
\
                 -channel       $chan \
                 -topoInstance  $topo \
                 -agentTrace    ON \
                 -routerTrace   ON \
                 -macTrace      ON \
                 -movementTrace ON \
                 -energyModel "EnergyModel"
\
                 -initialEnergy 50.0
```

Gambar 4.8 Konfigurasi node

Diperlukan penggunaan *energyModel* pada konfigurasi agar energi pada *node* dapat diakses. Setiap *node* pada simulasi memiliki energi sebesar 50 joule. Konfigurasi *node* untuk simulasi dapat dilihat pada gambar 4.8 dan kode skenario NS-2 dapat dilihat pada lampiran A.1.

4.3 Implementasi Metrik Analisis

Dari hasil simulasi menggunakan NS-2, dihasilkan sebuah file berupa *trace file* yang berisikan segala informasi yang terjadi ketika simulasi berjalan. Dari *file* tersebut dapat diambil data berupa *packet delivery ratio*, *routing overhead* dan *end-to-end delay*. Kode sumber untuk implementasi dapat dilihat pada lampiran A.3.

4.3.1 Implementasi *Packet Delivery Ratio*

Untuk mengetahui PDR dari sebuah simulasi, diperlukan dua data yaitu jumlah paket yang dikirimkan dan jumlah paket yang diterima. Untuk mendapatkan informasi terbsebut dari *trace file* hasil simulasi, diperlukan pencocokan kata kunci tertentu. Berikut adalah cara untuk mendapatkan jumlah paket yang dikirimkan dan jumlah paket yang diterima dari *trace file*.

```
# CALCULATE PACKET DELIVERY FRACTION
  if (( $1 == "s") &&  ( $7 == "cbr" ) && (
$4=="AGT" )) {  sends++; }

  if (( $1 == "r") &&  ( $7 == "cbr" ) && (
$4=="AGT" ))   {  recvs++; }
```

Gambar 4.9 Implementasi data yang dikirim dan diterima

Untuk mendapatkan jumlah paket yang dikirim, diperlukan untuk me-*filter output* pertama (*output* pada *trace file* dipisahkan dengan spasi) berupa karakter “s”, output ke 7 berupa “cbr” dan output ke 4 berupa “AGT”. Untuk mendapatkan jumlah paket yang

diterima sama dengan cara di atas hanya saja *output* pertama berupa "r".

Setelah mendapatkan data tersebut maka dilakukan perbandingan antara jumlah paket yang diterima dan jumlah paket yang dikirim. Implementasi penghitungan PDR dapat dilihat pada gambar 4.10.

```
PDR = (recvs/sends)*100
```

Gambar 4.10 Implementasi penghitungan PDR

4.3.2 Implementasi Rata-Rata *End-to-End Delay*

Untuk mendapatkan rata rata *end-to-end delay*, data yang diperlukan dari *trace file* adalah waktu kapan paket dikirimkan dan waktu ketika paket diterima oleh *node* tujuan. Implementasi dari pengambilan data tersebut adalah sebagai berikut

```
if ( start_time[packet_id] == 0 )
     start_time[packet_id] = time;

if (( $1 == "r") &&  ( $7 == "cbr" ) && (
$4=="AGT" ))
     {  end_time[packet_id] = time;  }

else
     {  end_time[packet_id] = -1;  }
```

Gambar 4.11 Implementasi pengambilan data delay

Setelah didapatkan data tersebut, diperlukan penghitungan rata-rata *delay*. Untuk itu, data yang diperlukan adalah total waktu *delay* yang didapatkan dari menghitung selisih antara waktu terima dan waktu kirim, serta jumlah paket yang diterima. Berikut adalah implementasi pengambilan data tersebut dalam AWK s*cript*.

```
for ( i in end_time ){
      start = start_time[i];
      end = end_time[i];
      packet_duration = end - start;
      if ( packet_duration > 0 )
      {
            sum += packet_duration;
            recvnum++;
      }
}
```

Gambar 4.12 Implementasi penghitungan rata-rata E2E delay

4.3.3 Implementasi *Routing Overhead*

Untuk mendapatkan *routing overhead*, diperlukan penghitungan seluruh paket *routing* baik itu RREQ, RREP maupun RERR. Jumlah tersebut didapat dari *trace file* dengan menggunakan skrip AWK pada gambar 4.13.

```
if (($1 == "s" || $1 == "f") && $4 == "RTR"
&& ($7 =="AODV" || $7 =="AOMDV"))
      routing_packets++;
```

Gambar 4.13 Implementasi penghitungan routing overhead

BAB V
UJI COBA DAN EVALUASI

Pada bab ini akan dilakukan uji coba dan evaluasi sesuai dengan apa yang sudah dirancang dan diimplementasikan. Dari hasil uji coba, akan dilakukan evaluasi sehingga dapat ditarik kesimpulan pada bab selanjutnya.

5.1 Lingkungan Uji Coba

Uji coba dilakukan pada perangkat dengan spesifikasi seperti yang tertera pada **Error! Reference source not found.**.

Komponen	**Spesifikasi**
CPU	Intel(R) Core (TM) i7-6700HQ CPU @ 2.60GHz 2.59 GHz
Sistem Operasi	Ubuntu 16.04.3 LTS
Linux Kernel	4.4.0-43-Microsoft
Memori	16.0 GB
Penyimpanan	1000 GB

Tabel 5.1 Spesifikasi perangkat simulasi

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*.

5.2 Hasil Uji Coba

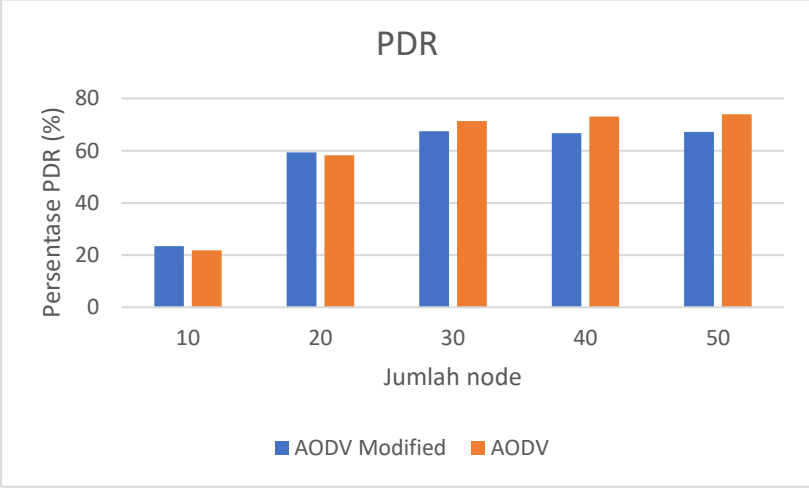
Setelah dilakukan uji coba untuk masing-masing algoritma selama 5 kali, berikut adalah hasil untuk masing-masing metrik:

5.2.1 *Packet Delivery Ratio*

Setelah dilakukan uji coba, data yang didapatkan direpresentasikan dalam table 5.2 dan grafik yang dapat dilihat pada gambar 5.1.

Protokol / Jumlah *Node*	**10**	**20**	**30**	**40**	**50**
AODV Modified	23.408	59.322	67.418	66.674	67.188
AODV	21.862	58.204	71.384	73.086	73.898

Tabel 5.2 Hasil uji coba (PDR)



Gambar 5.1 Grafik perbandingan PDR

Untuk uji coba dengan 10 *node*, dapat dilihat bahwa AODV modifikasi sedikit lebih unggul dari AODV asli. Berdasarkan tabel, AODV modifikasi memiliki keunggulan sebesar 7.071631%

dibandingkan AODV asli. AODV modifikasi juga unggul pada hasil uji coba menggunakan 20 *node*, yaitu memiliki PDR 1.92083% lebih tinggi dari AODV asli. Namun, PDR AODV modifikasi untuk uji coba pada 30 *node* lebih rendah dari AODV asli, yaitu sebesar 5.55587% di bawah AODV asli. Begitu juga dengan hasil uji coba untuk *node* berjumlah 40 dan 50, AODV modifikasi memiliki PDR lebih rendah, yakni 8.77323% dan 9.08008% secara berurutan.

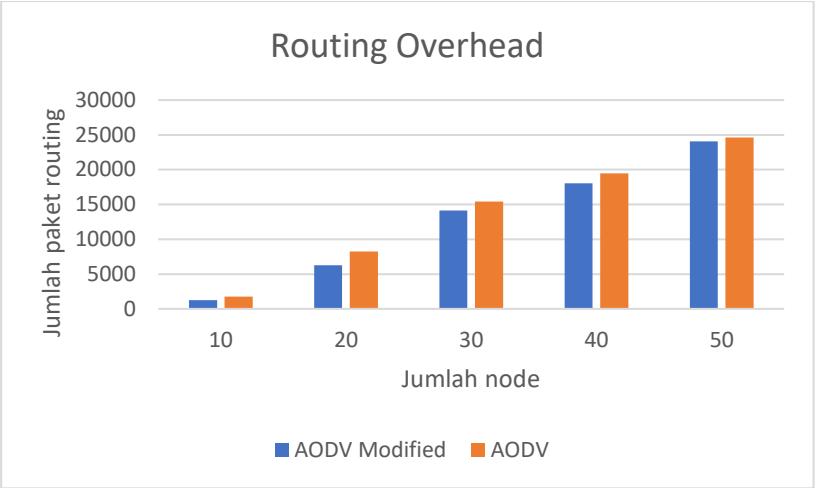
Dapat dilihat bahwa PDR dari AODV yang dimodifikasi memiliki keunggulan ketika jumlah *node* yang ada pada jaringan berjumlah 10 dan 20 *node*. Namun, untuk *node* yang berjumlah 30, 40 dan 50, PDR protokol *routing* AODV asli melebihi dari AODV modifikasi. Dengan modifikasi AODV, dapat terjadi suatu skenario dimana rute yang dipilih adalah rute dengan jumlah *node* yang lebih banyak daripada ketika menggunakan AODV asli. Dikarenakan semakin banyaknya jumlah *node* yang terlibat, maka kemungkinan sebuah paket dapat di-*drop* di suatu *node* semakin besar. Oleh karena itulah PDR dari AODV modifikasi lebih buruk pada 30, 40 dan 50 node.

5.2.2 *Routing Overhead*

Data perbandingan hasil uji coba untuk mendapatkan *routing overhead* dapat dilihat pada table 5.3 dan direpresentasikan dalam bentuk grafik yang dapat diamati pada gambar 5.2.

Protokol / Jumlah *Node*	**10**	**20**	**30**	**40**	**50**
AODV Modified	1259.2	6260.8	14129.8	18036.6	24066.2
AODV	1779.4	8237.8	15402.2	19485.4	24635.6

Tabel 5.3 Hasil uji coba (Routing Overhead)



Gambar 5.2 Grafik perbandingan routing overhead

Untuk uji coba dengan jumlah *node* 10, AODV modifikasi memiliki *routing overhead* yang 29.23457% lebih kecil dari AODV asli. Untuk hasil uji coba pada *node* dengan jumlah 20, 30, 40 dan 50 AODV modifikasi juga memiliki *routing overhead* yang selalu lebih kecil dari AODV asli, yaitu sebesar 23.99913%, 8.261157%, 7.435311% dan 2.311289% secara berurutan.

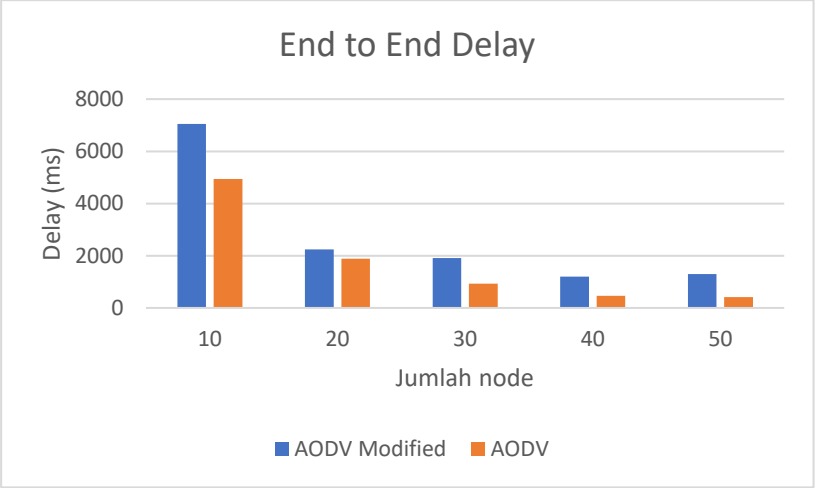
Berdasarkan hasil uji coba tersebut, dapat dilihat bahwa jumlah paket *routing* yang dikirimkan dalam jaringan untuk algoritma AODV modifikasi selalu lebih kecil dari AODV asli. Itu artinya AODV modifikasi membutuhkan lebih sedikit mengirimkan paket *control routing* seperti RREQ, RREP maupun RERR daripada AODV asli.

5.2.3 Rata-Rata *End-to-End Delay*

Hasil dari penghitungan rata-rata *end-to-end delay* dapat dilihat pada table 5.5 dan direpresentasikan dengan grafik yang dapat diamati pada gambar 5.3.

Protokol / Jumlah *Node*	**10**	**20**	**30**	**40**	**50**
AODV Modified	7045.764	2246.172	1915.856	1196.562	1302.538
AODV	4935.262	1890.17	935.418	463.918	408.836

Tabel 5.4 Hasil uji coba (E2E delay)



Gambar 5.3 Grafik perbandingan E2E delay

Untuk uji coba dengan 10 *node*, AODV modifikasi memiliki rata-rata *end-to-end delay* 42.7637% lebih tinggi dari AODV modifikasi. Untuk *node* 20, 30, 40 dan 50, AODV modifikasi juga sama-sama memiliki *delay* yang lebih tinggi dari AODV asli, yaitu sebesar 18.8344%, 104.813%, 157.925%, 218.597% secara berurutan.

Berdasarkan grafik hasil uji coba, dapat dilihat bahwa *delay* dari AODV modifikasi selalu lebih tinggi daripada AODV asli. Hal ini disebabkan karena rute yang dipilih oleh AODV modifikasi

bukanlah sebuah rute yang tercepat, maka dari itu membutuhkan waktu yang lebih lama agar paket dapat mencapai *node* tujuan.

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:

1. Dengan mengubah pemilihan rute AODV berdasarkan total energi rute akan meningkatkan PDR untuk jumlah *node* yang relatif kecil dan pertumbuhan PDR akan semakin menurun ketika jumlah *node* semakin bertambah.
2. AODV hasil modifikasi membutuhkan lebih sedikit pengiriman paket kontrol untuk *routing* daripada AODV asli berdasarkan jumlah *routing overhead* yang ditransmisikan pada simulasi.
3. AODV hasil modifikasi memiliki rata-rata *end-to-end delay* yang lebih tinggi daripada AODV asli.

6.2 Saran

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

1. Uji coba dilakukan lebih dari 5 kali untuk masing-masing skenario.
2. Menambahkan batas energi paling rendah yang dimiliki sebuah *node* untuk me-*forward* paket *request*.
3. Menambahkan batas maksimal jumlah *node* yang terlibat dalam suatu rute.

(Halaman ini sengaja dikosongkan)

DAFTAR PUSTAKA

[1] "Techopedia," 1 1 2018. [Online]. Available: https://www.techopedia.com/definition/5868/ad-hoc-network.

[2] J. Anuradha dan P. Praneeth, "Compromising AODV for Better Performance Improve Energy Efficiency in AODV," *National Conference on Technology and Management (NCTM),* p. 4, 2017.

[3] "rfc3561," 1 1 2018. [Online]. Available: https://www.ietf.org/rfc/rfc3561.txt.

[4] E. M. R. Charles E. Perkins, "Ad-hoc On-Demand Distance Vector Routing".

[5] "ns," 1 1 2018. [Online]. Available: https://www.isi.edu/nsnam/ns/.

[6] GNU, "Executable awk Programs," [Online]. Available: https://www.gnu.org/software/gawk/manual/html_node/Executable-Scripts.html. [Diakses 2018].

(Halaman ini sengaja dikosongkan)

LAMPIRAN

A.1 Kode Skenario NS-2

```
set val(chan)   Channel/WirelessChannel    ;#
channel type
set val(prop)   Propagation/TwoRayGround   ;#
radio-propagation model
set val(netif)  Phy/WirelessPhy            ;#
network interface type
set val(mac)    Mac/802_11                 ;#
MAC type
set val(ifq)    Queue/DropTail/PriQueue    ;#
interface queue type
set val(ll)     LL                         ;#
link layer type
set val(ant)    Antenna/OmniAntenna        ;#
antenna model
set val(ifqlen) 50                         ;#
max packet in ifq
set val(nn)     10                         ;#
number of mobilenodes
set val(rp)     AOMDV                       ;#
routing protocol
set val(x)      600                      ;# X
dimension of topography
set val(y)      600                      ;# Y
dimension of topography
set val(stop)   1000.0
;# time of simulation end
set val(cp)     "cbr-10-test-1"
set val(sc)     "scen-10-test-1"

#Create a ns simulator
set ns_ [new Simulator]
```

```
#Setup topography object
set topo       [new Topography]
$topo load_flatgrid $val(x) $val(y)
# create-god $val(nn)
set god_ [create-god $val(nn)]

set tracefile [open out-10-1.tr w]
$ns_ trace-all $tracefile

set namfile [open out-10-1.nam w]
$ns_ namtrace-all $namfile
$ns_ namtrace-all-wireless $namfile
$val(x) $val(y)
set chan [new $val(chan)];#Create wireless
channel

$ns_ node-config -adhocRouting  $val(rp) \
                 -llType        $val(ll) \
                 -macType       $val(mac) \
                 -ifqType       $val(ifq) \
                 -ifqLen
$val(ifqlen) \
                 -antType       $val(ant) \
                 -propType      $val(prop)
\
                 -phyType       $val(netif)
\
                 -channel       $chan \
                 -topoInstance  $topo \
                 -agentTrace    ON \
                 -routerTrace   ON \
                 -macTrace      ON \
                 -movementTrace ON \
                 -energyModel "EnergyModel"
\
                 -initialEnergy 50.0
```

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

#
# Define node movement model
#
puts "Loading connection pattern..."
source $val(cp)

#
# Define traffic model
#
puts "Loading scenario file..."
source $val(sc)

#===================================
#        Termination
#===================================
#Define a 'finish' procedure
proc finish {} {
    global ns_ tracefile namfile
    $ns_ flush-trace
    close $tracefile
    close $namfile
    # exec nam out.nam &
    exit 0
}
```

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

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

$ns_ at $val(stop) "$ns_ nam-end-wireless
$val(stop)"
$ns_ at $val(stop) "finish"
$ns_ at $val(stop) "puts \"done\" ; $ns_
halt"
$ns_ run
```

A.2 Kode Konfigurasi *Traffic*

```
#
# nodes: 10, max conn: 1, send rate: 1.0,
seed: 1
#
#
# 1 connecting to 2 at time
2.5568388786897245
#
set udp_(0) [new Agent/UDP]
$ns_ attach-agent $node_(8) $udp_(0)
set null_(0) [new Agent/Null]
$ns_ attach-agent $node_(9) $null_(0)
set cbr_(0) [new Application/Traffic/CBR]
$cbr_(0) set packetSize_ 512
$cbr_(0) set interval_ 1.0
$cbr_(0) set random_ 1
$cbr_(0) set maxpkts_ 10000
$cbr_(0) attach-agent $udp_(0)
$ns_ connect $udp_(0) $null_(0)
$ns_ at 2.5568388786897245 "$cbr_(0) start"
#
#Total sources/connections: 1/1
#
```

A.3 Kode Skrip AWK

```
BEGIN {
       sends=0;
       recvs=0;
       routing_packets=0.0;
       droppedBytes=0;
       droppedPackets=0;
       highest_packet_id =0;
       sum=0;
       recvnum=0;
     }

  {
  time = $2;
  packet_id = $6;

  # CALCULATE PACKET DELIVERY FRACTION
  if (( $1 == "s") &&  ( $7 == "cbr" ) && (
$4=="AGT" )) {  sends++; }

  if (( $1 == "r") &&  ( $7 == "cbr" ) && (
$4=="AGT" ))   {  recvs++; }

  # CALCULATE DELAY
  if ( start_time[packet_id] == 0 )
start_time[packet_id] = time;
  if (( $1 == "r") &&  ( $7 == "cbr" ) && (
$4=="AGT" )) {  end_time[packet_id] = time;
}
       else {  end_time[packet_id] = -1;  }

  # CALCULATE TOTAL AODV OVERHEAD
  if (($1 == "s" || $1 == "f") && $4 == "RTR"
&& ($7 =="AODV" || $7 =="AOMDV"))
routing_packets++;
}
```

```
END {

  for ( i in end_time )
  {
  start = start_time[i];
  end = end_time[i];
  packet_duration = end - start;
  if ( packet_duration > 0 )
  {    sum += packet_duration;
       recvnum++;
  }
  }

     delay=sum/recvnum;
     PDF = (recvs/sends)*100;  #packet delivery
ratio[fraction]
     printf("Routing Packets =
%.2f\n",routing_packets++);
     printf("Packet Delivery Function =
%.2f\n",PDF);
     printf("Average end to end delay(ms)=
%.2f\n",delay*1000);
  }
```

A.4 Kode aomdv.h

```
#define AOMDV_PACKET_SALVAGING
#define AOMDV_MAX_SALVAGE_COUNT  10
#define AOMDV_EXPANDING_RING_SEARCH
#define AOMDV_NODE_DISJOINT_PATHS
#define AOMDV_LOCAL_REPAIR
#define AOMDV_LINK_LAYER_DETECTION
```

```
#define AOMDV_USE_LL_METRIC
#define AOMDV_USE_GOD_FEEDBACK


class AOMDV;

#define MY_ROUTE_TIMEOUT        10
// 100 seconds
#define ACTIVE_ROUTE_TIMEOUT    10          //
50 seconds
#define REV_ROUTE_LIFE          6            //
5  seconds
// AODV ns-2.31 code
#define BCAST_ID_SAVE           6            //
3 seconds


// No. of times to do network-wide search before
timing out for
// MAX_RREQ_TIMEOUT sec.
#define RREQ_RETRIES            3
// timeout after doing network-wide search
RREQ_RETRIES times
#define MAX_RREQ_TIMEOUT   10.0 //sec

/* Various constants used for the expanding ring
search */
// AOMDV code
#ifdef AOMDV_EXPANDING_RING_SEARCH
#define TTL_START             5           // 5
#define TTL_INCREMENT         2           // 2
#else // NO EXPANDING RING SEARCH
#define TTL_START             30
#define TTL_INCREMENT         30
#endif // NO EXPANDING RING SEARCH
#define TTL_THRESHOLD 7

// Should be set by the user using best guess
(conservative)
#define NETWORK_DIAMETER        30
```

```
...

void            sendReply(nsaddr_t ipdst,
u_int32_t hop_count,
                                  nsaddr_t
rpdst, u_int32_t rpseq,
                                  double
lifetime, double timestamp,
              nsaddr_t nexthop, u_int32_t
bcast_id, nsaddr_t rp_first_hop, double
rp_totalEnergy);
        void            sendError(Packet *p,
bool jitter = true);

        /*
         * Packet RX Routines
         */
        void            recvAOMDV(Packet *p);
        void            recvHello(Packet *p);
        void            recvRequest(Packet *p);
        void            recvReply(Packet *p);
        void            recvError(Packet *p);

        double          get_energy(nsaddr_t
addr);

...
```

A.5 Kode aomdv_rtable.h

```
class aomdv_rt_entry {
        friend class aomdv_rtable;
        friend class AOMDV;
      friend class LocalRepairTimer;
 public:
        aomdv_rt_entry();
        ~aomdv_rt_entry();

        void            nb_insert(nsaddr_t id);
        AOMDV_Neighbor*  nb_lookup(nsaddr_t id);

 ...

// AOMDV code
        aomdv_paths      rt_path_list;     //
list of paths
        u_int32_t       rt_highest_seqno_heard;
        int             rt_num_paths_;
      bool rt_error;

      /* list of precursors */
        aomdv_precursors rt_pclist;
        double          rt_expire;
      // when entry expires
        double          rt_totalEnergy;
```

A.6 Kode aomdv_packet.h

```
struct hdr_aomdv_request {
        u_int8_t        rq_type;    // Packet
Type
        u_int8_t        reserved[2];
        u_int8_t        rq_hop_count;   // Hop
Count
        u_int32_t       rq_bcast_id;    //
Broadcast ID

        nsaddr_t        rq_dst;         //
Destination IP Address
        u_int32_t       rq_dst_seqno;   //
Destination Sequence Number
        nsaddr_t        rq_src;         //
Source IP Address
        u_int32_t       rq_src_seqno;   //
Source Sequence Number

        double          rq_timestamp;   // when
REQUEST sent;
                              // used to compute
route discovery latency
// AOMDV code
        nsaddr_t        rq_first_hop;  // First
Hop taken by the RREQ
        double          rq_totalEnergy;
...
```

```
...

struct hdr_aomdv_reply {
        u_int8_t        rp_type;        //
Packet Type
        u_int8_t        reserved[2];
        u_int8_t        rp_hop_count;
// Hop Count
        nsaddr_t        rp_dst;
// Destination IP Address
        u_int32_t       rp_dst_seqno;
// Destination Sequence Number
        nsaddr_t        rp_src;
// Source IP Address
        double             rp_lifetime;
// Lifetime

        double          rp_timestamp;
// when corresponding REQ sent;
                                        // used to
compute route discovery latency
// AOMDV code
        u_int32_t       rp_bcast_id;
// Broadcast ID of the corresponding RREQ
        nsaddr_t        rp_first_hop;
        double          rp_totalEnergy;

...
```

A.7 Kode aomdv.cc

```
...
#include <aomdv/aomdv.h>
#include <aomdv/aomdv_packet.h>
#include <random.h>
#include <cmu-trace.h>
#include <energy-model.h>
...
void
AOMDV::recvRequest(Packet *p) {
double temp = rq->rq_totalEnergy +
get_energy(index);
            sendReply(rq->rq_src,
// IP Destination
                                  0,
// Hop Count
                                  index,
// (RREQ) Dest IP Address
                                  seqno,
// Dest Sequence Num

MY_ROUTE_TIMEOUT,         // Lifetime
rq->rq_timestamp,         // timestamp
                                  ih->saddr(),
// nexthop
                                  rq-
>rq_bcast_id,          // broadcast id to
identify this route discovery
ih->saddr(),
temp);
...
rq->rq_totalEnergy += get_energy(index);
...
forward((aomdv_rt_entry*) 0, p,
AOMDV_DELAY);
```

```
void
AOMDV::recvReply(Packet *p) {
...

if ((rt->rt_seqno < rp->rp_dst_seqno) ||
((rt->rt_seqno == rp->rp_dst_seqno) && (rt-
>rt_totalEnergy < rp->rp_totalEnergy))) {
            // printf("%d : Update route -
current = %f - new = %f\n", index, rt-
>rt_totalEnergy, rp->rp_totalEnergy);
            rt->rt_seqno = rp-
>rp_dst_seqno;
            rt->rt_advertised_hops =
INFINITY;
            rt->path_delete();
            rt->rt_flags = RTF_UP;
            /* Insert forward path to RREQ
destination. */
            forward_path = rt-
>path_insert(rp->rp_src, rp-
>rp_hop_count+1, CURRENT_TIME + rp-
>rp_lifetime, rp->rp_first_hop);
            // CHANGE
            rt->rt_last_hop_count = rt-
>path_get_max_hopcount();
            rt->rt_totalEnergy = rp-
>rp_totalEnergy;
            // CHANGE
   }

...
```

```
void
AOMDV::sendRequest(nsaddr_t dst) {
...
rq->rq_type = AOMDVTYPE_RREQ;
      // AOMDV code
      rq->rq_hop_count = 0;
      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->rq_totalEnergy =
get_energy(index);

...

void
AOMDV::sendReply(nsaddr_t ipdst, u_int32_t
hop_count, nsaddr_t rpdst,
                                  u_int32_t
rpseq, double lifetime, double timestamp,
                                  nsaddr_t
nexthop, u_int32_t bcast_id, nsaddr_t
rp_first_hop, double rp_totalEnergy) {

...
```

```
rp->rp_type = AOMDVTYPE_RREP;
      //rp->rp_flags = 0x00;
      rp->rp_hop_count = hop_count;
      rp->rp_dst = rpdst;
      rp->rp_dst_seqno = rpseq;
      rp->rp_src = index;
      rp->rp_lifetime = lifetime;
      rp->rp_timestamp = timestamp;
   rp->rp_bcast_id = bcast_id;

rp->rp_first_hop = rp_first_hop;
   rp->rp_totalEnergy = rp_totalEnergy;

...

double AOMDV::get_energy(nsaddr_t addr) //
edited by me
{
  Node* thisnode =
Node::get_node_by_address(addr);
  double energy ;
  energy = thisnode->energy_model()-
>energy();
  //printf("energy of node %d at time %f is
%f\n",ra_addr_,CURRENT_TIME,energy);
  return energy;
}
```

BIODATA PENULIS

Ahmad Bilal lahir di Lumajang pada 29 September 1995. Penulis menempuh pendidikan formal SDN Tempeh Tengah 01 dan SDN Tompokersan 01, SMPN 1 Lumajang (2008-2011), SMAN 2 Lumajang (2011-2014), dan melanjutkan studi S1 di Departemen Informatika ITS. Bidang studi yang diambil oleh penulis pada saat berkuliah di Departemen Informatika ITS adalah Arsitektur Jaringan Komputer (AJK). Penulis dapat dihubungi melalui email bilal290995@gmail.com.