TUGAS AKHIR – IF184802

Implementasi *Double Verification Method* pada Route Discovery Ad-hoc On demand Distance Vector (AODV) untuk mengatasi Black Hole Attack pada MANET

RAFI R. RAMADHAN
NRP 05111540000158

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

Dosen Pembimbing II
Prof.Ir. Supeno Djanali, M.Sc Ph.D.

DEPARTEMEN INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabava 2019

TUGAS AKHIR – IF184802

Implementasi *Double Verification Method* pada Route Discovery Ad-hoc On demand Distance Vector (AODV) untuk mengatasi Black Hole Attack pada MANET

RAFI R. RAMADHAN
NRP 05111540000158

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

Dosen Pembimbing II
Prof.Ir. Supeno Djanali, M.Sc Ph.D.

DEPARTEMEN INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya 2019

(Halaman ini sengaja dikosongkan)

UNDERGRADUATE THESIS – IF184802

DOUBLE VERIFICATION METHOD IMPLEMENTATION IN Ad-hoc On demand Distance Vector (AODV) for Resolve Black Hole Attack ON MANET

RAFI R. RAMADHAN
NRP 05111540000158

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

Second Advisor
Prof.Ir. Supeno Djanali, M.Sc Ph.D.

INFORMATICS DEPARTMENT
Faculty of Information Communication and Technology
Institut Teknologi Sepuluh Nopember
Surabaya 2019

(Halaman ini sengaja dikosongkan)

LEMBAR PENGESAHAN

Implementasi *Double Verification Method* pada Route Discovery Ad-hoc On demand Distance Vector (AODV) untuk mengatasi Black Hole Attack pada MANET

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:
RAFI R. RAMADHAN
NRP: 05111540000158

Disetujui oleh Pembimbing *tugas akhir*:

1. Dr.Eng. Radityo Anggoro, S.Kom., M.Sc. .....
   (NIP. 19841016200812 1002) (Pembimbing 1)

2. Prof.Ir. Supeno Djanali, M.Sc Ph.D. .....
   (NIP. 19480619 197301 1 001) (Pembimbing 2)

SURABAYA
JUNI, 2019

(Halaman ini sengaja dikosongkan)

Implementasi *Double Verification Method* pada Route Discovery Ad-hoc On demand Distance Vector (AODV) untuk mengatasi Black Hole Attack pada MANET

Nama Mahasiswa	**:**	**Rafi R. Ramadhan**
NRP	**:**	**05111540000158**
Departemen	**:**	**Informatika FTIK ITS**
Dosen Pembimbing 1	**:**	**Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.**
Dosen Pembimbing 2	**:**	**Prof.Ir. Supeno Djanali, M.Sc Ph.D.**

Abstrak

Mobile Ad Hoc Network (MANET) adalah jaringan nirkabel ad hoc yang melakukan konfigurasi mandiri secara kontinyu dan tidak memiliki infrastruktur yang tetap. Salah satu algoritma *routing* pada MANET adalah *Ad-hoc On demand Distance Vector* (AODV). AODV merupakan algoritma *routing protocol reactive* yang akan bekerja ketika ada permintaan pengiriman data agar mengurangi *control overhead* dari *routing protocol* proaktif. Algoritma AODV menjaga *timer-based state* pada setiap *node* sesuai dengan penggunaan tabel *routing*. Tabel *routing* akan kadaluarsa jika jarang digunakan. Untuk membuat tabel *routing*, AODV akan melakukan *route discovery*. Sementara untuk Menjaga rute, akan dilakukan *route maintenance*. Protokol AODV rawan akan *Blackhole Attack*, yaitu penyerangan *node* jahat yang mengaku sebagai rute terbaik, lalu melakukan *drop* pada seluruh paket yang dikirim. Ini dapat terjadi karena *route* yang berubah ubah pada saat pengiriman data.

Modifikasi akan dilakukan pada konsep penentuan *route*, yang semula hanya dilakukan satu kali pengecekan, diubah menjadi dua kali. Hal ini dilakukan dengan cara menambahkan beberapa variabel pengamanan seperti *VERIFYPCKT dan CHECKVRF* yang nantinya akan dibandingkan sebelum menetapkan suatu *route* aman untuk dilewati. Jika pesan sesuai, maka *route* aman untuk dilalui sebagai jalur pengiriman data.

Sebaliknya, jika pesan tidak sesuai atau *reply* yang diterima bernilai default, maka *route* tidak aman.

Hasil dari implementasi menggunakan skenario simulasi NS-2 berupa peningkatan rata-rata *Packet Delivery Ratio* sebesar 1775 %, penurunan rata-rata *Forwarded Route Request* sebesar 11,98 %, dan penurunan rata-rata *Packet Loss* sebesar 77,42 %.

Dari hasil di atas, dapat diketahui bahwa modifikasi *Double Verification Method* pada AODV dapat diimplementasikan dan memengaruhi kinerja algoritma *routing* AODV pada lingkungan MANET.

Kata kunci : Blackhole Attack, MANET, NS-2, AODV.

DOUBLE VERIFICATION METHOD IMPLEMENTATION IN Ad-hoc On demand Distance Vector (AODV) for Resolve Black Hole Attack ON MANET

Student's Name : Rafi R. Ramadhan
Student's ID : 05111540000158
Department : Informatika FTIK-ITS
First Advisor : Dr.Eng. Radityo Anggoro, S.Kom, M.Sc.
Second Advisor : Prof.Ir. Supeno Djanali, M.Sc Ph.D.

Abstract

Mobile Ad Hoc Network (MANET) is an ad hoc wireless network that performs continuous self-configuration and does not have a fixed infrastructure. One of the routing algorithms in MANET is Ad-hoc On-demand Distance Vector (AODV). AODV is a reactive routing protocol algorithm that works on request of packet delivery to reduce the control overhead of proactive routing protocols. AODV algorithm keeps the timer-based state of each node value match with the usage of the routing table. The routing table will be expired if rarely used. To create routing table, AODV will perform route discovery. While to keep the route, route maintenance will be performed. AODV protocol is highly vulnerable to Blackhole Attack, which attack from malicious node that pretend to be the best route, then drop all the packet that sent. It happened because the route that often changed when start to deliver the data packet.

Modifications will be made to the concept of route discovery, which was originally used only one time checking, converted into double verification checking. This is done by adding several safety variables like VERIFYPCKT and CHECKVRF that later will be compared before set the route is safe to be used. If the message matches, then the route is safe to be used as a delivery route. On the other hand, if the message does not match or the reply that sent to the source node has a default value, the route is not safe.

The results of the implementation using the NS-2 simulation scenario in the form of an increase in the average Packet Delivery Ratio of 1775%, a decrease in the Forwarded Route Request average of 11,98%, and a decrease in the average Packet Loss of 77,42%.

From the results above, it can be seen that the modification of Double Verification Method in AODV can be implemented and affect the performance of the AODV routing algorithm in the MANET environment.

Keyword: Blackhole Attack, MANET, NS-2, ZRP.

KATA PENGANTAR

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

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

"Implementasi *Double Verification Method* pada Route Discovery Ad-hoc On demand Distance Vector (AODV) untuk mengatasi Black Hole Attack pada MANET".

Harapan dari penulis, semoga apa yang tertulis di dalam buku tugas akhir ini dapat bermanfaat bagi pengembangan ilmu pengetahuan saat ini dan ke depannya, serta dapat memberikan kontribusi yang nyata.

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

1. Allah SWT. dan Nabi Muhammad SAW. yang telah membimbing penulis selama hidup.
2. Keluarga penulis yang selalu memberikan dukungan baik berupa 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.Eng.Sc. dan Bapak Prof.Ir. Supeno Djanali, M.Sc Ph.D. 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 dosen wali dan kepala Departemen Informatika ITS.
5. Bapak dan Ibu Dosen yang telah memberikan ilmunya selama penulis berkuliah di Informatika ITS.
6. Teman-teman penulis GAES dan KONS (Abyan Dafa, Andrea Prahita Janardana, Drajad Bima Ajipangestu, Fajar Maulana Firdaus, GD Wahyu Nugraha Subagia, Ivan Fadhila, M. Azka

Yasin, Muh. Akram Abdullah, Muhammad Adib Arinanda, Muhammad Pandu Praadha, Narendra Haryo Bismo, Pandito HARS, Pradipta Baskara, Reinardus Wandya K, Rizaldy Primanta Putra, Satriyo Nugroho, Unggul Widodo Wijayanto, Widhera Yoza Mahana Putra, Yuga Mitra Hakiki, Zahri Rusli) yang selalu memberikan semangat secara tidak langsung kepada penulis, selalu memberikan hiburan, selalu menemani hari-hari penulis saat senang maupun susah, dan juga menjadi keluarga baru penulis saat berkuliah di Departemen Informatika ITS.

7. Teman-teman dari keluarga besar Laboratorium MIS (Muhajir, Ayas, Fino, Naja, Yuda, Zevi, Fasma, Shania, Andre, Fajar, Purina, PD, Ariya) yang telah menemani, memberi semangat, doa, serta hiburan dikala penulis sedang jenuh saat pengerjaan tugas akhir ini.
8. Teman-teman Core Team Developer Student Club (Ana, Daus, Zahri, Hana) yang telah bersama sama menyebarkan semangat pengembangan bersama Google Developer selama kurang lebih 2 tahun.
9. Teman-teman Tim Inti CommTECH Camp Insight 2017 (Mas Muh. Wahyu Islami Pratama, Adolft Afwari Rahman, Erica Maulidina Bening, dan Mbak Hannah Febriani) yang telah bersama sama menyukseskan short program ITS.
10. Teman-teman dari Kabinet Semangat Berpadu BEM FTIK periode 2017/2018 yang telah mewarnai hari-hari penulis dan mengajarkan makna ikhlas dan sabar, yang sudah berjuang bersama penulis selama kurang lebih satu setengah tahun, yang terkadang membuat penulis kesal namun tidak apa-apa.
11. Teman-teman IA Bahagia BEM FTIf Presisi Bermanfaat yang telah menjadi teman organisasi penulis, menambah rumah baru bagi penulis.
12. Teman-teman Medfo HMTC Kreasi dan Inspirasi yang telah menjadi teman berhimpun penulis.

13. Teman-teman pejuang SW 120 yang selalu memberikan informasi penting dan semangat kepada penulis untuk menyelesaikan tugas akhir.
14. Teman-teman angkatan 2015 (Masamalas) yang sudah menjadi saksi hidup perjalanan karir penulis selama berkuliah di Informatika ITS.
15. Teman-teman PILAR 45 yang tersebar diseluruh Indonesia.
16. Untuk orang-orang yang tidak dapat disebutkan satu persatu oleh penulis dan pembaca buku tugas akhir ini.

Penulis telah berusaha sebaik-baiknya dalam menyusun tugas akhir ini. Namun, penulis memohon maaf apabila terdapat kekurangan, kesalahan maupun kelalaian yang telah penulis lakukan. Kritik dan saran yang membangun dapat disampaikan sebagai bahan perbaikan selanjutnya. Tetap semangat dalam menjalani kehidupan, jangan menyerah, karena Allah masih ingin melihat kita berjuang. Semoga kita semua selalu diberi kebahagiaan lahir dan batin dan kesuksesan dunia akhirat. Aamiin.

Surabaya, 20 Juni 2019

Rafi R. Ramadhan

(Halaman ini sengaja dikosongkan)

DAFTAR ISI

Abstrak ..... **vii**
Abstract ..... **ix**
KATA PENGANTAR ..... **xi**
DAFTAR ISI ..... **xv**
DAFTAR GAMBAR ..... **xix**
DAFTAR TABEL ..... **xxi**
KODE SUMBER ..... **xxiii**
BAB I PENDAHULUAN **1**
1.1 Latar Belakang **1**
1.2 Rumusan Masalah **2**
1.3 Batasan Permasalahan **2**
1.4 Tujuan **3**
1.5 Manfaat **3**
1.6 Metodologi **3**
1.6.1 Penyusunan Proposal Tugas Akhir 3
1.6.2 Studi Literatur 3
1.6.3 Implementasi Sistem 4
1.6.4 Pengujian dan Evaluasi 4
1.6.5 Penyusunan Buku 4
1.7 Sistematika Penulisan Laporan **4**
BAB II TINJAUAN PUSTAKA **7**
2.1 MANET **7**
2.2 *Ad-hoc On demand Distance Vector* (AODV) **8**
2.3 *Network Simulator 2* (NS-2) **8**
2.3.1 Instalasi 9
2.3.2 *Trace File* 12
2.4 AWK **13**
BAB III PERANCANGAN **15**
3.1 Deskripsi Umum **15**
3.2 Daftar Istilah **17**
3.3 Perancangan Simulasi pada NS-2 ..... **18**
3.4 Perancangan Modifikasi *Routing Protocol* AODV **18**

3.4.1 Perancangan Pemilihan Pengiriman VERIFYPCKT dan Intermediate Node 19
3.4.2 Perancangan Pengiriman CHECKVRF 20
3.5 Perancangan Simulasi pada NS-2 21
3.6 Perancangan Metrik Analisis.....22
3.6.1 *Packet Delivery Ratio* (PDR) 22
3.6.2 *Forwarded Route Request (RREQ F)*.....22
3.6.3 *Packet Loss*.....23
BAB IV IMPLEMENTASI.....25
4.1 Lingkungan Pembangunan Sistem.....25
4.2 Implementasi Modifikasi *Routing Protocol* AODV.....25
4.2.1 Implementasi Inisialisasi *Blackhole node* pada rute 26
4.2.2 Implementasi Pemilihan Intermediate Node 27
4.2.3 Implementasi paket VERIFYPCKT dan CHECKVRF 28
4.3 Implementasi Simulasi pada NS-2 31
4.4 Implementasi Metrik Analisis 36
4.4.1 Implementasi *Packet Delivery Ratio* 36
4.4.2 Implementasi *Forwarded Route Request* 37
4.4.3 Implementasi *Packet Loss*.....38
BAB V UJI COBA DAN EVALUASI.....41
5.1 Lingkungan Uji Coba.....41
5.2 Hasil Uji Coba.....41
5.2.1 Hasil Uji Coba Interval 1 Paket/detik 42
5.2.2 Hasil Uji Coba Interval 2 Paket/detik 47
5.2.3 Hasil Uji Coba Interval 3 Paket/detik 52
5.3 Dampak Blackhole Node pada Skenario Pengiriman Paket.....57
BAB VI KESIMPULAN DAN SARAN.....59
6.1 Kesimpulan.....59
6.2 Saran 59
DAFTAR PUSTAKA 61
LAMPIRAN.....63

1. Kode Fungsi recvRequest() 63
2. Kode Fungsi sendRequest() 73
3. Kode Fungsi recvReply() 77
4. Kode Fungsi sendReply() 82
5. Kode Konfigurasi hdr_aodv_request 84
6. Kode Konfigurasi aodv_rtable 85
7. Kode Skenario NS-2 86
8. Kode Skrip AWK *Packet Delivery Ratio* 89
9. Kode Skrip AWK *Forwarded Route Request* 90
10. Kode Skrip AWK *Packet Loss* 91

BIODATA PENULIS 93

(Halaman ini sengaja dikosongkan)

DAFTAR GAMBAR

Gambar 3.1 Diagram Rancangan Simulasi AODV Modifikasi . 16
Gambar 3.2 Diagram Alur Proses Double Verification Method 17
Gambar 3.3 Pseudocode Pengiriman VERIFYPCKT dan Pemilihan Intermediate node 20
Gambar 3.4 Pseudocode Pengiriman CHECKVRF 21
Gambar 5.1 Grafik Rata-rata Packet Delivery Ratio pada Simulasi NS-2 Interval 1 43
Gambar 5.2 Grafik Rata-rata Forwarded Route Request pada Simulasi NS-2 Interval 1 45
Gambar 5.3 Grafik Rata-rata Packet Loss pada Simulasi NS2 Interval 1 46
Gambar 5.4 Grafik Rata-rata Packet Delivery Ratio pada Simulasi NS-2 Interval 2 48
Gambar 5.5 Grafik Rata-rata Forwarded Route Request pada Simulasi NS-2 Interval 2 50
Gambar 5.6 Grafik Rata-rata Packet Loss pada Simulasi NS2 Interval 2 51
Gambar 5.7 Grafik Rata-rata Packet Delivery Ratio pada Simulasi NS-2 Interval 3 53
Gambar 5.8 Grafik Rata-rata Forwarded Route Request pada Simulasi NS-2 Interval 3 55
Gambar 5.9 Grafik Rata-rata Packet Loss pada Simulasi NS2 Interval 3 56
Gambar 5.10 Perhitungan Dampak Double Verification Method 57

(Halaman ini sengaja dikosongkan)

DAFTAR TABEL

Tabel 2.1 Detail Penjelasan Trace File AODV.....12
Tabel 3.1 Daftar Istilah.....17
Tabel 3.2 Parameter Lingkungan Simulasi.....21
Tabel 4.1 Tabel Lingkungan Pembangunan Sistem 25
Tabel 5.1 Spesifikasi Perangkat yang Digunakan 41
Tabel 5.2 Hasil Rata - Rata PDR Interval 1.....42
Tabel 5.3 Hasil Rata - Rata RREQ-F Interval 1 42
Tabel 5.4 Hasil Rata - Rata Packet Loss Interval 1 43
Tabel 5.5 Hasil Rata - Rata PDR Interval 2.....47
Tabel 5.6 Hasil Rata - Rata RREQ-F Interval 2 47
Tabel 5.7 Hasil Rata - Rata Packet Loss Interval 2 48
Tabel 5.8 Hasil Rata - Rata PDR Interval 3.....52
Tabel 5.9 Hasil Rata - Rata RREQ-F Interval 3 52
Tabel 5.10 Hasil Rata - Rata Packet Loss Interval 3 53
Tabel 5.11 Dampak Blackhole pada Skenario Pengiriman Paket 57

(Halaman ini sengaja dikosongkan)

KODE SUMBER

Kode Sumber 2.1 Instalasi gcc 9
Kode Sumber 2.2 Memasang *dependency* yang dibutuhkan 10
Kode Sumber 2.3 Mengunduh NS-2 versi 2.35.....10
Kode Sumber 2.4 Memodifikasi file ls.h sesuai kebutuhan.....10
Kode Sumber 2.5 Melakukan Install NS-2.....11
Kode Sumber 4.1 Modifikasi Fungsi *AODV::recvRequest()* dalam kode sumber aodv.cc 26
Kode Sumber 4.2 Mengirim pesan *reply* setelah mendapat instruksi dari *Blackhole node* 27
Kode Sumber 4.3 Pemilihan Intermediate Node dan Inisiasi Pesan VERIFYPCKT 28
Kode Sumber 4.4 Source Node menerima request untuk mengirimkan paket CHECKVRF.....29
Kode Sumber 4.5 Destination Node menerima paket CHECKVRF dan melakukan komparasi. Jika cocok mengirim Final Reply....30
Kode Sumber 4.6 Pengaturan Parameter Lingkungan Simulasi 31
Kode Sumber 4.7 Pengaturan Inisialisasi NS-2.....32
Kode Sumber 4.8 Pengaturan Konfigurasi Parameter Node.....33
Kode Sumber 4.9 Posisi Statis Node Sumber dan Node Tujuan untuk skenario 50 node.....34
Kode Sumber 4.10 Konfigurasi Koneksi Simulasi 35
Kode Sumber 4.11 Konfigurasi Mengakhiri Simulasi.....36
Kode Sumber 4.12 Perintah Menjalankan Simulasi TCL.....36
Kode Sumber 4.13 Pseudocode Menghitung PDR 37
Kode Sumber 4.14 Pseudocode Menghitung RREQ-F.....38
Kode Sumber 4.15 Pseudocode Menghitung Delivery Delay ...39

(Halaman ini sengaja dikosongkan)

BAB I
PENDAHULUAN

1.1 Latar Belakang

Pada *Mobile Ad Hoc Network* (MANET), setiap node berperan sebagai end-system dan router, sehingga setiap node dapat meneruskan paket data ke node berikutnya. Routing dalam MANET merupakan suatu tantangan yang menarik karena MANET memiliki fitur yang dinamis, serta dibatasi oleh bandwidth dan energi. Pergerakan node akan meyebabkan perubahan topologi jaringan. Oleh karena itu, untuk proses discovery dan pemeliharaan routing pada lingkungan MANET merupakan hal yang sulit [1].

Pada protokol perutingan di MANET pada umumnya hanya menggunakan satu path tunggal untuk setiap node asal dan tujuan yang akan berkomunikasi. Tapi, dengan adanya mobilitas setiap node yang mengakibatkan topologi jaringannya berubah-ubah, atau membuat rute yang sudah ada sebelumnya menjadi terputus sementara dan node harus kembali membentuk rute baru. Oleh karena itu, diterapkanlah routing multipath, yang dapat memberikan lebih dari satu rute ke node tujuan. Sehingga node sumber dan node perantara dapat menggunakan rute ini sebagai rute utama maupun sebagai rute cadangan.

Routing protocol dalam MANET dibedakan menjadi dua model, yaitu proactive dan reactive routing. Proactive routing adalah protokol yang bekerja dengan cara membentuk tabel routing dan melakukan update setiap saat pada selang waktu tertentu. Sedangkan reactive routing adalah merupakan mekanisme routing yang membentuk tabel routing jika ada permintaan pengiriman data atau terjadinya perubahan rute dalam setiap jaringan. Contoh reactive routing protocol adalah Adhoc On-Demand Distance Vector (AODV), Dynamic Source Routing (DSR), dan Temporary Order Routing Algortihm (TORA).

Pencarian rute menjadi suatu mekanisme yang penting untuk mendukung mobilitas di MANET. Pemilihan rute yang stabil saat

proses pencarian rute sangat diperlukan untuk memperpanjang waktu penggunaan rute. Salah satu cara dapat dilakukan adalah dengan memilih rute yang memiliki kemungkinan kecil terputus.

Namun pada AODV performa pengiriman paket dapat di pengaruhi oleh *Black Hole*. *Black Hole* adalah node jahat yang menunggu tetangganya untuk mengirim RREQ messages. Ketika Black Hole node menerima pesan RREQ, *Black Hole node* akan langsung melakukan Reply dan mengaku sebagai rute tercepat dan terpendek untuk menuju Destination node. Ketika data mulai dikirim, node ini akan melakukan Drop Packet secara total.

Pada Tugas Akhir ini diimplementasikan suatu mekanisme routing discovery pada *reactive routing* AODV untuk memperoleh rute yang paling stabil dan mencegah pengiriman data kepada *Black Hole node* menggunakan *Double Verification Method* yang melibatkan *Intermediate node*. Disini, *Double Verification Method* diharapkan dapat mengatasi penyerangan oleh *Black Hole node*. Hasil akhir yang diharapkan adalah mengetahui perbandingan kinerja antara AODV dan AODV yang telah dimodifikasi diukur berdasarkan performansi *Packet Delivery Ratio* (PDR), *Forwarded Route Request* (RREQ F), dan *Packet Loss*.

1.2 Rumusan Masalah

Rumusan masalah yang diangkat dalam tugas akhir ini dapat dipaparkan sebagai berikut:

1. Bagaimana mencegah pengiriman paket data menuju *Black Hole node* dengan menggunakan *Double Verification Method?*
2. Bagaimana mengevaluasi dampak penambahan Double Verification terhadap performansi AODV yang diukur berdasarkan *Packet Delivery Ratio* (PDR), *Forwarded Route Request* (RREQ F), dan *Packet Loss*.

1.3 Batasan Permasalahan

Berdasarkan masalah yang diuraikan oleh penulis, maka batasan masalah pada tugas akhir ini adalah:

1. Jaringan yang digunakan adalah MANET.

2. *Routing protocol* yang diujicobakan yaitu AODV.
3. Uji coba menggunakan *Network Simulator* 2 (NS-2).

1.4 Tujuan

Tujuan dari pembuatan tugas akhir ini adalah sebagai berikut:

1. Meningkatkan *Packet Delivery Ratio (PDR)* pada AODV.
2. Mencegah pengiriman paket kepada *Black Hole node*.

1.5 Manfaat

Dengan dibuatnya Tugas Akhir ini akan memberikan sebuah manfaat dalam meningkatkan kestablian rute yang dibentuk berdasarkan yang terseleksi serta mengurangi potensi kegagalan dalam pengiriman paket.

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 gambaran tentang tugas akhir yang akan dibuat. Pendahuluan proposal tugas akhir meliputi hal yang menjadi latar belakang diajukannya usulan tugas akhir, rumusan masalah yang diangkat, batasan masalah yang menjadi konstrain dari tugas akhir, tujuan pembuatan tugas akhir, dan manfaat dari hasil tugas akhir. Di dalam proposal tugas akhir juga dijabarkan mengenai tinjauan pustaka yang menjadi referensi pendukung dalam pembuatan tugas akhir ini. Terdapat pula sub bab jadwal kegiatan yang menjelaskan jadwal pengerjaan tugas akhir.

1.6.2 Studi Literatur

Pada studi literatur, akan dilakukan pengumpulan informasi dan referensi yang digunakan dalam pengerjaan tugas akhir yaitu mengenai *Mobile Ad Hoc Network* (MANET), *Ad-hoc on-Demand*

Distance Vector (AODV), *Black hole, Network Simulator* 2 (NS-2), dan AWK.

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 simulator jaringan, bahasa C/C++ sebagai bahasa pemrograman untuk uji coba mengimplementasikan metode yang sudah diajukan.

1.6.4 Pengujian dan Evaluasi

Pengujian dilakukan dengan menggunakan MANET simulator untuk membuat simulasi keadaan topologi yang diujikan. Simulator yang digunakan adalah NS-2 dan akan menghasilkan keluaran berupa *trace file*. Dari *trace file* tersebut akan dihitung besar *Packet Delivery Ratio* (PDR), *Forwarded Route Request* (RREQ F), dan *Packet Loss* untuk mengetahui performa kinerja *routing protocol* yang telah dimodifikasi.

1.6.5 Penyusunan Buku

Pada tahap ini dilakukan penyusunan buku sebagai dokumentasi dari pelaksanaan tugas akhir yang mencakup 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 berisi penjelasan mengenai latar belakang, rumusan masalah, batasan permasalahan, 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, *library*, dan *tools* yang digunakan dalam penyusunan

tugas akhir ini. Kajian teori yang dimaksud berisi tentang penjelasan singkat mengenai *Mobile Ad Hoc Network* (MANET), *Ad-hoc on-Demand Distance Vector* (AODV), *Black hole, Network Simulator* 2 (NS-2), dan AWK.

3. Bab III. Perancangan
   Bab ini berisi perancangan skenario yang akan diimplementasikan dalam tugas akhir. Berupa pendefinisian terhadap *Black hole node*, perancangan skenario dan pemilihan rute menggunakan AODV yang telah dimodifikasi, perancangan simulasi pada NS-2, dan perancangan metrik analisis yang terdiri dari *Packet Delivery Ratio* (PDR), *Forwarded Route Request* (RREQ F), dan *Packet Loss*
4. Bab IV. Implementasi
   Bab ini menjelaskan implementasi yang berbentuk kode sumber dari proses modifikasi protokol AODV, melakukan simulasi menggunakan NS-2, dan penghitungan metrik analisis.
5. Bab V. Pengujian dan Evaluasi
   Bab ini berisi hasil uji coba dan evaluasi dari implementasi modifikasi pada *routing protocol* AODV yang telah dilakukan. Pengujian dilakukan dengan skenario yang telah dirancang dan dijalankan di NS-2, yang selanjutnya akan didapatkan hasil untuk dianalisis menggunakan skrip AWK yang nantinya akan menghasilkan metrik analisis berupa PDR, RREQ F, dan Packet Loss.
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 tugas akhir 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 alat yang digunakan dalam tugas akhir. Penjelasan ini bertujuan untuk memberikan gambaran secara umum terhadap program yang dibuat dan berguna sebagai penunjang dalam pengembangan riset yang berkaitan.

2.1 MANET

Mobile Ad Hoc Network (MANET) adalah jaringan nirkabel ad hoc yang melakukan konfigurasi mandiri secara kontinyu dan tidak memiliki infrastruktur yang tetap. Setiap komponen pada MANET dapat bergerak secara bebas ke segala arah, hal ini menyebabkan perubahan pada topologi secara acak. Karakteristik MANET yang dinamis membuat MANET banyak dimanfaatkan sebagai alat bantu komunikasi dalam situasi yang tidak memiliki infrastruktur seperti upaya rekonstruksi bencana alam, evakuasi Tim SAR, hingga operasi militer [2].

Tipe jaringan MANET adalah desentralisasi, maknanya setiap *node* memiliki tanggung jawab dalam proses pengiriman dan penerimaan paket data. *Node-node* tersebut akan meneruskan paket data ke *node* lain secara dinamis berdasarkan konektivitas jaringan dan algoritma *routing* yang digunakan. Algoritma *routing* pada MANET sendiri diklasifikasikan ke dalam tiga kategori, yaitu *routing protocol* proaktif, reaktif, dan hibrid. *Node-node* pada *routing protocol* proaktif bekerja secara kontinyu untuk mencari rute ke *node* tujuan sehingga tabel *routing* akan selalu ter-*update*. Algoritma *routing protocol* proaktif antara lain *Destination-Sequenced Distance-Vector Routing* (DSDV), *Optimized Link State Routing* (OLSR), *Fisheye State Routing* (FSR), dan *Fuzzy Sighted Link State* (FSLS). Pada *routing protocol* reaktif, *node* baru akan bekerja ketika ada permintaan pengiriman data. Algoritma yang termasuk ke dalam *routing protocol* reaktif antara lain *Dynamic Source Routing* (DSR) dan *Ad Hoc On Demand Distance*

Vector Routing (AODV). *Routing protocol* hibrid merupakan kombinasi dari *routing protocol* proaktif dan reaktif. Protokol ini menggunakan klaster-klaster untuk menentukan rute dari setiap *node*. *Routing protocol* proaktif digunakan untuk pengiriman paket data antar *node* dalam satu klaster, sedangkan *routing protocol* reaktif digunakan untuk pengiriman paket data antar klaster [3] [4].

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

Ad-hoc On demand Distance Vector (AODV) adalah salah satu *routing* protokol yang termasuk dalam klasifikasi *reactive routing protocol*. Sebuah protokol yang hanya me-*request* sebuah rute saat dibutuhkan. AODV dikembangkan oleh C. E. Perkins, E.M. Belding-Royer dan S. Das pada RFC 3561.

Ciri utama dari AODV adalah menjaga *timer-based state* pada setiap node sesuai dengan penggunaan tabel *routing*. Tabel *routing* akan kadaluarsa jika jarang digunakan. Ada dua tahapan dalam AODV yaitu *route discovery* dan r*oute maintenance*. *Route discovery* memiliki dua pesan yang berupa *Route Request* (RREQ) dan *Route Request* (RREP). Sedangkan *Route Maintenance* berupa *Route Error* (RERR).

AODV adalah sebuah metode *routing* pesan antar node yang memungkinkan node-node tersebut untuk melewatkan pesan melalui lingkungannya ke node yang tidak dapat dihubungi secara langsung. AODV melakukan ini dengan cara menemukan rute yang bisa dilalui oleh pesan. Selain itu AODV juga memastikan rute ini tidak mengandung perulangan (*loop*), menangani perubahan rute, dan membuat rute baru apabila terjadi *error* [5].

2.3 *Network Simulator 2* (NS-2)

Network Simulator (NS) pertama kali dibangun sebagai varian dari REAL *Network Simulator* pada tahun 1989 di UCB (University of California Berkeley). NS sendiri bersifat *open source* di bawah GPL (*Gnu Public License*), sehingga NS dapat diunduh dan digunakan secara gratis. Sifat *open source* ini juga berdampak pada pengembangan NS menjadi lebih dinamis.

Pemodelan media, protokol, komponen jaringan, dan perilaku trafik cukup lengkap bila dibandingkan dengan perangkat lunak lain yang sejenis. Hal ini disebabkan oleh pengembangan NS yang dilakukan oleh banyak periset dunia [6].

Network simulator memiliki beberapa versi, salah satunya adalah *Network Simulator* 2 (NS-2). NS-2 merupakan alat bantu simulasi berbasis aktivitas yang banyak digunakan pada penelitian jaringan kabel dan nirkabel. NS-2 menggunakan dua bahasa utama, yaitu C++ dan *Object-oriented Tool Command Language* (OTCL). Bahasa C++ digunakan untuk menggambarkan mekanisme internal (*backend*) pada objek simulasi, sedangkan OTCL digunakan untuk menggambarkan lingkungan simulasi eksternal (*frontend*) pada perakitan dan konfigurasi objek simulasi. Hasil dari simulasi menggunakan NS-2 berupa *trace file* (*.tr) dan *network animator* (*.nam) [7] [8].

Pada tugas akhir ini digunakan NS-2 versi 2.35 untuk melakukan simulasi lingkungan MANET menggunakan *routing protocol* AODV yang sudah dimodifikasi. *Trace file* yang dihasilkan oleh NS-2 juga digunakan sebagai informasi untuk mengukur performa *routing protocol* AODV yang sudah dimodifikasi.

2.3.1 Instalasi

Sebelum melakukan instalasi NS-2, terlebih dahulu harus memastikan bahwa *compiler* yang terpasang sudah sesuai dengan standar yang digunakan oleh NS-2, yaitu gcc-4.8. Untuk memasang *compiler* tersebut dapat dilakukan dengan perintah sebagai berikut:

```
sudo apt install gcc-4.8 g++-4.8
```

Kode Sumber 2.1 Instalasi gcc

NS-2 membutuhkan beberapa *package* yang harus sudah terpasang pada perangkat komputer sebelum memulai instalasi NS-

2. Untuk memasang *dependency* yang dibutuhkan dapat dilakukan dengan perintah sebagai berikut :

```
sudo apt update

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

Kode Sumber 2.2 Memasang *dependency* yang dibutuhkan

Kemudian, unduh NS-2 versi 2.35 yang dapat dilakukan melalui tautan berikut :

```
https://sourceforge.net/projects/nsnam/fi
les/allinone/ns-allinone-2.35/ns-
allinone-2.35.tar.gz/download
```

Kode Sumber 2.3 Mengunduh NS-2 versi 2.35

lalu ekstrak berkas unduhan NS-2 pada direktori yang diinginkan. Selanjutnya, masuk ke dalam direktori ns-allinone-2.35/ns-2.35, lalu ubah baris kode ke-137 pada *file* ls.h di folder linkstate menjadi seperti perintah berikut :

```
gedit Makefile.in
ubah baris kode ke-36 dan 37 menjadi:
CC    = gcc-4.8
CPP   = g++-4.8

gedit linkstate/ls.h
ubah baris kode ke-137 menjadi:
void eraseAll(){this->erase
(baseMap::begin(),baseMap::end());}
```

Kode Sumber 2.4 Memodifikasi file ls.h sesuai kebutuhan

Setelah mengubah baris kode, kembali ke direktori ns-allinone-2.35 dan instal NS-2 dengan menjalankan perintah berikut:

```
./install
```

Kode Sumber 2.5 Melakukan Install NS-2

Kemudian, *environment* sistem pada NS-2 diatur agar NS-2 dapat dijalankan. Pengaturan tersebut dapat dilakukan melalui perintah sebagai berikut:

```
sudo gedit ~/.bashrc
```

Copy skrip berikut diakhir file:

```
# LD_LIBRARY_PATH
OTCL_LIB=/home/userta/ns-allinone-
2.35/otcl-1.14
NS2_LIB=/home/userta/ns-allinone-2.35/lib
X11_LIB=/usr/X11R6/lib
USR_LOCAL_LIB=/usr/local/lib
export
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$OTCL_LI
B:$NS2_LIB:$X11_LIB:$USR_LOCAL_LIB

# TCL_LIBRARY
TCL_LIB=/home/userta/ns-allinone-
2.35/tcl8.5.10/library
USR_LIB=/usr/lib
export TCL_LIBRARY=$TCL_LIB:$USR_LIB

# PATH
XGRAPH=/home/userta/ns-allinone-
2.35/bin:/home/userta/ns-allinone-
2.35/tcl8.5.10/unix:/home/userta/ns-
allinone-2.35/tk8.5.10/unix
NS=/home/userta/ns-allinone-2.35/ns-2.35/
NAM=/home/userta/ns-allinone-2.35/nam-
1.15/
PATH=$PATH:$XGRAPH:$NS:$NAM
```

Lalu masukkan perintah `source ~/.bashrc` dan `ns` pada terminal.

2.3.2 *Trace File*

Trace file merupakan dokumen hasil simulasi NS-2 yang berisikan informasi detail mengenai pengiriman paket data. *Trace file* digunakan untuk menganalisis performa *routing protocol* yang disimulasikan. Penjelasan mengenai *trace file* ditunjukkan pada Tabel 2.1.

Tabel 2.1 Detail Penjelasan Trace File AODV

Kolom ke-	**Penjelasan**	**Isi**
1	*Event*	s: *send* 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 Layer ARP: Paket *link layer Address Resolution Protocol*
8	Ukuran	Ukuran paket pada *layer* saat itu

Kolom ke-	Penjelasan	Isi
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)*

2.4 AWK

AWK merupakan sebuah program filter untuk teks berupa bahasa pemrograman pada *shell* atau C yang memiliki karakteristik sebagai alat untuk melakukan filter atau manipulasi data. 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*. Secara umum AWK dapat digunakan untuk mengelola *database* sederhana, membuat laporan, memvalidasi data, dan membuat algoritma yang digunakan untuk mengubah bahasa komputer ke bahasa lainnya. Dengan kata lain, AWK menyediakan fasilitas yang dapat memudahkan untuk memecah bagian data untuk proses selanjutnya, mengurutkan data, dan menampilkan komunikasi jaringan yang sederhana. Pada tugas akhir ini, AWK digunakan untuk membuat skrip menghitung metrik analisis berupa *Packet Delivery Ratio* (PDR), *Forwarded Route Request* (RREQ F), dan *Packet Loss*. dari hasil simulasi menggunakan NS-2 [9].

(Halaman ini sengaja dikosongkan)

BAB III
PERANCANGAN

Bab ini membahas mengenai perancangan implementasi sistem yang dibangun pada tugas akhir. Bagian yang akan dijelaskan pada bab ini berawal dari deskripsi umum sistem, perancangan skenario, alur, hingga gambaran implementasi sistem yang diterapkan pada *Network Simulator* 2 (NS-2).

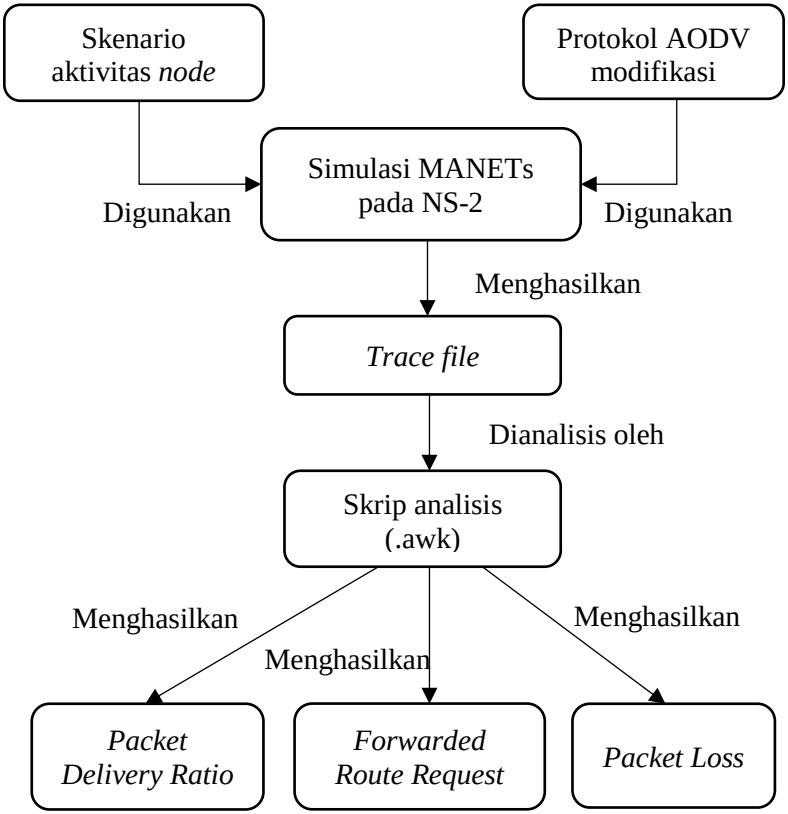
3.1 Deskripsi Umum

Pada tugas akhir ini akan dilakukan implementasi *routing protocol* AODV yang dimodifikasi dengan menerapkan metode verifikasi ganda yang harus dilakukan sebelum pengiriman paket ke node tujuan. Pembuatan skenario MANET ini menggunakan simulator yaitu *Network Simulator* 2 (NS-2).

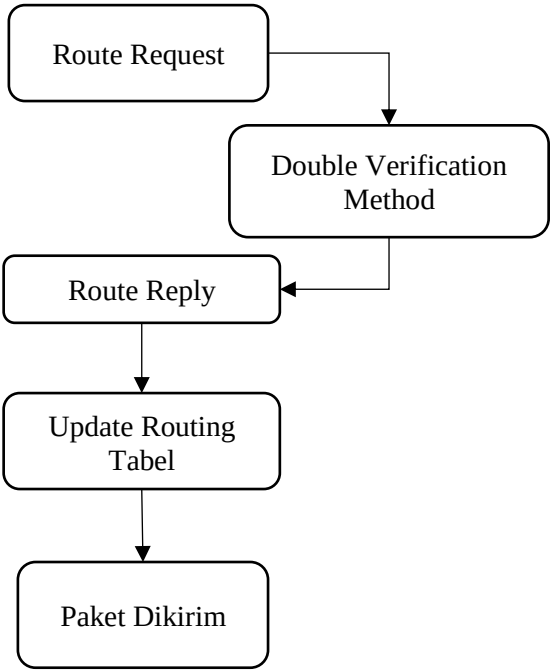
Perancangan metode verifikasi diawali dengan melakukan pengiriman route request, namun berbeda dari biasanya, *intermediate node* berperan untuk mengirim sebuah pesan verifikasi (*VERIVYPCKT)* setelah mendapatkan reply (berbentuk request dengan tipe 2) dari destination node. VERIVYPCKT ini dikirimkan kepada *destination node* bersamaan dengan meneruskan request kepada source node untuk meminta CHECKVRF agar dikirimkan melalui jalur yang sama kepada *Destination node*. Setelah memiliki VERIVYPCKT dan CHECKVRF, *Destination node* akan membandingkan nilai kedua paket tersebut (berbentuk string). Jika cocok, maka *Destination node* akan mengirimkan *Final Reply* untuk meminta pengiriman paket yang sesungguhnya.

Skenario pengiriman paket data menggunakan *routing protocol* AODV akan dimulai segera setelah *Source node* menerima reply dari *Destination node*. Simulasi yang dilakukan akan menghasilkan *trace file,* yang kemudian dianalisis menggunakan skrip AWK untuk mendapatkan *Packet Delivery Ratio* (PDR), *Forwarded Route Request* (RREQ F), dan *Packet Loss* Analisis tersebut dapat mengukur performa *routing protocol*

AODV yang telah dimodifikasi dibandingkan dengan *routing protocol* AODV asli. Diagram rancangan simulasi dengan *routing protocol* AODV yang telah dimodifikasi dapat dilihat pada Gambar 3.1 dan Gambar 3.2.



Gambar 3.1 Diagram Rancangan Simulasi AODV Modifikasi



Gambar 3.2 Diagram Alur Proses Double Verification Method

3.2 Daftar Istilah

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	*Mobile Adhoc Network* (MANET)	*Mobile Ad Hoc Network* (MANET) merupakan jaringan nirkabel ad hoc yang melakukan konfigurasi mandiri secara kontinyu dan tidak memiliki infrastruktur yang tetap.

2	*Ad-hoc On demand Distance Vector* (AODV)	*Ad-hoc On demand Distance Vector* (AODV) adalah salah satu *routing* protokol yang termasuk dalam klasifikasi *reactive routing protocol*. Sebuah protokol yang hanya me-*request* sebuah rute saat dibutuhkan.
3	*Network Simulator* 2 (NS-2)	NS-2 merupakan alat bantu simulasi berbasis aktivitas pada penelitian jaringan kabel maupun nirkabel.
4	*Packet Delivery Ratio* (PDR)	PDR merupakan teknik penghitungan perbandingan jumlah paket yang diterima oleh *node* tujuan dengan jumlah paket yang dikirim oleh *node* sumber.
5	*Forwarded Route Request* (RREQ F)	*Forwarded Route Request* adalah jumlah paket kontrol *route request* yang di*forward* per data paket ke *node* tujuan selama simulasi terjadi
6	*Route Request* (RREQ)	*Route Request*. Paket *request* pada AODV yang dikirim untuk mendapatkan rute
7	*Route Reply* (RREP)	*Route Reply*. Paket *reply* pada AODV yang dikirim ke *node* sumber melalui rute yang sudah terbuat.

3.3 Perancangan Simulasi pada NS-2

Simulasi MANET pada NS-2 dilakukan dengan menggunakan skenario mobilitas dan digabungkan dengan skrip TCL yang berisikan konfigurasi mengenai lingkungan simulasi

3.4 Perancangan Modifikasi *Routing Protocol* AODV

Protokol AODV yang diajukan pada Tugas Akhir ini merupakan modifikasi dari protokol AODV yang mengubah mekanisme *route discovery* pada protokol tersebut. Pada protokol AODV, mekanisme pencarian *node* dengan RREQ untuk pengiriman paket langsung dikirim begitu saja setelah menerima

RREP, maka pada AODV yang dimodifikasi ini akan ada proses penyeleksian *node*. Seleksi *node* dilakukan dengan cara *node* sumber menunggu *Intermediate node* mengirimkan *VERIFYPCKT* kepada *Destination node* sekaligus menerusken Request bertipe 1 kepada *Source node*. Request bertipe 1 ini merupakan tanda untuk meminta *Source node* mengirimkan *CHECKVRF* yang akan di lakukan komparasi oleh *Destination node* untuk memastikan rute aman. Setelah *VERIFYPCKT* dan *CHECKVRF* dipastikan memiliki nilai sama, barulah *Destination node* mengirimkan *Final Reply* dengan tipe 1 kepada *Source node.* Lalu paket akan dikirim.

3.4.1 Perancangan Pemilihan Pengiriman VERIFYPCKT dan Intermediate Node

Ketika *Destination node* Menerima Request default dari *Source node* (request type = 0) maka *Destination node* akan mengirimkan paket dengan type 1 untuk meminta *VERIFYPCKT* kepada *Intermediate node.* Intermediate node sendiri didefinisikan sebagai node dengan *Hop Count* = 1 dihitung dari *Destination node.* Ketika *Intermediate node* menerima Request ini, *Intermediate node* akan melakukan 2 tugas secara bersamaan, Mengirimkan *VERIFYPCKT* kepada *Destination node* dan meneruskan request *CHECKVRF* oleh *Destination node* ke *Source node.* Untuk *Pseudocode* dapat dilihat pada Gambar 3.2

```
if (rq->rq_dst == index && rq-
>rq_request_type == 0)
    if (f_destination == -1)
        f_destination = index;
    rtable.rt_request_type = 1;
    sendRequest(rq->rq_src);
←==========VERIFYPCKT==========→

if (rq->rq_request_type == 1 && rq-
>rq_hop_count == 1)
    char sent_message[25] = "iamsafe";
    rtable.rt_request_type = 2;
    strcpy(rtable.rt_saved_VERIFYPCKT,
sent_message);
    sendRequest(rq->rq_src);
```

Gambar 3.3 Pseudocode Pengiriman VERIFYPCKT dan Pemilihan Intermediate node

3.4.2 Perancangan Pengiriman CHECKVRF

Pengiriman paket *CHECKVRF* akan dilakukan setelah *Source node* menerima request dengan tipe 1 yang diteruskan oleh *Intermediate node*. Paket ini akan dikirimkan langsung kepada *Destination node* yang pada proses sebelumnya posisinya sudah disimpan pada variabel *f_destination. Pseudocode* untuk pengiriman CHECKVRF dapat dilihat pada Gambar 3.3.

```
if (rq->rq_dst == index && rq-
>rq_request_type == 1)
    char CHECKVRF[25] = "iamsafe";

    strcpy(rtable.rt_saved_CHECKVRF,
CHECKVRF);
    rtable.rt_request_type = 3;
    sendRequest(rq->rq_src);
    rtable.rt_delete(rq->rq_src);
```

Gambar 3.4 Pseudocode Pengiriman CHECKVRF

3.5 Perancangan Simulasi pada NS-2

Simulasi MANET pada NS-2 dilakukan dengan menggunakan skrip TCL dengan parameter yang berisi konfigurasi untuk membangun lingkungan simulasi. Parameter simulasi perancangan sistem MANET dapat dilihat pada Tabel 3.2.

Tabel 3.2 Parameter Lingkungan Simulasi

No.	Parameter	Spesifikasi
1.	*Network Simulator*	NS-2 versi 2.35
2.	*Routing protocol*	AODV
3.	Waktu Simulasi	200 detik
4.	Waktu Pengiriman Paket Data	2,5 – 200 detik
5.	Area Simulasi	800 m x 800 m
6.	Banyak *Node*	50, 100, 150
7.	Kecepatan Pergerakan *Node*	20 m/s
8.	Radius Transmisi	200 m
9.	Tipe Koneksi	UDP
10.	Tipe Data	*Constant Bit Rate* (CBR)
11.	Kecepatan Generasi Paket	1, **2**, 3 paket per detik
12.	Ukuran Paket Data	512 *bytes*

13.	Protokol MAC	IEEE 802.11
14.	Tipe Antena	OmniAntenna
15.	Tipe Interface Queue	Droptail/PreQueue
16.	Tipe Kanal	*Wireless Channel*
17.	Tipe Trace	Old Format Trace

3.6 Perancangan Metrik Analisis

Berikut ini merupakan parameter-parameter yang akan dianalisis pada tugas akhir ini untuk mengetahui pengaruh dari implementasi *Double Verification Method* pada *routing protocol* AODV.

3.6.1 *Packet Delivery Ratio* (PDR)

Packet Delivery Ratio (PDR) merupakan teknik penghitungan perbandingan antara jumlah paket yang diterima oleh *destination node* dengan jumlah paket yang dikirim oleh *source node*. PDR dihitung dengan persamaan 3.1.

$$PDR = \frac{packet\ received}{packet\ sent} x\ 100\ \% \quad \textbf{(3.1)}$$

Keterangan :
Packet Received = jumlah total paket yang diterima oleh *node* tujuan.
Packet Sent = jumlah total paket yang dikirim oleh *node* sumber.

3.6.2 *Forwarded Route Request (RREQ F)*

Forwarded Route Request adalah jumlah paket kontrol *route request* yang di*forward* per data paket ke *node* tujuan selama simulasi terjadi. RREQ F didapatkan dengan menjumlahkan semua paket control *routing* yang ditransmisikan khususnya *route request* bagian *sent* (RREQ F). Perhitungan RREQ F dapat dilihat dengan persamaan 3.2.

$$Forwarded\ Route\ Request = \sum_{n=1}^{rreqsent} packet\ sent \quad \textbf{(3.2)}$$

Keterangan :
Rreqsent = Jumlah RREQ yang dikirim oleh seluruh *node*
Packet sent = Jumlah total paket yang dikirim oleh *node* sumber

3.6.3 *Packet Loss*

Packet Loss menunjukkan jumlah paket yang hilang diantara *Source node* dengan *Destination node* dan diukur dalam *Packet Loss Ratio*. Pengukuran *Packet Loss* sebagai bahan Analisa jaringan pada komunikasi data secara *real time* cukup penting. Trafik komunikasi *real time* yang menggunakan transport protocol UDP tidak dapat menjamin sebuah paket data dapat diterima oleh node tujuan dengan baik. Berbeda dengan pengiriman paket data menggunakan protocol TCP yang proses pengiriman datanya melalui proses three-way-handshaking. Dengan demikian perlu dipastikan kualitas sebuah jaringan untuk komunikasi data *real time* yang disebut sebagai QoS (Quality of Service). *Packet Loss* dihitung dengan persamaan 3.3. Sementara untuk rationya dapat dihitung dengan persamaan 3.4.

$$total\_packet\_lost = (total\_packet\_sent - total\_packet\_received) \quad \textbf{(3.3)}$$

$$Packet\_loss\_rate = \left(\frac{total\_packet\_lost}{total\_packet\_sent}\right) * 100 \quad \textbf{(3.4)}$$

Keterangan :
Total_Packet_Lost = Jumlah paket yang hilang pada proses pengiriman.
Total_Packet_Sent = Jumlah paket yang berhasil terkirim.
Total_Packet_Received = Jumlah paket yang berhasil diterima

(Halaman ini sengaja dikosongkan)

BAB IV
IMPLEMENTASI

Bab ini membahas mengenai implementasi sistem yang dilakukan berdasarkan rancangan yang telah dijabarkan pada bab sebelumnya. Implementasi berupa kode sumber untuk membangun program. Implementasi yang dijelaskan meliputi lingkungan pembangunan sistem, implementasi skenario *Blackhole*, implementasi modifikasi *routing protocol* AODV, implementasi simulasi pada NS-2, dan implementasi metrik analisis.

4.1 Lingkungan Pembangunan Sistem

Lingkungan sistem yang digunakan untuk membangun implementasi skenario dapat dilihat pada Tabel 4.1.

Tabel 4.1 Tabel Lingkungan Pembangunan Sistem

Perangkat	Komponen	Spesifikasi
Perangkat Keras	*Processor*	Intel(R) Core(TM) i3-2120 CPU @ 3.30GHz
	Memori	RAM 8,00 GB
	Penyimpanan	90.1 GB
Perangkat Lunak	Sistem Operasi	Ubuntu 16.04.6 LTS 64 bit
	Simulator	*Network Simulator* 2 versi 2.35

4.2 Implementasi Modifikasi *Routing Protocol* AODV

Pada tugas akhir ini dilakukan modifikasi pada *routing protocol* AODV agar dapat meningkatkan *Packet Delivery Ratio* (PDR) pada scenario yang terdapat *Blackhole node* di dalamnya. Hal tersebut dilakukan dengan cara memilih rute yang akan dilakukan dengan 2 kali verifikasi sehingga dipastikan rute yang akan dilewati terbebas dari *Blakhole node.* Sehingga dapat dilihat peningkatan performa pada *routing* AODV yang telah dimodifikasi.

Implementasi modifikasi routing protocol AODV ini dibagi menjadi 3 bagian yaitu:

- Implementasi Inisialisasi *Blackhole node* pada rute.
- Implementasi Pemilihan *Intermediate node*
- Implementasi paket *VERIFYPCKT* dan *CHECKVRF*

4.2.1 Implementasi Inisialisasi *Blackhole node* pada rute

Langkah awal yang dilakukan untuk melakukan inisialisasi *Blackhole node* seperti yang sudah di definisikan sebelumnya adalah menentukan sifat sifat dasar dari *Blackhole node*.

Pada *routing protocol* AODV, terdapat fungsi *AODV::recvRequest*. Fungsi ini berfungsi untuk melakukan pengecekan siapa yang mengirimkan request dan apa yang harus dilakukan selanjutnya. Jika tujuan (rq->rq_dst) bernilai sama dengan *index* saat pengecekan, berarti request telah sampai kepada *Destination node*. Jika tidak, akan dilakukan pengecekan terhadap beberapa variabel seperti *Hop Count*, *Destination Sequence Number* (DSN), dan apakah routing tabel menuju *Destination node* sudah ada atau belum. 3 Variabel tersebut yang dimanfaatkan untuk memberi sifat dasar pada *Blackhole node* dan memanfaatkan fungsi *recvRequest()* pada kode sumber aodv.cc yang terdapat dalam folder ns-2.35/aodv. Potongan kode modifikasi dapat dilihat pada Kode Sumber 4.1.

```
if (index == 2 || index == 4 || index == 7 ||
index == 12) {
        if(rt == 0) rt = rtable.rt_add(rq-
        >rq_dst);
        if(rt->rt_seqno == 0) rt->rt_seqno=9999;
        rt->rt_seqno = rt->rt_seqno+5;
        rt->rt_hops = 0;
    }
```

Kode Sumber 4.1 Modifikasi Fungsi *AODV::recvRequest()* dalam kode sumber aodv.cc

Dengan ini, setiap node 2, 4, 7, dan 12 akan mendapatkan sifat blackhole yang memiliki rute ke *rq->rq_dst*, *rt->rt_seqno* dengan nilai tertinggi, dan juga *rt->rt_hops* dengan nilai terkecil sehingga pasti dipilih sebagai rute terbaik dan tercepat menuju *Destination node*.

Selanjutnya, setelah paket *request* sampai pada node blackhole, akan di terima oleh salah satu *if statement* yang akan mengantarkan pesan bahwa rute tercepat dan terbaik menuju *Destination node* adalah melalui *Blackhole node* dan mengirimkan pesan *reply* kepada *Source node*. Untuk potongan kode tersebut dapat dilihat pada Kode Sumber 4.2.

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

Kode Sumber 4.2 Mengirim pesan *reply* setelah mendapat instruksi dari *Blackhole node*

4.2.2 Implementasi Pemilihan Intermediate Node

Intermediate node dipilih berdasarkan jumlah *Hop Count* terhitung dari *Destination node.* Dengan memanfaatkan fungsi *AODV::recvRequest()* yang sudah di modifikasi dan memiliki beberapa tipe. Seperti yang sudah di bahas pada bagian 3.4.1 , Ketika *Destination node* menerima request dengan tipe 0 (default) maka akan

membalas dengan request bertipe 1. Node pertama yang menerima request dengan tipe 1, akan di tunjuk sebagai *Intermediate node* untuk meneruskan pesan meminta *CHECKVRF* dan memberikan pesan *VERIFYPCKT*. Untuk potongan kode tersebut dapat dilihat pada Kode Sumber 4.3.

```
if (rq->rq_request_type == 1 && rq-
>rq_hop_count == 1) {

      char sent_message[25] = "iamsafe";
      rtable.rt_request_type = 2;
      strcpy(rtable.rt_saved_VERIFYPCKT,
      sent_message);

      sendRequest(rq->rq_src);

    }
```

Kode Sumber 4.3 Pemilihan Intermediate Node dan Inisiasi Pesan VERIFYPCKT

4.2.3 Implementasi paket VERIFYPCKT dan CHECKVRF

Pada tahap selanjutnya setelah *Intermediate node* mendapatkan request dengan tipe 1 oleh *Destination node, request ini* diteruskan menuju *Source node.* Sesaat setelah request diterima, *Source node* akan melakukan pengecekan tipe request. Jika sesuai, maka paket *CHECKVRF* akan di kirim kepada *Destination node.* Untuk potongan kode tersebut dapat dilihat pada Kode Sumber 4.4

```
else if (rq->rq_dst == index && rq-
>rq_request_type == 1) {

        char CHECKVRF[25] = "iamsafe";
        strcpy(rtable.rt_saved_CHECKVRF,
        CHECKVRF);
        rtable.rt_request_type = 3;
        sendRequest(rq->rq_src);
        rtable.rt_delete(rq->rq_src);
        Packet::free(p);

    }
```

Kode Sumber 4.4 Source Node menerima request untuk mengirimkan paket CHECKVRF

Untuk paket *VERIFYPCKT*, saat ini sudah berada pada *Destination node*, dikirimkan ketika *Intermediate node* menerima request dengan tipe 1. Bisa dilihat pada 4.2.2 untuk proses pengirimannya. Setelah *Destination node* menerima paket *CHECKVRF*, lalu akan dilakukan perbandingan nilai kedua paket tersebut. Ketika paket sesuai, maka rute dipastikan aman dan *Final Reply* untuk meminta pengiriman data akan segera di kirim kepada *Source node.* Untuk potongan kode tersebut dapat dilihat pada Kode Sumber 4.5

```
else if (rq->rq_dst == index &&
         rq->rq_request_type == 3 &&
         strcmp(rtable.rt_saved_VERIFYPCKT,
         rtable.rt_saved_CHECKVRF) == 0) {

      seqno = max(seqno, rq->rq_dst_seqno) + 1;

      if (seqno % 2) seqno++;
      rtable.rt_reply_type = 1;
      sendReply(rq->rq_src,
                1,
                index,
                seqno,
                MY_ROUTE_TIMEOUT,
                rq->rq_timestamp);
      Packet::free(p);

    }
```

Kode Sumber 4.5 Destination Node menerima paket CHECKVRF dan melakukan komparasi. Jika cocok mengirim Final Reply

Dengan diterimanya *Final reply* oleh *Source node*, lalu pengiriman akan segera dimulai dengan rute yang sudah dipastikan tidak terdapat *Blackhole node* di dalamnya.

4.3 Implementasi Simulasi pada NS-2

Implementasi simulasi pada NS-2 dijalankan menggunakan kode program TCL. Dokumen ini berisi konfigurasi yang dibutuhkan untuk setiap *node* dan langkah-langkah yang dilakukan selama simulasi. Kode program untuk konfigurasi lingkungan simulasi dapat dilihat pada Kode Sumber 4.6.

```
# channel type
set val(chan)     Channel/WirelessChannel;
# radio-propagation model
set val(prop)     Propagation/TwoRayGround;
# network interface type
set val(netif)    Phy/WirelessPhy;
# MAC type
set val(mac)      Mac/802_11;
# interface queue type
set val(ifq)      Queue/DropTail/PriQueue;
# link layer type
set val(ll)       LL;
# antenna model
set val(ant)      Antenna/OmniAntenna;
# X dimension of the topography
set opt(x)        800;
# Y dimension of the topography
set opt(y)        800;
# max packet in ifq
set val(ifqlen)   50;
# how many nodes are simulated
set val(nn)       50;
# seed cbr
set val(seed)     1.0;
# routing protocol
set val(adhocRouting)   AODV;
# simulation time
set val(stop)     200;
#<-- traffic file
set val(cp)       "cbr50node.tcl";
#<-- mobility file
set val(sc)       "setdest50.tcl";
```

Kode Sumber 4.6 Pengaturan Parameter Lingkungan Simulasi

Setelah melakukan konfigurasi lingkungan simulasi, langkah selanjutnya adalah melakukan inisialisasi program untuk skenario MANET. Kode program untuk inisialisasi skenario MANET dapat dilihat pada Kode Sumber 4.7.

```
# create simulator instance
set ns_ [new Simulator]

# setup topography object
set topo [new Topography]

# create trace object for ns and nam
set tracefd [open result.tr w]
set namtrace [open result.nam w]

$ns_ trace-all $tracefd
$ns_ namtrace-all-wireless $namtrace $opt(x)
$opt(y)

# set up topology object
set topo [new Topography]
$topo load_flatgrid $opt(x) $opt(y)

# Create God
set god_ [create-god $val(nn)]
```

Kode Sumber 4.7 Pengaturan Inisialisasi NS-2

Pada Kode Sumber 4.7, ns_ merupakan variabel baru untuk simulator. Pengaturan topografi juga dilakukan untuk menentukan luas area yang akan digunakan saat simulasi, di mana dalam simulasi ini digunakan topografi dengan koordinat x dan y. Untuk menyimpan hasil simulasi, digunakan perintah *trace-all* di mana nantinya hasil simulasi akan dimasukkan ke dalam dokumen berekstensi .tr. Pada NS-2 juga disediakan *Network Animator* (biasa disebut NAM), yang memiliki sistem kerja hampir sama dengan dokumen .tr. Keduanya sama-sama melakukan *trace* terhadap skenario pada NS-2. Perbedaannya terletak pada hasil keluarannya, di mana NAM

berbentuk visual dengan dokumen berekstensi .nam, sedangkan .tr berbentuk teks.

Sebelum membuat *node* untuk simulasi, hal pertama yang dilakukan adalah melakukan konfigurasi parameter *node* yang akan digunakan untuk simulasi. Konfigurasi *node* dapat terdiri dari tipe ad hoc *routing protocol*, *link layer*, MAC *layer*, dan lain sebagainya. Kode program untuk konfigurasi parameter *node* dapat dilihat pada Kode Sumber 4.8.

```
$ns_ node-config
      -adhocRouting $val(adhocRouting) \
      -llType $val(ll) \
      -macType $val(mac) \
      -ifqType $val(ifq) \
      -ifqLen $val(ifqlen) \
      -antType $val(ant) \
      -propType $val(prop) \
      -phyType $val(netif) \
      -channelType $val(chan) \
      -energyModel $val(energy_mod) \
      -initialEnergy $val(energy_init) \
      -txPower $val(tx_power) \
      -rxPower $val(rx_power) \
      -idlePower $val(idle_power) \
      -sleepPower $val(sleep_power) \
      -topoInstance $topo \
      -agentTrace ON \
      -routerTrace ON \
      -macTrace ON \
      -movementTrace ON
```

Kode Sumber 4.8 Pengaturan Konfigurasi Parameter Node

Setelah melakukan konfigurasi parameter *node*, selanjutnya adalah membuat *node* dengan mendefinisikan *node-node* yang digunakan seperti menentukan posisi *Source node*, pergerakan *node*, dan pemberian label untuk masing-masing *node*. Pada tugas akhir ini,

posisi *Source node* dan *Destination node* diatur secara statis, sedangkan node lainnya diatur secara acak. Kode program untuk mendefinisikan *node* sumber dan tujuan dapat dilihat pada Kode Sumber 4.9.

```
# nodes: 48, speed type: 1, min speed: 20.00,
max speed: 60.00
# avg speed: 31.95, pause type: 1, pause:
1.00, max x: 500.00, max y: 500.00
#
$node_(0) set X_ 296.247851364210
$node_(0) set Y_ 413.375222585640
$node_(0) set Z_ 0.000000000000
$node_(1) set X_ 276.083282177644
$node_(1) set Y_ 424.165885133321
$node_(1) set Z_ 0.000000000000
.
.
.
$node_(47) set X_ 393.913907682079
$node_(47) set Y_ 480.320977141793
$node_(47) set Z_ 0.000000000000
#Source node
$node_(48) set X_ 500.000000000000
$node_(48) set Y_ 0.000000000000
$node_(48) set Z_ 0.000000000000
#Destination node
$node_(49) set X_ 0.000000000000
$node_(49) set Y_ 500.000000000000
$node_(49) set Z_ 0.000000000000
```

Kode Sumber 4.9 Posisi Statis Node Sumber dan Node Tujuan untuk skenario 50 node

Pada Kode Sumber 4.9, posisi pergerakan node di *generate* secara otomatis menggunakan script *setdest.* Pada saat di *generate* node yang di buat hanya n-2 (jumlah scenario dikurangi 2), setelah kode *setdest* di dapatkan lalu ditambahkan 2 node terakhir sebagai

Source node dan *Destination node*. Selanjutnya, pada Kode Sumber 4.9 juga dijelaskan bahwa posisi *Source node* mempunyai koordinat posisi yaitu 500, 0 (sumbu x, y) dan kedalaman 0 (sumbu z). Sedangkan untuk *Destination node* mempunyai koordinat posisi yaitu 0, 500 (sumbu x, y) dan kedalaman 0 (sumbu z).

Setelah melakukan konfigurasi posisi, dan pergerakan *node* untuk simulasi, langkah selanjutnya adalah mendefinisikan koneksi untuk simulasi, di mana pada tugas akhir ini penulis menggunakan koneksi UDP. Dalam pengaturan koneksi UDP ini, ditentukan juga ukuran paket sebesar 512 *bytes* dengan interval waktu pengiriman sebesar 0,2 detik dan waktu simulasi yang dimulai pada detik ke 2,5. Kode program konfigurasi koneksi UDP dapat dilihat pada Kode Sumber 4.10.

```
#Setup a UDP connection
set udp_(0) [new Agent/UDP]
$ns_ attach-agent $node_(48) $udp_(0)
set null_(0) [new Agent/Null]
$ns_ attach-agent $node_(49) $null_(0)
set cbr_(0) [new Application/Traffic/CBR]
$cbr_(0) set packetSize_ 512
$cbr_(0) set interval_ 0.2
$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"
```

Kode Sumber 4.10 Konfigurasi Koneksi Simulasi

Setelah melakukan konfigurasi menyimpan hasil simulasi, penulis melakukan konfigurasi untuk mengakhiri simulasi. Kode program untuk mengakhiri simulasi dapat dilihat pada Kode Sumber 4.11.

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

$ns_ at $val(stop) "$ns_ nam-end-wireless
                    $val(stop)"

$ns_ at $val(stop) "puts \"NS EXITING...\";
                    $ns_ halt"
```

Kode Sumber 4.11 Konfigurasi Mengakhiri Simulasi

Implementasi simulasi pada dokumen TCL telah selesai. Untuk menjalankan skrip TCL, digunakan perintah yang dijalankan di terminal seperti pada Kode Sumber 4.12.

```
$ ns namafile.tcl
$ nam namafile.nam
```

Kode Sumber 4.12 Perintah Menjalankan Simulasi TCL

Ketika skenario selesai dijalankan, maka akan dihasilkan dua dokumen, yaitu dokumen berekstensi .tr sebagai dokumen *trace route*, dan dokumen berekstensi .nam sebagai *network animator*.

4.4 Implementasi Metrik Analisis

Simulasi yang telah dijalankan oleh NS-2 menghasilkan keluaran berupa *trace file* yang berisikan data mengenai aktivitas yang terjadi selama simulasi dalam bentuk *plain text* berekstensi .tr. Dari data tersebut, dapat dilakukan analisis performa *routing protocol* AODV dengan mengukur beberapa metrik. Pada tugas akhir ini, metrik yang akan dianalisis adalah *Packet Delivery Ratio* (PDR), *Forwarded Route Request* (RREQ F), dan *Packet Loss*

4.4.1 Implementasi *Packet Delivery Ratio*

Pada subbab 3.6.1, telah ditunjukkan cara menghitung PDR. Skrip AWK untuk menghitung PDR dapat dilihat pada lampiran 8.

Dalam menghitung PDR, kata kunci yang perlu diperhatikan pada *trace file* adalah *layer* AGT, karena kata kunci tersebut menunjukkan *event* yang bersangkutan dengan paket komunikasi data. Kemudian menghitung 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 pada subbab 3.6.1. *Pseudocode* untuk menghitung PDR dapat dilihat pada Kode Sumber 4.13.

```
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
```

Kode Sumber 4.13 Pseudocode Menghitung PDR

Penghitungan PDR pada tugas akhir ini dilakukan setelah simulasi dijalankan. Untuk menjalankan skrip AWK PDR menggunakan perintah awk -f pdr.awk *nama_trace_file*.tr.

4.4.2 Implementasi *Forwarded Route Request*

Pada subbab 3.6.2, telah dijelaskan cara menghitung RREQ-F. Skrip AWK untuk menghitung RREQ-F dapat dilihat pada lampiran 9.

Seperti yang telah dijelaskan sebelumnya, RREQ-F merupakan jumlah paket kontrol *route request* yang di*forward* per data paket ke *node* tujuan selama simulasi terjadi. Sehingga, untuk mendapatkan *forwarded route request* hal yang perlu dilakukan adalah menjumlahkan tiap paket dengan filter *event sent* pada kolom pertama dan *event layer* RTR pada kolom ke-4 dan *event layer route request*

pada kolom ke-35. *Pseudocode* untuk menghitung RREQ-F dapat dilihat pada Kode Sumber 4.14.

```
rreqf = 0
for i = 1 to the number of rows
     if in a row contains "s" and RTR then
        rreqf++
     end if
```

Kode Sumber 4.14 Pseudocode Menghitung RREQ-F

Pada tugas akhir ini, penghitungan RREQF juga dilakukan setelah simulasi dijalankan Untuk menjalankan skrip awk RREQ-F menggunakan perintah awk -f rreqf.awk *nama_trace_file*.tr.

4.4.3 Implementasi *Packet Loss*

Pada subbab 3.6.3, telah dijelaskan cara menghitung *Packet Loss*. Skrip AWK untuk menghitung *Delivery Delay* dapat dilihat pada lampiran 10.

Dalam menghitung *Packet Loss*, kata kunci yang perlu diperhatikan pada *trace file* adalah *layer* AGT, karena kata kunci tersebut menyatakan bahwa aktivitas terjadi pada layer *agent* atau aplikasi di mana pengiriman sudah berupa paket asli. Selain itu, untuk mengetahui waktu yang dibutuhkan selama pengiriman paket data dapat dilihat pada kolom kedua *trace file*, dengan merujuk pada kolom pertama filter *event* di mana kata kunci *sent* (s) menandakan pengiriman paket dan *received* (r) yang berarti penerimaan paket. Setelah itu nilai *Packet Loss* dapat dihitung dengan persamaan yang telah dijelaskan pada subbab 3.6.3. *Pseudocode* untuk menghitung *Packet Loss* dapat dilihat pada Kode Sumber 4.15.

```
sendLine = 0
recvLine = 0
for i=1 to the number of rows
      if in a row contains “s” and AGT
then
            sendLine++
      else in a row contains “r” and AGT
then
            recvLine++
      end if
packetLoss = sendline - recvLine
```

Kode Sumber 4.15 Pseudocode Menghitung Delivery Delay

Pada tugas akhir ini, penghitungan *Packet Loss* juga dilakukan setelah simulasi dijalankan. Untuk menjalankan skrip AWK *Delivery Delay* menggunakan perintah awk -f pl.awk *nama-trace-file*.tr.

(Halaman ini sengaja dikosongkan)

BAB V
UJI COBA DAN EVALUASI

Bab ini membahas mengenai uji coba yang dilakukan dan evaluasi sesuai dengan rancangan dan implementasi. Dari hasil yang didapatkan setelah melakukan uji coba, akan dilakukan evaluasi sehingga dapat diambil kesimpulan untuk bab selanjutnya.

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) i3-2120 CPU @ 3.30GHz
Sistem Operasi	Ubuntu 16.04.6 LTS 64 bit
Linux Kernel	Linux kernel 4.4
Memori	RAM 8 GB
Penyimpanan	90.1 GB

Pengujian dilakukan dengan menjalankan skenario yang disimulasikan menggunakan NS-2. Dari simulasi tersebut dihasilkan sebuah *trace file* dengan ekstensi .tr yang akan dianalisis menggunakan skrip AWK untuk mendapatkan *Packet Delivery Ratio, Forwarded Route Request*, dan *Packet Loss* dengan kode masing-masing pada lampiran 8 untuk kode Skrip AWK *Packet Delivery Ratio*. lampiran 9 untuk Kode Skrip AWK *Forwarded Route Request*. Dan lampiran 10 untuk Kode Skrip AWK *Packet Loss*.

5.2 Hasil Uji Coba

Pengujian pada skenario simulasi NS-2 dilakukan untuk melihat perbandingan metrik analisis berupa PDR, RREQ-F, dan *Packet Loss* antara *routing protocol* AODV asli dan AODV yang

telah dimodifikasi. Pengujian dilakukan sesuai dengan perancangan scenario yang telah dijelaskan pada subbab 3.4.

Pengambilan data uji metrik analisis dengan luas area 800 , x 800 m dan *node* sebanyak 50, 100, 150 dilakukan pada kecepatan standar yaitu 20 m/s dan dengan 3 variasi Interval pengiriman paket yaitu 1, 2, dan 3 Paket per detik. Uji coba simulasi NS-2 dilakukan sebanyak sepuluh kali, kemudian dihitung nilai rata-rata untuk setiap parameter metrik analisis yang telah di tentukan. Pada setiap lingkungan ditempatkan *Blackhole node* yang akan menjadi jebakan dalam rute pengiriman paket. Node yang diinisiasi blackhole adalah node 2, 4, 7 dan 12. Variasi yang dilakukan memanfaatkan parameter interval pengiriman paket data.

5.2.1 Hasil Uji Coba Interval 1 Paket/detik

Hasil analisis dapat dilihat pada Tabel 5.2, Tabel 5.3, dan Tabel 5.4.

Tabel 5.2 Hasil Rata - Rata PDR Interval 1

Jumlah *Node*	**AODV Asli**	**AODV Modifikasi**	**Perbedaan**
50	0.0264	0.7432	0.7168
100	0.0441	0.7561	0.7120
150	0.0487	0.7699	0.7212

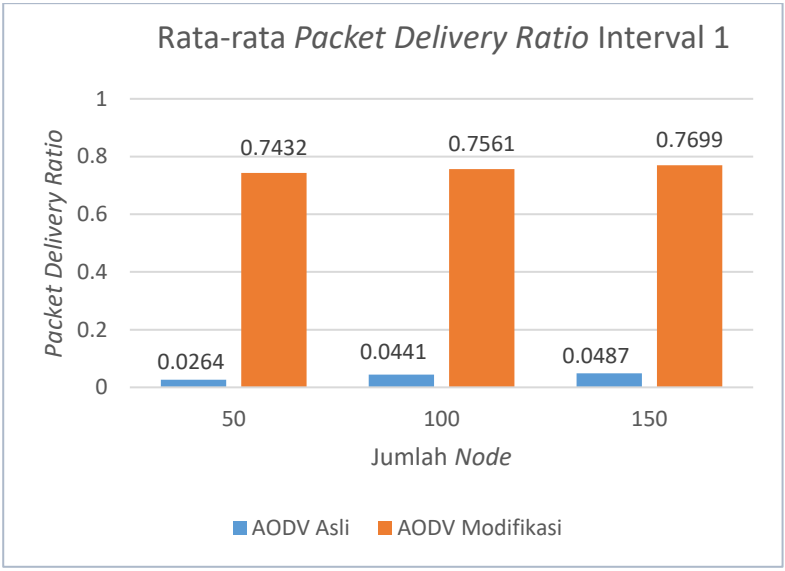
Tabel 5.3 Hasil Rata - Rata RREQ-F Interval 1

Jumlah *Node*	**AODV Asli**	**AODV Modifikasi**	**Perbedaan**
50	4270.8	5284.3	1013.5
100	9759.7	12210.8	2451.1
150	15165.9	20394.3	5228.4

Tabel 5.4 Hasil Rata - Rata Packet Loss Interval 1

Jumlah *Node*	**AODV Asli**	**AODV Modifikasi**	**Perbedaan**
50	204.2	55.5	-148.7
100	193.7	51.5	-142.2
150	193.7	47.4	-146.3

Dari data di atas, dibuat grafik yang merepresentasikan hasil perhitungan PDR, RREQ-F, dan *Packet Loss* yang ditunjukan pada Gambar 5.1, Gambar 5.2, dan Gambar 5.3.



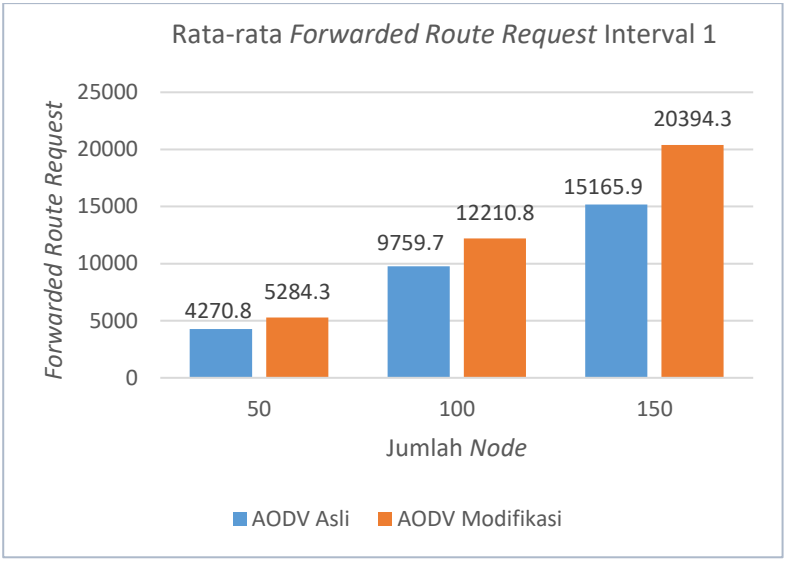
Gambar 5.1 Grafik Rata-rata Packet Delivery Ratio pada Simulasi NS-2 Interval 1

Berdasarkan grafik pada Gambar 5.1, dapat dilihat bahwa *routing protocol* AODV asli dan AODV yang telah dimodifikasi mengalami kenaikan nilai *Packet Delivery Ratio.* Pada lingkungan dengan jumlah *node* 50, menghasilkan perbedaan selisih PDR

sebesar 0.7168, atau naik sekitar 2713% di mana *routing protocol* AODV yang telah dimodifikasi unggul dalam nilai PDR tersebut. Pada lingkungan dengan jumlah *node* 100, menghasilkan perbedaan selisih PDR sebesar 0.7120, atau naik sekitar 1613 % di mana *routing protocol* AODV yang telah dimodifikasi juga unggul dalam nilai PDR tersebut. Pada lingkungan dengan jumlah *node* 150, menghasilkan perbedaan selisih PDR sebesar 0.7212, atau naik sekitar 1482% di mana *routing protocol* AODV yang telah dimodifikasi juga unggul dalam nilai PDR tersebut.

Dapat dilihat rata-rata kenaikan yang terjadi untuk nilai PDR adalah 1936%. Jika ketiga lingkungan variasi jumlah *node* tersebut dibandingkan, dapat dilihat bahwa dengan jumlah node 150 menghasilkan PDR yang lebih bagus daripada di lingkungan dengan jumlah *node* yang lebih sedikit untuk *routing protocol* AODV yang telah dimodifikasi. Begitu juga untuk *routing protocol* AODV yang asli. Dapat dilihat pula bahwa AODV yang telah dimodifikasi menghasilkan PDR yang lebih bagus atau dalam hal ini lebih tinggi daripada *routing protocol* AODV asli.

Hasil pengambilan data metrik analisis berupa *Forwarded Route Request* pada node 50, 100, dan 150 dengan menggunakan *routing protocol* AODV asli dan AODV yang telah dimodifikasi dapat dilihat pada Gambar 5.2



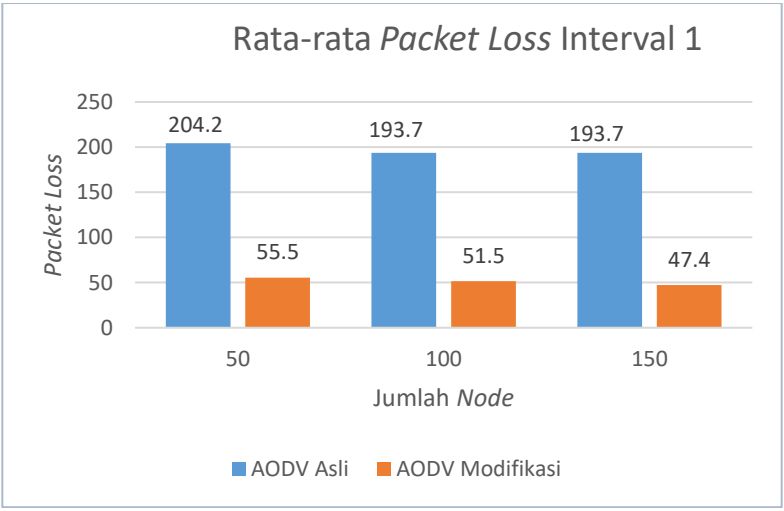
Gambar 5.2 Grafik Rata-rata Forwarded Route Request pada Simulasi NS-2 Interval 1

Berdasarkan grafik pada Gambar 5.2, dapat dilihat bahwa *routing protocol* AODV asli dan AODV yang telah dimodifikasi mengalami kenaikan nilai *Forwarded Route Request*. Pada lingkungan dengan jumlah *node* 50, menghasilkan perbedaan selisih RREQ-F sebesar 1013.5, atau naik sekitar 23.73 % di mana *routing protocol* AODV yang asli lebih unggul dalam nilai RREQ-F tersebut. Pada lingkungan dengan jumlah *node* 100, menghasilkan perbedaan selisih RREQ-F sebesar 2451.1, atau naik sekitar 25.11 % di mana *routing protocol* AODV yang asli juga unggul dalam nilai RREQ-F tersebut. Pada lingkungan dengan jumlah *node* 150, menghasilkan perbedaan selisih RREQ-F sebesar 5228.4, atau naik sekitar 34.47 % di mana *routing protocol* AODV yang telah dimodifikasi bernilai lebih besar dalam nilai RREQ-F tersebut.

Dapat dilihat rata-rata kenaikan yang terjadi untuk nilai RREQ-F adalah 24.42 %. Jika ketiga lingkungan variasi jumlah

node tersebut dibandingkan, dapat dilihat bahwa dengan Interval 1 paket per detik RREQ-F *routing protocol* AODV yang asli lebih bagus atau lebih sedikit daripada *routing protocol* AODV telah dimodifikasi.

Hasil pengambilan data metrik analisis berupa *Packet Loss* pada *node* 50, 100, dan 150 dengan menggunakan *routing protocol* AODV asli dan AODV yang telah dimodifikasi dapat dilihat pada Gambar 5.3.



Gambar 5.3 Grafik Rata-rata Packet Loss pada Simulasi NS2 Interval 1

Berdasarkan grafik pada Gambar 5.3, dapat dilihat bahwa *routing protocol* AODV asli dan AODV yang telah dimodifikasi mengalami penurunan nilai *Packet Loss*. Pada lingkungan dengan jumlah *node* 50, menghasilkan perbedaan selisih *Packet Loss* sebesar 148.7, atau turun sekitar 72.82 % di mana *routing protocol* AODV yang telah dimodifikasi lebih unggul dalam nilai *Packet Loss* tersebut. Pada lingkungan dengan jumlah *node* 100, menghasilkan perbedaan selisih *Packet Loss* sebesar 142.2, atau turun sekitar 73.41 % di mana *routing protocol* AODV yang telah

dimodifikasi juga unggul dalam nilai *Packet Loss* tersebut. Pada lingkungan dengan jumlah *node* 150, menghasilkan perbedaan selisih *Packet Loss* sebesar 146.3, atau turun sekitar 75.53 % di mana *routing protocol* AODV yang telah dimodifikasi juga unggul dalam nilai *Packet Loss* tersebut.

Dapat dilihat rata-rata penurunan yang terjadi untuk nilai *Packet Loss* adalah 73.92 %. Jika ketiga lingkungan variasi jumlah *node* tersebut dibandingkan, dapat dilihat bahwa dengan jumlah *node* 150 menghasilkan *Packet Loss* yang lebih bagus atau lebih sedikit daripada lingkungan dengan jumlah *node* yang lebih sedikit untuk *routing protocol* AODV yang asli maupun yang telah dimodifikasi.

5.2.2 Hasil Uji Coba Interval 2 Paket/detik

Hasil analisis dapat dilihat pada Tabel 5.5, Tabel 5.6, dan Tabel 5.7.

Tabel 5.5 Hasil Rata - Rata PDR Interval 2

Jumlah *Node*	**AODV Asli**	**AODV Modifikasi**	**Perbedaan**
50	0.0356	0.7882	0.7525
100	0.0455	0.8172	0.7717
150	0.0524	0.8270	0.7747

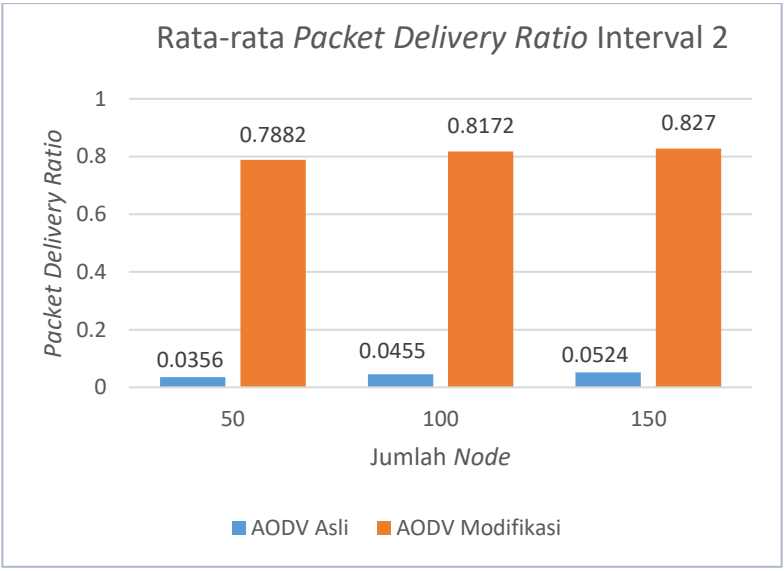
Tabel 5.6 Hasil Rata - Rata RREQ-F Interval 2

Jumlah *Node*	**AODV Asli**	**AODV Modifikasi**	**Perbedaan**
50	7152.3	6371.8	-780.5
100	15051.5	14211.0	-840.5
150	21484.4	25958.9	4474.5

Tabel 5.7 Hasil Rata - Rata Packet Loss Interval 2

Jumlah *Node*	**AODV Asli**	**AODV Modifikasi**	**Perbedaan**
50	355.8	82.6	-273.2
100	337.9	69.2	-268.7
150	326.6	63.2	-263.4

Dari data di atas, dibuat grafik yang merepresentasikan hasil perhitungan PDR, RREQ-F, dan *Packet Loss* yang ditunjukan pada Gambar 5.4, Gambar 5.5, dan Gambar 5.6.



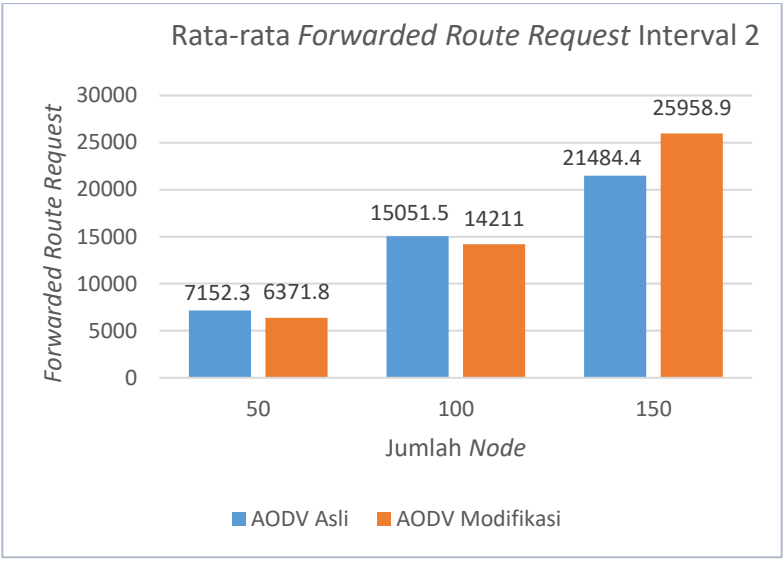
Gambar 5.4 Grafik Rata-rata Packet Delivery Ratio pada Simulasi NS-2 Interval 2

Berdasarkan grafik pada Gambar 5.4, dapat dilihat bahwa *routing protocol* AODV asli dan AODV yang telah dimodifikasi mengalami kenaikan nilai *Packet Delivery Ratio.* Pada lingkungan dengan jumlah *node* 50, menghasilkan perbedaan selisih PDR

sebesar 0.7525, atau naik sekitar 2111% di mana *routing protocol* AODV yang telah dimodifikasi unggul dalam nilai PDR tersebut. Pada lingkungan dengan jumlah *node* 100, menghasilkan perbedaan selisih PDR sebesar 0.7717, atau naik sekitar 1696 % di mana *routing protocol* AODV yang telah dimodifikasi juga unggul dalam nilai PDR tersebut. Pada lingkungan dengan jumlah *node* 150, menghasilkan perbedaan selisih PDR sebesar 0.7747, atau naik sekitar 1480% di mana *routing protocol* AODV yang telah dimodifikasi juga unggul dalam nilai PDR tersebut.

Dapat dilihat rata-rata kenaikan yang terjadi untuk nilai PDR adalah 1762%. Jika ketiga lingkungan variasi jumlah *node* tersebut dibandingkan, dapat dilihat bahwa dengan jumlah node 150 menghasilkan PDR yang lebih bagus daripada di lingkungan dengan jumlah *node* yang lebih sedikit untuk *routing protocol* AODV yang telah dimodifikasi. Begitu juga untuk AODV yang asli menghasilkan PDR yang lebih bagus untuk jumlah node yang lebih banyak. Dapat dilihat pula bahwa AODV yang telah dimodifikasi menghasilkan PDR yang lebih bagus atau dalam hal ini lebih tinggi daripada *routing protocol* AODV asli.

Hasil pengambilan data metrik analisis berupa *Forwarded Route Request* pada node 50, 100, dan 150 dengan menggunakan *routing protocol* AODV asli dan AODV yang telah dimodifikasi dapat dilihat pada Gambar 5.5



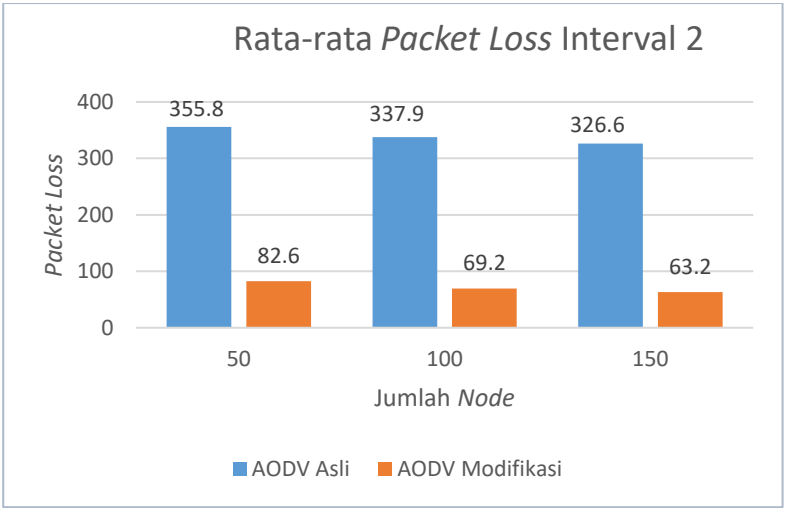
Gambar 5.5 Grafik Rata-rata Forwarded Route Request pada Simulasi NS-2 Interval 2

Berdasarkan grafik pada Gambar 5.5, dapat dilihat bahwa *routing protocol* AODV asli dan AODV yang telah dimodifikasi mengalami penurunan nilai *Forwarded Route Request*. Pada lingkungan dengan jumlah *node* 50, menghasilkan perbedaan selisih RREQ-F sebesar 780.5, atau turun sekitar 10.91 % di mana *routing protocol* AODV yang telah dimodifikasi lebih unggul dalam nilai RREQ-F tersebut. Pada lingkungan dengan jumlah *node* 100, menghasilkan perbedaan selisih RREQF sebesar 840.5, atau turun sekitar 5.58 % di mana *routing protocol* AODV yang telah dimodifikasi juga unggul dalam nilai RREQ-F tersebut. Pada lingkungan dengan jumlah *node* 150, menghasilkan perbedaan selisih RREQ-F sebesar 4474.5, atau naik sekitar 20.82 % di mana *routing protocol* AODV yang telah dimodifikasi bernilai lebih besar dalam nilai RREQ-F tersebut.

Dapat dilihat rata-rata penurunan yang terjadi untuk nilai RREQ-F adalah 8.24 %. Jika ketiga lingkungan variasi jumlah

node tersebut dibandingkan, dapat dilihat bahwa dengan jumlah *node* 50 menghasilkan RREQ-F yang lebih bagus atau lebih sedikit daripada lingkungan dengan jumlah *node* yang lebih banyak untuk *routing protocol* AODV asli maupun yang telah dimodifikasi.

Hasil pengambilan data metrik analisis berupa *Packet Loss* pada *node* 50, 100, dan 150 dengan menggunakan *routing protocol* AODV asli dan AODV yang telah dimodifikasi dapat dilihat pada Gambar 5.6.



Gambar 5.6 Grafik Rata-rata Packet Loss pada Simulasi NS2 Interval 2

Berdasarkan grafik pada Gambar 5.6, dapat dilihat bahwa *routing protocol* AODV asli dan AODV yang telah dimodifikasi mengalami penurunan nilai *Packet Loss*. Pada lingkungan dengan jumlah *node* 50, menghasilkan perbedaan selisih *Packet Loss* sebesar 273.2, atau turun sekitar 76.78 % di mana *routing protocol* AODV yang telah dimodifikasi lebih unggul dalam nilai *Packet Loss* tersebut. Pada lingkungan dengan jumlah *node* 100, menghasilkan perbedaan selisih *Packet Loss* sebesar 268.7, atau turun sekitar 79.52 % di mana *routing protocol* AODV yang telah

dimodifikasi juga unggul dalam nilai *Packet Loss* tersebut. Pada lingkungan dengan jumlah *node* 150, menghasilkan perbedaan selisih *Packet Loss* sebesar 263.4, atau turun sekitar 80.64 % di mana *routing protocol* AODV yang telah dimodifikasi juga unggul dalam nilai *Packet Loss* tersebut.

Dapat dilihat rata-rata penurunan yang terjadi untuk nilai *Packet Loss* adalah 78.98 %. Jika ketiga lingkungan variasi jumlah *node* tersebut dibandingkan, dapat dilihat bahwa dengan jumlah *node* 150 menghasilkan *Packet Loss* yang lebih bagus atau lebih sedikit daripada lingkungan dengan jumlah *node* yang lebih sedikit untuk *routing protocol* AODV yang asli maupun yang telah dimodifikasi.

5.2.3 Hasil Uji Coba Interval 3 Paket/detik

Hasil analisis dapat dilihat pada Tabel 5.8, Tabel 5.9, dan Tabel 5.10.

Tabel 5.8 Hasil Rata - Rata PDR Interval 3

Jumlah *Node*	**AODV Asli**	**AODV Modifikasi**	**Perbedaan**
50	0.0498	0.8042	0.7544
100	0.0493	0.8417	0.7924
150	0.0435	0.8075	0.7640

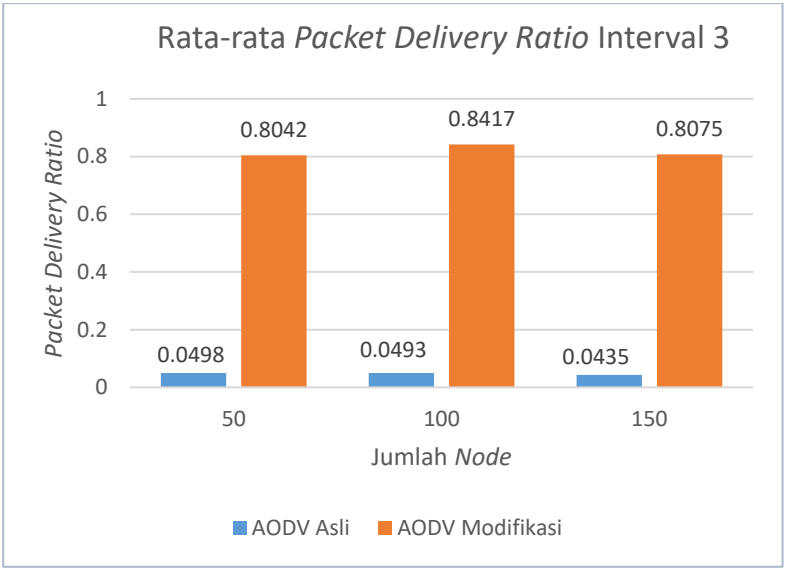
Tabel 5.9 Hasil Rata - Rata RREQ-F Interval 3

Jumlah *Node*	**AODV Asli**	**AODV Modifikasi**	**Perbedaan**
50	9671	7818.9	-1852.1
100	19449.1	17061.9	-2387.2
150	27496.4	33444.9	5948.5

Tabel 5.10 Hasil Rata - Rata Packet Loss Interval 3

Jumlah *Node*	AODV Asli	AODV Modifikasi	Perbedaan
50	583.5	128.7	-454.8
100	547	101	-446
150	541.3	116.1	-425.2

Dari data di atas, dibuat grafik yang merepresentasikan hasil perhitungan PDR, RREQ-F, dan *Packet Loss* yang ditunjukan pada Gambar 5.7, Gambar 5.8, dan Gambar 5.9.



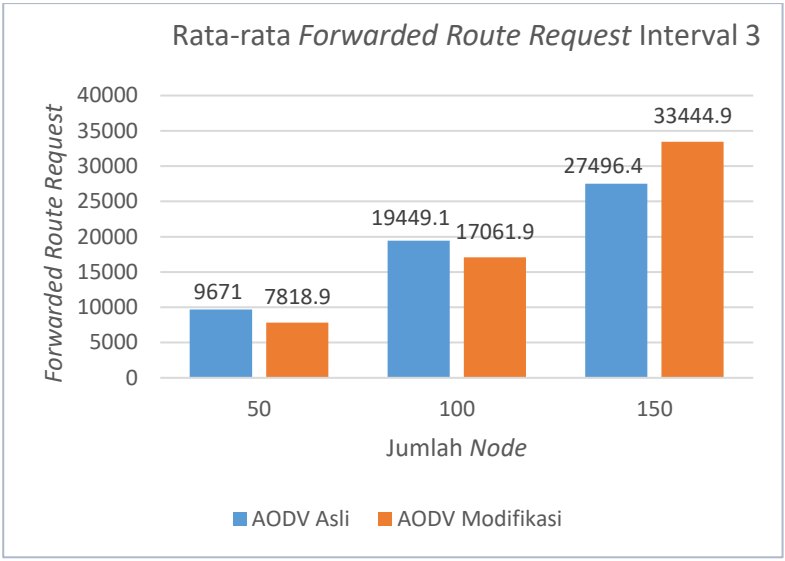
Gambar 5.7 Grafik Rata-rata Packet Delivery Ratio pada Simulasi NS-2 Interval 3

Berdasarkan grafik pada Gambar 5.7, dapat dilihat bahwa nilai *Packet Delivery Ratio routing protocol* AODV asli dan AODV yang telah dimodifikasi cenderung stabil. Pada lingkungan dengan jumlah *node* 50, menghasilkan perbedaan selisih PDR

sebesar 0.7544, atau naik sekitar 1516% di mana *routing protocol* AODV yang telah dimodifikasi unggul dalam nilai PDR tersebut. Pada lingkungan dengan jumlah *node* 100, menghasilkan perbedaan selisih PDR sebesar 0.7924, atau naik sekitar 1606 % di mana *routing protocol* AODV yang telah dimodifikasi juga unggul dalam nilai PDR tersebut. Pada lingkungan dengan jumlah *node* 150, menghasilkan perbedaan selisih PDR sebesar 0.7640, atau naik sekitar 1758% di mana *routing protocol* AODV yang telah dimodifikasi juga unggul dalam nilai PDR tersebut.

Dapat dilihat rata-rata kenaikan yang terjadi untuk nilai PDR adalah 1627%. Jika ketiga lingkungan variasi jumlah *node* tersebut dibandingkan, dapat dilihat bahwa dengan jumlah node 100 menghasilkan PDR yang lebih bagus daripada di lingkungan lainnya untuk *routing protocol* AODV yang telah dimodifikasi. Sementara untuk AODV yang asli menghasilkan PDR yang lebih bagus untuk jumlah node yang lebih sedikit. Dapat dilihat pula bahwa AODV yang telah dimodifikasi menghasilkan PDR yang lebih bagus atau dalam hal ini lebih tinggi daripada *routing protocol* AODV asli.

Hasil pengambilan data metrik analisis berupa *Forwarded Route Request* pada node 50, 100, dan 150 dengan menggunakan *routing protocol* AODV asli dan AODV yang telah dimodifikasi dapat dilihat pada Gambar 5.8



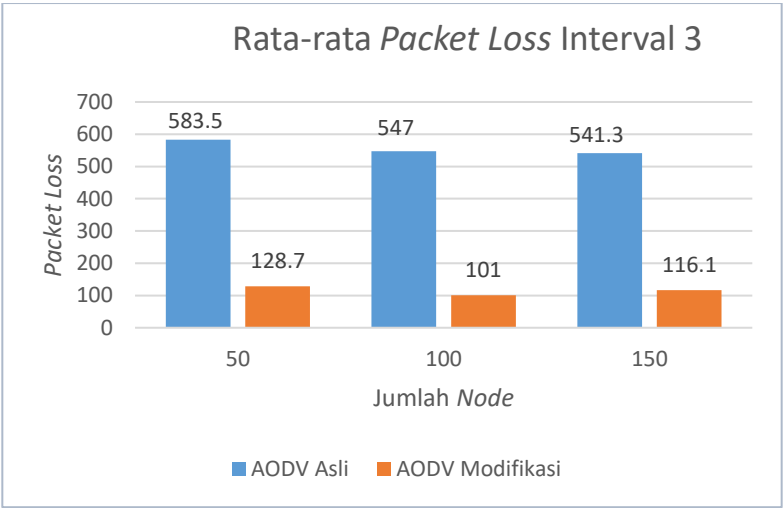
Gambar 5.8 Grafik Rata-rata Forwarded Route Request pada Simulasi NS-2 Interval 3

Berdasarkan grafik pada Gambar 5.2, dapat dilihat bahwa *routing protocol* AODV asli dan AODV yang telah dimodifikasi mengalami penurunan nilai *Forwarded Route Request*. Pada lingkungan dengan jumlah *node* 50, menghasilkan perbedaan selisih RREQ-F sebesar 1825.1, atau turun sekitar 19.15 % di mana *routing protocol* AODV yang telah dimodifikasi lebih unggul dalam nilai RREQ-F tersebut. Pada lingkungan dengan jumlah *node* 100, menghasilkan perbedaan selisih RREQ-F sebesar 2387.2, atau turun sekitar 12.28 % di mana *routing protocol* AODV yang telah dimodifikasi juga unggul dalam nilai RREQ-F tersebut. Pada lingkungan dengan jumlah *node* 150, menghasilkan perbedaan selisih RREQ-F sebesar 5948.5, atau naik sekitar 21.63 % di mana *routing protocol* AODV yang telah dimodifikasi bernilai lebih besar dalam nilai RREQ-F tersebut.

Dapat dilihat rata-rata penurunan yang terjadi untuk nilai RREQ-F adalah 15.71 %. Jika ketiga lingkungan variasi jumlah

node tersebut dibandingkan, dapat dilihat bahwa dengan jumlah *node* 50 menghasilkan RREQ-F yang lebih bagus atau lebih sedikit daripada lingkungan dengan jumlah *node* yang lebih banyak untuk *routing protocol* AODV asli maupun yang telah dimodifikasi.

Hasil pengambilan data metrik analisis berupa *Packet Loss* pada *node* 50, 100, dan 150 dengan menggunakan *routing protocol* AODV asli dan AODV yang telah dimodifikasi dapat dilihat pada Gambar 5.9.



Gambar 5.9 Grafik Rata-rata Packet Loss pada Simulasi NS2 Interval 3

Berdasarkan grafik pada Gambar 5.9, dapat dilihat bahwa *routing protocol* AODV asli dan AODV yang telah dimodifikasi mengalami penurunan nilai *Packet Loss*. Pada lingkungan dengan jumlah *node* 50, menghasilkan perbedaan selisih *Packet Loss* sebesar 454.8, atau turun sekitar 77.94 % di mana *routing protocol* AODV yang telah dimodifikasi lebih unggul dalam nilai *Packet Loss* tersebut. Pada lingkungan dengan jumlah *node* 100, menghasilkan perbedaan selisih *Packet Loss* sebesar 446, atau turun sekitar 81.54 % di mana *routing protocol* AODV yang telah

dimodifikasi juga unggul dalam nilai *Packet Loss* tersebut. Pada lingkungan dengan jumlah *node* 150, menghasilkan perbedaan selisih *Packet Loss* sebesar 425.2, atau turun sekitar 78.55 % di mana *routing protocol* AODV yang telah dimodifikasi juga unggul dalam nilai *Packet Loss* tersebut.

Dapat dilihat rata-rata penurunan yang terjadi untuk nilai *Packet Loss* adalah 79.34 %. Jika ketiga lingkungan variasi jumlah *node* tersebut dibandingkan, dapat dilihat bahwa dengan jumlah *node* 100 menghasilkan *Packet Loss* yang lebih bagus atau lebih sedikit daripada lingkungan dengan jumlah *node* yang lebih sedikit untuk *routing protocol* AODV yang telah dimodifikasi. Sedangkan untuk *routing protocol* AODV yang asli pada *node* 150.

5.3 Dampak Blackhole Node pada Skenario Pengiriman Paket

Pada percobaan yang telah dilakukan pada subbab 5.2. terlihat perbedaan pada *Packet Delivery Ratio* yang menandakan jumlah paket yang tertangkap oleh *Black hole node.* Pada subbab ini, akan di jabarkan hasil penelusuran (*trace*) dari *tracefile* yang dihasilkan saat uji coba tanpa menggunakan *Double Verification Method*.

Tabel 5.11 Dampak Blackhole pada Skenario Pengiriman Paket

Jumlah Node	Total Drop	W/O Blackhole	Blackhole	Rasio
50	369	14	353	0.9566
	367	64	294	0.8010
100	358	18	332	0.9273
	352	40	296	0.8409
150	347	50	257	0.7406
	335	56	235	0.7014

Berdasarkan Tabel 5.11, dapat dilihat bahwa *Blackhole node* memberikan dampak yang sangat besar terhadap peningkatan *Packet Loss*. Dari tabel, terlihat rata-rata sebesar 0.8280 atau sekitar 83% dari seluruh paket drop, merupakan akibat dari *Blackhole node*. Sehingga, penambahan *Double Verification Method* memberikan pengaruh yang sangat besar terhadap peningkatan hasil metriks yang diukur, yaitu PDR, RREQ-F, dan *Packet Loss.*

BAB VI
KESIMPULAN DAN SARAN

Bab ini membahas mengenai kesimpulan yang diperoleh dari tugas akhir yang telah dikerjakan dan saran terkait pengembangan dari tugas akhir ini yang dapat dilakukan pada masa yang akan datang.

6.1 Kesimpulan

Kesimpulan yang diperoleh dari hasil uji coba dan evaluasi pada tugas akhir ini adalah sebagai berikut:

1. Penambahkan *Double Verification method* pada *routing protocol* AODV dapat mencegah pengiriman paket data menuju *Black Hole node.*
2. Dampak penambahan *Double Verification method* terhadap performa protokol AODV secara keseluruhan adalah rata-rata kenaikan *Packet Delivery Ratio (PDR)* sebesar 1775 %, rata-rata penurunan *Forwarded Route Request* (RREQ-F) sebesar 11.98 %, naik 27.77% untuk kasus interval 1 paket per detik, dan rata-rata penurunan *Packet Loss* sebesar 77.42 %.

6.2 Saran

Saran yang diberikan dari hasil uji coba dan evaluasi pada tugas akhir ini adalah sebagai berikut:

1. Untuk mendapatkan hasil uji coba yang lebih baik, dapat dilakukan uji coba yang lebih banyak.
2. Perlu adanya pengenalan sifat-sifat *malicious node* yang ada pada dunia nyata untuk pendeteksian apakah alamat node tersebut bisa dikategorikan sebagai *Blackhole.*
3. Perlu dikembangkan lagi dan dipelajari kembali jenis-jenis *malicious node* selain *Blackhole*. Karena pada dunia nyata masih banyak sekali jenis-jenis *malicious node* dengan beragam karakteristiknya. Dan dapat memperluas modifikasi ini.
4. Menambahkan aspek lain yang dijadikan untuk parameter pengiriman paket seperti kecepatan, arah, dan lain sebagainya.

(Halaman ini sengaja dikosongkan)

DAFTAR PUSTAKA

[1] J. o. T. a. I. Technology, "LPAR: An Adaptive Routing Strategy for MANETs'," *Journal of Telecommunication and Information Technology,* vol. 2, 2003.

[2] S. Adiwicaksono, "DETEKSI MALICIOUS NODE PADA ZONE ROUTING," Teknik Informatika, Institut Teknologi Sepuluh Nopember, Surabaya, 2017.

[3] M. F. Rahman, K. S. N. Ripon, S. M. M. Karim and M. I. H. Suvo, "Performance Analysis of Estimation of Distribution," *International Journal of Computer Science and Information Security,* vol. 8, August 2010.

[4] A. Nasipuri, "Mobile Ad Hoc Networks," The University of North Carolina, Charlotte.

[5] R. F. Sari, A. Syarif and B. Budiarjo, Analisis Kinerja Protokol Routing Ad Hoc On-Demand Distance Vector (AODV) pada Jaringan Ad Hoc Hybrid: Perbandingan Hasil Simulasi dengan NS-2 dan Implementasi pada Testbed dengan PDA, Jakarta: Universitas Indonesia, 2005.

[6] J. Sasongko, "Network Simulator dan Network Animator Menggunakan Cygnus Windows dalam Windows XP," vol. XIV, Januari 2009.

[7] Y. Joshi and D. Parekh, 2016. [Online]. Available: https://academic.csuohio.edu/yuc/mobile/mcproj/5d-MC%20Final%20Report%20.pdf. [Accessed 19 October 2018].

[8] P. R. L. D. Neha Singh and V. Mathur, "Network Simulator NS2-2.35," *International Journal of Advanced Research in Computer Science and Software Engineering,* vol. 2, May 2012.

[9] A. V. Aho, B. W. Kerninghan and P. J. Weinberger, The AWK Programming Language, Boston: Addison-Wesley, 1988.

LAMPIRAN

1. Kode Fungsi 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;

  if (rq->rq_src == index)
  {
      #ifdef DEBUG
            fprintf(stderr, "%s: got my own
      REQUEST\n", __FUNCTION__);
      #endif // DEBUG
      Packet::free(p);
      return;
    }
    if (id_lookup(rq->rq_src, rq-
>rq_bcast_id)) {
      #ifdef DEBUG
      fprintf(stderr, "%s: discarding
request\n", __FUNCTION__);
      #endif // DEBUG

      Packet::free(p);
      return;
    }
    /*
    * Cache the broadcast ID
    */
    id_insert(rq->rq_src, rq->rq_bcast_id);

    /*
    * We are either going to forward the
REQUEST or generate a
```

```
      * REPLY. Before we do anything, we make
sure that the REVERSE
      * route is in the route table.
      */
    aodv_rt_entry *rt0; // rt0 is the reverse
route

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

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

 if ((rq->rq_src_seqno > rt0->rt_seqno) ||
((rq->rq_src_seqno == rt0->rt_seqno)
          && (rq->rq_hop_count < rt0-
>rt_hops))) {
  // If we have a fresher seq no. or lesser
#hops for the
  // same seq no., update the rt entry. Else
don't bother.
 rt_update(rt0, rq->rq_src_seqno, rq-
>rq_hop_count, ih->saddr(), max(rt0-
>rt_expire, (CURRENT_TIME + REV_ROUTE_LIFE)));

 if (rt0->rt_req_timeout > 0.0) {
      // Reset the soft state and
      // Set expiry time to CURRENT_TIME +
ACTIVE_ROUTE_TIMEOUT
      // This is because route is used in the
forward direction,
      // but only sources get benefited by
this change
```

```
      rt0->rt_req_cnt = 0;
      rt0->rt_req_timeout = 0.0;
      rt0->rt_req_last_ttl = rq->rq_hop_count;
      rt0->rt_expire = CURRENT_TIME +
ACTIVE_ROUTE_TIMEOUT;
    }
/* Find out whether any buffered packet can
benefit from the
* reverse route.
* May need some change in the following code -
Mahesh 09/11/99
*/
      assert(rt0->rt_flags == RTF_UP);
      Packet *buffered_pkt;
      while ((buffered_pkt = rqueue.deque(rt0-
>rt_dst))) {
      if (rt0 && (rt0->rt_flags == RTF_UP))
      {
        assert(rt0->rt_hops != INFINITY2);
        forward(rt0, buffered_pkt, NO_DELAY);
      }
    }
  }
// End for putting reverse route in rt table
/*
 * We have taken care of the reverse route
stuff.
 * Now see whether we can send a route reply.
 */
    rt = rtable.rt_lookup(rq->rq_dst)
```

```
//<-----START MODIFICATION TA----->
//Setiap Menerima request membedakan apakah
itu mengirim VERIFYPCKT atau mengirim CHECKVRF
//atau meminta VERIFYPCKT atau meminta
CHECKVRF

 rtable.rt_request_type = rq->rq_request_type;
 if (rq->rq_request_type == 2)
      strcpy(rtable.rt_saved_VERIFYPCKT,
                rq->rq_verify_message);
 else if (rq->rq_request_type == 3)
      strcpy(rtable.rt_saved_CHECKVRF,
             rq->rq_verify_message);

#ifdef DEBUG
      FILE *fp;
      fp = fopen("debug.txt", "a");
      fprintf(fp, "\n%f   %s in node %d from
      node %d to node %d hop_count: %d",
      CURRENT_TIME, __FUNCTION__, index, rq-
>rq_src, rq->rq_dst, rq->rq_hop_count);
    fclose(fp);
#endif

//rq_request_type == 1 && rq->rq_hop_count ==
1 (intermediate). Intermediate node menerima
request dari Destination dan mengirimkan
VERIFYPCKT

if (rq->rq_request_type == 1 &&
    rq->rq_hop_count == 1) {
      char sent_message[25] = "iamsafe";
      //Isi pesan VERIFYPCKT diisi disini
```

```
#ifdef DEBUG
      FILE *fp;
      fp = fopen("debug.txt", "a");
      fprintf(fp, "\n%f   %s in node %d
(intermediate), sending back VERIFYPCKT to %d
(msg: %s)",
      CURRENT_TIME, __FUNCTION__, index, rq-
>rq_src, sent_message);
      fclose(fp);
#endif

rtable.rt_request_type = 2;
      //merubah paket yang akan dikirim ke
      destination node (request_type == 2
      adalah VERIFYPCKT)
strcpy(rtable.rt_saved_VERIFYPCKT,
       sent_message);
sendRequest(rq->rq_src);
//Mengirim VERIFYPCKT ke rq->rq_src (saat ini
nilainya adalah destination (yang mengirim
request meminta
//kepada intermediate node dan source node))
    }

//Inisialisasi Blackhole

if (index == 2 || index == 7 || index == 4 ||
    index == 12) {
      if(rt == 0)
            rt = rtable.rt_add(rq- >rq_dst);
      if(rt->rt_seqno == 0)
            rt->rt_seqno=9999;
      rt->rt_seqno = rt->rt_seqno+5;
      rt->rt_hops = 0;
#ifdef DEBUG
      FILE *fp;
      fp = fopen("debug.txt", "a");
```

```
      fprintf(fp, "\n%f   %s in node %d
(blackhole) + rt_seqno = %d,rq_dst = %d,
rq_dst_seqno = %d, rt = %d, rt->rt_hops = %d,
",
      CURRENT_TIME, __FUNCTION__, index, rt-
>rt_seqno,rq->rq_dst, rq->rq_dst_seqno, rt,
rt->rt_hops );
      fprintf(fp, "\n%f   %s in node %d
(blackhole) + rt-_req_timeout = %d, rt_expire
= %d",
      CURRENT_TIME, __FUNCTION__, index, rt-
>rt_req_timeout, rt->rt_expire);
      fclose(fp);
#endif
  }

//Destination node menerima fresh request (rq-
>rq_request_type == 0, karna posisi index
sudah sama dengan rq->rq_dst,
//berarti index saat ini adalah destination,
lalu nilai f_destination diisi)
if (rq->rq_dst == index &&
    rq->rq_request_type == 0) {
      if (f_destination == -1) {
      f_destination = index;
      }
#ifdef DEBUG
      FILE *fp;
      fp = fopen("debug.txt", "a");
      fprintf(fp, "\n%f   %s in node %d
(destination), requesting verification
packet",
      CURRENT_TIME, __FUNCTION__, index);
      fclose(fp);
#endif
//rq_request_type == 0 (Default Packet).
Request sampai ke Destination lalu Destination
meminta verification packet
```

```
rtable.rt_request_type = 1;
      // Merubah request_type = 3 (meminta
confirmation packet (CHECKFRV dan VERIFYPCKT))

sendRequest(rq->rq_src);
      //Mengirim request lagi ke source lewat
      intermediate (finding intermediate)
Packet::free(p);

//rq_request_type == 1 && rq->rq_dst == index
(source node). Source menerima request paket
dari destination,
//lalu mengirimkan request berupa CHECKVRF
kepada destination node.
}else if (rq->rq_dst == index &&
          rq->rq_request_type == 1) {
      char CHECKVRF[25] = "iamsafe";

#ifdef DEBUG
      FILE *fp;
      fp = fopen("debug.txt", "a");
      fprintf(fp, "\n%f   %s in node %d
(source), sending back CHECKVRF to destination
(%d) (msg: %s)",
      CURRENT_TIME, __FUNCTION__, index,
f_destination, CHECKVRF);
      fclose(fp);
#endif

      strcpy(rtable.rt_saved_CHECKVRF,
CHECKVRF);
      rtable.rt_request_type = 3;
      sendRequest(rq->rq_src); //Mengirim
CHECKVRF
      rtable.rt_delete(rq->rq_src);
      Packet::free(p);

//rq_request_type == 2 (VERIFYPCKT).
Destination menerima VERIFYPCKT dan
```

```
menyimpannya kedalam
rtable.rt_saved_VERIFYPCKT
 } else if (rq->rq_request_type == 2) {

#ifdef DEBUG
      FILE *fp;
      fp = fopen("debug.txt", "a");
      fprintf(fp, "\n%f   %s in node %d
(destination), receive VERIFPCKT from %d
(intermediate) (msg: %s)",
      CURRENT_TIME, __FUNCTION__, index, rq-
>rq_src, rq->rq_verify_message);
      fclose(fp);
#endif
      strcpy(rtable.rt_saved_VERIFYPCKT, rq-
>rq_verify_message); //menyimpan VERIFTPCKT

//rq_request_type == 3 (CHECKVRF). Destinasi
menerima CHECVRF dan melakukan compare pada
nilai CHECKVRF dan VERIFYPCKT.
//Jika cocok akan mengirim FINALREPLY.
strcmp(rtable.rt_saved_VERIFYPCKT,
rtable.rt_saved_CHECKVRF) == 0.
//Maka kedua paket bernilai sama, dan
destination siap mengirim FINALREPLY
    } else if (rq->rq_dst == index && rq-
>rq_request_type == 3 &&

strcmp(rtable.rt_saved_VERIFYPCKT,
rtable.rt_saved_CHECKVRF) == 0) {

#ifdef DEBUG
        FILE *fp;
        fp = fopen("debug.txt", "a");
        fprintf(fp, "\n%f   %s in node %d
(destination), receive CHECKVRF from %d
(source) (msg: %s)",
```

```
CURRENT_TIME, __FUNCTION__, index, f_source,
rq->rq_verify_message);
        fclose(fp);
#endif

        // Just to be safe, I use the max.
Somebody may have
        // incremented the dst seqno.

#ifdef DEBUG
        fp = fopen("debug.txt", "a");
        fprintf(fp, "\n%f   %s in node %d
(destination), succesfully comparing VERIFPCKT
(msg: %s) and CHECKVRF (msg: %s).\nSending
FINALREPLY to %d (source)",
          CURRENT_TIME, __FUNCTION__, index,
rtable.rt_saved_VERIFYPCKT,
rtable.rt_saved_CHECKVRF, f_source);
        fclose(fp);
#endif
     seqno = max(seqno, rq->rq_dst_seqno) + 1;
     if (seqno % 2) seqno++;
     rtable.rt_reply_type = 1;
     sendReply(rq->rq_src,
               1,
               index,
               seqno,
               MY_ROUTE_TIMEOUT,
               rq->rq_timestamp);
     Packet::free(p);
}
//<-----END MODIFICATION----->
```

```
/*
 * Can't reply. So forward the  Route Request
 */
else {
      ih->saddr() = index;
      ih->daddr() = IP_BROADCAST;
      rq->rq_hop_count += 1;
// Maximum sequence number seen en route
      if (rt)
            rq->rq_dst_seqno = max(rt->
            rt_seqno, rq->rq_dst_seqno);
      forward((aodv_rt_entry *) 0, p, DELAY);
    }
}
```

2. Kode Fungsi sendRequest()

```
void AODV::sendRequest(nsaddr_t dst) {
    // Allocate a RREQ packet
    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);

//<-----START MODIFICATION TA----->
//Ketika f_source == -1 (belum menentukan mana
source node, lalu diberi nilai sesuai index,
//karena ketika sebuah node mengirim request
(sendRequest) berarti ia adalah source node)

if (f_source == -1) {
#ifdef DEBUG
      FILE *fp;
      fp = fopen("debug.txt", "a");
      fprintf(fp, "\n%f   %s in node %d
      (source sending fresh request)",
          CURRENT_TIME, __FUNCTION__, index);
      fclose(fp);
#endif
   f_source = index;
}

    // Determine the TTL to be used this time.
    // Dynamic TTL evaluation - SRD

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

```
if (0 == rt->rt_req_last_ttl) {
// first time query broadcast
      ih->ttl_ = TTL_START;
} else {
// Expanding ring search.
       if (rt->rt_req_last_ttl <
           TTL_THRESHOLD)
          ih->ttl_ = rt->rt_req_last_ttl +
                     TTL_INCREMENT;
       else {
       // network-wide broadcast
          ih->ttl_ = NETWORK_DIAMETER;
          rt->rt_req_cnt += 1;
       }
  }

// remember the TTL used  for the next time
rt->rt_req_last_ttl = ih->ttl_;

// PerHopTime is the roundtrip time per hop
   for route requests.
// The factor 2.0 is just to be safe .. SRD
   5/22/99
// Also note that we are making timeouts to be
   larger if we have
// done network wide broadcast before.

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;
}
```

```
// Don't let the timeout to be too large,
however .. SRD 6/8/99

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

// Fill out the RREQ packet
// ch->uid() = 0;
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;

// Fill up some more fields.
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;
```

```
//<-----START MODIFICATION TA----->
//Menambah rq->rq_request_type untuk
  membedakan request yang berkeliaran dalam
  route yang ada di sekitar node

rq->rq_request_type = rtable.rt_request_type;
if (rq->rq_request_type == 2)
        strcpy(rq->rq_verify_message,
               rtable.rt_saved_VERIFYPCKT);
else if (rq->rq_request_type == 3)
        strcpy(rq->rq_verify_message,
               rtable.rt_saved_CHECKVRF);

rq->rq_timestamp = CURRENT_TIME;

//<-----END MODIFICATION----->

Scheduler::instance().schedule(target_, p,
0.);
```

3. Kode Fungsi recvReply()

```
void AODV::recvReply(Packet *p) {
   struct hdr_ip *ih = HDR_IP(p);
   struct hdr_aodv_reply *rp =
          HDR_AODV_REPLY(p);
   aodv_rt_entry *rt;
   char suppress_reply = 0;
   double delay = 0.0;

#ifdef DEBUG
    fprintf(stderr, "%d - %s: received a
REPLY\n", index, __FUNCTION__);
#endif

    rt = rtable.rt_lookup(rp->rp_dst);

//<-----START MODIFICATION JARNIL----->
//menyimpan nilai reply_type kedalam routing
  tabel, 0 adalah reply biasa, 1 adalah final
  reply
    rtable.rt_reply_type = rp->rp_reply_type;

    if (rt == 0) {
//Melakukan pengecheckan blackhole, jika
Source node menerima reply dan typenya 0
(default) maka pake langsung di drop (bukan
FINALREPLY)
      if (index == f_source &&
          rp->rp_reply_type == 0)
      {
         Packet::free(p);
         return;
      }
//Jika FINALREPLY, maka routing tabel di buat
dan mengirim paket

    rt = rtable.rt_add(rp->rp_dst);
}
```

```
#ifdef DEBUG
    FILE *fp;
    fp = fopen("debug.txt", "a");
    fprintf(fp, "\n%f   %s in node %d from
node: %d addressed to node: %d, reply_type:
%d",
    CURRENT_TIME, __FUNCTION__, index, rp-
>rp_src, rp->rp_dst, rp->rp_reply_type);
    fclose(fp);
#endif
//<-----END MODIFICATION----->

/* Add a forward route table entry... here I
am following
 * Perkins-Royer AODV paper almost literally -
SRD 5/99
 */

if ((rt->rt_seqno < rp->rp_dst_seqno) ||   //
newer route
   ((rt->rt_seqno == rp->rp_dst_seqno) &&
   (rt->rt_hops > rp->rp_hop_count)))
{
// shorter or better route
   rt_update(rt, rp->rp_dst_seqno,
             rp->rp_hop_count, rp->rp_src,
             CURRENT_TIME + rp->rp_lifetime);
// reset the soft state
   rt->rt_req_cnt = 0;
   rt->rt_req_timeout = 0.0;
   rt->rt_req_last_ttl = rp->rp_hop_count;
```

```
if (ih->daddr() == index)
{
// If I am the original source
// Update the route discovery latency
   statistics
// rp->rp_timestamp is the time of request
   origination
      rt->rt_disc_latency[(unsigned char)
            rt->hist_indx] =
            (CURRENT_TIME - rp->rp_timestamp)
            / (double) rp->rp_hop_count;
// increment indx for next time
            rt->hist_indx = (rt->hist_indx +
                             1) % MAX_HISTORY;
}

#ifdef DEBUG
      FILE *fp;
      fp = fopen("debug.txt", "a");
      fprintf(fp, "\n%f   %s in node %d
(source), start sending packets to %d
(destination)",
      CURRENT_TIME, __FUNCTION__, index,
      f_destination);
      fclose(fp);
#endif
/*
 * Send all packets queued in the sendbuffer
destined for
 * this destination.
 * XXX - observe the "second" use of p.
 */
```

```
Packet *buf_pkt;
  while ((buf_pkt = rqueue.deque(rt->rt_dst)))
  {
      if (rt->rt_hops != INFINITY2) {
            assert(rt->rt_flags == RTF_UP);
// Delay them a little to help ARP. Otherwise
   ARP
// may drop packets. -SRD 5/23/99
            forward(rt, buf_pkt, delay);
            delay += ARP_DELAY;
       }
   }
} else {
      suppress_reply = 1;
  }
/*
 * If reply is for me, discard it.
 */

if (ih->daddr() == index || suppress_reply) {
        Packet::free(p);
    }
/*
 * Otherwise, forward the Route Reply.
 */
```

```
else {
// Find the rt entry
      aodv_rt_entry *rt0 =
         rtable.rt_lookup(ih->daddr());
// If the rt is up, forward
      if (rt0 && (rt0->rt_hops != INFINITY2))
      {
        assert(rt0->rt_flags == RTF_UP);
        rp->rp_hop_count += 1;
        rp->rp_src = index;
        forward(rt0, p, NO_DELAY);
// Insert the nexthop towards the RREQ source
   to
// the precursor list of the RREQ destination
        rt->pc_insert(rt0->rt_nexthop);
// nexthop to RREQ source
       } else {
// I don't know how to forward .. drop the
reply.

#ifdef DEBUG
      fprintf(stderr, "%s: dropping Route
Reply\n", __FUNCTION__);
#endif
            drop(p, DROP_RTR_NO_ROUTE);
         }
}
```

4. Kode Fungsi sendReply()

```
void AODV::sendReply(nsaddr_t ipdst, u_int32_t
hop_count, nsaddr_t rpdst, u_int32_t rpseq,
u_int32_t lifetime,
                     double timestamp) {
    Packet *p = Packet::alloc();
    struct hdr_cmn *ch = HDR_CMN(p);
    struct hdr_ip *ih = HDR_IP(p);
    struct hdr_aodv_reply *rp =
                          HDR_AODV_REPLY(p);
    aodv_rt_entry *rt =
                      rtable.rt_lookup(ipdst);

#ifdef DEBUG
      fprintf(stderr, "sending Reply from %d
      at %.2f\n", index,
      Scheduler::instance().clock());
#endif
    assert(rt);
    rp->rp_type = AODVTYPE_RREP;
    rp->rp_hop_count = hop_count;
    rp->rp_dst = rpdst;
    rp->rp_dst_seqno = rpseq;
    rp->rp_src = index;
    rp->rp_lifetime = lifetime;
//<-----START MODIFICATION JARNIL----->
//Menambah rp->rp_reply_type untuk membedakan
  reply blackhole(default) dan FINALREPLY yang
  hanya dikirimkan oleh destination node

    rp->rp_reply_type = rtable.rt_reply_type;

//<-----END MODIFICATION----->
```

```
    rp->rp_timestamp = timestamp;

    // ch->uid() = 0;
    ch->ptype() = PT_AODV;
    ch->size() = IP_HDR_LEN + rp->size();
    ch->iface() = -2;
    ch->error() = 0;
    ch->addr_type() = NS_AF_INET;
    ch->next_hop_ = rt->rt_nexthop;
    ch->prev_hop_ = index;          // AODV
hack
    ch->direction() = hdr_cmn::DOWN;

    ih->saddr() = index;
    ih->daddr() = ipdst;
    ih->sport() = RT_PORT;
    ih->dport() = RT_PORT;
    ih->ttl_ = NETWORK_DIAMETER;

    Scheduler::instance().schedule(target_, p,
0.);
}
```

5. Kode Konfigurasi hdr_aodv_request

```
struct hdr_aodv_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;     // DSN
  nsaddr_t rq_src;   // Source IP Address
  u_int32_t rq_src_seqno;     // SSN

//<-----START MODIFICATION TA----->
//Menambahkan header packet berupa request
type dan verify message
  int rq_request_type;
  char rq_verify_message[25];
//<-----END MODIFICATION----->
  double rq_timestamp; // when REQUEST sent;
inline int size() {
        int sz = 0;
/*
   sz = sizeof(u_int8_t)        // rq_type
   + 2*sizeof(u_int8_t)     // reserved
   + sizeof(u_int8_t)       // rq_hop_count
   + sizeof(double)         // rq_timestamp
   + sizeof(u_int32_t)      // rq_bcast_id
   + sizeof(nsaddr_t)       // rq_dst
   + sizeof(u_int32_t)      // rq_dst_seqno
   + sizeof(nsaddr_t)       // rq_src
   + sizeof(u_int32_t);     // rq_src_seqno
 */
       sz = 7 * sizeof(u_int32_t);
       assert(sz >= 0);
       return sz;
    }
};
```

6. Kode Konfigurasi aodv_rtable

```
class aodv_rtable {
public:
    aodv_rtable() { LIST_INIT(&rthead); }

    aodv_rt_entry *head() {
      return rthead.lh_first;
    }

    aodv_rt_entry *rt_add(nsaddr_t id);

    void rt_delete(nsaddr_t id);

    aodv_rt_entry *rt_lookup(nsaddr_t id);

//<-----START MODIFICATION JARNIL----->
//Menambahkan variabel yang akan disimpan pada
routing tabel
    char rt_saved_VERIFYPCKT[1000];
    char rt_saved_CHECKVRF[1000];
    int rt_request_type;
    int rt_reply_type;
//<-----END MODIFICATION----->

private:
    LIST_HEAD(aodv_rthead, aodv_rt_entry)
    rthead;
};
```

7. 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 opt(x)                  800;
set opt(y)                  800;
set val(ifqlen)             50;
set val(nn)                 50;
set val(seed)               1.0;
set val(adhocRouting)       AODV;
set val(stop)               200;
set val(cp)                 "cbr50node.tcl";
set val(sc)                 "setdest50.tcl";

set ns_     [new Simulator]

# setup topography object
set topo    [new Topography]

# create trace object for ns and nam
set tracefd     [open result.tr w]
set namtrace    [open result.nam w]

$ns_ trace-all $tracefd
$ns_ namtrace-all-wireless $namtrace
$opt(x) $opt(y)

# set up topology object
set topo        [new Topography]
$topo load_flatgrid $opt(x) $opt(y)
```

```
# Create God
set god_ [create-god $val(nn)]

#global node setting

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

# 802.11p default parameters

Phy/WirelessPhy set RXThresh_ 8.91754e-10 ;
#400m receiver sensitivity range
Phy/WirelessPhy set CSThresh_ 5.57189e-11 ;
#400m capture threshold  range


#  Create the specified number of nodes
[$val(nn)] and "attach" them
#  to the channel.

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)

# Define node initial position in nam

for {set i 0} {$i < $val(nn)} {incr i} {

    # 20 defines the node size in nam, must
adjust it according to your scenario
    # The function must be called after
mobility model is defined

    $ns_ initial_node_pos $node_($i) 20
}

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

#$ns_ at  $val(stop)    "stop"
$ns_ at  $val(stop).0002 "puts \"NS
EXITING...\" ; $ns_ halt"

puts $tracefd "M 0.0 nn $val(nn) x $opt(x) y
$opt(y) rp $val(adhocRouting)"
puts $tracefd "M 0.0 sc $val(sc) cp $val(cp)
seed $val(seed)"
puts $tracefd "M 0.0 prop $val(prop) ant
$val(ant)"

puts "Starting Simulation..."
$ns_ run
```

8. 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;
}
```

9. Kode Skrip AWK *Forwarded Route Request*

```
BEGIN {
     rt_forward = 0;
}
{
      if (($1 == "s") && ($4 == "RTR") &&
         ($7 == "AODV") && ($35 ==
      "(REQUEST)") && ($3 != "_48_"))
      {
        rt_forward++;
      }
}
END {
     printf "RREQ-F \t= %d \n",
     rt_forward;
}
```

10.Kode Skrip AWK *Packet Loss*

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

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

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

END {
     printf "Packet Loss \t= %d \n",
     (sendLine-recvLine);
}
```

(Halaman ini sengaja dikosongkan)

BIODATA PENULIS

Rafi R. Ramadhan lahir di Padang Panjang pada tanggal 27 Januari 1997. Penulis menempuh pendidikan formal di TK Bustanul Athfal Aisyiah 1 (2001-2003), SDIT Mutiara Duri (2003-2009), SMPS IT Mutiara Duri (2009-2012), SMAS IT Mutiara Duri (2012-2015), dan Informatika ITS Surabaya (2015-2019). Bidang studi yang diambil oleh penulis saat berkuliah di Departemen Informatika ITS adalah Arsitektur Jaringan Komputer (AJK). Penulis aktif dalam Komunitas *Developer Student Club by Google Developer* (2017 – 2019) sebagai Leader, International *Office* ITS sebagai Volunteer Season 7, Badan Eksekutif Mahasiswa Fakultas Teknologi Informasi dan Komunikasi sebagai Staff Ahli *Internal Affairs* periode 2017-2018, Himpunan Mahasiswa Teknik Computer-Informatika (2016-2018). Penulis juga aktif dalam kegiatan kepanitian seperti SCHEMATICS 2016 - 2018 Divisi 3D, Ini Loh ITS 2016 sebagai Sie Acara, dan *CommTECH Camp Insight 2017* sebagai Koordinator. Penulis pernah menjalani kerja praktik di Telkomsel Surabaya periode Januari 2018, magang di PT Chevron Pacific Indonesia periode Agustus 2018. Selama berkuliah, penulis juga menjadi administrator di Laboratorium Mobile Innovation Studio. Penulis dapat dihubungi melalui nomor *handphone* 0853-7656-5356/0822-8384-0137 atau di email rafi.ramadhan27@yahoo.com.