PERBANDINGAN KINERJA PROTOKOL AOMDV DAN MP-DSR PADA *MOBILE AD HOC NETWORK* (MANET)

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Anugerah Wijaya
NIM: 155150201111095

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2021

PENGESAHAN

PERBANDINGAN KINERJA PROTOKOL AOMDV DAN MP-DSR PADA *MOBILE AD HOC NETWORK* (MANET)

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Anugerah Wijaya
NIM: 155150201111095

Skripsi ini telah diuji dan dinyatakan lulus pada
12 Juli 2021
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing 2
Ir. Primantara Hari Trisnawan, M.Sc.	Reza Andria Siregar, S.T, M.Kom.
NIP: 19680912 199403 1 002	NIP: 19790621 200604 1 003

Mengetahui
Ketua Jurusan Teknik Informatika

Achmad Basuki, S.T., M.MG., Ph.D.
NIP: 19741118 200312 1 002

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar referensi.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 12 Juli 2021

Anugerah Wijaya

NIM: 155150201111095

ABSTRAK

Anugerah Wijaya, Perbandingan Kinerja Protokol AOMDV dan MPDSR pada Mobile Ad Hoc Network (MANET)

Pembimbing: Ir. Primantara Hari Trisnawan, M.Sc. dan Reza Andria Siregar, S.T., M.Kom.

Pengembangan dari teknologi jaringan *ad-hoc* adalah *Mobile Ad-Hoc Network* (MANET). MANET terbentuk dari kumpulan node yang menggunakan antarmuka nirkabel untuk dapat melakukan komunikasi antara satu *node* dengan *node* lainnya. *Routing* protokol mampu mencakup pertukaran data untuk dapat memberikan jalur *routing* secara optimal. Dalam penelitian ini, protokol routing yang digunakan yaitu AOMDV dan MP-DSR. Saat *source node* protokol AOMDV ingin mengirim sebuah paket menuju *destination node*, proses *route discovery* dimulai ketika *source node* mengirim *route request* (RREQ) menuju *destination node* melalui beberapa *node* terdekatnya. Setiap *node* protokol MP-DSR mencari lebih dari satu path ke *node* tujuan, sehingga jika terjadi kesalahan pada saat transmisi, *node* sudah mempunyai jalur alternatif lain. Parameter kinerja routing protokol yang diukur berupa *packet delivery ratio*, *end to end delay*, *throughput, normalized routing load* dan *convergence time* dengan menggunakan *Network Simulator* 2. Hasil yang diperoleh pada penelitian ini adalah protokol *routing* AOMDV unggul pada nilai paramater *throughput, end to end delay* dan *normalized routing load* dengan nilai rata-rata throughput 26.200 kbps, nilai rata-rata end to end delay 159,3425 m/s dan normalized routing load 0,7272. Sedangkan, protokol MP-DSR unggul pada nilai parameter *packet delivery ratio* dan *convergence time*, dengan nilai rata-rata *packet delivery ratio* sebesar 0,998775 % dan *convergence time* 169.7896 m/s. *Routing* protokol AOMDV lebih unggul dari *routing* protokol MP-DSR pada setiap protokol berdasarkan nilai kinerja *Quality of Service* (QoS) terhadap penambahan *node* dan variasi ukuran paket data.

Kata Kunci: *MANET, AOMDV, MP-DSR, throughput, end to end delay, packet delivery ratio, normalized routing load, convergence time, Network Simulator* 2

ABSTRACT

Anugerah Wijaya, Perbandingan Kinerja Protokol AOMDV dan MPDSR pada *Mobile Ad Hoc Network* (MANET)

Supervisors: Ir. Primantara Hari Trisnawan, M.Sc. and Reza Andria Siregar, S.T., M.Kom.

The Development of the ad-hoc network technology is Mobile Ad-Hoc Network (MANET). MANET is formed from a collection of nodes that use a wireless interface to be able to communicate between one node and another node. Protocol routing competent cover data exchange to be able provide optimal routing path. In this research, the routing protocols are AOMDV and MP-DSR. When the source node AOMDV protocol wants to send a packet to the destination node, the route discovery process starts when the source node sends a route request (RREQ) to the destination node through its closest nodes. Each node MP-DSR protocol searches more than one path to the destination node, so if an error occurs while transmitting, the node already has another alternate path. The performance parameters of the routing protocols are measured in the form of packet delivery ratio, end to end delay, throughput and normalized routing load, and convergence time using Network Simulator 2.35. The results obtained on this research are AOMDV routing protocols superior at the throughput parameters value, end to end delay and normalized routing load with average throughput of 26,200 Kbps, end to end average delay of 159.3425 m/s and 0,7272 normalized routing load. Protocol MP-DSR, superior with the average value of packet delivery ratio and convergence time with the average value of packet delivery ratio of 0.998775% and 169.7896 m/s convergence time. Routing protocols AOMDV is superior to the routing of the MP-DSR protocol on each protocol based on the performance value of Quality of Service (QoS) against the addition of nodes and data packet size variations.

***Keywords**: MANET, AOMDV, MP-DSR, throughput, end to end delay, packet delivery ratio, normalized routing load, convergence time, Network Simulator 2*

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR LAMPIRAN

BAB 1 PENDAHULUAN

1.1 Latar Belakang

Mobile Ad-Hoc Network (MANET) merupakan jaringan *wireless multihop* yang terdiri dari kumpulan *mobile node* yang bersifat dinamik. Sistem yang digunakan pada MANET mampu mengatur diri sendiri serta dibentuk oleh sekumpulan *node* atau terminal yang dihubungkan oleh jalur-jalur *nirkabel*. Dalam suatu jaringan, *konektivitas* beberapa *node* dapat menghilang karena jarak yang terlalu jauh dan muncul *node* baru dalam satu waktu dikarenakan pergerakan bebas *node-node* tersebut (Benardi, 2009).

Jika *node-node* bergerak bebas maka akan menyebabkan topologi jaringan berubah. Keadaan ini mengakibatkan jaringan tidak memiliki infrastruktur atau dikenal dengan istilah jaringan *ad-hoc*. Pesan yang dikirim dalam lingkungan jaringan ini akan terjadi antara dua *node* dalam cakupan transmisi masing-masing. Dan juga secara tidak langsung dihubungkan oleh *multiple hop* melalui beberapa *node* perantara jika kedua *node* itu tidak dapat berhubungan atau berada di luar jangkauannya. *Node* pada MANET tidak hanya berperan sebagai pengirim atau penerima data saja, namun dapat juga difungsikan sebagai penghubung *node* yang lain. Untuk mengatur seluruh proses *routing* pada topologi MANET tidak memerlukan *router*/*node*, karena setiap *device* berfungsi sebagai *router* untuk menentukan arah yang akan di tentukan (Amilia, Marzuki, & Agustina, 2014). Sehingga pada proses komunikasi pada jaringan MANET dibutuhkan sebuah aturan berupa protokol *routing* untuk menentukan rute pengiriman paket, agar *node* dapat mengirimkan paket data yang dibutuhkan oleh jaringan MANET tersebut.

AOMDV merupakan pengembangan dari protokol *routing Ad-Hoc on Demand Distance Vector* (AODV) yang menerapkan konsep *multiple path*. Pada *AODV* saat *source node* ingin mengirim sebuah paket menuju *destination node*, proses *route discovery* dimulai ketika *source node* mengirim *route request* (RREQ) menuju *destination node* melalui beberapa *node* terdekatnya. Duplikat RREQ ditandai menggunakan nomor urut (*sequence number*) yang berbeda. *Node* terdekat menerima paket RREQ non-duplikat, lalu membuat sebuah *reverse path* menuju *source node* menggunakan *hop* sebelumnya dari *RREQ*. Inti dari metode routing AOMDV, yaitu dalam proses penentuan beberapa rute alternatif yang digunakan harus bersifat *loop-free* dan saling lepas (*disjoint*) (Marina, 2001).

Protokol *routing* MP-DSR adalah pengembangan dari protokol *routing* DSR. Sama halnya dengan protokol DSR semua paket data yang dikirim sudah berisi daftar dari *node* yang akan dilewati. Pada MP-DSR *node* sumber akan terus mengirim RREQ sampai RREQ tersebut bertemu dengan *node* tujuan. Node tujuan akan memberikan RREP *packet* berisi daftar dari *node* yang harus dilewati. Node tujuan akan menerima lebih dari 1 RREP. Pemilihan RREP pada *node* sumber bisa dilihat dari jumlah *hop count* yang lebih sedikit dan juga latensi (Sangi, 2010).

Protokol *routing* AOMDV dan MP-DSR memiliki perbedaan cara kerja pada proses *routing* jaringan yang menyebabkan adanya perbedaan kinerja untuk kedua protokol. Kedua *routing* protokol tersebut akan disimulasikan menggunakan Network Simulator 2 dan *xgraph* sebagai simulator. Hasil simulasi yang diuji dapat dijadikan sebagai gambaran kinerja untuk mengetahui sejauh mana cara kerja yang dihasilkan dari protokol AOMDV dan MP-DSR dengan menggunakan Network Simulator 2.

Mengacu pada permasalahan yang telah disampaikan, judul yang diangkat dalam penelitian ini adalah Perbandingan Kinerja Protokol AOMDV dan MP-DSR pada Mobile Ad Hoc Network (MANET). Diharapkan tema penelitian yang diangkat dapat memberikan hasil perbandingan untuk kerja *routing* protokol pada jaringan MANET.

1.2 Identifikasi Masalah

Dari latar belakang yang telah ditulis, terdapat identifikasi masalah yang dapat diangkat pada penelitian ini. Permasalahan tersebut diantaranya yaitu untuk mengukur parameter jaringan diantara dua protokol AOMDV dan MP-DSR yang mencakup *throughput*, *end to end delay*, *packet delivery ratio*, *normalized routing load*, dan *convergence time.* Protokol AOMDV dan MP-DSR sama-sama memliki *Routing Multipath* yang dimana protokol *routing* 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.

1.3 Rumusan Masalah

1. Bagaimana melakukan simulasi *mobile ad hoc network* dengan NS-2.35 (*Network Simulator*-2.35) ?
2. Bagaimana perbandingan kinerja yang dihasilkan dari masing-masing protokol *routing* ?
3. Bagaimana kinerja yang lebih baik pada saat melakukan pengiriman paket data antar protokol *routing* AOMDV dan MP-DSR ?

1.4 Tujuan

Tujuan:

1. Dapat melakukan implementasi protokol *routing* AOMDV dan MP-DSR dengan menggunakan NS-2.35 (*Network Simulator*-2.3.5).
2. Membandingkan kinerja yang dihasilkan dari masing-masing protokol *routing* AOMDV dan MP-DSR.

3. Mengetahui kinerja mana yang lebih baik pada saat melakukan pengiriman paket data antar protokol *routing* AOMDV dan MP-DSR.

1.5 Manfaat

Manfaat dari dilakukannya penelitian adalah sebagai berikut.

a. Penulis
   - Menambah pengetahuan serta pengalaman baru terkait dengan cara kerja *routing* protokol AOMDV dan MP-DSR.
   - Menambah pengetahuan dan pengalaman terkait dengan penerapan protokol AOMDV dan MP-DSR pada *Network Simulator*-2 terkhusus pada versi 2.35.

b. Pembaca
   - Mendapatkan wawasan terkait dengan *routing* protokol AOMDV dan MP-DSR.
   - Mendapatkan wawasan terkait kinerja yang lebih baik antara AOMDV dan MP-DSR.

1.6 Batasan Masalah

Untuk menghindari melebarnya pokok bahasan, penulis hanya memfokuskan pada ruang lingkup yang diterapkan, sebagai berikut.

1. Penelitian ini fokus pada dua protokol *routing* yaitu AOMDV dan MP-DSR.
2. Jumlah *node* yang di analisis sudah ditetapkan di awal sebelum dilakukan pengujian.
3. Waktu simulasi yang akan dijalankan sudah ditetapkan di awal sebelum dilakukan pengujian.
4. Parameter pengukur kinerja protokol *routing* (*Quality of Service*) dalam pengujian meliputi *Packet Delivery Ratio* (PDR), *End to End Delay*, *Normalized Routing Load*, *Throughput*, dan *Convergence Time*.
5. Simulasi skenario menggunakan *Network Simulator* versi 2.35.

1.7 Sistematika pembahasan

Sistematika pembahasan untuk proposal skripsi ini sebagai berikut:

BAB I PENDAHULUAN

Menjelaskan tentang latar belakang, merumuskan masalah, menjabarkan tujuan, menjelaskan manfaat penelitian, dan pembatasan masalah penelitian.

BAB II KAJIAN PUSTAKA

Membahas tentang kajian pustaka yang mendukung penelitian dan menjelaskan tentang dasar teori penelitian.

BAB III METODOLOGI PENELITIAN

Menjelaskan tentang langkah kerja yang harus dilakukan dalam penelitian, mulai dari studi literatur, perancangan dan implementasi lingkungan pengujian, analisis hasil, dan pengambilan kesimpulan.

BAB IV PERANCANGAN DAN IMPLEMENTASI LINGKUNGAN PENGUJIAN

Menjelaskan tentang tahapan perancangan lingkungan pengujian dan implementasinya dalam Network Simulator 2 (NS2) beserta proses pengukuran kinerja berdasarkan parameter yang telah ditentukan.

BAB V HASIL DAN PEMBAHASAN

Menjelaskan tentang hasil pengujian dari beberapa skenario yang telah dilakukan pada penelitian ini.

BAB VI PENUTUP

Menjelaskan tentang hasil kesimpulan dan saran yang didapat dari penelitian yang telah dilakukan.

BAB 2 LANDASAN KEPUSTAKAAN

2.1 Kajian Pustaka

Dalam menunjang proses pengerjaan skripsi ini, digunakan beberapa jurnal dan *paper* penelitian dari berbagi sumber. Penelitian yang pernah dilakukan oleh para peneliti sebelumnya diharapkan dapat memberikan data-data yang valid.

Penelitian yang dilakukan oleh (Sony Candra, 2013) membahas tentang kinerja protokol *routing Ad Hoc On-Demand Distance Vector* (AODV) dan *Optimized Link State Routing* (OLSR) pada jaringan *Mobile Ad Hoc Network* (MANET*)*. Penelitian ini menggunakan beberapa skenario diantaranya, kapasitas jaringan yaitu *node* yang digunakan sebanyak 10 *node* sampai 50 *node*, mobilitas dengan menggunakan *pause time* 20s, 40s, 60s, 80s, 100s dan perubahan volume trafik dilakukan dengan melakukan perubahan data *trafik rate* dimana simulasi akan dilakukan pada tingkat *trafik rate* 2, 5, dan 10 paket/detik. Kesimpulan yang didapat dari penelitian ini adalah hasil protokol OLSR lebih baik diimplementasi pada jaringan yang besar dan tingkt mobilitas yang tinggi. Protokol OLSR memiliki kelebihan *packet delivery end to end delay* dan *control overhead*, tetapi nilai *packet delivery ratio* masih rendah. Sedangkan protokol AODV memiliki kelebihan pada presentase *packet delivery ratio* yang relatif tinggi di berbagai kondisi jaringan yaitu berkisar di atas 86% sedangkan kekurangannya adalah nilai *control overhead* relatif rendah dan nilai *average end to end delay* yang sering berubah-ubah.

Penelitian selanjutnya yang berjudul Analisis Kinerja *Routing* Protokol AODV dan *DSDV* pada Manet (*Mobile Ad Hoc Network*) (Tanudjaya, 2016). Penelitian ini memiliki ukuran performansi yang digunakan dalam simulasi antara lain *routing overheadI, packet delivery ratio*, *dan average end to end delay* dengan skenario yang terdiri dari perubahan jumlah *node*, waktu jeda dan luas area jaringan yang berbeda. Pada simulasi MANET yang dilakukan, dapat disimpulkan DSDV secara umum lebih unggul dari AODV, dilihat dari nilai *average end to end delay, packet delivery ratio, dan routing overhead*nya. DSDV juga lebih cocok diaplikasikan pada jaringan *ad hoc* yang memiliki mobilitas rendah karena nilai PDR nya lebih tinggi lebih stabil dibandingkan dengan AODV tanpa tepengaruh dengan peningkatan jumlah *node*. Akan tetapi Performansi AODV lebih baik dibandingkan DSDV apabila diaplikasikan pada luas area jangkauan lebih luas.

Pada penelitian yang lain mengenai Analisis Kinerja Pola-Pola Trafik pada Beberapa Protokol *Routing* dalam jaringan MANET (Didik Imawan, 2009) dimana pada penelitian tersebut membandingkan antara protokol *Routing* AODV, DSR, dan DSDV dengan hasil dari penelitian tersebut protokol *routing* DSR memiliki kinerja yang lebih baik dari pada AODV dan DSDV, dimana kinerja tersebut dilihat dari perubahan kapasitas jaringan ditujukan dengan nilai *routing overhead* yang relatif kecil. Sedangkan untuk kekurangan DSR sendiri yaitu nilai *average delay* meningkat sangat besar pada peningkatan *volume* trafik dibandingkan dengan AODV dan DSDV cenderung lebih kecil pada beberapa tingkat *volume* trafik

Berdasarkan penelitian-penelitian di atas, dapat diketahui bahwa telah banyak dilakukan pengujian yang memanfaatkan protokol *routing* pada MANET maupun media pengujian berupa *Network Simulator*. Pada penelitian ini, difokuskan pada pengujian kinerja yang menghasilkan perbandingan performansi lebih baik antara protokol *routing Ad Hoc On-Demand Multipath Distance Vector* (AOMDV) dan *Multi*Path-Dynamic *Source Routing* (MP-DSR) pada *Mobile Ad Hoc Network* menggunakan *Network Simulator* 2.35.

2.2 Dasar Teori

2.2.1 *Mobile Ad Hoc Network*

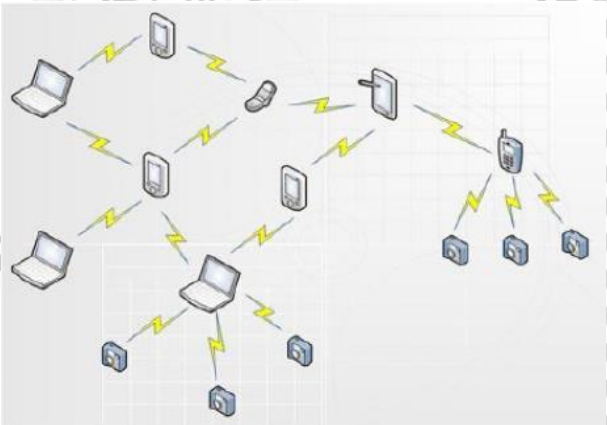
Mobile Ad Hoc Network (MANET) merupakan kumpulan perangkat nirkabel yang sifatnya dinamis dan temporari.Pada jaringan ini, setiap *node* dapat dijadikan suatu *host* ataupun *router* yang mampu meneruskan paket data menuju perangkat yang lain (Imawan, 2009). Setiap *node* yang berada dalam jaringan MANET bebas untuk bergerak ke segala arah dan *node* tersebut dapat dijadikan sebagai penghubung antara *node* satu degan *node* yang lainnya. Pada *Mobile Ad Hoc Network* (MANET), terdapat dua *node* lebih yang mampu berkomunikasi dengan *node* yang lain, akan tetapi masih tetap dalam jangkauan *node* tersebut. Beberapa karakteristik yang ada pada jaringan MANET, yaitu (Aarti, 2013):

a. Topologi yang dinamis

   Setiap *node* dapat bergerak secara bebas dan tidak dapat diprediksi.

b. *Scalability*

   Dapat menggunakan berbagai ukuran topologi jaringan sesuai dengan kebutuhan.



Gambar 2.1 Skema Ilustrasi MANET

Sumber : (Victor, 2013)

Sesuai Gambar 2.1 MANET terdiri dari berbagai *mobile* platform, yang disederhanakan sebagai *node* yang bergerak bebas secara acak. Pada *mobile ad hoc network*, terdapat beberapa keuntungan, beberapa di antaranya adalah (Demir, 2001):

a) Dukungan *backbone* infrastruktur tidak dibutuhkan pada jaringan ini, sehingga dengan mudah dapat diimplementasikan dan sangat berguna meskipun tidak terdapat infrastruktur atau infrastruktur tersebut tidak dapat berfungsi lagi.

b) *Mobile node* memiliki sifat dinamis, dapat mengakses informasi secara *real time* pada saat berhubungan dengan *mobile node* lainnya, sehingga pertukaran data dan pengambilan keputusan bisa segera dilakukan.

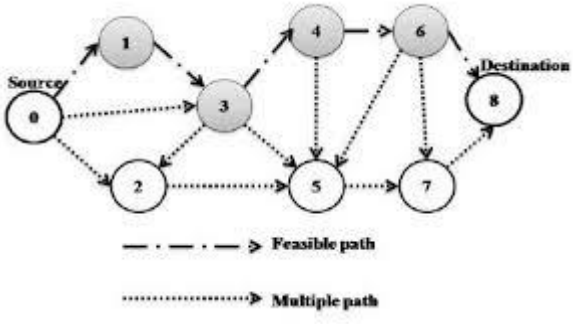
c) Fleksibel, karena jaringan ini memiliki sifat yang sementara.

d) Topologi dapat di rekonfigurasikan untuk jumlah *user* yang kecil maupun jumlah *user* yang besar sesuai dengan aplikasi dan *scalability*.

2.2.2 *Ad Hoc On-Demand Multipath Distance Vector (*AOMDV*)*

Protokol routing *Ad Hoc On-Demand Multipath Distance Vector* (AOMDV) adalah protokol routing pengembangan dari AODV (*Ad hoc On-demand Distance Vector*) yang termasuk dalam klasifikasi reaktif *routing* protokol. Berbeda dengan AODV, protokol AOMDV memiliki multipath dalam setiap proses pencarian rute. Rute yang diperoleh dijadikan rute utama dan rute cadangan berdasarkan minimal hop. Apabila rute yang diperoleh berisi *node* yang memiliki mobilitas menjauh dari jangkauan *node* lain, maka kerusakan rute dapat terjadi yang menyebabkan perpindahan rute.

Berbeda denga AODV proses *routing* AOMDV dapat memelihara beberapa rute melalui prosedur *route discovery* tunggal. Di dalam AODV seluruh duplikat RREQ dapat digunakan dalam upaya pencarian beberapa rute alternatif. Pada AOMDV propagasi RREQ dari *source node* menuju masing-masing duplikat RREQ dibangun menggunakan beberapa *reverse paths* baik pada *node intermediate* maupun *destination node*. Beberapa RREP yang melintasi *reverse path* ini akan kembali untuk membentuk beberapa jalur penerus menuju tujuan pada *node* sumber dan *node intermedite* dengan rute alternatif yang ditemukan sebagai upaya dalam mengurangi frekuensi *route discovery*. Inti dari proses *routing* AOMDV, yaitu dalam proses penentuan rute yang ditemukan harus bersifat *loop-free* dan saling lepas (disjoint).



Gambar 2.2 Skema Ilustrasi AOMDV

Sumber : (Ayyasamy, 2013)

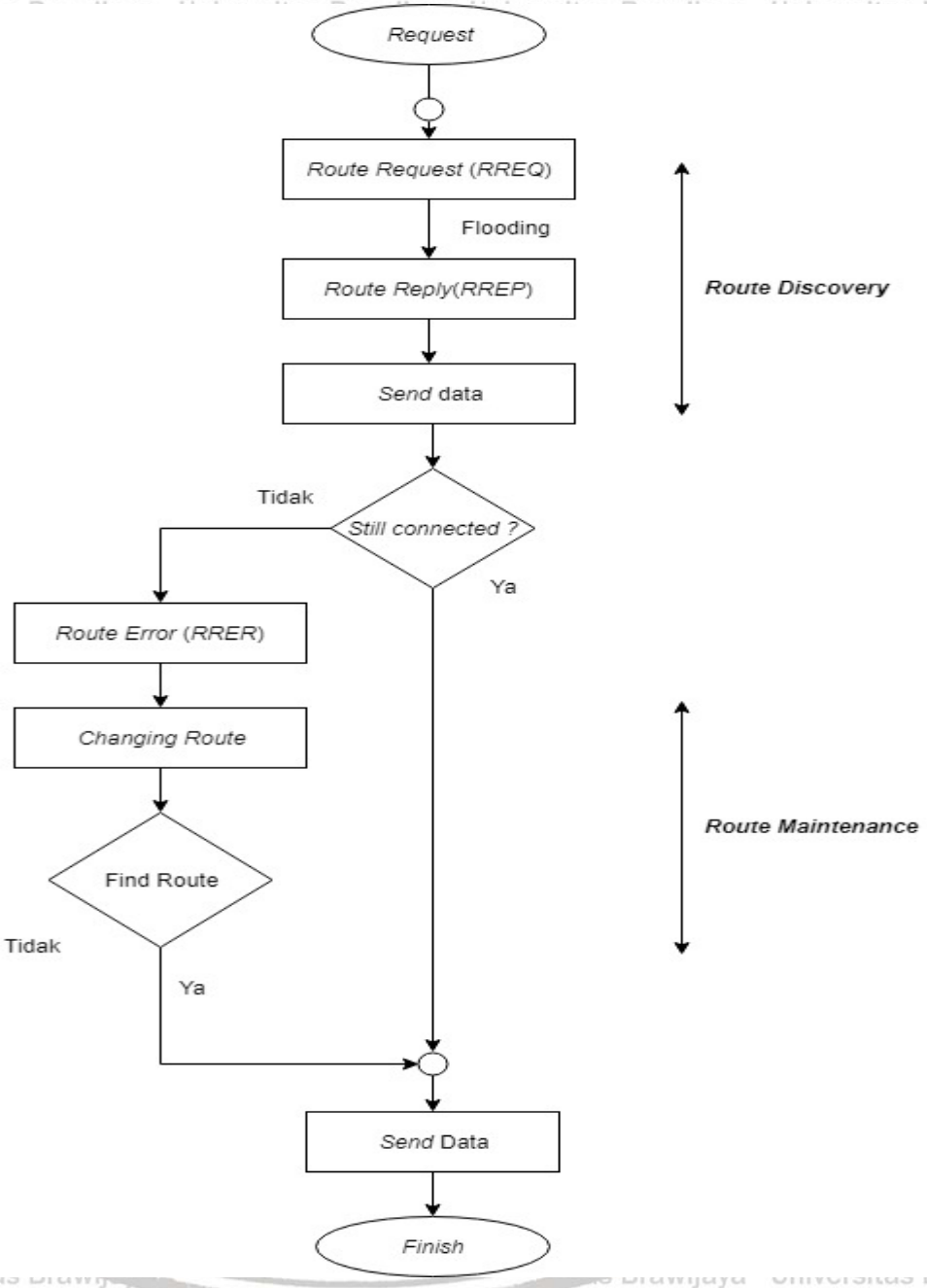
Gagasan utama dari terciptanya AOMDV adalah dalam upaya mengakumulasikan beberapa rute (*multiple paths*) selama proses *route discovery* berjalan. Hal tersebut dirancang untuk jaringan Ad-Hoc yang dinamis, di mana akan sering terjadi kegagalan dalam terhubung maupun terputus dalam jaringan. *Route discovery* sangat diperlukan untuk dalam menanggapi setiap rute yang terputus karena dalam proses *route discovery* yang berjalan kurang efisien akan menimbulkan *overhead* dan *latency* yang tinggi. Hal tersebut dapat dihindari dengan menyediakan beberapa rute yang serupa sehingga proses *route discovery* baru hanya akan diperlukan ketika semua jalur yang serupa menuju tujuan terputus. Protokol AOMDV memiliki dua komponen utama (Marina, 2001);

1. Proses pembaruan rute diatur dalam membangun dan memelihara beberapa jalur *loop-free* di setiap *node*.
2. Protokol didistribusikan untuk menemukan *link-disjoint paths*.

2.2.3 *Multipath-Dynamic Source Routing* (MP-DSR)

Protokol *routing Multipath-Dynamic Source Routing* (MP-DSR) adalah pengembangan dari protokol *routing* DSR. Sama halnya dengan protokol DSR semua paket data yang dikirim sudah berisi daftar dari *node* yang akan dilewati. Pada MP-DSR *node* sumber akan terus mengirim RREQ sampai RREQ tersebut bertemu dengan *node* tujuan. Node tujuan akan memberikan RREP *packet* berisi daftar dari *node* yang harus dilewati. Node tujuan akan menerima lebih dari 1 RREP. Pemilihan RREP pada *node* sumber bisa dilihat dari jumlah *hop count* yang lebih sedikit dan juga latensi.

Node sumber pada MP-DSR akan terus mengirim RREQ sampai RREQ tersebut bertemu dengan *node* tujuan. *Node* tujuan akan memberikan RREP paket berisi daftar dari *node* yang harus dilewati. *Node* tujuan akan menerima lebih dari 1 RREQ demikian juga *node* sumber akan akan menerima lebih dari 1 RREP. Pemilihan RREP pada *node* sumber bisa dilihat dari jumlah *hop count* yang lebih sedikit dan juga *latensi*. *Flowchart* dari *Route Discovery* dan *Route Maintenance* protokol MP-DSR sama dengan protokol DSR, perbedaanya terletak pada saat dilakukan *Route Discovery Path* yang dicari lebih dari 1 *path* sehingga ketika pada *Route Maintenance* terjadi masalah pada *path* utama maka *path* cadangan akan digunakan untuk mengirim data. Berikut merupakan *flowchart* dari protokol MP-DSR seperti pada gambar berikut:



Gambar 2.3 *Flowchart* Perancangan Protokol MP-DSR

Route Discovery yaitu *node* mengirimkan paket data ke tujuan yang belum diketahui rutenya. Sehingga sumber mengirim *Route Request* (RREQ). RREQ akan melakukan proses *flooding* yaitu proses pengiriman data atau control message ke setiap *node* pada jaringan untuk mencari rute ke tujuan. RREQ akan menyebar ke seluruh *node* dalam jaringan. Tiap *node* akan mengirim paket RREQ ke node lain kecuali *node* tujuan. Kemudian *node-node* yang menerima RREQ akan mengirim paket *Route Reply* (RREP) ke *node* yang mengirim RREQ tadi. Setelah rute ditemukan *node* sumber mulai mengirim data.

Route Maintenance yaitu mekanisme dimana sumber akan mendeteksi adanya perubahan topologi jaringan sehingga pengiriman paket mengalami kongesti. Hal

ini disebabkan karena salah satu *node* yang terdaftar dalam rute sebelumnya bergerak menjauh dari *range node* yang lain. Saat *route maintenance* mendeteksi masalah pada rute yang ada, paket *Route Error* (RERR) akan dikirim pada node pengirim. Pada saat RERR diterima, *hop* ke *node* yang menjauh akan dihilangkan dari *route cache*. Kemudian rute lain yang masih tersimpan di *cache* akan digunakan. Jika tidak ada rute lagi maka protokol MP-DSR akan melakukan proses *route discovery* lagi untuk menemukan rute baru.

2.2.4 *Multipath*

Pada protokol *routing* 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.

2.2.5 *Random way point*

Pada pergerakan *Random Way Point*. *Node-node* yang ada tersebar dan berjalan menuju arah yang acak (*random*). Model ini menyertakan *pause time* dalam *node* pergerakannya dan *node-node* dalam suatu area, bergeral acak menuju tujuannya dengan distribusi kecepatan antara 0 hingga kecepatan maksimum tertentu (m/s). Pada model mobilitas random waypoint (RWP), node didistribusikan secara acak dalam jaringan. Setiap node menyebar secara independen dari node yang lain. Prosedur yang ditempuh oleh RWP dalam memulai pergerakan yaitu setiap node memilih tujuan secara acak.

2.2.6 *Packet Delivery Ratio* (PDR)

Packet Delivery Ratio adalah perbandingan antara banyaknya jumlah paket yang diterima oleh *node* tujuan dengan total paket yang dikirimkan dalam suatu periode waktu tertentu. *Packet Delivery Ratio* menunjukkan tingkat keberhasilan sebuah protokol *routing* apabila semakin tinggi nilai PDR yang dihasilkan. Hal ini disebabkan keberhasilan sebuah protokol *routing* dalam melakukan pencarian dan pemeliharaan rutenya. *Packet Delivery Ratio* dapat dicari dengan persamaan berikut ini:

$$\text{PDR} = \frac{Pr}{\text{Ps}} x\ 100\ \%$$

$0 \leq t \leq T$ dimana : Pr = Paket yang diterima

Ps = Paket yang dikirim

T = Waktu simulasi (detik)

t = Waktu pengambilan sampel (detik)

Implementasi nilai dari *Packet Delivery Ratio* pada masing-masing protokol didapatkan dari perhitungan setiap *event* yang melakukan pengiriman dan penerimaan paket data yang dikirim dan paket data yang diterima akan dibedakan dari kolom pertama pada baris yang telah disaring. Setelah pembacaan setiap *event* dalam *trace file* selesai dilakukan, maka selanjutnya dilakukan perhitungan *Packet Delivery Ratio* (Denatama, 2016).

2.2.7 *End to End Delay*

End-to-end delay adalah jumlah waktu yang digunakan oleh sebuah paket ketika paket tersebut dikirimkan oleh sebuah *node* dan diterima pada *node* tujuan. *End to end delay* merupakan jumlah dari waktu pengiriman, propagasi, proses dan antrian dari suatu paket pada setiap *node* dalam jaringan

Faktor utama yang mempengaruhi *end to end delay* adalah waktu untuk penemuan rute. Hal ini berguna sebelum pesan dikirmkan, *node* harus mengetahui jalur atau rute yang akan dituju untuk mencapai *destination*. Faktor lain yang mempengaruhi *end to end delay* adalah *delay* proses. Ketika suatu *node* menerima sebuah pesan, *node* tersebut akan menganalisa *header* untuk mengetahui kepada siapa paket akan ditujukan, kemudian dilakukan pengecekan *node* untuk menentukan kemana paket tersebut akan diteruskan. *End to end delay* juga bisa terjadi ketika jumlah komunikasi mengalami penambahan, maka *node* lain sebagai perantara untuk mengirimkan layanan dari *node* sumber menuju *node* tujuan cenderung bertambah. Sehingga paket-paket dari layanan harus melalui *buffer* Hal ini yang membuat waktu pengiriman menjadi semakin lama. Untuk menghitung rasio *end to end delay* yang terjadi pada saat pengiriman paket, dapat dihitung dengan persamaan:

$$\text{Delay} = \sum_{i=0}^{n} \frac{t\ received\ [i] - t\ sent\ [i]}{jumlah\ paket\ yang\ dikirim}$$

Dimana : i = nomor paket yang berhasil diterima

t *received* = waktu ketika pakei i dikirim

t *sent* = waktu ketika paket i diterima

Implementasi nilai dari *end-to-end delay* pada masing-masing protokol didapatkan dari pembacaan baris pada *trace file*. Ada 5 (lima) kolom yang harus diperhatikan, yaitu kolom pertama yang berisi penanda *event* pengiriman atau penerimaan, kolom kedua yang berisi waktu terjadinya *event*, kolom keempat yang berisi informasi *layer* komunikasi paket, kolom keenam yang berisi ID paket dan kolom ketujuh yang berisi tipe paket (Anisia, 2016)

2.2.8 *Normalized Routing Load*

Normalized Routing Load adalah suatu nilai perbandingan antara banyaknya paket *routing* yang dikirim *source node* dan diteruskan (*forwarding*) dengan jumlah paket data yang diterima pada *destinatiton node*. Semakin tingggi nilai perbandingan paket *routing* terhadap paket data yang diterima maka semakin

kurang efisien kinerja suatu protocol *routing* (Perkins, 2001). Secara matematis *Normalized Routing Load* dapat dijabarkan dengan rumus sebagai berikut:

1. $$Normalized\ Routing\ Loads = \frac{Jumlah\ total\ paket\ routing\ (send\ \&\ forwarded)}{Jumlah\ total\ seluruh\ paket\ yang\ diterima}$$

2.2.9 *Throughtput*

Throughput merupakan banyaknya *byte* yang diterima dalam selang waktu tertentu dengan satuan *byte per second* yang merupakan kondisi data *rate* sebenarnya dalam suatu jaringan. Besarnya selang waktu pengukuran dapat mempengaruhi hasil gambaran perilaku jaringan. Secara umum, *throughtput* dinyatakan dalam persamaan seperti berikut:

$$Throughtput = \frac{Jumlah\ paket\ yang\ diterima}{Total\ waktu\ pengamatan}\ x\ ukuran\ paket$$

Ada beberapa hal yang mempengaruhi nilai *throughput*, salah satunya adalah ketahanan dari *link* yang terbentuk dan proses pencarian jalur hingga jalur tersebut terbentuk(Rianda & Rendy, 2016). Semakin tinggi nilai *throughput*, maka performa jaringan tersebut semakin baik. Penurunan *throughput* bisa terjadi karena adanya jarak antara *node* memiliki informasi *node* sumber yang cukup jauh sehingga melibatkan bertambahnya jumlah *hop routing*. Masing-masing *node* memiliki informasi *node* sangat berpengaruh pada *routing* protokol, karena apabila semakin besar jumlahnya maka dapat mempengaruhi trafik yang bisa menyebabkan *congestion* pada *node* perantara. *Congestion* tersebut dapat membuat beberapa paket mengalami *drop* sehingga mengakibatkan *throughput* koneksi utama menjadi turun.

2.2.10 *Convergence*

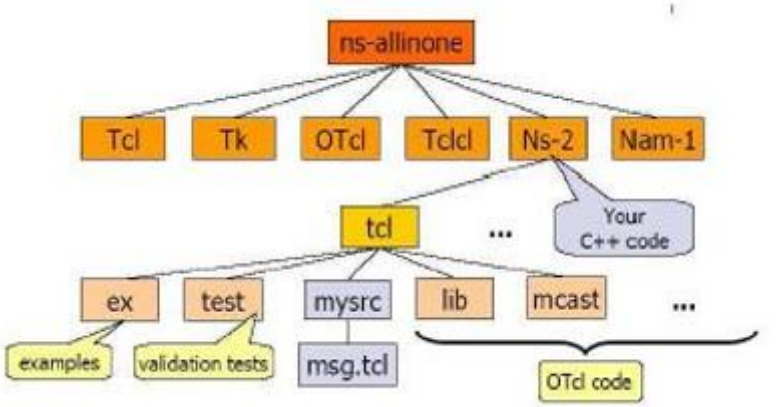
Convergence adalah proses *node-node* menyetujui rute yang paling optimal untuk meneruskan paket dan untuk melengkapi tabel *routing* dari masing-masing *node* tersebut. *Convergence time* merupakan kondisi dimana semua *node* mendapatkan kembali semua informasi tabel *routing* setelah terjadi perubahan topologi. *Convergence time* merupakan rata-rata waktu yang dibutuhkan secara keseluruhan setiap *node* mendapatkan *broadcast* dari *node* yang menyebarkan informasi tabel *routing*. *Convergence* terjadi sebagai hasil dari perubahan topologi jaringan, ketika *convergence* terjadi masing-masing *router* menjalankan algoritma *routing* untuk menghitung ulang *metric* dan membangun tabel *routing* yang baru berdasarkan informasi tersebut.

$$Convergence = \sum_i^n \sum_j^n \frac{tij}{n}$$

Untuk *i≠j* dan *tij* adalah waktu informasi *routing* diterima oleh *node destination* – waktu informasi *routing* dikirim oleh *node source*. Sedangkan n adalah jumlah *node* yang menerima pesan waktu informasi *routing*.

2.2.11 *Network Simulator* 2

Network Simulator 2 (NS-2) merupakan program simulasi jaringan yang bersifat *open source*. NS2 dibangun dari 2 bahasa pemrograman, yaitu bahasa pemrograman C++ yang digunakan untuk *event scheduler*, *protocol*, *network components*, dan Tcl/OTcl yang merupakan bahasa pemrograman untuk menulis *script* simulasi.



Gambar 2.4 Komponen Pembangun NS2

Sumber : (Issariyakul, 2013)

Penjelasan dari Gambar 2.2 adalah komponen-komponen NS-2 yang terdiri dari:

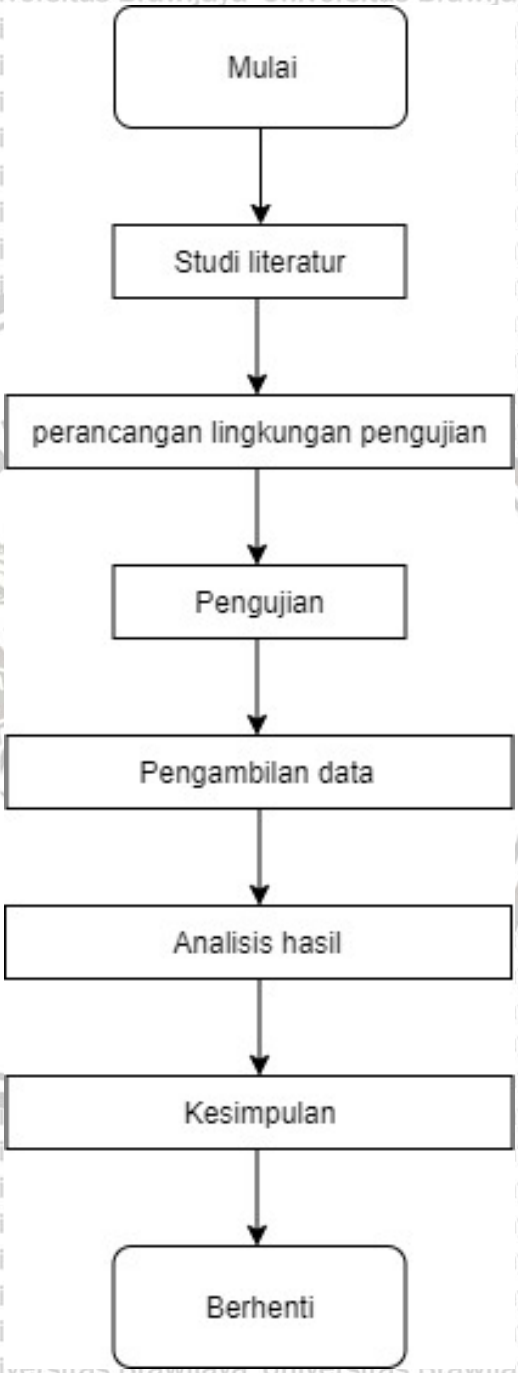
1. Tcl (*Tool command language*) merupakan *scripting programming* untuk konfigurasi *network simulator.*
2. Otcl (*Object Tcl*) merupakan Tcl *Interpreter* yang melakukan inisiasi *event scheduler*, membangun teopologi jaringan berbasis objek serta memberitahu sumber trafik saat memulai dan mengakhiri pengiriman paket melalui *event scheduler.*
3. TK (*Tool Kit*)
4. Tclcl merupakan bahasa pemrograman untuk menyediakan *linkage* antara C++ dan Otcl *berupa class hierarchy*, *object instantiation*, *variable binding*, *command dispatching.*
5. NS disebut sebagai simulator.
6. NAM adalah *network animator* yang berfungsi untuk memvisualisasikan keluaran dari NS2. *Editor* NAM merupakan *Interface GUI* yang dipanggil sebagai *file* berektensi *.nam* pada *script Tcl*.
7. Pre-processing berfungsi untuk membangkitkan trafik dan topologi jaringan
8. Post-processing merupakan analisa hasil simulasi yang ditampilkan pada file .tr dimana sebagian dari hasil simulasi tersebut dapat di filter menggunakan perintah awk.

2.2.12 AWK *Programming*

Awk adalah pemrograman yang dapat menangani tugas-tugas dengan kode program yang sangat singkat. Program awk adalah serangkaian pola dan aksi yang menjelaskan apa yang haru dicari dalam *input* data dan apa yang harus dilakukan ketika hal ini ditemukan. Program Awk mencari satu set *file* untuk baris yang cocok dengan salah satu pola. Selanjutnya ketika baris yang cocok ditemukan , maka tindakan yang sesuai akan dilakukan. Pola dapat memilih baris dengan kombinasi dari ekspresi reguler dan operasi-operasi bandingan pada *string*, angka, bidang, variabel, dan elemen *array*. Aksi yang dilakukan dapat meliputi melakukan pengolahan dengan bebas pada baris yang dipilih. Program Awk memindai *file input* dan membagi setiap baris *input*, pemisahan bidang, manajemen penyimpanan, dan inisialisasi, maka program Awk biasanya jauh lebih keicl dari bahasa yang lebih konvensional (Alfred V. Aho, 1988)

BAB 3 METODOLOGI

Metodologi merupakan bab yang akan menjelaskan tentang proses-proses yang diperlukan dalam menjalankan penelitian untuk membandingkan *Routing Protokol* AOMDV dan MP-DSR.



Gambar 3.1 Skema Alur Metode Penelitian

Sesuai dengan Gambar 3.1 Penelitian akan dimulai dengan Studi Literatur dari penelitian-penelitian terdahulu untuk menjadi panduan dalam penelitian, dilanjutkan dengan melakukan perancangan lingkungan pengujian. Setelah itu

pengujian dan pengambilan data. Selanjutnya adalah menganalisa hasil pengujian yang telah didapatkan. Dan tahap terakhir yang dilakukan adalah mengambil kesimpulan atas hasil penelitian yang telah dilakukan dan pemberian saran untuk penelitian selanjutnya.

3.1 Studi Literatur

Studi literatur menggambarkan seluruh dasar teori yang ada serta menjadi pendukung dalam perancangan dan pengimplementasian sistem. Untuk dapat merancang sistem tersebut. Berikut adalah dasar teori pendukung yang digunakan.

1. Protokol AOMDV
2. Protokol MP-DSR
3. Network Simulator *2*
4. *Packet Delivery Ratio* (PDR)
5. *End to End Delay*
6. *Normalized Routing Load*
7. *Throughput*
8. *Convergence Time*
9. Jenis Mobility : *Random Way point*
10. *File trace*
11. *File awk*

3.2 Perancangan dan Implementasi Lingkungan Pengujian

Perancangan dan implementasi lingkungan pengujian merupakan tahapan yang penting dalam merancang dan mengimplementasikan skenario lingkungan pengujian yang akan diterapkan pada Network Simulator 2. Dalam tahap ini dijabarkan langkah-langkah dalam menentukan beberapa parameter pengujian serta tahapan-tahapan dalam merancang skenario pengujian yang akan diterapkan. Selanjutnya dijabarkan juga langkah-langkah dalam mengimplementasikan lingkungan pengujian dalam Network Simulator 2 sesuai skenario yang telah dirancang.

3.2.1 Perancangan Lingkungan Pengujian

Dalam perancangan lingkungan pengujian dijelaskan bagaimana menentukan parameter lingkungan pengujian seperti jumlah *node*, luas area, durasi pengujian, kecepatan pergerakan *node*, ukuran paket yang dikirim dan model transport yang digunakan serta jumlah konektivitas yang dibangun. Dalam tahap ini juga

dijelaskan tahapan dalam merancang skenario lingkungan pengujian beserta topologi yang akan berjalan

3.2.2 Implementasi Lingkungan Pengujian

Setelah proses perancangan lingkungan pengujian telah dilakukan, maka langkah selanjutnya adalah mengimplementasikan rancangan tersebut pada Network Simulator 2. Dalam mengimplemntasikan rancangan tersebut dibutuhkan beberapa konfigurasi parameter lingkungan pengujian, topologi jaringan, pergerakan *node*, pengiriman paket, dan pemrosesan data *output*.

3.2.3 Pengujian

Setelah lingkungan dan skenario pengujian telah berhasil dirancang dan diimplementasikan berikutnya pengujian dilakukan untuk menguji kinerja protokol AOMDV dan MP-DSR berdasarkan skenario pengujian yang telah dirancang.

3.2.4 Pengambilan data

Proses pengambilan data dilakukan berdasarkan parameter pengukuran yang telah ditentukan, yakni *packet delivery ratio*, *end-to-end delay*, *throughput*, *Normalized Routing Load,* dan *convergence time* dengan menggunakan Bahasa pemrograman AWK.

3.3 Hasil dan Analisis

Setelah proses pengambilan data hasil pengujian telah berhasil, maka dilakukan penjabaran hasil beserta analisis kinerja masing-masing protokol *routing* yang dipisah berdarkan skenario lingkungan pengujian yang telah diimplementasikan dalam Network Simulator 2.

3.3.1 Analisis Pengaruh Variasi kepadatan *node* dengan luas Area

Analisis pengaruh variasi kepadatan *node* dilakukan dengan menjabarkan data hasil pengujian beserta analisis perbandingan kinerja protokol *routing* AOMDV dan MP-DSR berdasarkan skenario luas area 1000x1000 meter.

3.4 Kesimpulan dan Saran

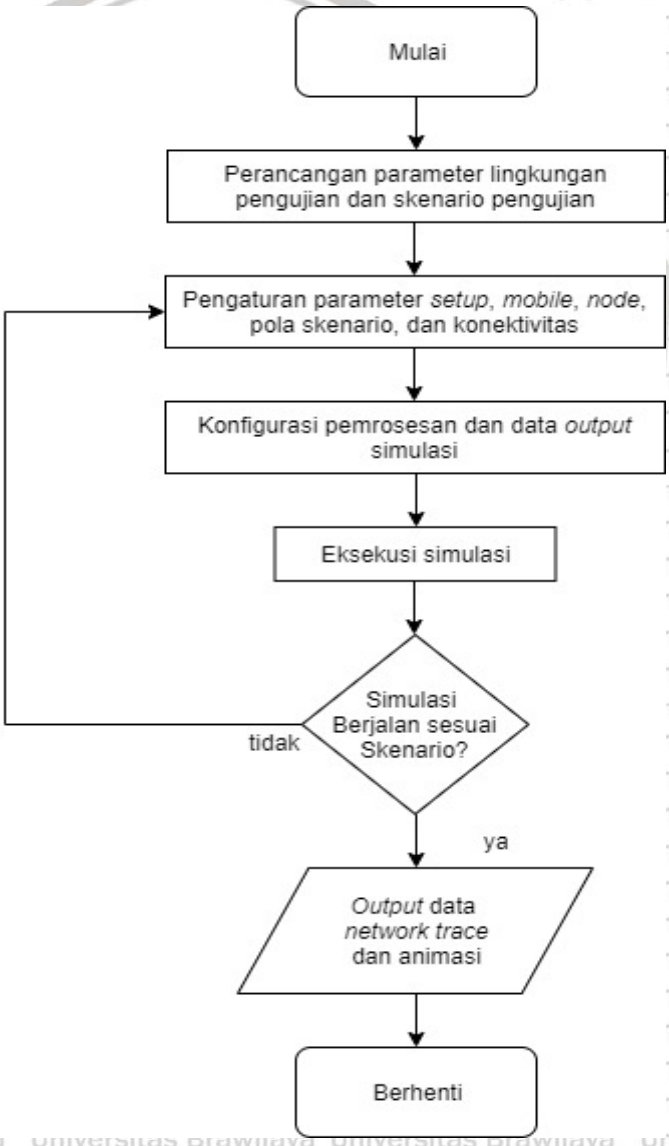
Berdasarkan tahapan proses penelitian mulai dari perancangan, implementasi lingkungan pengujian, dan analisis hasil pengujian. Di dalam bab ini dirangkum kesimpulan secara ringkas dan detail dari tiap poin analisis hasil pengujian beserta saran untuk pengembangan penelitian selanjutnya yang akan dijabarkan dalam kesimpulan dan saran.

BAB 4 PERANCANGAN DAN IMPLEMENTASI LINGKUNGAN PENGUJIAN

Pada bab ini akan dijabarkan tahapan dalam perancangan dan implementasi lingkungan pengujian jaringan MANET. Implementasi lingkungan pengujian dalam penelitian diterapkan pada Network Simulator 2 (NS2).

4.1 Perancangan Lingkungan Pengujian

Perancangan lingkungan pengujian merupakan tahapan penting sebelum mengimplementasikan lingkungan pengujian yang akan dijalankan. Karena penelitian ini murni bersifat simulatif, maka metode perancangan menyesuaikan lingkungan dari simulator yang digunakan, yaitu Network Simulator 2.



Gambar 4.1 Skema Alur Tahapan Perancangan Lingkungan Pengujian pada NS2

4.1.1 Perancangan Parameter Lingkungan Pengujian

Perancangan parameter lingkungan pengujian dilakukan untuk menentukan parameter-parameter yang nantinya akan digunakan selama proses simulasi protokol *routing* AOMDV dan MP-DSR dalam jaringan MANET. Skema perancangan parameter pengujian akan dijabarkan pada tabel berikut:

Tabel 4.1 Perancangan Parameter Lingkungan Pengujian

Parameter	Nillai
Jumlah *Node*	20, 30, 40, 50
Jenis Protokol	AOMDV dan MP-DSR
Simulator	*Network Simulator* 2 Versi 2.35
Jenis Mobilitas	*Random Way Point*
Model Transpor	UDP (CBR)
Ukuran Paket Data	512 *bytes*, 1024 *bytes*
Luas Area Jaringan	1000 m x 1000 m
Kecepatan *node*	Acak, maks 10 m/s
Waktu Simulasi	1000 s
Mac	Mac/802_11
Transmission rate	0,1 Mb

4.1.2 Perancangan Skenario Lingkungan Pengujian

Skenario lingkungan pengujian MANET dilakukan terhadap 2 jenis protokol yang berbeda, yaitu AOMDV dan MP-DSR. Pengujian dilakukan dengan atu koneksi menggunakan transpor jaringan UDP dengan model *traffic constant bit-rate* (CBR). Besar *transmission rate* ditentukan sebesar 0,1 Mb. Skenario lingkungan pengujiannya, yaitu dengan luas area sebesar 1000x1000 meter. Tiap skenario akan disimulasikan dengan variasi jumlah *node* yang berbeda-beda, yakni 20, 30, 40 dan 50 *node* dengan durasi waktu 1000 detik. Besar paket yang dikirimkan dibagi menjadi dua, yaitu 512 *byte* dan 1024 *byte*. Dalam skenario pengujian paket mulai dikirimkan oleh *source node* menuju *destination node* sejak detik ke-120 sampai ke-1000. Detail skenario lingkungan pengujian dalam penelitian ini dijabarkan pada tabel berikut ini:

Tabel 4.2 Perancangan Skenario Lingkungan Pengujian

Skenario	Luas area simulasi	Jumlah *node*	Kecepatan pergerakan *node*	Durasi simulasi
1	1000x1000 meter	20, 30, 40, dan 50	Acak, maks 10 m/s	1000 *sec*

Dalam penelitian ini arah pergerakan dan kecepatan tiap *node* dijalankan secara acak sesuai karakteristik dalam jaringan MANET yang dinamis. Di dalam penelitian ini kecepatan pergerakan dibatasi dengan kecepatan maksimal 10 m/s.

Tabel 4.3 Skenario Berdasarkan Ukuran Paket Data

Jumlah *Node*	Kecepatan *Node* (m/s)	Luas Area Jaringan	Ukuran Paket Data (*bytes*)
20	Acak, maks 10 m/s	1000 m x 1000 m	512
30	Acak, maks 10 m/s	1000 m x 1000 m	512
40	Acak, maks 10 m/s	1000 m x 1000 m	512
50	Acak, maks 10 m/s	1000 m x 1000 m	512

Perancangan ukuran paket data pada setiap skenario disusun sama, baik untuk protokol *routing* AOMDV dan MP-DSR. Ukuran paket data menunjukan jumlah keseluruhan paket data yang ditransmisikan setiap detiknya.

Tabel 4.4 Skenario Berdasarkan Ukuran Paket Data

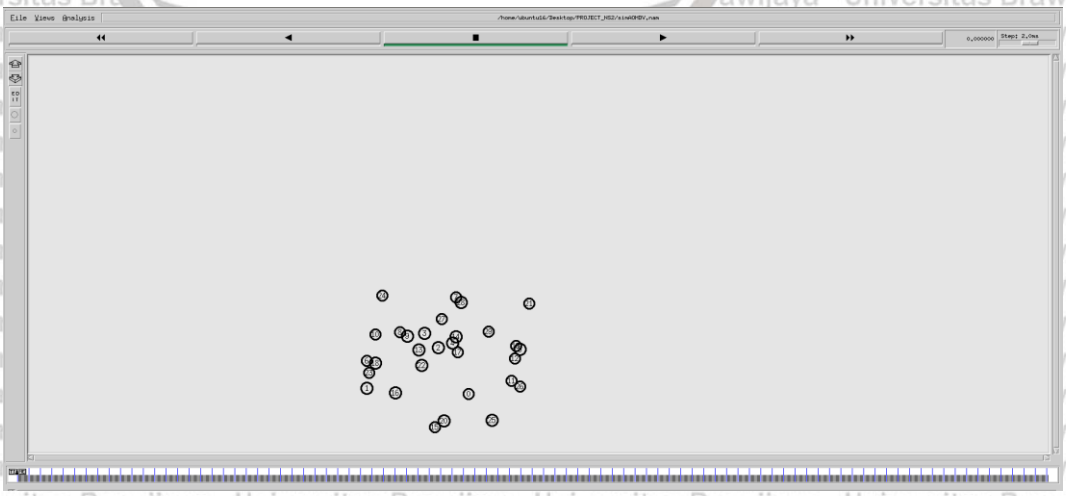
Jumlah *Node*	Kecepatan *Node* (m/s)	Luas Area Jaringan	Ukuran Paket Data (*bytes*)
20	Acak, maks 10 m/s	1000 m x 1000 m	1024
30	Acak, maks 10 m/s	1000 m x 1000 m	1024
40	Acak, maks 10 m/s	1000 m x 1000 m	1024

50	Acak, maks 10 m/s	1000 m x 1000 m	1024

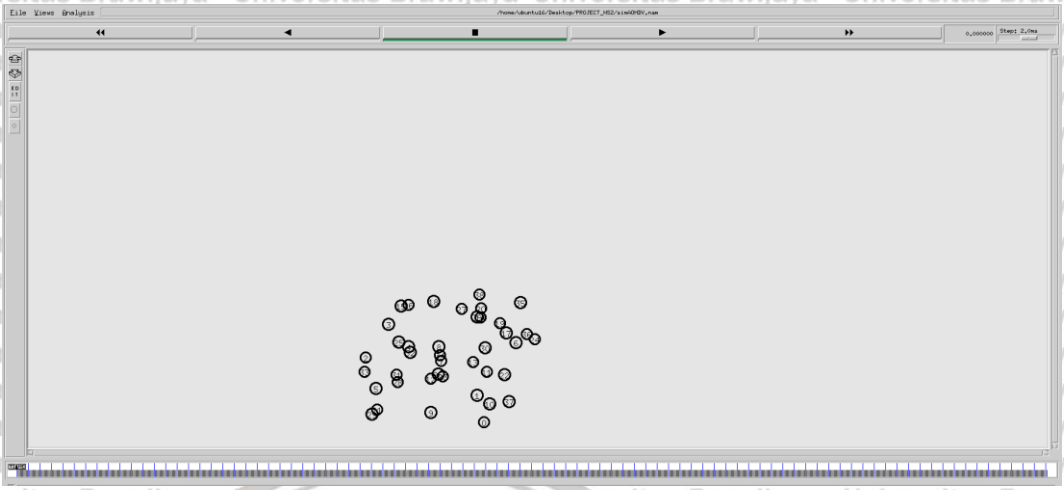
4.1.3 Perancangan Topologi Jaringan

Mobile Ad Hoc Network (MANET) memiliki karakteristik di mana tiap *node* dapat bergerak bebas secara acak dengan kecepatan yang tidak konstan sehingga menyebabkan topologi dalam jaringan MANET selalu bersifat dinamis. Dalam penelitian ini topologi dibuat dengan skema posisi dan pergerakan tiap *node* yang diatur secara acak, sehingga tiap skenario pengujian yang dijalankan akan berbeda topologi maupun pergerakan. Namun, untuk mengurangi inkonsistensi posisi *source node* dan *destination node* yang akan menyebabkan akurasi data yang berbanding jauh, maka dalam skenario penjauhan ini posisi awal *source node* dan *destination node* ditetapkan saling berjauhan. Hal tersebut ditetapkan tanpa mengubah posisi dan pergerakan selanjutnya yang tetap diatur secara acak. Berikut merupakan beberapa sampel topologi jaringan dengan jumlah *node* 20, 30, 40 dan 50 yang diatur secara acak sesuai skenario lingkungan pengujian.

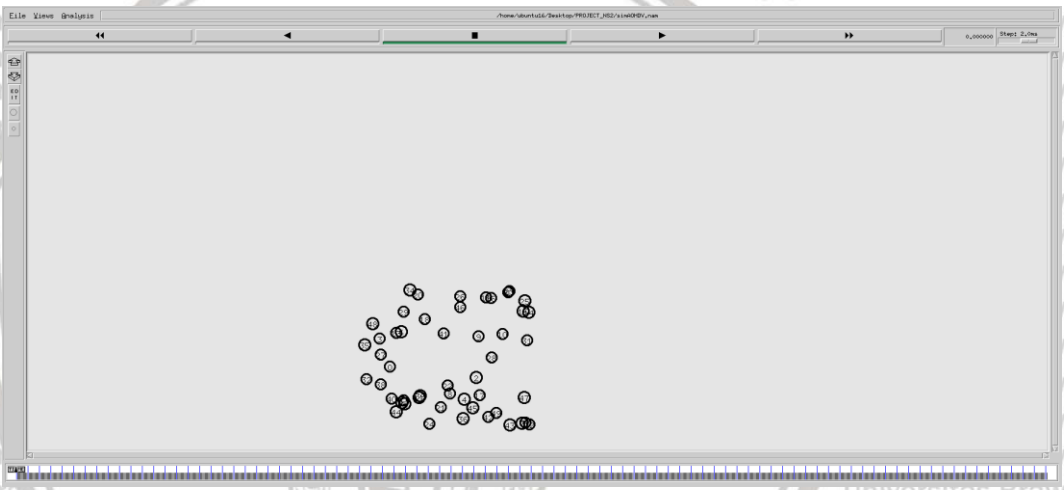
Gambar 4.2 Screenshot Topologi Jaringan dengan 20 node



Gambar 4.3 Screenshot Topologi Jaringan dengan 30 node



Gambar 4.4 Screenshot Topologi Jaringan dengan 40 node



Gambar 4.5 Screenshot Topologi Jaringan dengan 50 node

4.2 Implementasi Lingkungan Pengujian

Dalam subbab ini akan dijelaskan mengenai tahapan-tahapan dalam implementasi lingkungan pengujian yang telah dirancang pada subbab ini sebelumnya. Implementasi lingkungan pengujian akan diterapkan pada Network Simulator 2.

1. Melakukan pengaturan parameter untuk simulasi

Pengaturan parameter simulasi digunakan untuk mendefinisikan nilai dari parameter yang akan digunakan dan karakteristik yang akan disimulasikan pada *MANET*.

2. Melakukan Inisialisasi

Inisialisasi variabel digunakan untuk mengkonfigurasikan pada setiap node tentang apa saja yang dibutuhkan pada saat melakukan simulasi

```
1   set val (chan)   Channel/WirelessChannel
2   set val (prop)   Propagation/TwoRayGround
3   set val(netif) Phy/WirelessPhy
4   set val (mac)    Mac/802_11
5   set val (ifq)    Queue/Droptail/Priqueue
6   set val (ll)     LL
7   set val (ant)    Antenna/OmniAntenna
8   set val (ifqlen)50
9   set val (nn)     20, 30, 40, 50
10  set val (rp)     AOMDV, MP-DSR
11  set val (x)      1000
12  set val (y)      1000
13  set val (stop)   1000
```

Penjelasan :

Baris 1 : Mendefinisikan tipe channel

Baris 2 : Mendefinisikan model propagasi yang digunakan

Baris 3 : Mendefinisikan tipe jaringan *interface*

Baris 4 : Mendefinisikan tipe mac

Baris 5 : Mendefinisikan tipe *queue interface*

Baris 6 : Mendefinisikan tipe *link layer*

Baris 7 : Mendefinisikan model antenna yang digunakan

Baris 8 : jumlah ukuran paket *node*

Baris 9 : jumlah *mobile node*

Baris 10 : *Routing* protokol yang digunakan

Baris 11 : Luas area jaringan sumbu X

Baris 12 : Luas area jaringan sumbu Y

Baris 13 : Waktu henti simulasi

3. Melakukan *setting* NAM dan *trace file*

Setting NAM digunakan untuk memvisualisasikan keluaran simulasi berupa tampilan grafis animasi dan *trace file* digunakan untuk menganalisa numerik hasil simulasi yang telah dilakukan. *Trace file* akan dijadikan sebagai keluaran file.tr untuk melihat hasil numerik yang terjadi selama proses komunikasi pada folder *script* program *namfile* akan dijadikan sebagai keluaran *file.nam* untuk melihat tampilan hasil dari program.

```
1   set ns [new Simulator]
2   set tracefile [open out.tr w]
3   set trace-all $tracefilefi
4   set namfile [open out.nam w]
5   set namtrace-all $namfile
```

Penjelasan :

Baris 1 : Mendefinisikan sebuah variabel atau objek sebagai *instance* dari kelas simulator

Baris 2-3 : Medefinisikan nama *trace file* dan menampilkan *trace file*

Baris 4-5 : Mendefinisikan nama namfile dan menampilkan NAM *trace file*

4. Pembuatan *node*

Pembuatan *node* digunakan untuk membangkitkan *node-node yang akan dijalankan.* N0 merupakan variabel pointer dalam pembuatan *node*.

```
1  set n0 [$ns node]
2  Set n1 [$ns node]
3  set n2 [$ns node]
4  set n3 [$ns node]
5  set n4 [$ns node]
6  set n5 [$ns node]
```

Penjelasan :

Baris 1-6 : Mendefinisikan variabel *node*

5. Pembuatan aliran trafik data dan pembentukan koneksi

Aliran trafik data dan pembentukan koneksi digunakan untuk menjalankan simulasi terhadap proses terjadinya pengiriman data dan penerimaan data dari *node* satu ke *node* lainnya dengan menggunakan *Agent* UDP sebagai pengirim dan CBR sebagai trafik generator untuk pembentukan koneksi agent UDP.

```
1   set udp0 [new Agent/UDP]
2   $ns attach-agent $n0 $udp0
3   Set cbr0 [new Application/Traffic/CBR]
4   $cbr0 set packetSize_ 1Kb
5   $cbr0 set interval_ 0.0005
6   $cbr0 attach-agent $udp0
7   set null0 [new Agent/Null
8   $ns attach-agent $udp0 $null0
9   $ns attach-agent $ns1 $null0
10  $ns at 0.5 "$cbr0 start"
11  $ns at 4.5 "$cbr0 stop"
```

Penjelasan :

Baris 1 : Membentuk koneksi melalui protokol UDP

Baris 2 : Menggabungkan protokol pada sebuah *node*

Baris 3 : Membentuk trafik CBR pada protokol UDP

Baris 4 : Besar ukuran paket yang digunakan

Baris 5 : interval antar paket

Baris 6 : Menggabungkan aplikasi CBR pada protokol UDP

Baris 7-8 : Mendefinisikan tujuan data

Baris 9 : Membuat koneksi antara sumber dan tujuan

Baris 10 : Pengaturan jadwal *start* untuk koneksi CBR

Baris 11 : Pengaturan jadwal *stop* untuk koneksi CBR

6. Mengakhiri simulasi

Pada pembuatan simulasi, dilakukan pengaturan jadwal untuk dapat mengakhiri simulasi. Pengaturan jadwal dapat dilakukan dengan menetapkan waktu henti pada saat awal simulasi dan melakukan *reset*. Pada *node* agar dapat mengakhiri program.

```
1   proc finish {} {
2   global ns tracefile namfile
3   $ns attach-agent $udp0 $null0
4   close $tracefile
5   close $namfile
6   puts "running nam..."
7   exec nam out.nam &
8   exit 0
9   }
10
11  $ns at 0.5 "$cbr0 start"
12  $ns at 4.5 "$cbr0 stop"
```

Penjelasan :

Baris 1-2 : Mendeklarasikan prosedur *finish* pada simulasi

Baris 3-5 : Menyimpan semua data hasil simulasi ke dalam *tracefile* dan *namfile*

Baris 6-7 : Mengeksekusi NAM

Baris 8-9 : Mengakhiri aplikasi dan mengembalikan status dengan angka 0 kepada sistem

Baris 11 : Mendefinisikan waktu henti simulasi

Baris 12 : Menjalankan simulasi program

4.3 Pengujian

Dalam subbab ini membahas tentang skenario pengujian dan teknik dalam pengambilan data berdasarkan simulasi protokol *routing* yang telah dilakukan sesuai rancangan skenario lingkungan pengujian yang telah dibuat. Berdasarkan hasil simulasi tersebut didapatkan beberapa data *output* yang kemudian akan diukur berdasarkan dari parameter-parameter yang telah ditentukan.

4.3.1 Skenario pengujian

Pada penelitian ini pengukuran data hasil pengujian dilakukan dengan parameter yang telah ditentukan, yakni *packet delivery ratio*, *end-to-end delay*, *throughput*, *normalized routing load* dan *convergence time*. Protokol *routing yang* diuji, yaitu AOMDV dan MP-DSR. Kedua protokol tersebut diuji dengan skenario

lingkungan pengujian yang sama. Tiap skenario lingkungan pengujian tersebut diuji berdasarkan pengaruh variasi kepadatan *node* yang berjumlah, yaitu 20, 30, 40 dan 50. Pengujian dilakukan dengan mengirimkan paket sebesar 512 *byte* dan 1024 *byte* dengan *transmission rate* 0,1 Mb yang berjalan dalam durasi 1000 *sec*. Pengujian masing-masing skenario dilakukan sebanyak tiga kali dan selanjutnya diambil nilai rata-ratanya untuk menambah keakuratan data analisis. Hal tersebut dilakukan karena topologi MANET yang dinamis, sehingga menyebabkan data yang dihasilkan cenderung berbeda di tiap pengujian.

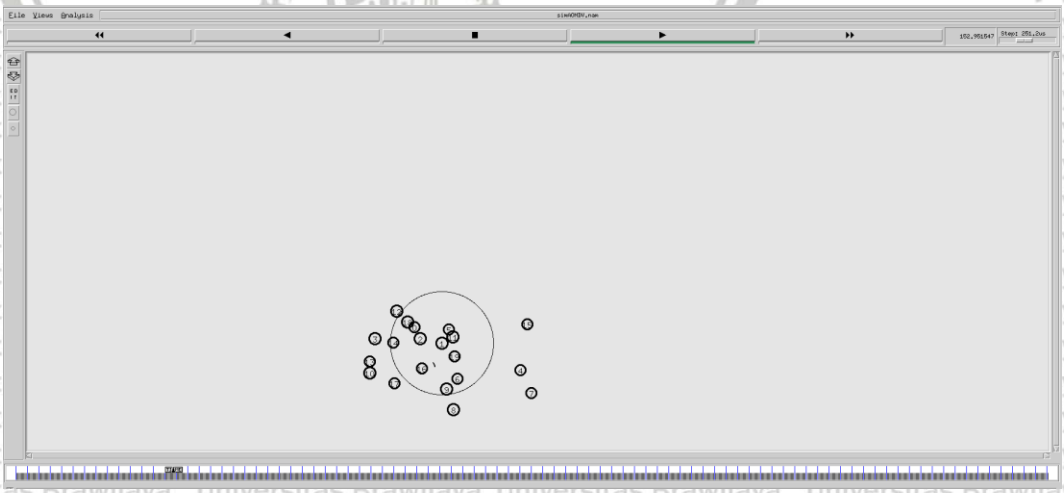
4.3.2 Eksekusi Simulasi Protokol *Routing* MANET

Simulasi protokol *routing* MANET dilakukan sebagai implementasi dari beberapa skenario lingkungan pengujian pada protokol *routing* AOMDV dan MP-DSR yang telah dirancang dan konfigurasi dalam Network Simulator 2. Simulasi dilakukan dengan cara mengeksekusi *file script Tcl* yang telah dibuat, yaitu AOMDV.tcl dan MPDSR.tcl. Hasil eksekusi berjalan seperti pada gambar berikut.

```
ubuntu16@ubuntu16-VirtualBox:~/Desktop/PROJECT_NS2$ ns AOMDV.tcl
num_nodes is set 20
warning: Please use -channel as shown in tcl/ex/wireless-mitf.tcl
INITIALIZE THE LIST xListHead
channel.cc:sendUp - Calc highestAntennaZ_ and distCST_
highestAntennaZ_ = 1.5,  distCST_ = 550.0
SORTING LISTS ...DONE!
```

Gambar 4.6 Hasil Eksekusi file script AOMDV.tcl

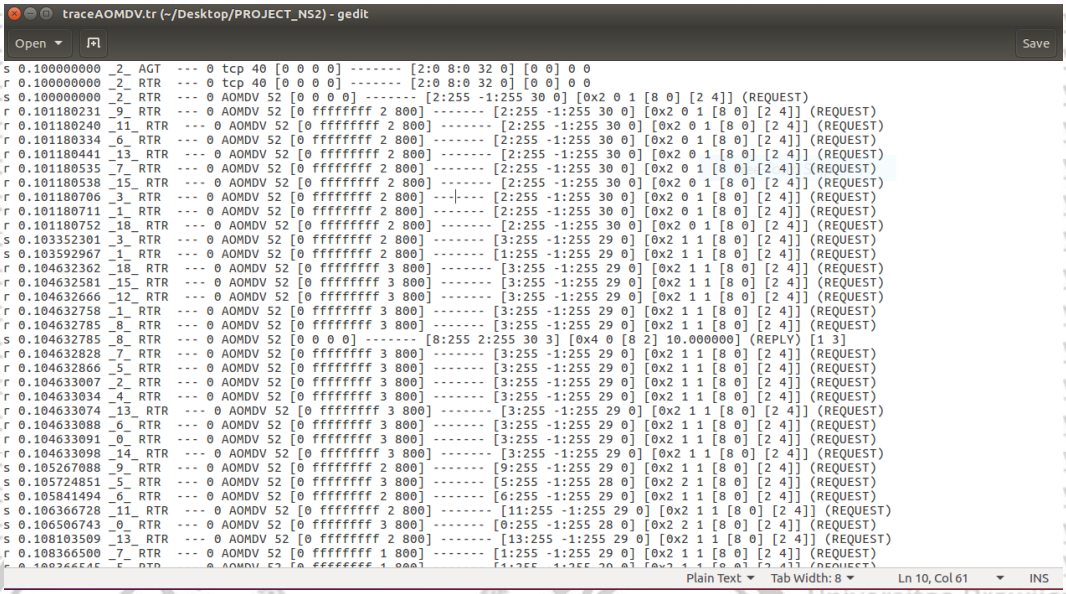
Dalam menjalankan *file script* Tcl yang telah dikonfigurasi pada Network Simulator 2 langkah yang diperlukan, yakni dengan menjalankan perintah "ns" dilanjutkan dengan nama *file script* Tcl yang telah dibuat. Jika telah berhasil dieksekusi, maka akan keluar beberapa notifikasi salah satunya berupa tulisan "sorting *list done*", maka setelah itu secara otomatis *output* berupa *network animation file* (.nam) akan tampil seperti pada gambar berikut.



Gambar 4.7 Hasil Output Network Animation File

Network animation file (.nam) akan menampilkan keseluruhan jalannya simulasi secara visual sesuai dengan rancangan yang telah dikonfigurasikan pada

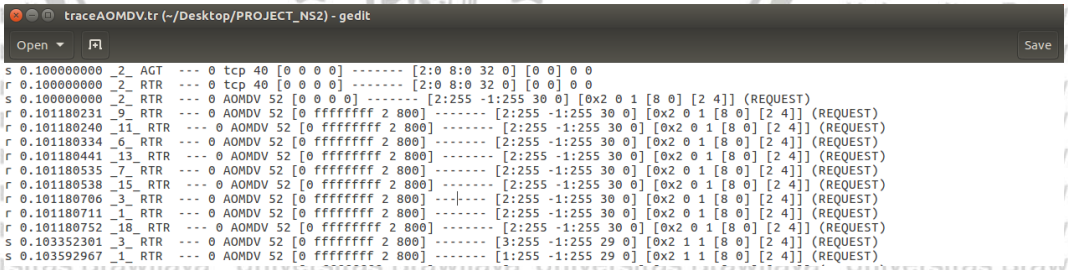
Tcl *script* pada Network Simulator 2. Sedangkan pada *output* lainnya, yakni *network trace file* (.tr) ditampilkan seluruh rekaman data lalu lintas jaringan selama jalannya simulasi secara detail seperti pada gambar berikut.



Gambar 4.8 Hasil Output Network Trace File

4.4 Pengambilan Data

Pengambilan data dilakukan dengan mengolah salah satu data *output* simulasi, yaitu *output network trace file* yang berisi rincian rekaman data lalu lintas jaringan selama proses simulasi dijalankan. Pengolahan data akan dijalankan dengan pemrograman AWK yang memproses data berdasarkan struktur kolom pada *output network trace file*. Sebelum masuk pada proses pemrograman AWK. Berikut merupakan struktur data *output network trace file* pada Network Simulator 2 beserta penjelasannya.



Gambar 4.9 Struktur Output Network Trace File

1. Pada kolom pertama berisi detail dari status paket yang dikirim. Status paket dapat berupa symbol s (*send*) yaitu paket telah dikirim, r (*received*) paket telah diterima, d (*dropped*) paket didrop, f (*forwarded*) paket diteruskan, dan c (*collision*) paket bertabrakan pada lapisan MAC.
2. Pada kolom kedua merupakan detail waktu aktivitas dilakukan dalam satuan waktu *second*.

3. Kolom ketiga menjelaskan identitas *node* tempat terjadinya aktivitas yang dirinci.
4. Kolom keempat menjelaskan dimana letak tingkatan lapisan jaringan tempat suatu aktivitas terjadi contohnya seperti AGT (*agent*) terdapat pada lapisan aplikasi, RTR (*routing*), MAC, IFQ (*interface queue*) merupakan antrian paket keluar yang terdapat diantara lapisan link dan MAC, LL (*link layer*), dan PHY (*physcal*).
5. Kolom keenam merupakan identitas paket dalam bentuk *sequence number*.
6. Kolom ketujuh merupakan penjelasan tipe paket dimana dapat berbentuk data stream CBR (*constant bitrate*), paket *routing* (AOMDV dan MP-DSR), ARP, RTS dan CTS.
7. Kolom kedelapan menjelaskan besaran paket yang diproses.
8. Kolom kesembilan berisi alamat *source node* dan tujuan MAC yang masing-masing adalah "0" dan "ffffffff". Protokol MAC mengasumsikan bahwa penundaan di atas saluran nirkabel yang mendasari dari nilai 0. Nilai 800 merupakan IP paket yang berjalan di atas jaringan *Ethernet*.
9. Kolom kesepuluh berisi alamt IP *source* dan *destination node* yakni 0:255 dan 1:255 beserta port masing-masing. Sedangkan 30 dan 0 merupakan *time to live* dan alamat dari *hop* berikutnya.
10. Kolom kesebelas dan seterusnya merupakan rekaman informasi *routing* AOMDV yang memiliki paket RREQ yang ditandai dengan ID "0x2" selanjutnya jumlah *hop* dihitung "0" dan *broadcast* ID adalah "1".

Berdasarkan penjelasan struktur data *output network trace file* di atas, maka pemrograman AWK berperan dalam mengolah dan mengakumulasi data berdasarkan kolom-kolom data informasi yang tertera berdasarkan rumusan matematis dari parameter yang telah ditentukan seperti *packet delivery ration*, *end-to-end delay*, *throughput*, *normalized routing load,* dan *convergence time*.

4.4.1 Pengambilan Data *Packet Delivery Ratio*

Proses pengambilan data *packet delivery ratio* dalam *output network trace file* Network Simulator 2 dilakukan dengan perancangan pemrograman AWK berdasarkan rumusan matematis dari *packet delivery ratio*.

4.4.2 Pengambilan Data *End-to-end Delay*

Proses pengambilan data *end-to-end delay* dalam output *network trace file* Network Simulator 2 dilakukan dengan perancangan pemrograman AWK berdasarkan rumusan matematis dari *end-to-end delay*.

4.4.3 Pengambilan Data *Throughput*

Proses pengambilan data *throughput* dalam *output network trace file network* Simulator 2 dilakukan dengan perancangan pemrograman AWK berdasarkan rumusan matematis dari *throughput*.

4.4.4 Pengambilan Data *Normalized Routing Load*

Proses pengambilan data *normalized routing load* dalam *output network trace file network* Simulator 2 dilakukan dengan perancangan pemrograman AWK berdasarkan rumusan matematis dari *normalized routing load.*

4.4.5 Pengambilan Data *Convergence time*

Proses pengambilan data *convergence time* dalam *output network trace file network* Simulator 2 dilakukan dengan perancangan pemrograman AWK berdasarkan rumusan matematis dari *convergence time.*

BAB 5 HASIL DAN PEMBAHASAN

Pada bab ini akan dijelaskan hasil dan pembahasan dari kinerja yang dilakukan pada *Mobile Ad hoc Network* (MANET) dengan adanya perubahan pada dua parameter, yaitu variiasi jumlah *node* dan ukuran paket data dengan memperhatika *link* atau *node* dan paket data sebagai pembanding untuk menganalisis kinerja protokol *routing*.

5.1 Perbandingan Kinerja berdasarkan Jumlah *Node*

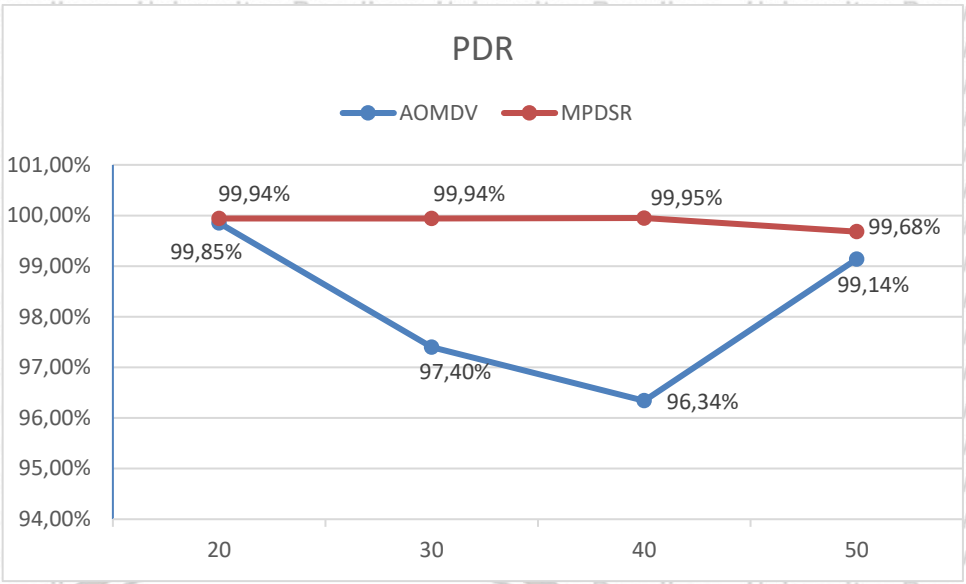
Pada skenario pertama, akan dijalankan kinerja dari protokol terhadap beban jaringan dengan variasi jumlah node sebanyak 20 *node*, 30 *node*, 40 *node*, 50 *node* secara bergantian menggunakan protokol *routing* AOMDV dan MP-DSR pada luas area 1000m x 1000m. Kepadatan *node* dalam jaringan digunakan sebagai parameter penelitian untuk mengetahui protokol *routing* mana yang cocok untuk jaringan dengan kepatan *node* tertentu.

5.1.1 Perbandingan Kinerja *Packet Delivery Ratio*

Packet Delivery Ratio (PDR) merupakan perbandingan antara paket data yang terkirim dengan jumlah paket data yang terkirim dengan jumlah paket data yang dikirimkan oleh *node* sumber. Hasil yang diperoleh paket dari *node* sumber ke tujuan ditunjukkan pada tabel 5.1 di bawah ini.

Tabel 5.1 Nilai PDR vs Jumlah *Node*

Node	AOMDV	MPDSR
20	99,85	99,94
30	97,40	99,94
40	96,34	99,95
50	99,14	99,68



Gambar 5.1 PDR vs Jumlah *Node*

Gambar 5.1 menjelaskan melalui simulasi berdasarkan penambahan variasi jumlah *node*, diperoleh hasil bahwa protokol MP-DSR memiliki kinerja lebih baik terhadap rasio jumlah paket yang diterima dengan jumlah paket yang dikirim dibandingkan dengan protokol MP-DSR. Pada protokol MP-DSR di dapatkan hasil yaitu 99,94% untuk 20 *node*, 99,94% untuk 30 *node*, 99,95% untuk 40 *node* dan 99,68% untuk 50 *node*. Sedangkan pada protokol AOMDV di dapatkan hasil yaitu 99,85% untuk 20 *node*, 97,40% untuk 30 *node*, 96,34% untuk 40 *node* dan 51,28% untuk 50 *node*.

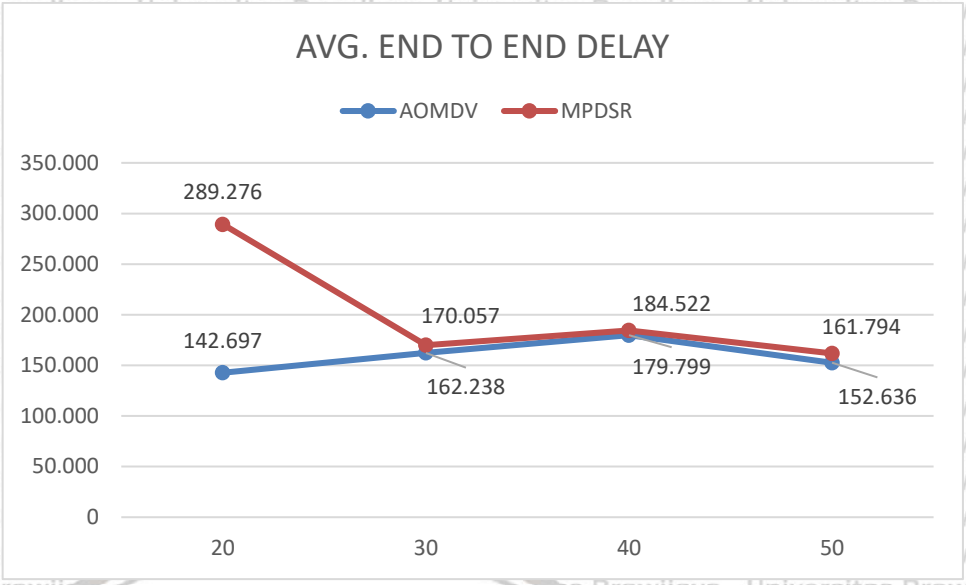
Hal ini disebabkan protokol MP-DSR dapat menjamin rute alternatif saling disjoint atau beririsan melalui komputasi yang terdistribusi pada tiap node tanpa perlu komputasi dari node sumber saja sehingga rute yang ditemukan diharapkan tidak akan terjadi loop.

5.1.2 Perbandingan Kinerja *Average End to End Delay*

Average End to End Delay meerupakan rata-rata waktu yang diperlukan mulai dari paket yang dikirimkan oleh *node* sumber ke sampai paket data tersebut berhasil diterima oleh *node* tujuan. Hasil analisa data dari kinerja *average End to End Delay* yang didapatkan berdasarkan penambahan variasi jumlah *node* ditunjukkan pada tabel 5.2 di bawah ini.

Tabel 5.2 Nilai *Avg. End to End Delay* vs Jumlah *Node*

Node	AOMDV	MP-DSR
20	142,697	289,276
30	162,238	170,057
40	179,799	184,522
50	152,636	161,794



Gambar 5.2 *Avg. End to End Delay* vs Jumlah *Node*

Gambar 5.2 menjelaskan bahwa kinerja dari protokol AOMDV lebih baik daripada protokol MP-DSR dengan nilai rata-rata *end to end delay* dari protokol AOMDV, yaitu 142,697 m/s untuk 20 *node*, 162,238 m/s untuk 30 *node*, 179,799 m/s untuk 40 *node* dan 152,639 m/s untuk 50 *node.*

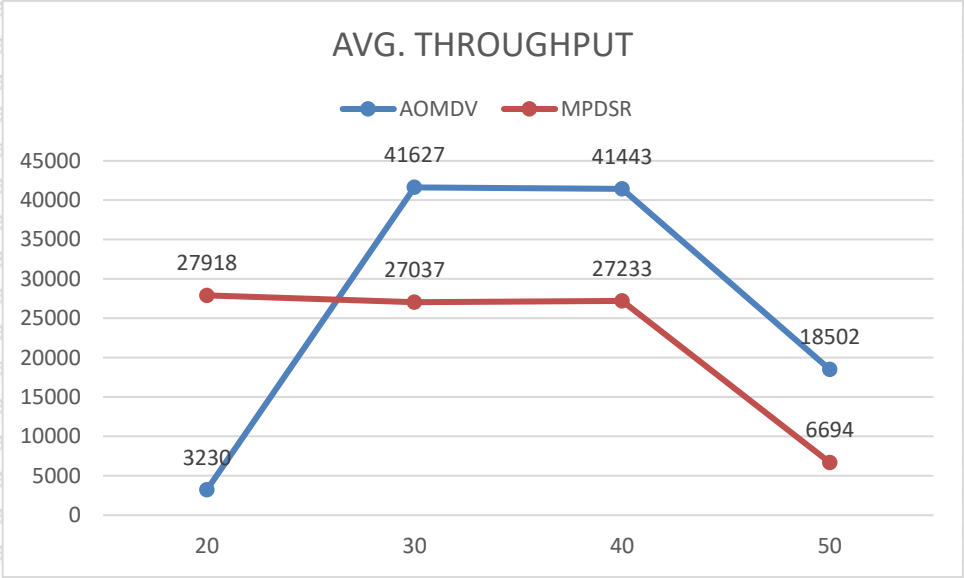
Hal ini disebabkan prtokol AOMDV dapat menghitung atau menemukan alternatif rute dengan tambahan overhead yang minim dibandingkan dengan protokol MP-DSR.

5.1.3 Perbandingan Kinerja *Average Throughput*

Average throughput merupakan rata-rata jumlah *byte* yang berhasil dikirim pada suatu jaringan dalam satuan waktu. Tabel 5.3 merupakan hasil kinerja *average throughput* berdasarkan analisis data yang didapatkan saat pengiriman paket dari *node* sumber ke *node* tujuan.

Tabel 5.3 Nilai *Avg. Throughput* vs Jumlah *Node*

Node	AOMDV	MP-DSR
20	3230	27918
30	41627	27037
40	41443	27233
50	18502	6694



Gambar 5.3 *Avg. Troughput* vs Jumlah *Node*

Gambar 5.3 menjelaskan bahwa nilai rata *throughput* dari protokol MP-DSR, yaitu 27918 kbps untuk 20 *node*, 27037 kbps untuk 30 *node*, 27233 untuk 30 *node*, 6694 kbps untuk 50 *node*. Sedangkan protokol AOMDV, yaitu 3230 kbps untuk 20 *node*, 41627 untuk 30 *node*, 41443 untuk 40 *node*, 18502 untuk 50 *node*. Dari nilai rata-rata yang diperoleh, dapat disimpulkan kinerja protokol AOMDV lebih baik daripada protokol MP-DSR.

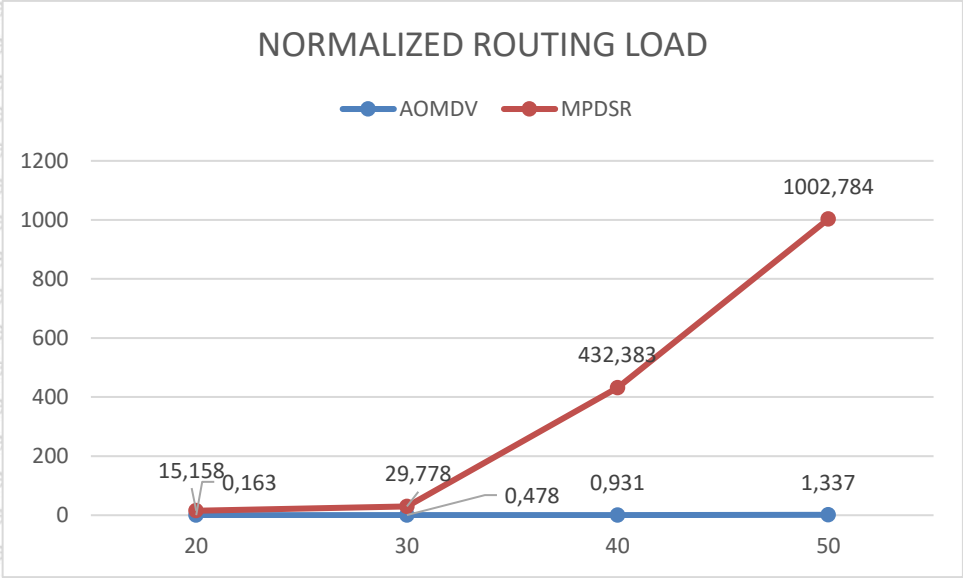
Hal ini disebabkan protokol AOMDV menyimpan informasi ke seluruh *node* dan secara berkala melakukan *broadcast* untuk memperbaharui *routing table* tersebut. Ketika salah satu *node* keluar dari rute, maka protokol AOMDV akan melakukan *broadcast* ke seluruh *node* untuk memberitahu terdapat rute yang rusak. Sedangkan MP-DSR, tidak menyimpan rute pada suatu *node* apabila melebihi batas *lifetime* dan menghemat penggunaan *bandwich*.

5.1.4 Perbandingan Kinerja *Normalized Routing Load*

Normalized routing load adalah suatu nilai perbandingan antara banyaknya paket *routing* yang dikirim *source node* dan diteruskan (*forwarding*) dengan jumlah paket data yang diterima pada *destinatiton node*.

Tabel 5.4 Nilai *Avg. Normalized Routing Load* vs Jumlah *Node*

Node	AOMDV	MP-DSR
20	0,163	15,158
30	0,478	29,778
40	0,931	432,383
50	1,337	1002,784



Gambar 5.4 *Avg. Normalized Routing Load* vs Jumlah Node

Gambar 5.4 menjelaskan bahwa hasil kinerja *normalized routing load* dari protokol *AOMDV*, yaitu 0,163 untuk 20 *node*, pada 30 *node* sebesar 0,478, 0,931 untuk 40 *node* dan 1,337 untuk 50 *node*. Sedangkan protokol *MP-DSR*, yaitu 15,158 untuk 20 *node*, 29,778 untuk 30 *node*, 432,383 untuk 40 *node*, 1002,784 untuk 50 *node*.

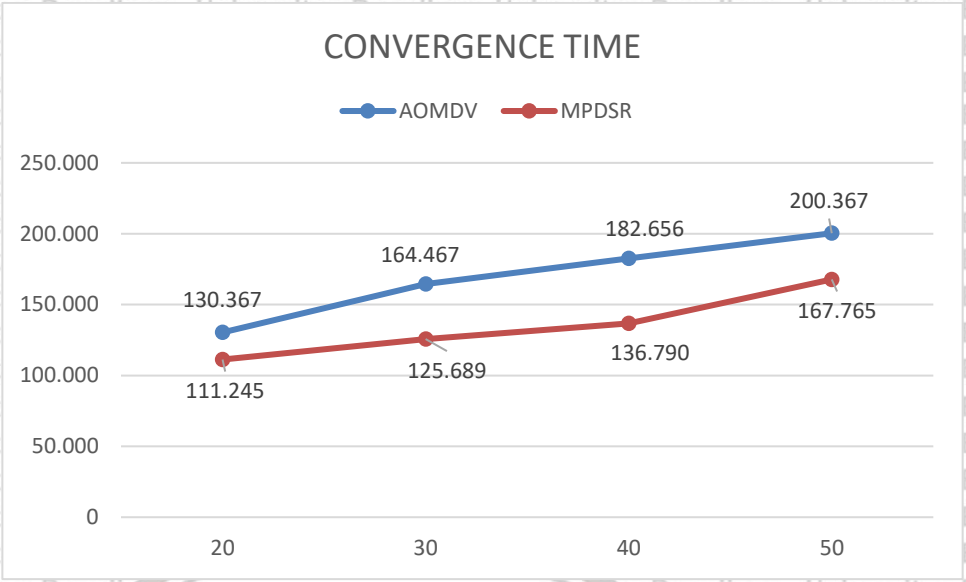
Hal ini disebabkan karena proses pencarian rute yang panjang dan membutuhkan waktu yang lama. Selain itu juga dipengaruhi oleh jarak antara *node* pengirim dengan *node* penerima. Semakin jauh jarak *node* pengirim dengan *node* penerima, maka paket yang hilang jumlahnya akan semakin besar. Kedua protokol *routing* menghasilkan *normalized routing load* yang besar ketika beban data layanan bertambah. Semakin besar beban data layanan yang digunakan, maka semakin besar kemungkinan dan antrian data selama proses transmisi berlangsung.

5.1.5 Perbandingan Kinerja *Convergence Time*

Convergence time merupakan rata-rata waktu yang dibutuhkan secara keseluruhan setiap *node* mendapatkan *broadcast* dari *node* yang menyebarkan informasi tabel *routing*.

Tabel 5.5 Nilai *Avg. Convergence Time* vs Jumlah *Node*

Node	AOMDV	MP-DSR
20	125,345	101,121
30	150,212	111,235
40	182,125	123,550
50	190,789	140,489



Gambar 5.5 *Avg. Convergence Time* vs Jumlah Node

Gambar 5.5 menjelaskan bahwa hasil kinerja *convergence time* dari protokol *AOMDV*, yaitu 130,367 m/s untuk 20 *node*, 165,467 m/s untuk 30 *node* , 182,656 m/s untuk 40 *node* dan 200,367 m/s untuk 50 *node*. Sedangkan protokol *MP-DSR*, yaitu 111,245 untuk 20 *node*, 125,689 m/s untuk 30 *node*, 136,790 m/s untuk 40 *node*, 167,765 m/s untuk 50 *node*.

Protokol MP-DSR memiliki nilai rata-rata terbaik. Nilai rata-rata *covergence time* pada protokol MP-DSR berada di bawah nilai *convergence time* dari AOMDV, dan tidak mengalami kenaikan yang signifikan. Sedangkan nilai rata-rata *convergence time* pada protokol routing AOMDV berada di atas nilai dari MP-DSR.

5.2 Perbandingan Kinerja terhadap Jumlah Ukuran Paket Data

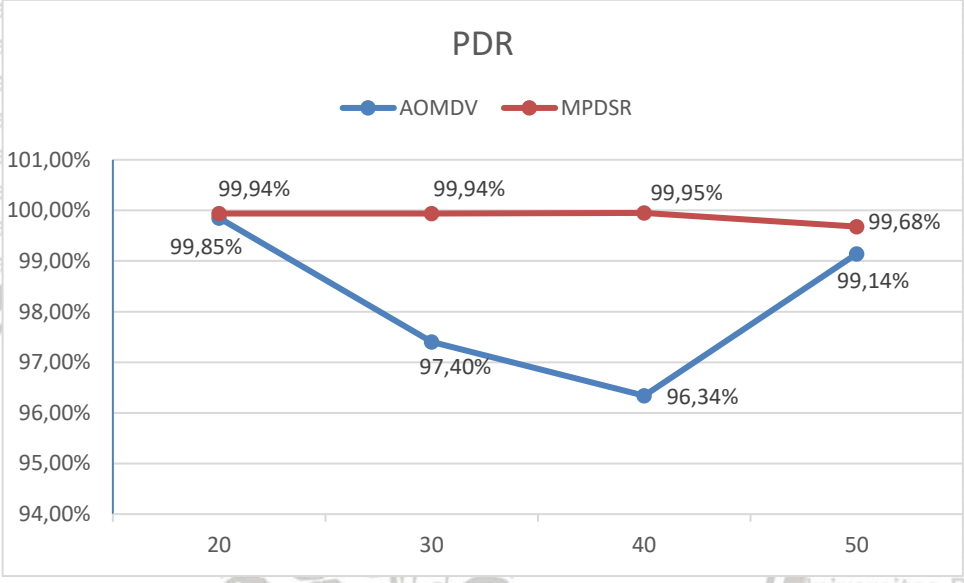
Pada skenario kedua, untuk mengetahui kinerja protokol terhadap ukuran paket data, simulasi dijalankan dengan variasi jumlah ukuran paket data sebesar 512 *bytes* dan 1024 *bytes* pada 20 *node*, 30 *node*, 40 *node*, 50 *node* secara bergantian menggunakan protokol *routing* AOMDV dan MP-DSR pada luas area 1000m x 1000m. Jenis trafik data yang digunakan adalah *constant Bit Rate* (CBR) atau UDP. Dengan pemodelan CBR, berarti setiap sumber trafik yang berada pada suatu jaringan akan mengirimkan datanya secara terus menerus.

5.2.1 Perbandingan Kinerja *Packet Delivery Ratio*

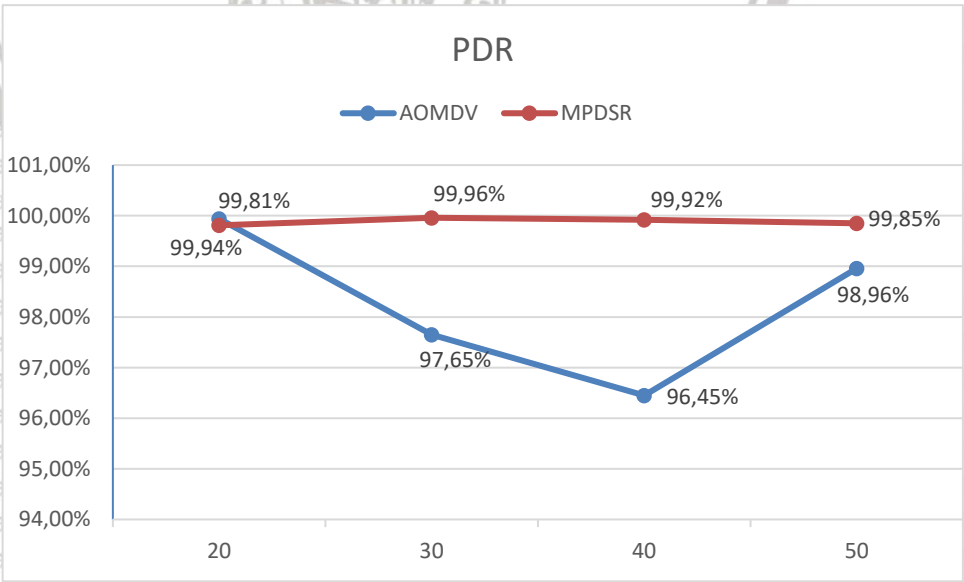
Kinerja dari *packet Delivery Ratio* (PDR) akan memberikan gambaran tentang seberapa baik suatu protokol dalam hal pengiriman paket data dengan variasi ukuran paket data. Pada tabel 5.6 menunjukkan hasil dari keseluruhan nilai *PDR* pada protokol AOMDV dan protokol MP-DSR.

Tabel 5.6 Nilai PDR vs Jumlah Ukuran Paket

Node	*512 bytes*		*1024 bytes*	
	AOMDV	MP-DSR	AOMDV	MP-DSR
20	99,85	99,94	99,94	99,81
30	97,40	99,94	97,65	99,96
40	96,34	99,95	96,45	99,92
50	99,14	99,68	98,96	99,85



Gambar 5.6a *Packet Delivery Ratio 512 bytes*



Gambar 5.6b *Packet Delivery Ratio 1024 bytes*

Gambar 5.6a, Gambar 5.6b menunjukkan bahwa setiap skenario yang dilakukan terhadap ukuran paket data, menunjukkan semakin besar ukuran paket data yang dikirimkan maka PDR yang dihasilkan akan semakin besar.

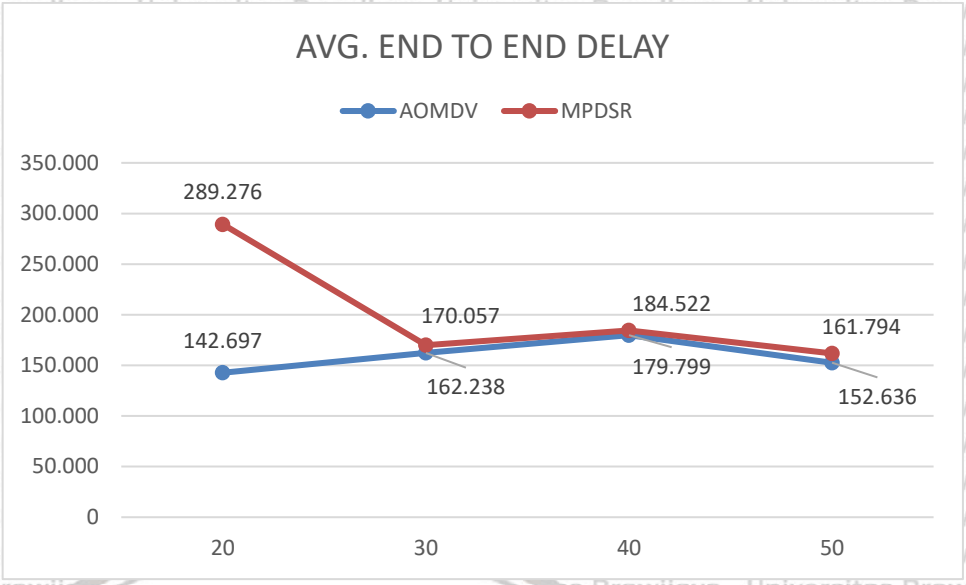
Hal ini disebabkan karena semakin besar ukuran paket, maka PDR akan membutuhkan waktu transfer yang lebih panjang dan ketika terjadi perubahan topologi saat transferdata dilakukan, akan terjadi paket drop yang akhirnya mempengaruhi jumlah data yang berhasil dikirimkan. Keberhasilan pengiriman paket data pada protokol MP-DSR lebih baik apabila dibandingkan dengan protokol AOMDV, ditujukkan pada protokol AOMDV yang memiliki rata-rata *ratio* pengiriman paket yang tidak terlalu signifikan dibandingkan dengan protokol MP-DSR, yaitu sebesar 0,9825 % untuk ukuran pakat data 1024 *bytes*, sedangkan protokol MP-DSR menghasilkan rata-rata *PDR* 0,99885 % untuk ukuran paket data 1024 *bytes*. Pada skenario dengan ukuran paket data 512 *bytes*, protokol MP-DSR menghasilkan rata-rata *PDR* sebesar 0,998775 %, sedangkan protokol AOMDV menghasilkan rata-rata 0,981825 % untuk ukuran paket data 512 *bytes*.

5.2.2 Perbandingan Kinerja *Average End to End Delay*

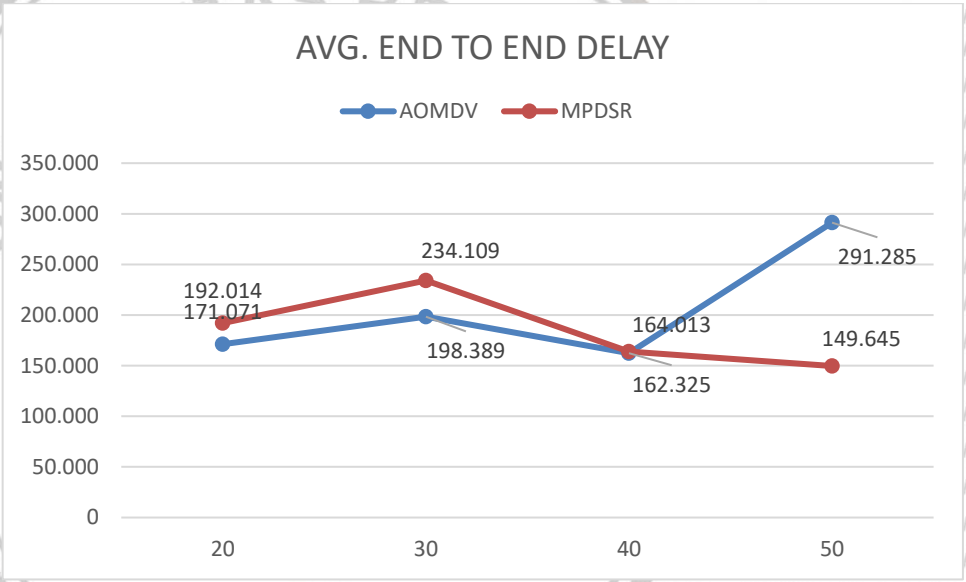
Rata-rata *delay* merupakan rata-rata waktu yang diperlukan mulai dari paket paket yang dikirimkan oleh *node* sumber sampai paket data tersebut berhasil diterima oleh *node* tujuan. Semakin kecil *delay* yang dialami, maka semakin baik kinerja dari jaringan tersebut. Hasil keseluruhan nilai *average end to end delay* pada protokol AOMDV dan protokol MP-DSR ditunjukkan pada tabel 5.7 di bawah ini.

Tabel 5.7 Nilai *Avg. End to End Delay* vs Ukuran Paket

Node	*512 bytes*		*1024 bytes*	
	AOMDV	MP-DSR	AOMDV	MP-DSR
20	142,697	289,276	171,071	192,014
30	162,238	170,057	198,389	234,109
40	179,799	184,522	162,325	164,013
50	152,636	161,794	291,285	149,645



Gambar 5.7a *Avg. End to End Delay 512 bytes*



Gambar 5.7b *Avg. End to End Delay 1024 bytes*

Gambar 5.7a, Gambar 5.7b menunjukkan hasil kinerja yang diperoleh protokol AOMDV lebih baik daripada protokol MP-DSR berdasarkan ukuran paket data 1024 *bytes*, dengan menghasilkan rata-rata *delay* 184,94525 m/s, sedangkan protokol MP-DSR menghasilkan 205,7675 m/s. Pada ukuran paket data 512 *bytes*, protokol AOMDV memiliki rata-rata *delay* 159,3425 m/s, sedangkan protokol MP-DSR memiliki rata-rata *delay* 201,41225 m/s.

Hal ini disebabkan karena *routing* protokol MP-DSR mengalami proses pencarian jalur lebih lama dan lebih panjang daripada *routing* protokol *MP-DSR* dan ukuran paket data yang dikirimkan semakin besar sehingga membutuhkan waktu yang lebih lama dari yang biasanya. MP-DSR menanggapi semua *RREQ* yang datang, sehingga kemacetan tidak dapat terhindarkan yang menyebabkan *delay* semakin besar. Sedangkan AOMDV hanya menanggapi RREQ pengiriman pertama.

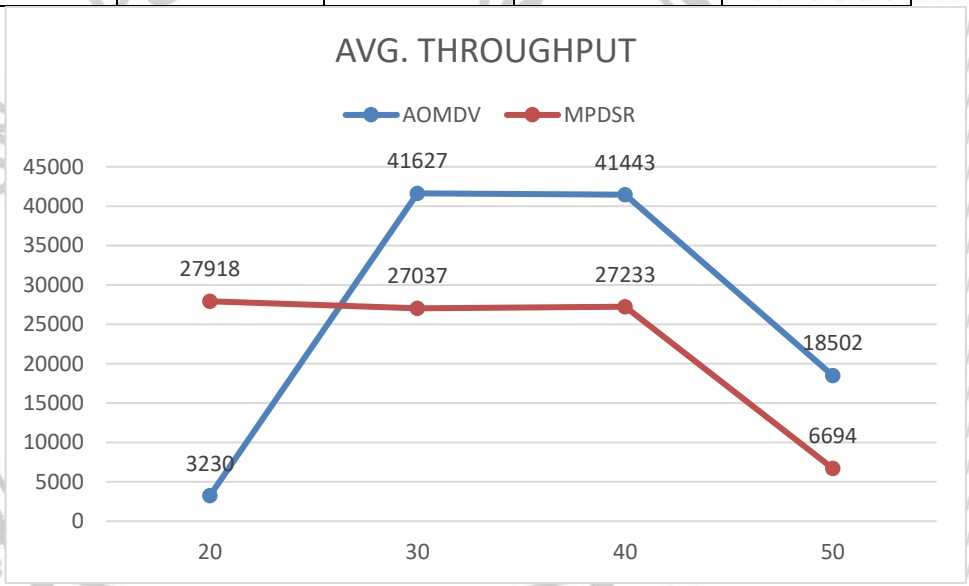
yang diterima dan mengabaikan RREQ selanjutnya dari *node* sumber yang berbeda.

5.2.3 Perbandingan Kinerja *Average Throughput*

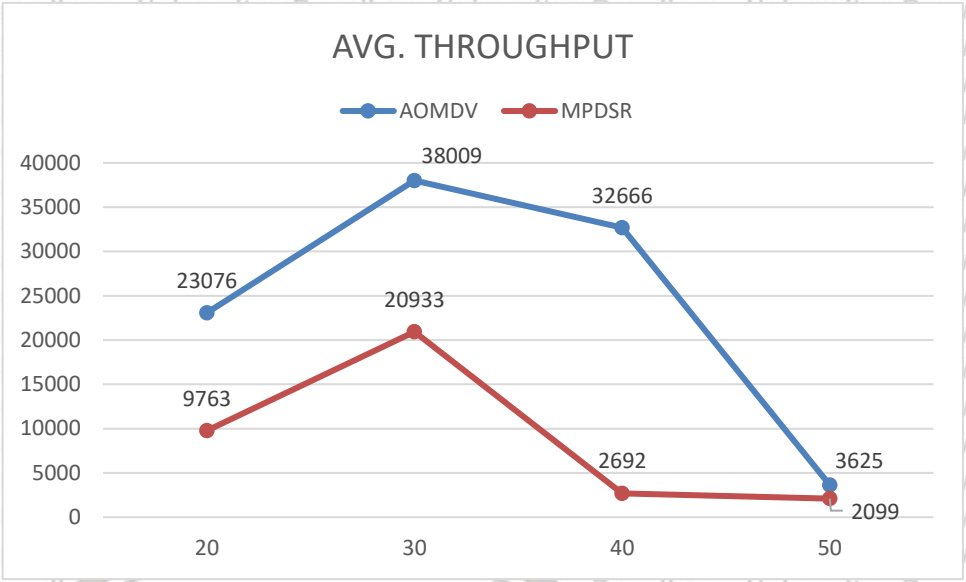
Semakin tinggi nilai *throughput* yang dihasilkan dalam sebuah jaringan, maka kinerja jaringan tersebut semakin baik. Pada Tabel 5.8 menunjukkan hasil keseluruhan nilai *throughput* protokol AOMDV dan protokol MP-DSR.

Tabel 5.8 Nilai *Avg. Troughtput* vs Ukuran Paket

Node	*512 bytes*		*1024 bytes*	
	AOMDV	MPDSR	AOMDV	MPDSR
20	3230	27918	23076	9763
30	41627	27037	38009	20933
40	41443	27233	32666	2692
50	18502	6694	3625	2099



Gambar 5.8a *Avg. Throughput 512 bytes*



Gambar 5.8b *Avg. Throughput 1024 bytes*

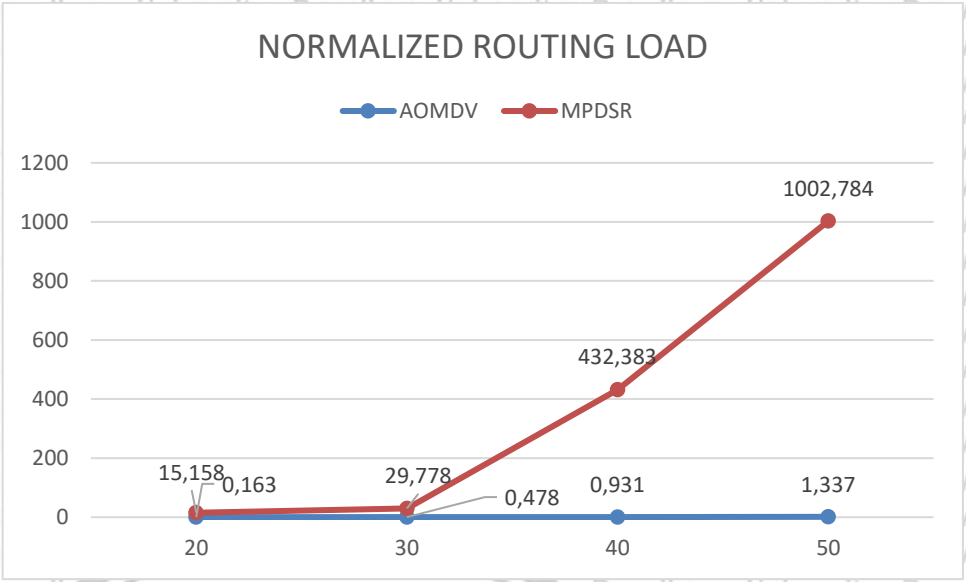
Gambar 5.8a, Gambar 5.8b menunjukkan bahwa kinerja protokol AOMDV lebih baik dari MP-DSR. Hal ini disebabkan hampir di setiap skenario yang disimulasikan, protokol AOMDV memiliki rata-rata *throughput* 24.344 kbps pada ukuran paket data 1024 *bytes* dan 26.200 kbps pada ukuran paket data 512 *bytes*. Sedangkan protokol MP-DSR memiliki rata-rata *throughput* 8.871 kbps untuk ukuran paket 1024 *bytes* dan 22.220 kbps dalam skenario dengan ukuran paket data 512 *bytes*. Berdasarkan gambar di atas, dapat disimpulkan bahwa jaringan masih mampu mem-*buffer* paket data sebelum dikirimkan ke *node* tujuan, meskipun ketika jumlah *node* bertambah dan ukuran paket data semakin besar.

5.2.4 Perbandingan Kinerja *Normalized Routing Load*

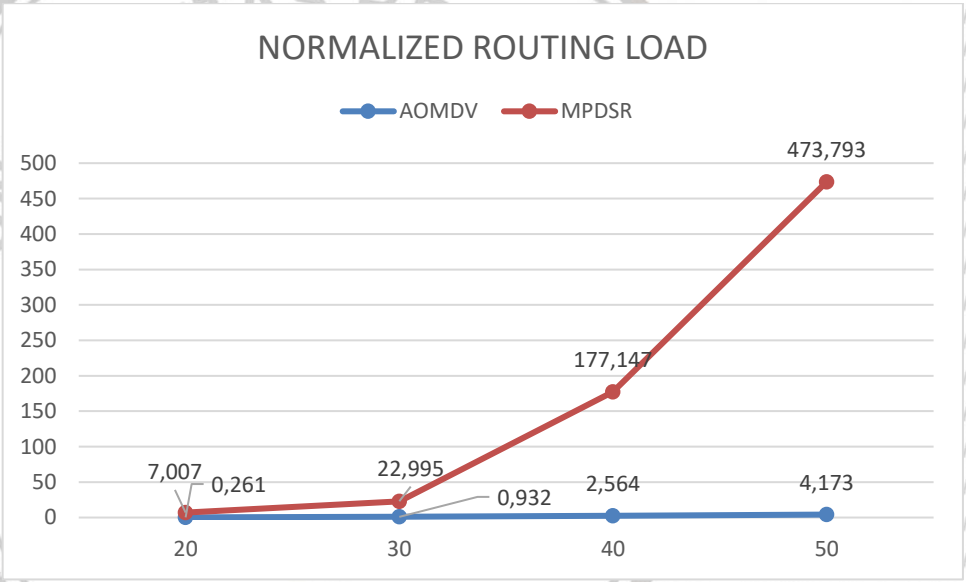
Pada simulasi yang dilakukan berdasarkan variasi ukuran paket data, hasil kinerja *normalized routing load* pada protokol AOMDV lebih baik dibandingkan protokol MP-DSR. Faktor yang mempengaruhi terjadinya *Normalized Routing Load* adalah karena *collision* (data saling bertabrakan) dan *congestion* (kemacetan) pada jaringan. Hasil keseluruhan nilai *normalized routing load* pada protokol AOMDV dan protokol MP-DSR ditunjukkan pada tabel 5.9 di bawah ini.

Tabel 5.9 Nilai *Avg. Normalized Routing Load* vs Ukuran Paket

Node	*512 bytes*		*1024 bytes*	
	AOMDV	MP-DSR	AOMDV	MP-DSR
20	0,163	15,158	0,261	7,007
30	0,478	29,778	0,932	22,995
40	0,931	432,383	2,564	177,147
50	1,337	1002,784	4,173	473,793



Gambar 5.9a *Normalized Routing Load 512 bytes*



Gambar 5.9b *Normalized Routing Load 1024 bytes*

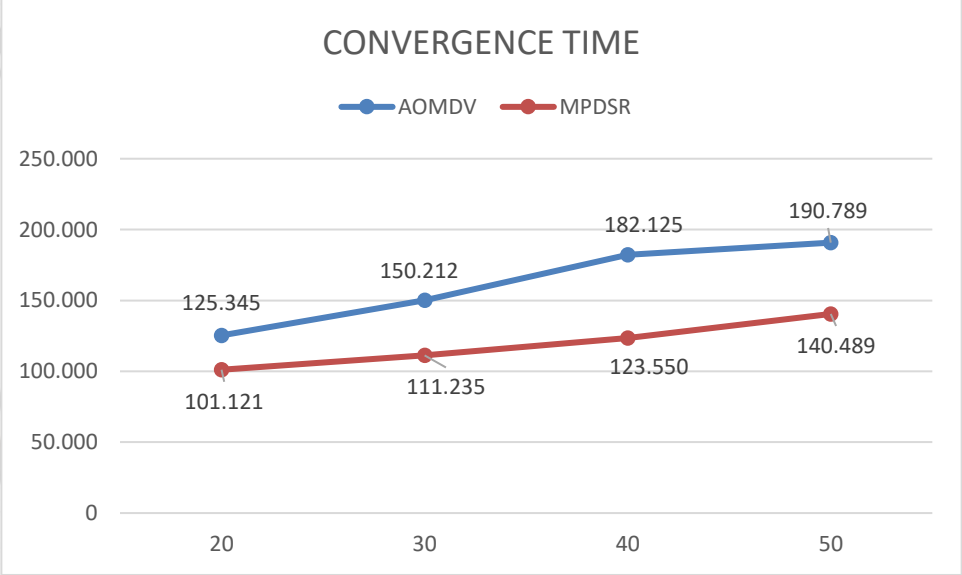
Gambar 5.9a, Gambar 5.9b menunjukkan semakin besar ukuran paket data, maka semakin besar nilai *Normalized Routing Load* yang didapatkan. Hal ini disebabkan karena nilai ukuran paket data berpengaruh pada pengiriman paket data ke *node* tujuan, yang memungkinkan paket data yang tidak sampai akan semakin meningkat. Sehingga dapat disimpulkan bahwa protokol AOMDV dengan ukuran paket data 512 *bytes* lebih baik daripada protokol MP-DSR, dengan nilai rata-rata *normalized routing load* 0,7272 dan protokol MP-DSR dengan nilai 370,025. Sedangkan pada protokol AOMDV, dengan ukuran paket data 1024 *bytes* memiliki rata-rata *normalized routing load* 1,9825 dan protokol MP-DSR dengan nilai 170,235.

5.2.5 Perbandingan Kinerja *Convergence Time*

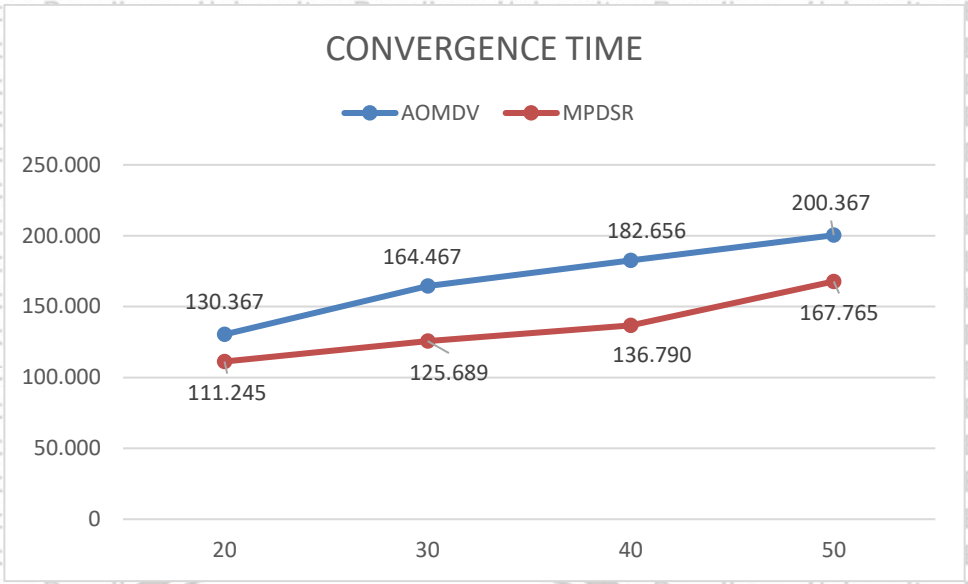
Pada simulasi yang dilakukan berdasarkan variasi ukuran paket data, hasil kinerja *convergence time* pada protokol MP-DSR lebih baik dibandingkan protokol AOMDV .

Tabel 5.1- Nilai *Avg. Convergence Time* vs Ukuran Paket

Node	*512 bytes*		*1024 bytes*	
	AOMDV	MP-DSR	AOMDV	MP-DSR
20	125,345	101,121	130,367	111,245
30	150,212	111,235	164,467	125,689
40	182,125	123,550	182,656	136,790
50	190,789	140,489	200,367	167,765



Gambar 5.10a *Convergence Time 512 bytes*



Gambar 5.10b *Convergence Time 1024 bytes*

Gambar 5.10a, Gambar 5.10b menunjukkan hasil kinerja yang diperoleh protokol MP-DSR lebih baik daripada protokol AOMDV berdasarkan ukuran paket data 1024 *bytes*, dengan menghasilkan rata-rata *convergence time* 135,37225 m/s, sedangkan protokol AOMDV menghasilkan 169,46425 m/s. Pada ukuran paket data 512 *bytes*, protokol MP-DSR memiliki rata-rata *convergence time* 119,09875 m/s, sedangkan protokol AOMDV memiliki rata-rata *delay* 162,11775 m/s.

BAB 6 PENUTUP

6.1 Kesimpulan

Dari hasil simulasi yang dijalankan secara keseluruhan menggunakan *Network Simulator* 2, pada skenario penambahan *node* berdasarkan nilai parameter kerja *throughput*, *Packet delivery ratio*, *end to end delay* dan *packet* kinerja *routing* protokol AOMDV lebih unggul dari *routing* protokol MP-DSR. Hal ini dikarenakan protokol AOMDV unggul pada nilai paramater *throughput*, *end to end delay* dan *normalized routing load*, dengan nilai rata-rata *throughput* 26.200 kbps, nilai rata-rata *end to end delay* 159,3425 m/s dan *normalized routing load* 0,7272. Sedangkan , protokol MP-DSR unggul pada nilai parameter *packet delivery ratio* dan *convergence time*, dengan nilai rata-rata *packet delivery ratio* sebesar 0,998775 % dan *convergence time* 119,09875 m/s.

Pada skenario penambahan ukuran paket data, kinerja dari protokol AOMDV juga unggul berdasarkan nilai parameter kerja *throughput* pada ukuran paket data 512 *bytes*, *Normalized routing load pada ukuran* paket data 512 bytes dan end *to end delay* pada paket ukuran data 512 *bytes*. Hal ini disebabkan semakin besar ukuran paket data, maka semakin besar waktu pengukuran yang dibutuhkan dalam suatu jaringan. Kondisi seperti ini dapat juga mempengaruhi hasil kinerja dari *throughput*. Sedangkan, protokol MP-DSR unggul pada *packet delivery ratio* dan *convergence time* dengan ukuran paket data 512 *bytes*. Hal ini disebabkan semakin besar ukuran paket data, maka waktu transfer yang dibutuhkan PDR dan *convergence* semakin lama.

Kinerja *routing* protokol berdasarkan nilai kinerja *Quality of services* (QOS) terhadap penambahan jumlah *node* dan variasi ukuran paket sangat berpengaruh pada saat melakukan pengiriman paket data. Hasil yang diperoleh dalam pengujian, dapat disimpulkan bahwa protokol AOMDV lebih baik pada saat melakukan pengiriman paket data dibandingkan dengan MP-DSR. Secara keseluruhan, protokol AOMDV dan protokol MPDSR memilki tingkat keunggulannya masing-masing. Protokol MP-DSR lebih cocok digunakan untuk kepadatan jaringan berskala kecil. Sedangkan, protokol AOMDV lebih cocok digunakan untuk kepadatan jaringan dengan skala menengah atas. Hal ini disebabkan karena penambahan *node* sangat berpengaruh pada penurunan nilai kinerja parameter QoS. Semakin besar kepadatan *node* maka semakin banyak hop yang terbentuk sehingga dapat menghambat proses pengiriman paket data menuju *node* tujuan.

6.2 Saran

Beberapa saran yang dapat dijadikan pertimbangan untuk pengembangan pengujian selanjutnya adalah :

1. Melakukan analisis dan pengujian lebih signifikan dengan menggunakan parameter yang berbeda, seperti jalur trafik yang berbeda agar mengetahui kinerja *routing* protokol yang lebih baik berdasarkan trafik yang digunakan, kecepatan *node*, model mobilitas dan jenis propagasi yang berbeda.
2. Melakukan pengujian yang lebih spesifik terhadap variasi jumlah *node* dan ukuran paket data yang memiliki pengaruh terhadap kinerja *routing* protokol
3. Analisis yang selanjutnya diharapkan bisa menjelaskan tentang keamanan pada jaringan *Mobile Ad Hoc Network* berdasarkan masing-masing *routing* protokol.

DAFTAR PUSTAKA

Aarti, 2013. Study of MANET: Characteristic, Challenges, Applications and Security Attacks. *IJARCSSE*, 3(5), pp. 252-257.

Alfred V. Aho, B. W. K. P. J. W., 1988. *The AWK Programming Languange*. 1 penyunt. New Jersey: ADDISON WESLEY PUBLISHING COMPANY.

Amilia, F., Marzuki, & Agustina. (2014). Analisis Perbandingan Kinerja Protokol Dynamic Source Routing (DSR) dan Geographic Routing Protocol (GRP) pada Mobile Ad-Hoc Network (MANET). Jurnal Sains, Teknologi dan Indonesia, Vol. 12(No. 1), 9-15.

Anisia, R., Muhadi, R. dan Negara, R.M.. 2016. Analsis Performasi Routing Protocol OLSR dan AOMDV pada Vehicular Ad Hoc Network (VARNET). Bandung : Jurusan Teknik Telekomunikasi, Fakultas Teknik, Universitas Telkom.

Ayyasamy, D.A., 2013.Performance Evalation of Load Based Channel Aware Routing in MANETs with Reusable Path. *International Journal of Engineering and Advanced of Technology.*

Benardi, B. (2009). Analisa Unjuk Kerja Jaringan Nirkabel Ad-Hoc dalam Beberapa Situasi yang Berbeda Ditinjau dari Sudut Pandang Routing. Universitas Mercubuana.

Demir, Tamer, "Simulation of Ad Hoc Networks with DSR Protocol". 2001.

Hantoro, G.D. (2009). WiFi (Wireless LAN) Jaringan Komputer Tanpa Kabel. Bandung: Penerbit informatika.

Denatama, M.I., 2016. Analasis Perbandingan Kinerja Protokol Routing DSDV dan OLSR Untuk Perubahan Kecepatan Mobilitas pada Standar IEEE 802.11ah. Infatel, 8(2), pp. 2085-3688.

Imawan, Didik., 2013. "Analisis Kinerja Pola-Pola Trafik Pada Beberapa Protokol Routing Dalam Jaringan MANET", Fakultas Teknologi Informasi, Institut Teknologi Sepuluh November Surabaya.

Issarriyakul, T., 2012. *Introduction to Network Simulator NS2*. Penyunt. New York: Springer Science+Business Media.

Johnson, D. Y.Hu, dan D. Maltz. Februari 2007. RFC 4728. The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4.

Marina, M. K., 2001. On-demand Multipath Distance Vector Routing in Ad Hoc Networks. IEEE 9th International Conference on Network Protocols (ICNP), Volume 1, pp. 14-23.

Michiardi, Pietro and R. Molva, "Simulation-based Analysis of Security Exposures in Mobile Ad Hoc Networks", European Wireless Conference, 2002.

NS-2, The NS Manual (formally known as NS Documentation) available at http://www.isi.edu/nsnam/ns/doc.NAM.

Perkins, C. E., 2001. Performance comparison of two on-demand routing protocols for ad hoc networks. *IEEE Personal Communications*, 8(1), pp. 16-28.

Sarkar, S. K., T.G. Barvasaraju dan C. Puttamadappa.(2008).Ad hoc mobile wireless networks : principles, protocols, and applications.Boca Raton:Auerbach Publications.

Tamilarasan, S dan Aramudan. "A Performance and Analysis of Misbehaving node in MANET using Intrusion Detection System". International Journal of Computer Science and Network Security.2011.

Tanudjaya T., R., 2016. Analisis Kinerja Protokol Routing Ad Hoc On-Demand Distance Vector (AODV) dan Destination Sequence Distance Vector (DSDV) pada MANET.

Toussaint, M.T., "Multipath Routing in Mobile Ad Hoc Networks".2003.

Victor, K. R., 2013. Routing and Reducing Pertubation in Mobile ad Hoc Networks (Manets) for Efficient Communication. *International Journal of Advanced Computer Research,* 4(4).

LAMPIRAN A HASIL SIMULASI

A.1 AOMDV 20 NODE 512 *bytes*

```
Sent Packets                = 51982
Received Packets            = 51902
Packets Delivery Ratio      = 99,85
Average End to End Delay    = 142,697
Average Throughput [kbps]   = 3230
Normalized Routing Load     = 0,163
Convergence Time            = 125,345
```

A.2 AOMDV 30 NODE 512 *bytes*

```
Sent Packets                = 54389
Received Packets            = 52976
Packets Delivery Ratio      = 97,40
Average End to End Delay    = 162,238
Average Throughput [kbps]   = 41627
Normalized Routing Load     = 0,478
Convergence Time            = 150,212
```

A.3 AOMDV 40 NODE 512 *bytes*

```
Sent Packets                = 50075
Received Packets            = 48240
Packets Delivery Ratio      = 96,34
Average End to End Delay    = 179,799
Average Throughput [kbps]   = 41443
Normalized Routing Load     = 0.931
Convergence Time            = 182,125
```

A.4 AODMV 50 NODE 512 *bytes*

```
Sent Packets                = 59296
Received Packets            = 58784
Packets Delivery Ratio      = 99,14
Average End to End Delay    = 152.636
Average Throughput [kbps]   = 18502
Normalized Routing Load     = 1,337
Convergence Time            = 190,789
```

A.5 MP-DSR 20 NODE 512 *bytes*

```
Sent Packets                = 105056
Received Packets            = 104996
Packets Delivery Ratio      = 99,94
Average End to End Delay    = 289,276
Average Throughput [kbps]   = 27918
Normalized Routing Load     = 15,158
Convergence Time            = 101,121
```

A.6 MP-DSR 30 NODE 512 *bytes*

```
Sent Packets                = 105761
Received Packets            = 105698
Packets Delivery Ratio      = 99,94
Average End to End Delay    = 170,057
Average Throughput [kbps]   = 27037
Normalized Routing Load     = 29,778
Convergence Time            = 111,235
```

A.7 MP-DSR 40 NODE 512 *bytes*

```
Sent Packets                = 107162
Received Packets            = 107106
Packets Delivery Ratio      = 99,95
Average End to End Delay    = 184,522
Average Throughput [kbps]   = 27233
Normalized Routing Load     = 432,383
Convergence Time            = 123,550
```

A.8 MP-DSR 50 NODE 512 *bytes*

```
Sent Packets                = 41682
Received Packets            = 41550
Packets Delivery Ratio      = 99,68
Average End to End Delay    = 161,794
Average Throughput [kbps]   = 6694
Normalized Routing Load     = 1002,784
Convergence Time            = 140,489
```

A.1 AOMDV 20 NODE 1024 *bytes*

```
Sent Packets                = 110834
Received Packets            = 110768
Packets Delivery Ratio      = 99,94
Average End to End Delay    = 23076
Average Throughput [kbps]   = 171,071
Normalized Routing Load     = 0,261
Convergence Time            = 130,367
```

A.1 AOMDV 30 NODE 1024 *bytes*

```
Sent Packets                = 69717
Received Packets            = 68079
Packets Delivery Ratio      = 97,65
Average End to End Delay    = 198,389
Average Throughput [kbps]   = 38009
Normalized Routing Load     = 0,932
Convergence Time            = 164,467
```

A.1 AOMDV 40 NODE 1024 *bytes*

```
Sent Packets                = 65393
Received Packets            = 63073
Packets Delivery Ratio      = 96,45
Average End to End Delay    = 162,325
Average Throughput [kbps]   = 32666
Normalized Routing Load                 = 20,97
Convergence Time            = 182,656
```

A.1 AOMDV 50 NODE 1024 *bytes*

```
Sent Packets                = 11596
Received Packets            = 11475
Packets Delivery Ratio      = 98,96
Average End to End Delay    = 291,285
Average Throughput [kbps]   = 3625
Normalized Routing Load     = 4,173
Convergence Time            = 200,367
```

A.13 MP-DSR 20 NODE 1024 *bytes*

```
Sent Packets                = 11475
Received Packets            = 51114
Packets Delivery Ratio      = 99,94
Average End to End Delay    = 289,276
Average Throughput [kbps]   = 27918
Normalized Routing Load     = 7,007
Convergence Time            = 111,245
```

A.14 MP-DSR 30 NODE 1024 *bytes*

```
Sent Packets                = 119708
Received Packets            = 119656
Packets Delivery Ratio      = 99,81
Average End to End Delay    = 192,014
Average Throughput [kbps]   = 9763
Normalized Routing Load     = 22,995
Convergence Time            = 125,689
```

A.15 MP-DSR 40 NODE 1024 *bytes*

```
Sent Packets                = 45421
Received Packets            = 45384
Packets Delivery Ratio      = 99,92
Average End to End Delay    = 164,013
Average Throughput [kbps]   = 2692
Normalized Routing Load     = 177,147
Convergence Time            = 136,790
```

A.16 MP-DSR 50 NODE 1024 *bytes*

```
Sent Packets                = 34942
Received Packets            = 34890
Packets Delivery Ratio      = 99,85
Average End to End Delay    = 149,645
Average Throughput [kbps]   = 2099
Normalized Routing Load     = 473,793
Convergence Time            = 167,765
```

LAMPIRAN B KODE PROGRAM

```
# Define options
set val(chan)           Channel/WirelessChannel    ;# channel type
set val(prop)         Propagation/TwoRayGround     ;# radio-
propagation model
set val(netif)          Phy/WirelessPhy            ;# network
interface type
set val(mac)            Mac/802_11                 ;# MAC type
set val(ifq)            Queue/DropTail/PriQueue    ;# interface
queue type
set val(ll)             LL                         ;# link layer
type
set val(ant)            Antenna/OmniAntenna        ;# antenna
model
set val(ifqlen)         50                         ;# max packet
in ifq
set val(nn)             20                         ;# number of
mobilenodes
set val(rp)             AOMDV                      ;# routing
protocol
set val(x)              1000                       ;# X dimension
of topography
set val(y)              1000                       ;# Y dimension
of topography
set val(stop)           1000                       ;# time of
simulation end

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

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

# set up topography object
set topo       [new Topography]

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

create-god $val(nn)

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

```
    -macTrace OFF \
    -movementTrace ON
for {set i 0} {$i < $val(nn) } { incr i } {
set node_($i) [$ns node]
$node_($i) set X_ [ expr 10+round(rand()*480) ]
$node_($i) set Y_ [ expr 10+round(rand()*380) ]
$node_($i) set Z_ 0.0
}
for {set i 0} {$i < $val(nn) } { incr i } {
$ns at [ expr 15+round(rand()*60) ] "$node_($i) setdest [ expr
10+round(rand()*480) ] [ expr 10+round(rand()*380) ] [ expr
2+round(rand()*15) ]"
}

set tcp [new Agent/TCP/Newreno]
$tcp set class_ 2
set sink [new Agent/TCPSink]
$ns attach-agent $node_(2) $tcp
$ns attach-agent $node_(8) $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp

set cbr [new Application/Traffic/CBR]
$cbr attach-agent $tcp
$cbr set type_ CBR
$cbr set packet_size_ 512
$cbr set rate_ 1mb
$cbr set random_ false

$ns at 0.1 "$cbr start"
$ns at 10.0 "$ftp start"

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

# Define node initial position in nam
for {set i 0} {$i < $val(nn)} { incr i } {
# 30 defines the node size for nam
$ns initial_node_pos $node_($i) 30
}

# Telling nodes when the simulation ends
for {set i 0} {$i < $val(nn) } { incr i } {
    $ns at $val(stop) "$node_($i) reset";
}
# ending nam and the simulation
$ns at $val(stop) "$ns nam-end-wireless $val(stop)"
```

```
$ns at $val(stop) "stop"
$ns at 1100 "puts \"end simulation\" ; $ns halt"
proc stop {} {
    global ns tracefd namtrace
    $ns flush-trace
    close $tracefd
    close $namtrace
    exec gawk -f pdr.awk traceAOMDV.tr &
    exec nam simAOMDV.nam &
}
$ns run

# Define options
set val(chan)           Channel/WirelessChannel    ;# channel type
set val(prop)         Propagation/TwoRayGround     ;# radio-
propagation model
set val(netif)          Phy/WirelessPhy            ;# network
interface type

set val(mac)            Mac/802_11                 ;# MAC type
set val(ifq)            CMUPriQueue    ;# interface queue type
set val(ll)             LL                         ;# link layer
type
set val(ant)            Antenna/OmniAntenna        ;# antenna
model
set val(ifqlen)         50                         ;# max packet
in ifq
set val(nn)             20                         ;# number of
mobilenodes
set val(rp)             MPDSR                      ;# routing
protocol
set val(x)              1000                       ;# X dimension
of topography
set val(y)              1000                       ;# Y dimension
of topography
set val(stop)           1000                       ;# time of
simulation end

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

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

# set up topography object
set topo       [new Topography]

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

create-god $val(nn)

# configure the nodes
$ns node-config -adhocRouting $val(rp) \
    -llType $val(ll) \
    -macType $val(mac) \
    -ifqType $val(ifq) \
```

```
    -ifqLen $val(ifqlen) \
    -antType $val(ant) \
    -propType $val(prop) \
    -phyType $val(netif) \
    -channelType $val(chan) \
    -topoInstance $topo \
    -agentTrace ON \
    -routerTrace ON \
    -macTrace OFF \
    -movementTrace ON

for {set i 0} {$i < $val(nn) } { incr i } {
set node_($i) [$ns node]
$node_($i) set X_ [ expr 10+round(rand()*480) ]
$node_($i) set Y_ [ expr 10+round(rand()*380) ]
$node_($i) set Z_ 0.0
}

for {set i 0} {$i < $val(nn) } { incr i } {
$ns at [ expr 15+round(rand()*60) ] "$node_($i) setdest [ expr
10+round(rand()*480) ] [ expr 10+round(rand()*380) ] [ expr
2+round(rand()*15) ]"

}

set tcp [new Agent/TCP/Newreno]
$tcp set class_ 2
set sink [new Agent/TCPSink]
$ns attach-agent $node_(2) $tcp
$ns attach-agent $node_(8) $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp

set cbr [new Application/Traffic/CBR]
$cbr attach-agent $tcp
$cbr set type_ CBR
$cbr set packet_size_ 1024
$cbr set rate_ 1mb
$cbr set random_ false

$ns at 0.1 "$cbr start"
$ns at 10.0 "$ftp start"

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

# Define node initial position in nam
for {set i 0} {$i < $val(nn)} { incr i } {
# 30 defines the node size for nam
$ns initial_node_pos $node_($i) 30
}
```

```
# Telling nodes when the simulation ends
for {set i 0} {$i < $val(nn) } { incr i } {
    $ns at $val(stop) "$node_($i) reset";
}
# ending nam and the simulation
$ns at $val(stop) "$ns nam-end-wireless $val(stop)"
$ns at $val(stop) "stop"
$ns at 1100 "puts \"end simulation\" ; $ns halt"
proc stop {} {
    global ns tracefd namtrace
    $ns flush-trace
    close $tracefd
    close $namtrace
    exec   gawk -f pdr.awk traceMPDSR.tr &
    exec nam simMPDSR.nam &
}

$ns run
```

```
[pdr.awk]
BEGIN {
        sendLine = 0;
        recvLine = 0;
        fw = 0;
      jarak = 0;
      TC =0;
      idle=0.0000018;
      pt=0.001;
      recvs = 0;
      routing_packets = 0;
}

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

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

$0 ~/^f.* RTR/ {
        jarak ++ ;
}
$0 ~/^f.* RTR/ {
        fw ++ ;
}
{
    if($4 == "AGT" && $1 == "s" && seqno < $6) {
        seqno = $6;
    }
    #end-to-end delay
```

```
        if($4 == "AGT" && $1 == "s") {
            start_time[$6] = $2;
        } else if(($7 == "tcp" || $7 == "cbr") && ($1 == "r"))
{
            end_time[$6] = $2;
        } else if($1 == "D" && ($7 == "tcp" || $7 == "cbr")) {
            end_time[$6] = -1;
        }
}
{
if (( $1 == "r") &&  ( $7 == "tcp" ||   $7 == "cbr") && ( $19=="4"
))  recvs++;
if (($1 == "s" || $1 == "r") && $4 == "RTR" && ($7 =="AOMDV" ||
$7 =="DSR"))    routing_packets++;
}

END {

    for(i=0; i<=seqno; i++) {

        if(end_time[i] > 0)
        {
            delay[i] = end_time[i] - start_time[i];
            count++;
        }
        else
        {
            delay[i] = -1;
        }
    }
    for(i=0; i<=seqno; i++) {
        if(delay[i] > 0) {
            n_to_n_delay = n_to_n_delay + delay[i];
        }
    }
    n_to_n_delay = n_to_n_delay/count;
t=fw*0.5;
```

```
        jarak1=jarak%50;
mpr=jarak%500;
jarakall=50+jarak1;
        bawah=4*3.14*jarakall;
        a=0.125/bawah;
        b=a*a;
        pr=pt*b;
        p=(pr+pt+idle)*mpr;

        printf "Packet Transmite \t= %d \n", sendLine;
        printf "Packet Recieved \t= %d \n", recvLine;
        printf "Packet PDR Ratio \t= %.4f \n", (recvLine/sendLine);
        printf "normalized routing load     \t=      %.4f      \n",
(sendLine-recvLine)/sendLine;
        printf "Energy          \t= %.9f \n", p/1.8;
        printf("Packet Routing  \t= %d\n", routing_packets);
        printf "Delay           \t= " n_to_n_delay * 1000 " ms \n";
        printf "Throughput      \t= %.2f \n", t;

}
```

```
[qos.awk]
BEGIN {
print("\n\n******** Network Statistics ********\n");

# Change array size from 50 to any number of nodes for which u
are doing simulation.
#  i.e.  change  values  of  arrays  packet_sent,  packet_drop,
packet_recvd, packet_forwarded, energy_left,
packet_sent[50] = 0;
packet_drop[50] = 0;
packet_recvd[50] = 0;
packet_forwarded[50] = 0;

# Change energy assigned to initial node (as per your simulation
tcl file)
# Initial Energy assigned to each node in Joules

energy_left[50] = 10000.000000;

total_pkt_sent=0;
total_pkt_recvd=0;
total_pkt_drop=0;
total_pkt_forwarded=0;
pkt_delivery_ratio = 0;
total_hop_count = 0;
avg_hop_count = 0;
overhead = 0;
start = 0.000000000;
end = 0.000000000;
packet_duration = 0.0000000000;
recvnum = 0;
delay = 0.000000000;
sum = 0.000000000;
i=0;
total_energy_consumed = 0.000000;
}
{
```

```
state       =       $1;
time        =       $3;
# For energy consumption statistics see trace file
node_num    =       $5;
energy_level        =       $7;

node_id     =       $9;
level       =       $19;
pkt_type    =       $35;
packet_id   =       $41;
no_of_forwards      =       $49;

# In for loop change values from 50 to number of nodes that u
specify for your simulation

if((pkt_type == "cbr") && (state == "s") && (level=="AGT")) {
      for(i=0;i<50;i++) {
            if(i == node_id) {
            packet_sent[i] = packet_sent[i] + 1;}
}
}else if((pkt_type == "cbr") && (state == "r") && (level=="AGT"))
{
      for(i=0;i<50;i++) {
            if(i == node_id) {
            packet_recvd[i] = packet_recvd[i] + 1; }
}
}else if((pkt_type == "cbr") && (state == "d")) {
      for(i=0;i<50;i++) {
            if(i == node_id) {
            packet_drop[i] = packet_drop[i] + 1; }
}
}else if((pkt_type == "cbr") && (state == "f")) {
      for(i=0;i<50;i++) {
            if(i == node_id) {
            packet_forwarded[i] = packet_forwarded[i] + 1; }
}
}


# To calculate total hop counts
if ((state == "r") && (level == "RTR") && (pkt_type == "cbr")) {
total_hop_count = total_hop_count + no_of_forwards; }

# Routing Overhead
if ((state == "s" || state == "f") && (level == "RTR") && (pkt_type
== "message")) { overhead = overhead + 1; }

# Calculating Average End to End Delay

#if ( start_time[packet_id] == 0 ) { start_time[packet_id] =
time; }
if (( state == "s") &&  ( pkt_type == "cbr" ) && ( level == "AGT"
))  { start_time[packet_id] = time; }

 if (( state == "r") &&  ( pkt_type == "cbr" ) && ( level == "AGT"
)) {  end_time[packet_id] = time;  }
  else {  end_time[packet_id] = -1;  }
```

```
# To Calculate Average Energy Consumption
# Change number of nodes in this for loop also

if(state == "N") {
      for(i=0;i<50;i++) {
            if(i == node_num) {
                                  energy_left[i] = energy_left[i] -
(energy_left[i] - energy_level);
                        }
                  }
}
}
# In this for loop also change

END {
for(i=0;i<50;i++) {
printf("%d %d \n",i, packet_sent[i]) > "pktsent.txt";
printf("%d %d \n",i, packet_recvd[i]) > "pktrecvd.txt";
printf("%d %d \n",i, packet_drop[i]) > "pktdrop.txt";
printf("%d %d \n",i, packet_forwarded[i]) > "pktfwd.txt";
printf("%d %.6f \n",i, energy_left[i]) > "energyleft.txt";

total_pkt_sent = total_pkt_sent + packet_sent[i];
total_pkt_recvd = total_pkt_recvd + packet_recvd[i];
total_pkt_drop = total_pkt_drop + packet_drop[i];
total_pkt_forwarded = total_pkt_forwarded + packet_forwarded[i];
total_energy_consumed = total_energy_consumed + energy_left[i];

}
printf("Total Packets Sent              :     %d\n",total_pkt_sent);
printf("Total Packets Received          :
     %d\n",total_pkt_recvd);
printf("Total Packets Dropped           :
     %d\n",total_pkt_drop);
printf("Total Packets Forwarded     :     %d\n",
total_pkt_forwarded);

pkt_delivery_ratio = (total_pkt_recvd/total_pkt_sent)*100;

printf("Packet Delivery Ratio           :
     %.2f%\n",pkt_delivery_ratio);

printf("The total hop counts are    :     %d\n",
total_hop_count);

avg_hop_count = total_hop_count/total_pkt_recvd;
printf("Average Hop Count           :     %d          hops\n",
avg_hop_count);

printf("Routing Overhead            :     %d\n", overhead);

printf("Normalized Routing Load     :     %.4f\n",
overhead/total_pkt_recvd);
```

```
printf("Througphut of the network(KBps)    :      %.4f\n",
((total_pkt_recvd/1000)*512)/1024);
# For End to End Delay

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

delay=sum/recvnum;
printf("Average End to End Delay    :%.9f ms\n", delay);

# Below change 50 to number of nodes that u want

printf("Total Energy Consumed              :%.6f\n",  (50*10000)-
total_energy_consumed);

# Below change 50 to number of nodes that u want

printf("Protocol Energy Consumption      :%.6f\n",   100.000000-
((total_energy_consumed/(50*10000.000000))*100.000000));

if(((total_pkt_recvd + total_pkt_drop)/total_pkt_sent)==1) {
printf("Statistics Correct !!!");
}
}

[simAOMDV.nam]
n -t * -s 0  -x 33 -y 112 -Z 0 -z 30  -v circle -c black
n -t * -s 1  -x 350 -y 99 -Z 0 -z 30  -v circle -c black
n -t * -s 2  -x 251 -y 25 -Z 0 -z 30  -v circle -c black
n -t * -s 3  -x 439 -y 146 -Z 0 -z 30  -v circle -c black
n -t * -s 4  -x 477 -y 363 -Z 0 -z 30  -v circle -c black
n -t * -s 5  -x 38 -y 338 -Z 0 -z 30  -v circle -c black
n -t * -s 6  -x 487 -y 276 -Z 0 -z 30  -v circle -c black
n -t * -s 7  -x 382 -y 214 -Z 0 -z 30  -v circle -c black
n -t * -s 8  -x 437 -y 111 -Z 0 -z 30  -v circle -c black
n -t * -s 9  -x 186 -y 312 -Z 0 -z 30  -v circle -c black
n -t * -s 10  -x 50 -y 239 -Z 0 -z 30  -v circle -c black
n -t * -s 11  -x 425 -y 151 -Z 0 -z 30  -v circle -c black
n -t * -s 12  -x 141 -y 167 -Z 0 -z 30  -v circle -c black
n -t * -s 13  -x 467 -y 314 -Z 0 -z 30  -v circle -c black
n -t * -s 14  -x 152 -y 211 -Z 0 -z 30  -v circle -c black
n -t * -s 15  -x 386 -y 221 -Z 0 -z 30  -v circle -c black
n -t * -s 16  -x 151 -y 307 -Z 0 -z 30  -v circle -c black
n -t * -s 17  -x 248 -y 49 -Z 0 -z 30  -v circle -c black
n -t * -s 18  -x 488 -y 187 -Z 0 -z 30  -v circle -c black
n -t * -s 19  -x 283 -y 120 -Z 0 -z 30  -v circle -c black
V -t * -v 1.0a5 -a 0
W -t * -x 1000 -y 1000
A -t * -n 1 -p 0 -o 0x7fffffff -c 30 -a 1
A -t * -h 1 -m 1073741823 -s 0
+ -t 0.100000000 -s 2 -d -1 -p tcp -e 40 -c 2 -a 0 -i 0 -k AGT
- -t 0.100000000 -s 2 -d -1 -p tcp -e 40 -c 2 -a 0 -i 0 -k AGT
```

```
h -t 0.100000000 -s 2 -d -1 -p tcp -e 40 -c 2 -a 0 -i 0 -k AGT
+ -t 0.100000000 -s 2 -d -1 -p AOMDV -e 52 -c 2 -a 0 -i 0 -k RTR
- -t 0.100000000 -s 2 -d -1 -p AOMDV -e 52 -c 2 -a 0 -i 0 -k RTR
h -t 0.100000000 -s 2 -d -1 -p AOMDV -e 52 -c 2 -a 0 -i 0 -k RTR
r -t 0.101180081 -s 17 -d -1 -p AOMDV -e 52 -c 2 -a 0 -i 0 -k RTR
r -t 0.101180334 -s 19 -d -1 -p AOMDV -e 52 -c 2 -a 0 -i 0 -k RTR
r -t 0.101180412 -s 1 -d -1 -p AOMDV -e 52 -c 2 -a 0 -i 0 -k RTR
r -t 0.101180599 -s 12 -d -1 -p AOMDV -e 52 -c 2 -a 0 -i 0 -k RTR
r -t 0.101180683 -s 8 -d -1 -p AOMDV -e 52 -c 2 -a 0 -i 0 -k RTR
+ -t 0.101180683 -s 8 -d 2 -p AOMDV -e 52 -c 2 -a 0 -i 0 -k RTR
- -t 0.101180683 -s 8 -d 2 -p AOMDV -e 52 -c 2 -a 0 -i 0 -k RTR
h -t 0.101180683 -s 8 -d 2 -p AOMDV -e 52 -c 2 -a 0 -i 0 -k RTR
r -t 0.101180702 -s 14 -d -1 -p AOMDV -e 52 -c 2 -a 0 -i 0 -k RTR
r -t 0.101180716 -s 11 -d -1 -p AOMDV -e 52 -c 2 -a 0 -i 0 -k RTR
r -t 0.101180745 -s 3 -d -1 -p AOMDV -e 52 -c 2 -a 0 -i 0 -k RTR
r -t 0.101180767 -s 7 -d -1 -p AOMDV -e 52 -c 2 -a 0 -i 0 -k RTR
r -t 0.101180782 -s 0 -d -1 -p AOMDV -e 52 -c 2 -a 0 -i 0 -k RTR
r -t 0.101180793 -s 15 -d -1 -p AOMDV -e 52 -c 2 -a 0 -i 0 -k RTR
+ -t 0.103352340 -s 3 -d -1 -p AOMDV -e 52 -c 2 -a 0 -i 0 -k RTR
- -t 0.103352340 -s 3 -d -1 -p AOMDV -e 52 -c 2 -a 0 -i 0 -k RTR
h -t 0.103352340 -s 3 -d -1 -p AOMDV -e 52 -c 2 -a 0 -i 0 -k RTR
+ -t 0.103593023 -s 7 -d -1 -p AOMDV -e 52 -c 2 -a 0 -i 0 -k RTR
- -t 0.103593023 -s 7 -d -1 -p AOMDV -e 52 -c 2 -a 0 -i 0 -k RTR
h -t 0.103593023 -s 7 -d -1 -p AOMDV -e 52 -c 2 -a 0 -i 0 -k RTR

[simMPDSR.nam]
n -t * -s 0  -x 201 -y 306 -Z 0 -z 30  -v circle -c black
n -t * -s 1  -x 287 -y 361 -Z 0 -z 30  -v circle -c black
n -t * -s 2  -x 75 -y 48 -Z 0 -z 30  -v circle -c black
n -t * -s 3  -x 327 -y 239 -Z 0 -z 30  -v circle -c black
n -t * -s 4  -x 435 -y 318 -Z 0 -z 30  -v circle -c black
n -t * -s 5  -x 456 -y 178 -Z 0 -z 30  -v circle -c black
n -t * -s 6  -x 171 -y 304 -Z 0 -z 30  -v circle -c black
n -t * -s 7  -x 307 -y 260 -Z 0 -z 30  -v circle -c black
n -t * -s 8  -x 100 -y 56 -Z 0 -z 30  -v circle -c black
n -t * -s 9  -x 205 -y 222 -Z 0 -z 30  -v circle -c black
n -t * -s 10  -x 89 -y 361 -Z 0 -z 30  -v circle -c black
n -t * -s 11  -x 490 -y 185 -Z 0 -z 30  -v circle -c black
n -t * -s 12  -x 331 -y 168 -Z 0 -z 30  -v circle -c black
n -t * -s 13  -x 146 -y 74 -Z 0 -z 30  -v circle -c black
n -t * -s 14  -x 345 -y 294 -Z 0 -z 30  -v circle -c black
n -t * -s 15  -x 311 -y 75 -Z 0 -z 30  -v circle -c black
n -t * -s 16  -x 368 -y 212 -Z 0 -z 30  -v circle -c black
n -t * -s 17  -x 33 -y 81 -Z 0 -z 30  -v circle -c black
n -t * -s 18  -x 21 -y 289 -Z 0 -z 30  -v circle -c black
n -t * -s 19  -x 299 -y 349 -Z 0 -z 30  -v circle -c black
V -t * -v 1.0a5 -a 0
W -t * -x 1000 -y 1000
A -t * -n 1 -p 0 -o 0x7fffffff -c 30 -a 1
A -t * -h 1 -m 1073741823 -s 0
+ -t 0.100000000 -s 2 -d -1 -p tcp -e 40 -c 2 -a 0 -i 0 -k AGT
- -t 0.100000000 -s 2 -d -1 -p tcp -e 40 -c 2 -a 0 -i 0 -k AGT
h -t 0.100000000 -s 2 -d -1 -p tcp -e 40 -c 2 -a 0 -i 0 -k AGT
+ -t 0.101862143 -s 2 -d -1 -p MPDSR -e 32 -c 2 -a 0 -i 1 -k RTR
- -t 0.101862143 -s 2 -d -1 -p MPDSR -e 32 -c 2 -a 0 -i 1 -k RTR
h -t 0.101862143 -s 2 -d -1 -p MPDSR -e 32 -c 2 -a 0 -i 1 -k RTR
+ -t 0.110649187 -s 8 -d 2 -p MPDSR -e 44 -c 2 -a 0 -i 2 -k RTR
- -t 0.110649187 -s 8 -d 2 -p MPDSR -e 44 -c 2 -a 0 -i 2 -k RTR
h -t 0.110649187 -s 8 -d 2 -p MPDSR -e 44 -c 2 -a 0 -i 2 -k RTR
```

```
+ -t 0.115552800 -s 2 -d 8 -p tcp -e 76 -c 2 -a 0 -i 0 -k RTR
- -t 0.115552800 -s 2 -d 8 -p tcp -e 76 -c 2 -a 0 -i 0 -k RTR
h -t 0.115552800 -s 2 -d 8 -p tcp -e 76 -c 2 -a 0 -i 0 -k RTR
r -t 0.117865062 -s 8 -d 8 -p tcp -e 40 -c 2 -a 0 -i 0 -k AGT
+ -t 0.117865062 -s 8 -d -1 -p ack -e 40 -c 2 -a 0 -i 4 -k AGT
- -t 0.117865062 -s 8 -d -1 -p ack -e 40 -c 2 -a 0 -i 4 -k AGT
h -t 0.117865062 -s 8 -d -1 -p ack -e 40 -c 2 -a 0 -i 4 -k AGT
+ -t 0.117865062 -s 8 -d 2 -p ack -e 76 -c 2 -a 0 -i 4 -k RTR
- -t 0.117865062 -s 8 -d 2 -p ack -e 76 -c 2 -a 0 -i 4 -k RTR
h -t 0.117865062 -s 8 -d 2 -p ack -e 76 -c 2 -a 0 -i 4 -k RTR
r -t 0.120577325 -s 2 -d 2 -p ack -e 40 -c 2 -a 0 -i 4 -k AGT
+ -t 0.120577325 -s 2 -d -1 -p tcp -e 1040 -c 2 -a 0 -i 5 -k AGT
- -t 0.120577325 -s 2 -d -1 -p tcp -e 1040 -c 2 -a 0 -i 5 -k AGT
h -t 0.120577325 -s 2 -d -1 -p tcp -e 1040 -c 2 -a 0 -i 5 -k AGT
+ -t 0.120577325 -s 2 -d -1 -p tcp -e 1040 -c 2 -a 0 -i 6 -k AGT
- -t 0.120577325 -s 2 -d -1 -p tcp -e 1040 -c 2 -a 0 -i 6 -k AGT

[traceAOMDV.tr]
s 0.100000000 _2_ AGT  --- 0 tcp 40 [0 0 0 0] ------- [2:0 8:0 32
0] [0 0] 0 0
r 0.100000000 _2_ RTR  --- 0 tcp 40 [0 0 0 0] ------- [2:0 8:0 32
0] [0 0] 0 0
s 0.100000000 _2_ RTR  --- 0 AOMDV 52 [0 0 0 0] ------- [2:255 -
1:255 30 0] [0x2 0 1 [8 0] [2 4]] (REQUEST)
r 0.101180081 _17_ RTR  --- 0 AOMDV 52 [0 ffffffff 2 800] ------
- [2:255 -1:255 30 0] [0x2 0 1 [8 0] [2 4]] (REQUEST)
r 0.101180334 _19_ RTR  --- 0 AOMDV 52 [0 ffffffff 2 800] -------
- [2:255 -1:255 30 0] [0x2 0 1 [8 0] [2 4]] (REQUEST)
r 0.101180412 _1_ RTR  --- 0 AOMDV 52 [0 ffffffff 2 800] -------
[2:255 -1:255 30 0] [0x2 0 1 [8 0] [2 4]] (REQUEST)
r 0.101180599 _12_ RTR  --- 0 AOMDV 52 [0 ffffffff 2 800] ------
- [2:255 -1:255 30 0] [0x2 0 1 [8 0] [2 4]] (REQUEST)
r 0.101180683 _8_ RTR  --- 0 AOMDV 52 [0 ffffffff 2 800] --------
[2:255 -1:255 30 0] [0x2 0 1 [8 0] [2 4]] (REQUEST)
s 0.101180683 _8_ RTR  --- 0 AOMDV 52 [0 0 0 0] ------- [8:255
2:255 30 2] [0x4 0 [8 2] 10.000000] (REPLY) [1 2]
r 0.101180702 _14_ RTR  --- 0 AOMDV 52 [0 ffffffff 2 800] ------
- [2:255 -1:255 30 0] [0x2 0 1 [8 0] [2 4]] (REQUEST)
r 0.101180716 _11_ RTR  --- 0 AOMDV 52 [0 ffffffff 2 800] -------
- [2:255 -1:255 30 0] [0x2 0 1 [8 0] [2 4]] (REQUEST)
r 0.101180745 _3_ RTR  --- 0 AOMDV 52 [0 ffffffff 2 800] -------
[2:255 -1:255 30 0] [0x2 0 1 [8 0] [2 4]] (REQUEST)
r 0.101180767 _7_ RTR  --- 0 AOMDV 52 [0 ffffffff 2 800] -------
[2:255 -1:255 30 0] [0x2 0 1 [8 0] [2 4]] (REQUEST)
r 0.101180782 _0_ RTR  --- 0 AOMDV 52 [0 ffffffff 2 800] --------
[2:255 -1:255 30 0] [0x2 0 1 [8 0] [2 4]] (REQUEST)
r 0.101180793 _15_ RTR  --- 0 AOMDV 52 [0 ffffffff 2 800] ------
- [2:255 -1:255 30 0] [0x2 0 1 [8 0] [2 4]] (REQUEST)
s 0.103352340 _3_ RTR  --- 0 AOMDV 52 [0 ffffffff 2 800] -------
[3:255 -1:255 29 0] [0x2 1 1 [8 0] [2 4]] (REQUEST)
s 0.103593023 _7_ RTR  --- 0 AOMDV 52 [0 ffffffff 2 800] --------
[7:255 -1:255 29 0] [0x2 1 1 [8 0] [2 4]] (REQUEST)
s 0.105266937 _17_ RTR  --- 0 AOMDV 52 [0 ffffffff 2 800] ------
- [17:255 -1:255 29 0] [0x2 1 1 [8 0] [2 4]] (REQUEST)
r 0.105792465 _2_ RTR  --- 0 AOMDV 52 [13a 2 8 800] ------- [8:255
2:255 30 2] [0x4 0 [8 2] 10.000000] (REPLY) [1 2]
s 0.105792465 _2_ RTR  --- 0 tcp 60 [0 0 0 0] ------- [2:0 8:0 30
8] [0 0] 0 0
```

```
s 0.105841571 _1_ RTR --- 0 AOMDV 52 [0 ffffffff 2 800] -------
[1:255 -1:255 29 0] [0x2 1 1 [8 0] [2 4]] (REQUEST)
s 0.106366822 _19_ RTR --- 0 AOMDV 52 [0 ffffffff 2 800] ------
- [19:255 -1:255 29 0] [0x2 1 1 [8 0] [2 4]] (REQUEST)
r 0.107818514 _8_ AGT --- 0 tcp 60 [13a 8 2 800] ------- [2:0
8:0 30 8] [0 0] 1 0
s 0.107818514 _8_ AGT --- 1 ack 40 [0 0 0 0] ------- [8:0 2:0 32
0] [0 0] 0 0
r 0.107818514 _8_ RTR --- 1 ack 40 [0 0 0 0] ------- [8:0 2:0 32
0] [0 0] 0 0
s 0.107818514 _8_ RTR --- 1 ack 60 [0 0 0 0] ------- [8:0 2:0 30
2] [0 0] 0 0
s 0.107954283 _15_ RTR --- 0 AOMDV 52 [0 ffffffff 2 800] ------
- [15:255 -1:255 29 0] [0x2 1 1 [8 0] [2 4]] (REQUEST)
s 0.108103667 _12_ RTR --- 0 AOMDV 52 [0 ffffffff 2 800] -------
- [12:255 -1:255 29 0] [0x2 1 1 [8 0] [2 4]] (REQUEST)
r 0.109122945 _7_ RTR --- 0 AOMDV 52 [0 ffffffff f 800] -------
[15:255 -1:255 29 0] [0x2 1 1 [8 0] [2 4]] (REQUEST)
r 0.109123185 _11_ RTR --- 0 AOMDV 52 [0 ffffffff f 800] ------
- [15:255 -1:255 29 0] [0x2 1 1 [8 0] [2 4]] (REQUEST)
r 0.109123224 _3_ RTR --- 0 AOMDV 52 [0 ffffffff f 800] --------
[15:255 -1:255 29 0] [0x2 1 1 [8 0] [2 4]] (REQUEST)
r 0.109123276 _18_ RTR --- 0 AOMDV 52 [0 ffffffff f 800] ------
- [15:255 -1:255 29 0] [0x2 1 1 [8 0] [2 4]] (REQUEST)
r 0.109123301 _6_ RTR --- 0 AOMDV 52 [0 ffffffff f 800] -------
[15:255 -1:255 29 0] [0x2 1 1 [8 0] [2 4]] (REQUEST)
r 0.109123322 _8_ RTR --- 0 AOMDV 52 [0 ffffffff f 800] --------
[15:255 -1:255 29 0] [0x2 1 1 [8 0] [2 4]] (REQUEST)
s 0.109123322 _8_ RTR --- 0 AOMDV 52 [0 0 0 0] ------- [8:255
2:255 30 15] [0x4 0 [8 2] 10.000000] (REPLY) [1 15]
r 0.109123329 _13_ RTR --- 0 AOMDV 52 [0 ffffffff f 800] ------
- [15:255 -1:255 29 0] [0x2 1 1 [8 0] [2 4]] (REQUEST)
r 0.109123342 _1_ RTR --- 0 AOMDV 52 [0 ffffffff f 800] --------
[15:255 -1:255 29 0] [0x2 1 1 [8 0] [2 4]] (REQUEST)
r 0.109123399 _19_ RTR --- 0 AOMDV 52 [0 ffffffff f 800] ------
- [15:255 -1:255 29 0] [0x2 1 1 [8 0] [2 4]] (REQUEST)
r 0.109123480 _4_ RTR --- 0 AOMDV 52 [0 ffffffff f 800] -------
[15:255 -1:255 29 0] [0x2 1 1 [8 0] [2 4]] (REQUEST)
r 0.109123650 _9_ RTR --- 0 AOMDV 52 [0 ffffffff f 800] --------
[15:255 -1:255 29 0] [0x2 1 1 [8 0] [2 4]] (REQUEST)
r 0.109123653 _17_ RTR --- 0 AOMDV 52 [0 ffffffff f 800] ------
- [15:255 -1:255 29 0] [0x2 1 1 [8 0] [2 4]] (REQUEST)
r 0.109123699 _14_ RTR --- 0 AOMDV 52 [0 ffffffff f 800] ------
- [15:255 -1:255 29 0] [0x2 1 1 [8 0] [2 4]] (REQUEST)
r 0.109123711 _2_ RTR --- 0 AOMDV 52 [0 ffffffff f 800] --------
[15:255 -1:255 29 0] [0x2 1 1 [8 0] [2 4]] (REQUEST)
s 0.109578968 _11_ RTR --- 0 AOMDV 52 [0 ffffffff 2 800] ------
- [11:255 -1:255 29 0] [0x2 1 1 [8 0] [2 4]] (REQUEST)
s 0.109691848 _0_ RTR --- 0 AOMDV 52 [0 ffffffff 2 800] -------
[0:255 -1:255 29 0] [0x2 1 1 [8 0] [2 4]] (REQUEST)
s 0.109801375 _14_ RTR --- 0 AOMDV 52 [0 ffffffff 2 800] -------
- [14:255 -1:255 29 0] [0x2 1 1 [8 0] [2 4]] (REQUEST)
r 0.110092972 _15_ RTR --- 0 AOMDV 52 [0 ffffffff 7 800] ------
- [7:255 -1:255 29 0] [0x2 1 1 [8 0] [2 4]] (REQUEST)
r 0.110093199 _11_ RTR --- 0 AOMDV 52 [0 ffffffff 7 800] ------
- [7:255 -1:255 29 0] [0x2 1 1 [8 0] [2 4]] (REQUEST)
r 0.110093241 _3_ RTR --- 0 AOMDV 52 [0 ffffffff 7 800] --------
[7:255 -1:255 29 0] [0x2 1 1 [8 0] [2 4]] (REQUEST)
```

```
[traceMPDSR.tr]
Sconfig 0.00000 tap: on snoop: rts? on errs? on
Sconfig 0.00000 salvage: on !bd replies? on
Sconfig 0.00000 grat error: on grat reply: on
Sconfig 0.00000 $reply for props: on ring 0 search: on
Sconfig 0.00000 using MOBICACHE
s 0.100000000 _2_ AGT  --- 0 tcp 40 [0 0 0 0] ------- [2:0
8:0 32 0] [0 0] 0 0
r 0.100000000 _2_ RTR  --- 0 tcp 40 [0 0 0 0] ------- [2:0
8:0 32 0] [0 0] 0 0
s 0.101862143 _2_ RTR  --- 1 MPDSR 32 [0 0 0 0] -------
[2:255 8:255 32 0] 1 [1 1] [0 1 0 0->0] [0 0 0 0->0]
r 0.102922230 _8_ RTR  --- 1 MPDSR 32 [0 ffffffff 2 800] -
------ [2:255 8:255 32 0] 1 [1 1] [0 1 0 0->0] [0 0 0 0->0]
r 0.102922321 _17_ RTR  --- 1 MPDSR 32 [0 ffffffff 2 800] -
------ [2:255 8:255 32 0] 1 [1 1] [0 1 0 0->0] [0 0 0 0->0]
r 0.102922395 _13_ RTR  --- 1 MPDSR 32 [0 ffffffff 2 800] -
------ [2:255 8:255 32 0] 1 [1 1] [0 1 0 0->0] [0 0 0 0->0]
r 0.102922867 _9_ RTR  --- 1 MPDSR 32 [0 ffffffff 2 800] -
------ [2:255 8:255 32 0] 1 [1 1] [0 1 0 0->0] [0 0 0 0->0]
r 0.102922934 _15_ RTR  --- 1 MPDSR 32 [0 ffffffff 2 800] -
------ [2:255 8:255 32 0] 1 [1 1] [0 1 0 0->0] [0 0 0 0->0]
r 0.102922966 _18_ RTR  --- 1 MPDSR 32 [0 ffffffff 2 800] -
------ [2:255 8:255 32 0] 1 [1 1] [0 1 0 0->0] [0 0 0 0->0]
s 0.110649187 _8_ RTR  --- 2 MPDSR 44 [0 0 0 0] -------
[8:255 2:255 254 2] 2 [0 1] [1 1 2 2->8] [0 0 0 0->0]
r 0.115552800 _2_ RTR  --- 2 MPDSR 44 [13a 2 8 800] ------
- [8:255 2:255 254 2] 2 [0 1] [1 1 2 2->8] [0 0 0 0->0]
SFESTs 0.115552800 _2_ 0 [2 -> 8] 1(1) to 8 [2 |8 ]
s 0.115552800 _2_ RTR  --- 0 tcp 76 [0 0 0 0] ------- [2:0
8:0 32 8] [0 0] 0 0
r 0.117865062 _8_ RTR  --- 0 tcp 76 [13a 8 2 800] -------
[2:0 8:0 32 8] [0 0] 1 0
r 0.117865062 _8_ AGT  --- 0 tcp 40 [13a 8 2 800] -------
[2:0 8:0 32 8] [0 0] 1 0
s 0.117865062 _8_ AGT  --- 4 ack 40 [0 0 0 0] ------- [8:0
2:0 32 0] [0 0] 0 0
r 0.117865062 _8_ RTR  --- 4 ack 40 [0 0 0 0] ------- [8:0
2:0 32 0] [0 0] 0 0
SFESTs 0.117865062 _8_ 4 [8 -> 2] 1(1) to 2 [8 |2 ]
s 0.117865062 _8_ RTR  --- 4 ack 76 [0 0 0 0] ------- [8:0
2:0 32 2] [0 0] 0 0
r 0.120577325 _2_ RTR  --- 4 ack 76 [13a 2 8 800] -------
[8:0 2:0 32 2] [0 0] 1 0
r 0.120577325 _2_ AGT  --- 4 ack 40 [13a 2 8 800] -------
[8:0 2:0 32 2] [0 0] 1 0
s 0.120577325 _2_ AGT  --- 5 tcp 1040 [0 0 0 0] ------- [2:0
8:0 32 0] [1 0] 0 0
r 0.120577325 _2_ RTR  --- 5 tcp 1040 [0 0 0 0] ------- [2:0
8:0 32 0] [1 0] 0 0
SFESTs 0.120577325 _2_ 5 [2 -> 8] 1(1) to 8 [2 |8 ]
s 0.120577325 _2_ AGT  --- 6 tcp 1040 [0 0 0 0] ------- [2:0
8:0 32 0] [2 0] 0 0
```

```
r 0.120577325 _2_ RTR --- 6 tcp 1040 [0 0 0 0] ------- [2:0
8:0 32 0] [2 0] 0 0
SFESTs 0.120577325 _2_ 6 [2 -> 8] 1(1) to 8 [2 |8 ]
s 0.120577325 _2_ RTR --- 5 tcp 1076 [0 0 0 0] ------- [2:0
8:0 32 8] [1 0] 0 0
s 0.120577325 _2_ RTR --- 6 tcp 1076 [0 0 0 0] ------- [2:0
8:0 32 8] [2 0] 0 0
r 0.130749587 _8_ RTR --- 5 tcp 1076 [13a 8 2 800] ------
- [2:0 8:0 32 8] [1 0] 1 0
r 0.130749587 _8_ AGT --- 5 tcp 1040 [13a 8 2 800] ------
- [2:0 8:0 32 8] [1 0] 1 0
s 0.130749587 _8_ AGT --- 7 ack 40 [0 0 0 0] ------- [8:0
2:0 32 0] [1 0] 0 0
r 0.130749587 _8_ RTR --- 7 ack 40 [0 0 0 0] ------- [8:0
2:0 32 0] [1 0] 0 0
SFESTs 0.130749587 _8_ 7 [8 -> 2] 1(1) to 2 [8 |2 ]
s 0.130749587 _8_ RTR --- 7 ack 76 [0 0 0 0] ------- [8:0
2:0 32 2] [1 0] 0 0
r 0.141161937 _8_ RTR --- 6 tcp 1076 [13a 8 2 800] ------
- [2:0 8:0 32 8] [2 0] 1 0
r 0.141161937 _8_ AGT --- 6 tcp 1040 [13a 8 2 800] ------
- [2:0 8:0 32 8] [2 0] 1 0
s 0.141161937 _8_ AGT --- 8 ack 40 [0 0 0 0] ------- [8:0
2:0 32 0] [2 0] 0 0
r 0.141161937 _8_ RTR --- 8 ack 40 [0 0 0 0] ------- [8:0
2:0 32 0] [2 0] 0 0
SFESTs 0.141161937 _8_ 8 [8 -> 2] 1(1) to 2 [8 |2 ]
s 0.141161937 _8_ RTR --- 8 ack 76 [0 0 0 0] ------- [8:0
2:0 32 2] [2 0] 0 0
r 0.143494200 _2_ RTR --- 7 ack 76 [13a 2 8 800] -------
[8:0 2:0 32 2] [1 0] 1 0
r 0.143494200 _2_ AGT --- 7 ack 40 [13a 2 8 800] -------
[8:0 2:0 32 2] [1 0] 1 0
s 0.143494200 _2_ AGT --- 9 tcp 1040 [0 0 0 0] ------- [2:0
8:0 32 0] [3 0] 0 0

[win.tr]
10.1 43.8724
10.109999999999999 43.918
10.119999999999999 43.9407
10.129999999999999 43.9635
10.139999999999999 43.9635
10.149999999999999 44.009
10.159999999999998 44.0317
10.169999999999998 44.0317
10.179999999999998 44.0544
10.189999999999998 44.0544
10.199999999999998 44.0998
10.209999999999997 44.1451
10.219999999999997 44.1451
10.229999999999997 44.1678
10.239999999999997 44.1904
10.249999999999996 44.213
```

10.259999999999996 44.2357
10.269999999999996 44.2809
10.279999999999996 44.2809
10.289999999999996 44.3035
10.299999999999995 44.3035
10.309999999999995 44.3035
10.319999999999995 44.326
10.329999999999995 44.3711
10.339999999999995 44.3711
10.349999999999994 44.3711
10.359999999999994 44.3937
10.369999999999994 44.3937
10.379999999999994 44.4162
10.389999999999993 44.4387
10.399999999999993 44.4387
10.409999999999993 44.4612
10.419999999999993 44.4837
10.429999999999993 44.4837
10.439999999999992 44.4837
10.449999999999992 44.5062
10.459999999999992 44.5511
10.469999999999992 44.5511
10.479999999999992 44.5511
10.489999999999991 44.5511
10.499999999999991 44.5511
10.509999999999991 44.5736
10.519999999999991 44.596
10.52999999999999 44.6184
10.53999999999999 44.6184
10.54999999999999 44.6184
10.55999999999999 44.6184
10.56999999999999 44.6408
10.579999999999989 44.6408

Kode Program AWK Packet Delivery Ratio

```
1   BEGIN {
2       send = 0;
3       recv = 0;
4   }{
5   If (( $1 == "s") && ( $4=="AGT" )) send++;
6   If (( $1 == "r") && ( $4=="AGT" )) recv++;
7   } END {
8   Printf "total paket dikirim:%.3f total paket diterima:%.3f,
      PDR:%.3f\n", send, recv, (recv/send);
```

Penjelasan:

1. Baris 1-4 merupakan inisialisasi awal program dengan variabel send dan recv yang digunakan untuk menampung nilai jumlah seluruh paket data yang dikirim dan diterima.
2. Baris 5 berfungsi dalam mengecek seluruh baris pada kolom *network trace file*, yakni pada kolom ke-1 dan ke-4 apakah status bernilai s (*send*) dan AGT. Jika benar maka variabel send akan bertambah.
3. Baris 6 berfungsi dalam mengecek seluruh baris pada kolom *network trace file*, yakni pada kolom ke-1 dan ke-4 apakah status bernilai r (*receive*) dan AGT jika benar maka variabel recv akan bertambah.
4. Baris 7-8 merupakan akhir akumulasi yang mana ditandai dengan instruksi END. Selanjutnya menampilkan jumlah variabel send dan recv beserta hasil akhir *packet delivery ratio* dengan membagi variabel recv dengan send.

Kode Program AWK End-to-end Delay

```
1   BEGIN {
2   seqno=-1;
3       cnt=0;
4       }{
5       if($4 == "AGT" && $1 == "s" && seqno < $6){
6           seqno = $6; }
7       if($4 == "AGT" && $1 == "s"){
8           start_time[$6] = $2; }
9       else if(($4 == "AGT") && ($1 == "r")){
10          end_time[6] = $2; }
11      else if($1 == "D" && $7 = "CBR"
12          end_time[$6] = -1;}
13      }end{
14      for(i=0;i<=seqno;i++){
15        if(end_time[i]>0){
16       delay[i]=end_time[i]-start_time[i];
17          cnt++;
18      }else{
19        delay[i]=-1; }}
20      for(i=0;i<seqno;i++){
21        if(delay[i]>0) {
22          ssdelay=ssdelay+delay[i];}}
23      ssdelay=ssdelay/(cnt);
24      printf( "average delay = %.3f,ssdelay); }
```

Penjelasan:

1. Baris 1-4 merupakan inisialisasi awal program dengan variabel seqno dan cnt yang digunakan untuk menampung nilai *sequence number* dari paket sejak awal paket dikirim hingga paket yang terakhir dikirim. Variabel cnt berfungsi dalam membagi nilai rata-rata *delay* pada akhir program.

2. Baris 5-6 brfungsi dalam mengisi nilai variabel seqno dengan kondisi jika kolom 1 bernilai s dan kolom 4 bernilai AGT dan nilai variabel seqno tidak lebih besar dari nilai kolom 6. Hal tersebut bertujuan untuk mendapatkan *sequence number* paket yang terakhir dikirim.
3. Baris 7-8 berfungsi untuk mengecek seluruh baris pada kolom *network trace file*, yakni kolom ke-1 dan ke-4 apakah status bernilai s (send) dan AGT. Jika benar, maka variabel array start_time dengan index sesuai nomer urut paket kolom 6 akan diisikan dengan nilai waktu berjalannya paket tersebut ketika mulai dikirim oleh *source node*.
4. Baris 9-10 berfungsi mengecek seluruh baris pada kolom *network trace file*, yakni kolom ke-1 dan ke-4 apakah status bernilai r (received) dan AGT. Jika benar, maka variabel array end_time dengan index sesuai nomer urut paket kolom 6 akan diisikan dengan nilai waktu berjalannya paket tersebut ketika diterima oleh *destination node*.
5. Baris 11-12 berfungsi mengecek seluruh baris pada kolom *network trace file*, yakni kolom ke-1 dan ke-4 apakah status bernilai D (dropped) dan CBR. Jika benar, maka variabel array end_time dengan index sesuai nomer urut paket akan diisikan dengan nilai -1 yang berarti paket gagal terkirim.
6. Baris 13-17 merupakan akhir pengumpulan data yang ditandai dengan instruksi END. Proses perhitungan estimasi waktu paket yang dilakukan dengan mengurangi waktu end_time dengan start_time serta mengabaikan paket drop yang bernilai -1. Kemudian nilai tersebut disimpan pada variabel *array delay* dengan index sesuai *sequence number* dari paket tersebut. Selanjutnya akan dilakukan perhitungan jumlah paket yang berhasil terkirim pada variabel cnt.
7. Baris 18-17 berfungsi dalam mengisi paket yang berstatus D (*dropped*) dengan nilai -1.
8. Baris 20-22 berfungsi menjumlahkan seluruh estimasi waktu paket terkirim yang disimpan pada variabel array delay dan mengabaikan paket yang berstatus *dropped*.
9. Baris 23-24 bberfungsi menampilkan hasil akhir rata-rata waktu (*second*) end-to-end delay dengan memagi hasil jumlah total *paket* terkirim yang disimpan pada variabel ssdelay dngan total seluruh paket yang terkirim yang tersimpan pada variabel cnt.

Kode Program AWK Throughput

```
1   BEGIN {
2       recvdSize = 0
3       start_time = 120
4       end_time = 0
5     }{
6     event = $1
7     time = $2
8     pkt_size = $8
9     level = $4
10  if (level == "AGT" && event == "s")
11      if (time < start_time) {
12          start_time = time }}
13      if (level == "AGT" && event == "r")
14          if (time > end_time) {
15          end_time = time }
16  recvdSize += 1000 }
17      } END {
18    Printf ("average Throughput[kbps] = %.3f\t\t StartTime=%.3f\t
    StopTime=%.3f \t totalpaket=%.3f\n",(recvdSize/ (end_time
    Start_time))*(8/1000), start_time,stop_time,recvdSize)
```

Penjelasan:

1. Baris 1-5 merupakan inisialisasi awal program dengan variabel start_time, end_time dan recvSize yang digunakan untuk menampung nilai waktu awal dimulai paket mulai_dikirim, waktu paket terakhir dikirim, dan jumlah seluruh paket yang terkirim
2. Baris 6-9 merupakan inisialisasi variabel merupakan nilai kolom 1, variabel event diisi dengan nilai kolom 1, variabel time diisi dengan nilai kolom 2, variabel pkt_size diisi dengan nilai kolom 8, dan variabel level diisi dengan nilai kolom 4.
3. Baris 10 -12 berfungsi menyimpan catatan waktu awal dimulai pengiriman paket data *source node* yang disimpan pada variabel start_time dengan kondisi apabila variabel event bernilai s dan level bernilai AGT.
4. Baris 13-15 berfungsi menyimpan catatan waktu akhir penerimaan paket pada *destination node* yang disimpan pada variabel end_time dengan kondisi apabila variabel event bernilai r dan level bernilai AGT.

5. Baris 16 merupakan kalkulasi banyaknya paket yang telah diterima oleh *destination node* dengan penambahan nilai 1000 pada variabel recvdSize dikarenakan ukuran paket yang dikirim 1kB (1000byte).
6. Baris 17-18 merupakan akhir pengumpulan data yang ditandai dengan instruksi END dan kalkulasi hasil akhir nilai *throughput* dilakukan dengan membagi total besaran paket yang terkirim pada *destination node* (rcvdSize) dibagi total waktu pengamatan (end_time – start_time) kemudian dikalikan 8/1000 agar satuan *throughput* dapat menjadi kbps (Kilobit per second).

Kode Program AWK Normalized Routing Load

```
1  BEGIN {
2  recv = 0;
3  trt_pkt = 0
4  }{
5  If (( $1 == "r") && ( $4=="AGT" )) recv++;
6  If (( $1 == "s" || $1 == "f") && $4 == "RTR" && ($7 =="AOMDV" || $7
     =="message" || $7 =="MPDSR")) rtr_pkt++;
7  }END{
8  Printf "Total paket routing = %.3f  total  paket  diterima  =
     %.3f\n", rtr_pkt,recv);
9  Printf(Normalized Routing Laoad = %.3f\n", rtr_pkt/recv);
```

Penjelasan:

1. Baris 1-4 merupakan inisialisasi awal program dengan variabel rtr_pkt dan rcvd yang digunakan untuk menampung nilai jumlah seluruh paket *routing* baik yang dikirim maupun diteruskan dan seluruh paket data yang diterima *desination node*.
2. Baris berfungsi dalam mengecek seluruh baris pada kolom *network trace file*, yakni kolom ke-1 dan ke-4 apakah status bernilai r (*received*) dan AGT jika benar, maka variabel rcvd akan bertambah.
3. Baris 6 berfungsi dalam mengecek seluruh baris pada kolom *network trace file*, yakni kolom ke-1 apakah status bernilai s (send) atau r (received) dan kolom ke-4 RTR (*routing request*). Selanjutnya mengecek kolom ke-7 apakah statusnya merupakan *agent routing* AOMDV dan MP-DSR jika benar, maka variabel rtr_pkt akan bertambah.
4. Baris 7-9 merupakan akhir akumulasi yang ditandai dengan instruksi END dan menampilkan jumlah variabel rtr_pkt dan recv. Hasil akhir *normalized routing load* diperoleh dengan membagi variabel variabel rtr_pkt dengan recv.

Kode Program AWK Convergence time

```
1  BEGIN {dps=0; dpa=0;routing_packets=0.0;oldVal=0;}
2     #data packet yang dikirim -> data packet sent (dps):
3        ($1==”d” && $4==”AGT”) {sender[$6} = $2; send++;}
4     #data packet yang diterima -> data packet arrived(dpa):
5     (($1==”r” && $4==”AGT”) {recv[$6} = $2; received++;}
6     #printf(“nilai received id = %d\n, $6);
7     #if( oldval > $6 )
8     #oldVal = $6; )
9       End{
10    #mencocokan Sender ID dengan Receier ID
11    #PROCINFO[“sorter_in”] = “@ind_num_asc”;
12    PROCINFO[“sorted_in”]=”@ind_num_asc”
13    no = 1;
14    prev = 0;
15    prev = 0;
16    not_match =0;
17    for(id in sender){
18       #printf (“Nomor %d     id %d\n”,no,id;
19       if(id in recv){
20       if (not_match == 1){
21          Konvergensi += (recv[id] - prev);
22          #print konvergensi;
23       not_match = 0;
24       prev = recv[id];
25       }else{
26       Prev = recv[id];
27         }
28       }else{
29       #   print “packet id yg hilang: ”id;
```

```
30          not match = 1;
31      }
32      #   no = no + 1;
33    }
34    Print "Total Waktu Konvergensi : "konvergensi;
35  }
```