ANALISIS KINERJA *ROUTING* PROTOKOL *MULTI-COPY* DENGAN MENGGUNAKAN *MANAGEMENT BUFFER FIRST IN-FIRST OUT* (FIFO) DAN *MOST FORWARDED FIRST* (MOFO) PADA *DELAY TOLERANT NETWORK* (DTN)

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Riski Manta Simanjorang
NIM: 155150201111172

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2020

PENGESAHAN

ANALISIS KINERJA *ROUTING* PROTOKOL *MULTI-COPY* DENGAN MENGGUNAKAN *MANAGEMENT BUFFER FIRST IN – FIRST OUT* (FIFO) DAN *MOST FORWARDED FIRST* (MOFO) PADA *DELAY TOLERANT NETWORK* (DTN)

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Riski Manta Simanjorang
NIM: 155150201111172

Skripsi ini telah diuji dan dinyatakan lulus pada
7 Januari 2020
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I

Ir. Heru Nurwarsito, M.Kom.
NIP: 19650402 199002 1 001

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T., M.T., Ph.D.
NIP: 19710518 200312 1 001

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, 8 Desember 2019

Riski Manta Simanjorang

NIM: 155150201111172

ABSTRAK

Riski Manta Simanjorang, Analisis Kinerja *Routing* Protokol *Multi-Copy* Dengan Menggunakan *Management Buffer First In – First Out* (FIFO) Dan *Most Forwarded First* (MOFO) Pada *Delay Tolerant Network* (DTN)

Pembimbing: Ir. Heru Nurwarsito, M.Kom

Tingginya *delay* pada jaringan merupakan salah satu masalah kegagalan pada saat pengiriman pesan. Masalah tersebut dapat diatasi dengan menggunakan *Delay Tolerant Network*. *Routing multi-copy* merupakan *routing* yang dapat meminimalkan *delay* pada jaringan serta memaksimalkan pengiriman data jika dibandingkan dengan *routing Single-Copy*. Namun, belum adanya manajemen *buffer* menyebabkan tidak efektifnya penggunaan *buffer* pada setiap *node*. *First In-First Out* (FIFO) dan *Most Forwarded First* (MOFO) merupakan manajemen *buffer* yang dapat mengelola *buffer* yang ada pada setiap node. Manajemen *buffer* FIFO dapat mengelola *buffer* berdasarkan waktu masuknya paket. Sedangkan manajemen *buffer* MOFO mengelola *buffer* berdasarkan jumlah penerusan paket. Berdasarkan pengujian kinerja *routing*, manajemen *buffer* dapat mengoptimalkan pengiriman paket pada DTN. Hasil simulasi menunjukkan bahwa nilai *delivery probability* tertinggi didapatkan sebesar 0.4799 yang terdapat pada manajemen *buffer* FIFO dengan ukuran *buffer* 12MB dengan *routing* Spray-and-Wait. Nilai *overhead ratio* terendah didapatkan sebesar 14.4873 yang terdapat pada manajemen *buffer* MOFO dengan ukuran *buffer* 12MB dengan *routing* Spray-and-Wait. Nilai *latency average* terendah didapatkan sebesar 2092.7180s yang terdapat pada *routing* Spray-and-Wait menggunakan manajemen *buffer* FIFO dengan ukuran *buffer* 5MB.

Kata kunci: *Delay Tolerant Network*, *Routing Multi-Copy*, *Management Buffer*, *First In-First Out, Most Forwarded First*

ABSTRACT

Riski Manta Simanjorang, ***Analysis of Multi-Copy Routing Protocol Performance Using First-In-First Out Buffer Management*** **(FIFO)** ***and Most Forwarded First*** **(MOFO)** ***on Delay Tolerant Network*** **(DTN)**

Supervisors**: Ir. Heru Nurwarsito, M.Kom**

The high delay on the network is one of the failure problems when sending messages. These problems can be overcome by using the Delay Tolerant Network. Routing multi-copy is a routing that minimizing network delay and maximizing data transmission when compared to routing Single-Copy. However, the absence of buffer management causes ineffective use of buffers each node. First In-First Out (FIFO) and Most Forwarded First (MOFO) is buffer management that can manage buffers that existeach node. FIFO buffer management can manage buffers based on the time the packet was entered. Whereas MOFO buffer management manages buffers based on the number of packet forwarding. Based on routing performance testing, buffer management can optimize packet delivery on DTN. The simulation results show that the highest value of delivery probability is 0.4799 which is found in the management of FIFO buffer with buffer size of 12MB with Spray-and-Wait routing. The lowest overhead ratio value obtained is 14.4873 found in the MOFO buffer management with a buffer size of 12MB with Spray-and-Wait routing. The lowest latency average value is 2092.7180s found in Spray-and-Wait routing using FIFO buffer management with 5MB buffer size.

Keywords: *Delay Tolerant Network*, *Routing Multi-Copy*, *Management Buffer*, *First In-First Out, Most Forwarded First*

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR LAMPIRAN

BAB 1 PENDAHULUAN

1.1 Latar Belakang

Seiring bejalannya waktu, kebutuhan manusia juga semakin bertambah. Berkomunikasi dan bertukar informasi, menjadi aktivitas yang sangat penting dilakukan. Intenet merupakan penghubung yang sangat popular dipakai untuk melakukan pertukaran informasi dan berkomunikasi. Pesan dapat dengan cepat dikirim dan diterima dengan menggunakan internet yang dibangun dengan infrastruktur baik. Namun, untuk beberapa lokasi, membangun internet yang stabil sulit untuk dilakukan. Letak geografis menjadi salah satu penyebab sulitnya membangun infrastruktur yang baik. Beberapa contohnya seperti daerah pegunungan dan daerah 3T (Terdepan, Terluar, Tertinggal). Kondisi tersebutlah yang menghambat untuk melakukan pertukaran informasi dengan baik. Namun, dengan memanfaatkan *Delay Tolerant Network*, kita dapat mengatasi kondisi dengan infrastruktur yang minim.

Delay Tolerant Network atau sering disingkat menjadi DTN merupakan jaringan yang memakai arsitektur khusus yang dikembangkan dengan upaya menyambungkan antar *device* yang mempunyai waktu *delay* tinggi dengan menggunakan koneksi berubah-ubah (Fall, 2003). Dengan menggunakan DTN, komunikasi jarak jauh dapat dilakukan tanpa memperdulikan *delay* yang tinggi. Metode yang digunakan DTN untuk mengirimkan paket adalah dengan menggunakan paradigma *store-carry-and*-forward. Agar dapat meneruskan paket data pada *node hop* berikutnya, *node* DTN menyimpan paket data pada penyimpanan persisten.

Terdapat dua jenis skema *routing* pada DTN, yaitu *single-copy* dan *multi-copy*. *Routing single-copy* bekerja dengan meneruskan paket yang ada pada *node* ke *node* yang ditemuinya. Sedangkan *routing multi-copy* bekerja dengan menyalin paket ke setiap *node* yang berada dalam area cakupannya (Spyropoulos, Psounis dan Raghavendra, 2005). Oleh karena itu, *routing multi-copy* memiliki probabilitas pengiriman yang lebih besar karena meneruskan paket ke setiap *node* yang di jumpai. Dengan tingginya probabilitas pengiriman yang dimiliki, *routing* membutuhkan sumber daya seperti *bandwidth, buffer,* dan energy yang lebih besar juga.

Penelitian sebelumnya dilakukan oleh Hutajulu, Yahya dan Pramukantoro, (2018) berjudul "Perbandingan Kinerja *Routing Multi Copy* dan *Routing First Contact* Dengan *Stationary Relay Node* pada *Delay Tolerant Network* (DTN)". Penelitian tersebut menganalisis mengenai perbandingan antara *routing multi-copy* dan *First Contact*. Selain itu, penulis juga menambahkan *Stationary Relay Node* pada jaringan DTN. Namun, penelitian tersebut menggunakan *buffer* yang besar yang sebenarnya pada banyak aplikasi dan teknologi yang menggunakan jaringan DTN hanya memberikan ukuran *buffer* yang terbatas. Hal ini menyebabkan tidak efektifnya penggunaan *buffer* pada setiap *node* sehingga dapat menurunkan kinerja *routing* pada jaringan tersebut. Beberapa kinerja yang

turun adalah rendahnya rasio pengiriman data dan tingginya *delay* pengiriman data. Maka diperlukan manajemen *buffer* yang dapat mengatur *buffer* sehingga dapat berjalan optimal dan meningkatkan pengiriman paket pada DTN. Berdasarkan hasil penelitian yang terdahulu, pada penelitian kali penulis melakukan perbandingan dan menganalisis kinerja pada DTN dengan menggunakan *routing multi-copy* dengan adanya manajemen *buffer*.

Manajemen *buffer* merupakan teknik untuk mengelola sumber daya sesuai dengan situasi dan mekanisme yang digunakan. Manajemen *buffer* yang efisien merupakan teknik yang dapat memutuskan langkah paket mana yang harus dibuang terlebih dahulu, ketika *buffer* penuh dan juga paket mana yang akan dikirim ketika *bandwidth* terbatas (Rani et al., 2014). Dengan adanya manajemen *buffer*, rasio pengiriman data dapat ditingkatkan dan pengelolaan *buffer* akan semakin baik. Sesuai dengan namanya, manajemen *buffer* adalah algoritma yang berfungsi untuk mengelola keluar masuknya paket, dimana jika *buffer* pada node DTN sudah penuh, maka dengan adanya manajemen *buffer* paket tersebut dapat di *drop*. Ada beberapa manajemen *buffer* yang dapat digunakan pada DTN, beberapa diantaranya adalah *First In – First Out* (FIFO) dan *Most Forwarded First* (MOFO). Manajemen *buffer FIFO* adalah teknik mengelola *buffer* berdasarkan waktu masuknya paket masuk dengan karateristik paket pertama yang masuk akan terlebih dahulu di proses (Rani et al., 2014). Sedangkan MOFO mempunyai karateristik menjatuhkan paket yang sudah diteruskan dalam jumlah waktu maksimum (Rani et al., 2014).

Berdasarkan permasalah tersebut, penelitian yang berjudul "Analisis Kinerja *Routing Protokol Multi-Copy* dengan Menggunakan *Management Buffer First In-First Out* (FIFO) dan *Most Forwarded First* (MOFO) pada *Delay Tolerant Network* (DTN)" dapat menganalis kinerja *routing* yang optimal menggunakan manajemen *buffer* FIFO dan MOFO. Simulator yang digunakan untuk melakukan pengujian pada penelitian ini adalah The One Simulator. Penelitian ini diharapkan mampu memberikan perbandingan kinerja dari *routing* dan manajemen *buffer* yang digunakan pada DTN dan dapat memberikan hasil analisis pengaruh adanya manajemen *buffer* terhadap kinerja DTN dengan adanya penambahan manajemen *buffer* FIFO dan MOFO. Selain itu, penelitian ini diharapkan dapat digunakan pada kondisi yang nyata dengan sumber daya jaringan yang terbatas.

1.2 Rumusan Masalah

Berikut ini merupakan poin-poin permasalahan yang dirumuskan berdasarkan latar belakang pada penelitian ini:

1. Bagaimana menerapkan manajemen *buffer First In – First Out* (FIFO) dan *Most Forwarded First* (MOFO) dengan *routing Multi-Copy* pada jaringan *Delay Tolerant Network*?
2. Bagaimana kinerja *routing Multi-Copy* dengan penambahan manajemen *buffer First In – First Out* (FIFO) *dan Most Forwarded First* (MOFO) pada jaringan *Delay Tolerant Network*?

1.3 Tujuan

Berikut ini merupakan tujuan berdasarkan rumusan masalah dari penelitian ini:

1. Mengimplementasikan manajemen *buffer First In – First Out* (FIFO) dan *Most Forwarded First* (MOFO) menggunakan *routing Multi-Copy* pada jaringan *Delay Tolerant Network*.
2. Mengetahui kinerja *routing Multi-Copy* dengan manajemen *buffer First In – First Out* (FIFO) dan *Most Forwarded First* (MOFO) pada jaringan *Delay Tolerant Network*.

1.4 Manfaat

Penelitian ini bermanfaat sebagai bahan pembelajaran dalam penerapan *routing multi-copy* pada jaringan DTN. Hasil pengujian pada penelitian ini juga dapat memberitahu routing mana yang cocok untuk masing-masing manajemen *buffer* yang diterapkan. Selain itu, penelitian ini juga dapat menjadi bahan referensi untuk penelitian selanjutnya yang berhubungan dengan penerapan manajemen *buffer* pada jaringan DTN.

1.5 Batasan Masalah

Berikut ini merupakan batasan-batasan masalah pada penelitian ini agar penelitian ini tidak terlalu melebar:

1. Simulasi jaringan dilakukan dengan menggunakan The ONE Simulator 1.6.0.
2. Skema *routing* yang digunakan adalah Epidemic, Spray-and-Wait dan MaxProp.
3. Menggunakan peta jalur pendakian Gunung Sibayak.
4. Menggunakan parameter uji *delivery probability, overhead ratio* dan *latency average.*
5. Menambahkan manajemen *buffer First In-First Out* (FIFO) dan *Most Forwarded First* (MOFO).

1.6 Sistematika Pembahasan

Berikut ini merupakan struktur sistematika pembahasan yang penulis gunakan pada penelitian ini:

BAB 1 PENDAHULUAN

Bab pertama dari penelitan yang berjudul "Perbandingan Kinerja *Routing Protokol Multi-Copy* dengan Menggunakan *Management Buffer First In-First Out* (FIFO) dan *Most Forwarded First* (MOFO) pada *Delay Tolerant Network* (Dtn)" memuat Latar belakang, Identifikasi Malasah, Rumusan Masalah, Tujuan, Manfaat, Batasan Masalah dan yang terakhir adalah Sistematika Pembahasan.

BAB 2 LANDASAN KEPUSTAKAAN

Bab kedua memuat tentang penjelasan dari teori-teori yang berhubungan dengan masalah yang dibahas pada penelitian ini yang dapat menunjang penelitian ini. Teori-teori yang ada didapatkan dari beragam referensi seperti buku, jurnal sumber referensi lainnya yang membahas tentang topik yang berkaitan dengan penelitian ini.

BAB 3 METODOLOGI PENELITIAN

Bab ketiga membahas mengenai alur kerja yang diterapkan pada penelitian ini. Bab ini mencakup studi literatur, analisis kebutuhan, perancangan, Implementasi, pengujian sistem, analisis hasil pengujian dan yang terakhir penutup dari Analisis Kinerja *Routing* Protokol *Multi-Copy* dengan Menggunakan *Management Buffer First In – First Out* (FIFO) dan *Most Forwarded First* (MOFO) pada *Delay Tolerant Network* (DTN).

BAB 4 PERANCANGAN DAN IMPLEMENTASI

Bab keempat memuat proses perancangan dan proses Implementasi. Dari perancangan dan Implementasi yang sudah dilaksanakan didapatkan hasil pengujian dari setiap skenario yang sudah dirancang berupa data yang nantinya dianalisis oleh penulis.

BAB 5 PENGUJIAN DAN ANALISIS

Proses pengujian terlebih dahulu dilakukan pada bab kelima. Pengujian dilakukan dengan menggunakan simulator yang sudah ditentukan oleh penulis. Data yang didapatkan, digambarkan dengan grafik yang nantinya dapat dianalisis.

BAB 6 PENUTUP

Bab keenam memuat tentang pengambilan kesimpulan dari hasil data dan analisis yang sudah dilakukan pada saat pengujian. Hasil dari analisis yang ada akan menjawab poin-poin rumusan masalah yang sudah disebutkan oleh penulis.

BAB 2 LANDASAN KEPUSTAKAAN

2.1 Kajian Pustaka

Tabel 2.1 Kajian pustaka

No.	Judul	Persamaan	Perbedaan	
			Penelitian Terdahulu	Rencana Penelitian
1	Perbandingan Kinerja *Routing Multi Copy* dan *Routing First Contact* dengan *Stationary Relay Node* pada *Delay Tolerant Network* (DTN). (Hutajulu, Yahya dan Pramukantoro, 2018)	Penulis menggunakan *routing Multi-Copy* pada DTN.	Penulis membandingkan antara *routing multi-copy* dan *routing first contact* pada jaringan DTN.	Penulis menerapkan *routing multi-copy* dan menambahkan manajemen *buffer* FIFO dan MOFO pada jaringan DTN.
2	Analisis *Routing Multi Copy* dengan *Stationary Relay Node* dan *Management Buffer First In – First Out* (FIFO) pada *Delay Tolerant Network* (DTN). (Reza et al., 2019)	Penulis menggunakan *routing multi-copy* dan menambahkan manajemen *buffer* FIFO pada jaringan DTN.	Penulis membandingkan dan menganalisis *routing* multi-copy dengan adanya *stationary relay node* dan manajemen *buffer* FIFO pada jaringan DTN.	Penulis membandingkan dan menganlisis *routing multi-copy* dengan adanya manajemen *buffer* FIFO dan MOFO pada jaringan DTN.
3	*Performance Evaluation of MOFO Buffer Management Technique With Different Routing Protocols In DTN Under Variable*	Peneliti menggunakan manajemen *buffer* MOFO dengan menggunakan *routing multi-copy* pada DTN.	Penelitian memuat tentang evaluasi kinerja teknik manajemen *buffer* MOFO dengan menggunakan *routing multi-*	Penulis membandingkan dan menganalisis *routing multi-copy* Epidemic, Spray-and-Wait dan MaxProp dengan adanya

	Message Buffer Size. (Rani, 2014)		*copy* Epidemic, MaxProp dan ProPHet pada jaringan DTN.	manajemen buffer FIFO da MOFO pada jaringan DTN.
4	*Performance Evaluation of MaxProp Routing Protocol with DL, FIFO, DLA and MOFO Buffer Management Techniques in DTN under Variable Message Buffer Size*. (Rani, 2014)	Penulis menggunakan manajemen *buffer* FIFO dan MOFO dengan *routing* MaxProp pada jaringan DTN.	Penulis membandingkan dan menganalisis manajemen *buffer* dengan menggunakan satu *routing* pada jaringan DTN.	Penulis membandingkan dan menganalisis manajemen *buffer* dengan menggunakan *routing multi-copy* Epidemic, Spray-and-Wait dan MaxProp pada jaringan DTN.
5	*Performance Analysis of Message Drop Control Source Relay (MDC-SR) in Maxprop DTN Routing*. (Putra, Vidya dan Tody, 2017)	Penulis menerapkan manajemen *buffer* dengan menggunakan *routing* MaxProp pada jaringan DTN.	Penulis menerapkan manajemen *buffer Drop Management Source Relay* (MDC-SR) dengan menggunakan *routing* MaxProp pada jaringan DTN.	Penulis membandingkan dan menganalisis *routing multi-copy* dengan adanya penambahan manajemen *buffer* FIFO dan MOFO pada jaringan DTN.

Tabel 2.1 menampilkan perbandingan antara permasalahan yang dibahas peneliti sebelumnya dan usulan penelitian oleh penulis. Kajian pustaka ini nantinya akan menjadi landasan yang digunakan untuk melakukan perancangan dalam pelaksanaan penelitian tentang menganalisis kinerja *routing multi-copy* dengan menggunakan manajemen FIFO dan MOFO pada jaringan DTN.

Sebagai pendukung penelitian dalam penulisan penelitian ini, penulis menggunakan lima penelitian sebelumnya. Penelitian pertama dilakukan oleh Hutajulu, Yahya dan Pramukantoro (2018) dengan judul "Perbandingan Kinerja *Routing Multi Copy* dan *Routing First Contact* dengan *Stationary Relay Node* pada *Delay Tolerant Network* (DTN)". Penelitian tersebut menganalis mengenai perbandingan kinerja antara *routing Multi-Copy* dan *routing First Contact* pada jaringan DTN. Dari hasil penelitiannya, penulis menyebutkan bahwa nilai dari *delivery probability* yang dihasilkan dengan menggunakan *ruting* Spray-and-Wait menghasilkan nilai paling tinggi jika dibandikan dengan *routing* yang lain.

Sedangkan, *routing First Contact* memberikan hasil *latency average* yang lebih besar jika dibandikan dengan *routing* yang lain (Hutajulu, Yahya dan Pramukantoro, 2018).

Penelitian selanjutnya diteliti oleh Reza et al. (2019) yang berjudul "Analisis *Routing Multi Copy dengan Stationary Relay Node* dan *Management Buffer First In – First Out* (FIFO) pada *Delay Tolerant Network* (DTN)". Peneliti membandingkan antara protokol *Multi-Copy*, yaitu Spray-and-Wait, Epidemic dan ProPhet dengan adanya penambahan *Stationary Relay Node* dan penambahan manajemen *buffer* FIFO pada jaringan DTN. Saat melakukan pengujian, peneliti menggunakan simulator The ONE Simulator. Penulis menyebutkan bahwa dari segi penyampaian paket, *routing* Spray-and-Wait merupakan *routing* yang lebih efektif jika dibandingkan dengan *routing* yang lain. Hal ini karena *routing* Spray-and-Wait menghasilkan nilai *overhead ratio* yang paling rendah jika dibandingkan dengan routing lainnya. Selain itu, *routing* Spray-and-Wait juga menghasilkan nilai *delivery* probability yang lebih tinggi dibandingkan dengan routing lainnya. Kondisi yang sama juga terjadi meskipun dengan skenario kondisi *buffer* yang terbatas. (Reza et al., 2019).

Penelitian berikutnya dilakukan oleh Rani A. (2014), dengan judul "*Performance Evaluation of* MOFO *Buffer Management Technique With Different Routing Protocols In* DTN *Under Variable Message Buffer Size*" membahas tentang evaluasi kinerja teknik manajemen *buffer* MOFO dengan tiga *routing* yaitu Epidemic, ProPHET dan MaxProp. Dari hasil pengujian yang sudah dilakukan peneliti, kinerja dari *routing* yang berbeda dapat bermanfaat untuk mengoptimalkan kinerja dari DTN dalam hal *latency average, overhead ratio*, *delivery probability* dan *hop count*. Dari ketiga *routing* tadi, *routing* MaxProp memberikan hasil terbaik pada parameter *overhead ratio, delivery probability* dan *latency time average*. Sedangkan untuk parameter *hop count average*, *routing* ProPHet memberikan hasil yang terbaik (Rani, A., 2014).

Penelitian berikutnya yang dilakukan oleh Rani A. (2014), dengan judul "*Performance Evaluation of* MaxProp *Routing Protocol with* DL, FIFO, DLA *and* MOFO *Buffer Management Techniques in* DTN *under Variable Message Buffer Size*". Penelitian ini membahas tentang evaluasi kinerja dari empat teknik manajemen *buffer* pada jaringan DTN, yaitu FIFO, DL, *Drop Largest* (DLA) dan MOFO dengan menggunakan protokol MaxProp pada *Delay* DTN. Penelitian ini menggunakan beberapa parameter uji, yaitu *delivery probability, latency average, overhead ratio* dan *hop count*. Hasil dari pengujian yang dilakukan menunjukkan bahwa *routing* Maxprop memberikan hasil terbaik pada parameter *overhead ratio, hop count* dan *delivery probability* jika adanya penambahan manajemen *buffer* MOFO. Sedangkan jika menggunakan manajemen *buffer* DLA, akan memberikan hasil terbaik pada parameter *latency time average* (Rani, A., 2014).

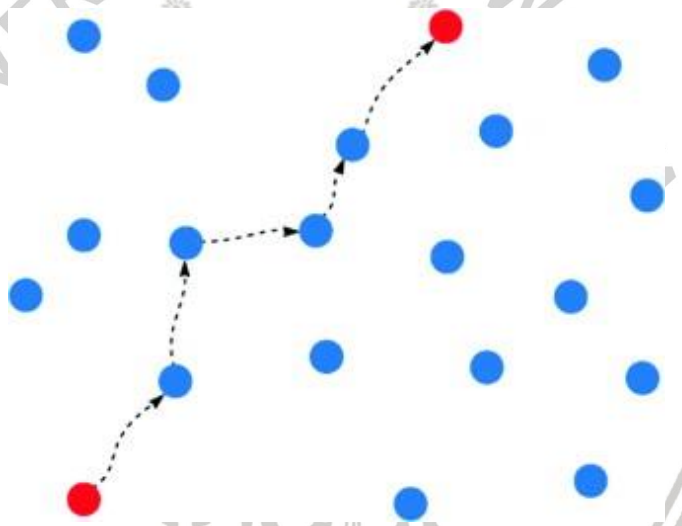
Penelitian terakhir yang dilakukan oleh Putra, Vidya dan Tody (2017), yang berjudul "*Performance Analysis of Message Drop Control Source Relay* (MDC-SR) *in* Maxprop DTN *Routing*" membahas tentang implementasi *management buffer Relay Drop Source Source* (MDC-SR) pada perutean MaxProp. Hasil dari penelitian

ini menunjukkan bahwa manajemen *buffer* kinerjanya terlihat meningkat pada parameter *delivery probability* dengan menggunakan *routing* MaxProp (Dzurovcak dan Yang, 2017).

2.2 Dasar Teori

2.2.1 Delay Tolerant Network

Delay Tolerant Network atau biasa disingkat DTN merupakan jaringan yang ditandai dengan adanya koneksi *intermitten* antar node. Jalur *end-to*-end yang biasanya dimiliki oleh jaringan pada umunya, tidak dimiliki oleh DTN. Oleh karena itu terlihat wajar jika *node* akan sering bergerak menjauh dari area jangkauan *node* yang lain. Konsep DTN berawal dari makalah yang di *publish* oleh Kevin Fall dengan judul "*A Delay Tolerant Network Architecture for Challenge Internets*". Kevin Fall menjelaskan DTN merupakan desain jaringan yang sangat bagus dipakai untuk jaringan yang memiliki koneksi terputus-putus. Jaringan DTN juga sangat cocok diterapkan pada jaringan yang memiliki waktu *delay* yang sangat lama ataupun jaringan yang mempunyai eror tinggi (Fall, 2003).



Gambar 2.1 Ilustrasi r*outing* DTN

Sumber: www.sciencedirect.com

Mekanisme yang digunakan jaringan DTN adalah *store-carry-forward*. Mekanisme ini merupakan mekanisme dimana paket yang sampai ke *node* akan disimpan dan dibawa sampai dapat dikirimkan ke *node* selanjutnya yang ditemuinya. Karena banyaknya kekurangan konektivitas *end-to-end* pada skema DTN, maka diusulkan berberapa skema yang dapat mengatasi masalah tersebut. Skema yang diusulkan untuk permasalahan tersebut adalah dengan menggunakan *routing single-copy* atau *routing multi-copy*. Cara kerja dari *routing single-copy* adalah meneruskan pesan yang ada pada *node* ke *node* lain yang dia jumpai. Berbeda dengan skema yang digunakan *routing single-copy*, *routing multi-copy* akan menyalin paket yang ada ke *node* lain yang dijumpai.

2.2.1.1 *Routing Single-Copy*

Routing single-copy akan meneruskan paket ke node selanjutnya yang dia jumpai. Dengan kata lain, *node* hanya akan memindahkan paket yang ada ke *node* selanjutnya. Cara kerja ini memberikan keuntungan yaitu akan mengurangi beban pada jaringan DTN. Namun ada kekurangan dari penggunaan *routing Single-Copy*, yaitu *delay* yang ada pada jaringan akan menjadi tinggi. Ada dua *routing* pada *Single-Copy*, yaitu *First Contact* dan *Direct Delivery (Mcmahon, Farrell dan College, 2009)*.

Ciri-ciri dari skema *routing* First Contact adalah paket yang ada pada *node* tersebut akan diteruskan ke *node* selanjutnya yang dijumpai. Dengan kata lain, setelah berhasil meneruskan paket ke node selanjutnya, tidak akan ada salinan paket yang tertinggal pada *node* sumber. Berbeda dengan *routing First Contact*, cara kerja dari *routing Direct Delivery* adalah hanya akan meneruskan paket ke *node* tujuan. Jika *node* yang dijumpai tidak *node* tujuan, maka tidak akan dilakukan proses pemindahan paket dari *node* sumber. Namun ada kemiripan *routing First Contact* dan *Direct Delivery*, paket yang diteruskan tidak disalin pada *node* sumber untuk mengurangi beban jaringan. Sebagai contoh, jika *node* A akan meneruskan paket ke *node* B, paket itu akan dikirimkan jika bertemu langsung dengan *node* B. Jika yang ditemui *node* tidak *node* B, *node* A tidak akan melakukan proses pengiriman paket. Namun karena melakukan satu transmisi paket, maka *delay* yang terjadi jika menggunakan skema routing ini tinggi dan tidak terbatas.

2.2.1.2 *Routing Multi-Copy*

Perbedaan antara *routing Single-Copy* dan *routing Multi-Copy* adalah pada proses pemindahan paket. Jika pada *routing Single-Copy* paket hanya akan dipindahkan ke *node* selanjutnya tanpa adanya salinan, *routing Multi-Copy* akan menyalin paket ke *node* selanjutnya. *Routing multi-copy* melakukan proses penyalinan paket ketika hendak mengirimkan paket ke *node* lain. Skema *routing Multi-Copy* adalah *routing* yang menyalin paket yang ada ke setiap *node* yang dijumpai pada area cakupannya. Dengan menggunakan *routing Multi-Copy*, dapat meminimalkan *delay* pada jaringan serta dapat memaksimalkan pengiriman data. Karena akan melakukan proses penyalinan ke setiap *node* yang dijumpai, *routing Multi-Copy* membutuhkan sumber daya yang lebih besar dibandingkan dengan *routing Single-Copy* (Spyropoulos, Psounis dan Raghavendra, 2005). Beban jaringan dengan menggunakan *routing Multi-Copy* juga lebih besar juga dibandingkan dengan *routing Single-Copy*. Namun, *routing Multi-Copy* lebih sering digunakan karena lebih maksimal dalam pengiriman data dan waktu *delay* dengan menggunakan *routing* ini lebih sedikit dibandingkan menggunakan *routing Single-Copy*. *Routing Multi-Copy* terbagi menjadi beberapa jenis *routing*, diantaranya Epidemic, Spray-andWait, MaxProp dan ProPHet.

Routing Epidemic merupakan *routing* yang dapat dianalogikan seperti penularan penyakit. Jika terdapat *node* yang tidak memiliki paket, maka *node* lain yang memiliki paket langsung mengirimkan paket jika bertemu dengan *node*

tersebut. Proses ini terus dilakukan ketika ada *node* yang masih belum memiliki paket.

Mekanisme *routing* Spray-and-Wait sedikit berbeda dengan *routing* yang lain. *Routing* Spray-and-Wait mempunyai dua mekanisme kerja dalam meneruskan paket. Pertama adalah mekanisme *Spray*. Mekanisme ini melakukan penyalinan paket ke *node* yang dijumpai dengan jumlah salinan yang sudah ditentukan. Mekanisme *Spray* mirip dengan mekanisme penyebaran paket yang digunakan *routing* Epidemic. Jika *node* sudah mencapai batas maksimal penyalinan paket, maka dilakukan proses *Wait*. Cara kerja proses *Wait* sama dengan *routing Direct Delivery*. *Node* akan meneruskan langsung pesan ke *node* tujuan ketika bertemu tanpa melakukan penyalinan paket lagi ke *node* lain atau meneruskan paket lagi ke node lain (Sazali dan Basuki, 2018).

Routing MaxProp merupakan *routing* yang menggunakan beberapa mekanisme secara bersamaan untuk meningkatkan *delivery probability* dan dapat mengurangi *latency average* dari paket yang dikirim. *Routing* MaxProp menggunakan beberapa mekanisme yang dapat menentukan urutan paket mana yang dikirim dan yang dihapus. *Routing* ini menggunakan *list cost* agar dapat mengatur urutan paket tadi. *Routing* dapat mencegah diterimanya paket yang sama dua kali dan akan memprioritaskan paket-paket baru untuk diterima (Burgess et al., 2006).

Routing ProPHET (*Probabilistic Routing Protocol Using History of Encounters and Transivity*) adalah routing yang memanfaatkan informasi yang sudah didapatkan sebelumnya untuk melakukan pengiriman paket yang akan datang. Sesuai dengan namanya, *routing* ini mengoptimalkan pengiriman paket dengan memanfaatkan pengetahuan yang sudah didapatkannya saat mengirimkan paket ke *node* lain (Hutajulu, Yahya dan Pramukantoro, 2018).

Untuk mengatasi konektivitas yang terputus-putus, jaringan DTN memiliki beberapa skema yag dapat digunakan. Gambar 2.2 menunjukkan struktur klasifikasi jaringan DTN.

Gambar 2.2 Struktur klasifikasi jaringan DTN

Sumber: Benhamida, Bouabdellah dan Challal (2017)

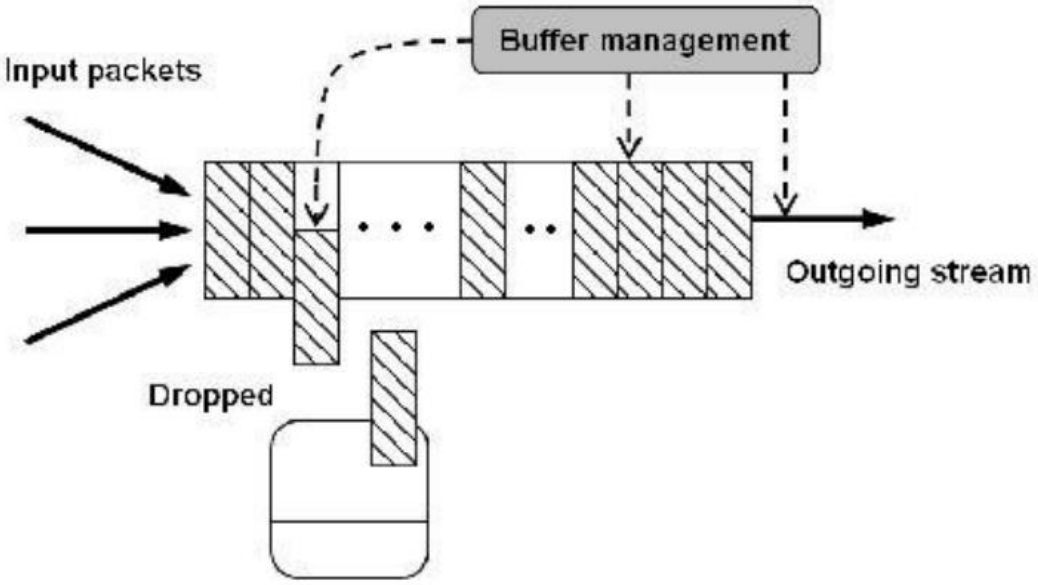
Berpatokan pada *Network Topology*, protokol DTN dapat diklasifikasikan kedalam skema *Deterministic, Stochastic* dan *Coding-based*. Topologi jaringan yang dikenal dengan apriori disebut dengan topologi *Deterministic*. Protokol *Stochastic* merupakan protokol ketika perilaku jaringannya acak. Sedangkan, protokol yang men-*encode* beberapa paket yang besar menjadi satu paket *network coding* disebut dengan *Coding-based*.

Berpatokan pada *Routing Strategy*, protokol DTN diklasifikasikan menjadi dua jenis, yaitu *Social* dan *Opportunistic*. Protokol berbasis *Social* meneruskan paket dengan cara setiap *node* memilih *relay node* untuk meneruskan paket menggunakan *social metrics*. Metrik menentukan kedekatan yang sama antar *node*. Sedangkan protokol yang berbasis *Opportunistic* memilih *node relay* berdasarkan faktor pola mobilitas informasi historis tentang perjumpaan antara *node* sumber dengan *node* tujuan, dan kemungkinan berjumpa lagi dengan *node* tersebut.

Sedangkan pada *Replication and Semantic*, protokol bergantung pada jumlah paket yang di-*inject* pada jaringan (*single* atau *multiple copies*). Selain itu, protokol ini juga bergantung pada pengiriman semantik seperti *multicast, unicast* dan *anycast*. Pertama yang dilakukan adalah, *node* sumber dapat memilih untuk mengirim paket dengan satu salinan paket, atau dengan cara menggandakannya sehingga dapat memperbesar kemungkinan paket dapat sampai ke tujuan. Berdasarkan jumlah salinan, *routing* DTN menggunakan jaringan semantik untuk mengirimkan paket. Transmisi berbasis *multicast* mengirimkan paket ke grup penerima yang tertarik dengan paket yang akan dikirim. Sedangkan transmisi berbasis *unicast* mengirimkan paket ke *node* tujuan yang unik. Sedangkan pada transmisi *anycast*, tujuan a*nycast* tidak dapat diinisialisasikan karena mungkin saja merupakan salah satu dari anggota grup *node* (Benhamida, Bouabdellah dan Challal, 2017) .

2.2.2 Buffer Management

Adanya *buffer management* yang efektif adalah salah satu factor yang mempengaruhi kinerja pada skema *routing* DTN. DTN terdiri dari perangkat genggam dan *node* sensor. Memori merupakan salah yang kendala yang sering dihadapi pada DTN. Paket yang penting seharusnya dapat disimpan dalam *buffer*. Tetapi jika *buffer* penuh, paket-paket tersebut tidak dapat disimpan lagi. Situasi seperti ini sering terjadi pada jaringan yang padat dan jaringan dengan *traffic* yang tinggi. Oleh karena itu *buffer management* sangat diperlukan untuk mengatur paket yang ada pada *buffer*. (Sobin, 2016). *Buffer management* bertugas untuk mengoptimalkan dengan menjadwalkan paket mana yang harus ditransfer terlebih dahulu. Gambar 2.3 menggambarkan fungsi *buffer management* pada jaringan DTN. Manajemen *buffer* akan memutuskan paket mana yang akan diterima dan yang akan di *drop* pada sebuah jaringan. *Buffer management* juga memilih paket yang tertunda dalam *buffer* untuk dikirim.

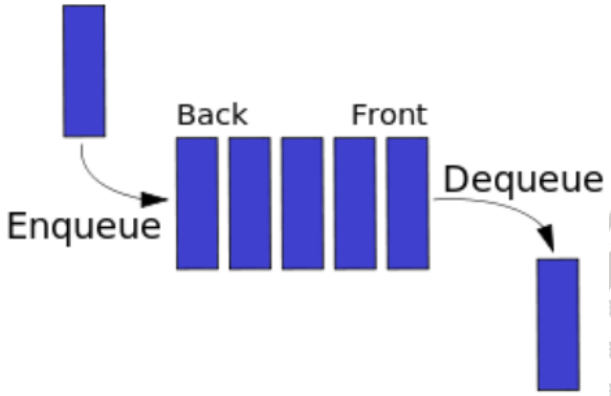


Gambar 2.3 Ilustrasi manajemen *buffer*

Sumber: Li (2008)

2.2.2.1 First In – First Out (FIFO) Buffer Management

Manajemen *buffer First In – First Out* (FIFO) merupakan metode menajeman *buffer* yang digunakan untuk mengatur dan memanipulasi data *buffer*. Teknik ini menjatuhkan paket berdasarkan urutan masuknya mereka ke dalam *buffer*, misalnya paket pertama yang masuk dalam antrian akan menjadi yang pertama kali dijatuhkan jika *buffer* penuh. (Rani, 2014).



Gambar 2.4 Sistem antrian algoritma FIFO

Gambar 2.4 menggambarkan bagaimana manajemen *buffer* FIFO bekerja. Paket yang pertama masuk diproses terlebih dahulu, kemudian antrian tersebut juga yang akan dijatuhkan terlebih dahulu (Reza et al., 2019). Paket ditulis dari "kepala" *buffer* kemudian dibaca dari "ekor" *buffer*. Jika "ekor" menuju ke kepala, maka *buffer* kosong. Tetapi, jika "kepala" berjalan menuju "ekor", maka *buffer* penuh. Dengan demikian maka paket terlama akan dibuang.

Proses ini dilakukan tanpa melihat tingkat kepentingan paket yang dikirim. Kinerja sistem *buffer* ini tergantung pada parameter berikut ini:

a. Tingkat kedatangan paket
b. Ukuran paket
c. Kapasitas layanan

2.2.2.2 Most Forwarded First (MOFO) Buffer Management

Most Forwarded First (MOFO) merupakan manajemen *buffer* yang mencoba untuk memaksimalkan penyebaran paket melalui jaringan dengan menjatuhkan paket yang telah diteruskan jumlah waktu maksimum. Sedemikian rupa paket dengan jumlah *hop* yang lebih rendah memungkinkan untuk melakukan perjalanan lebih jauh dalam jaringan. (Rani, 2014).

Namun, MOFO tidak mempertimbangkan masa berlaku paket. Paket dengan masa pakai yang tidak mencukupi untuk pengiriman tetapi belum diteruskan, sebagian besar tidak akan di-*drop*. Selain itu, sejumlah besar paket yang dijatuhkan dapat diteruskan kembali ke *node* yang sama di masa mendatang karena mode mobilitas. Hal ini cukup menyia-nyiakan sumber daya jaringan.

Tabel 2.2 *Snapshot* manajemen *buffer* MOFO *node* A

M_{id}	M_{Size}	NoF	TTL(*sec*)
M1	400K	7	100
M2	250K	12	50
M3	300K	12	55
M4	350K	5	120
M5	450K	12	40

Sumber: Diadaptasi dari Samyal dan Gupta (2018)

Tabel 2.3 *Snapshot* manajemen *buffer* MOFO *node* B

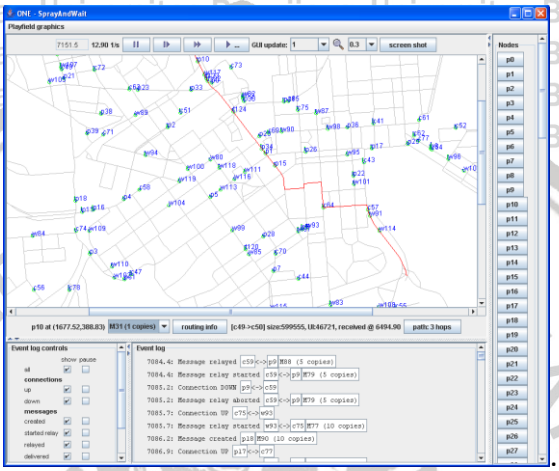
M_{id}	M_{Size}	NoF	TTL(*sec*)
M34	400K	6	110
M45	300K	2	120
M57	450K	4	100
M99	400K	5	90
M16	200K	1	180

Sumber: Diadaptasi dari Samyal dan Gupta (2018)

Tabel 2.2 dan Tabel 2.3 menunjukkan skenario urutan manajemen *buffer* MOFO dengan dua *node* A dan B. Dimana M_{id} mendefinisikan paket, M_{size} mendefinisikan ukuran paket, NoF menentukan jumlah *forwarding* dan TTL

menentukan waktu untuk hidup. Setiap kali kedua *node* A dan *node* B berada dalam jangkauan komunikasi, *node* B siap untuk meneruskan paket M34 ke *node* A. Tetapi ruang *buffer node* A penuh. Dengan skenario seperti itu, paket akan dijatuhkan berdasarkan kebijakan MOFO di *node* A untuk mengakomodasi paket baru M34 yang tiba dari *node* B. Sesuai dengan mekanisme manajemen *buffer* MOFO, *node* A akan menjatuhkan paket (M2, M3 dan M5) untuk mengakomodasi M34 karena pada *node* paket M2 dan M3 memiliki nilai *NoF* yang besar, yaitu 12. (Samyal dan Gupta, 2018).

2.2.3 The Opportunistic Network Environtment Simulator



Gambar 2.5 Tampilan aplikasi The ONE Simulator

Sumber: www.netlab.tkk.fi

Sebagai pendukung penelitian ini, The Opportunistic Network Environtment Simulator (The ONE Simulator) merupakan aplikasi yang digunakan untuk melakukan simulasi. Berikut ini merupakan beberapa fitur yang dimiliki oleh The ONE Simulator:

- Dapat memakai model yang berbeda untuk menggerakkan *node*.
- Melakukan *routing* dengan berbagai algoritma *routing* DTN.
- Dapat memakai berbagai macam tipe *node* sumber dan *node* tujuan
- Dengan menggunakan GUI secara *real time*, aplikasi ini mampu memvisualisasikan mobilitas dan pengiriman paket.
- Dapat meng-*import* data mobilitas dari dunia nyata atau menggunakan *generator* mobilitas lainnya.
- Memberikan berbagai macam laporan dari hasil simulasi

2.2.4 Parameter Performansi

Penilaian kinerja dari *routing* yang dilakukan pada saat melakukan simulasi dapat dilihat dengan menggunakan parameter performansi. Informasi dari parameter ini yang berupa angka yang dihasilkan dengan menggunakan rumus

yang dijalankan pada simulator. Penelitian ini menggunakan tiga parameter performansi untuk menilai kinerja dari setiap *routing* yang digunakan, yaitu *Delivery Probability*, *Overhead Ratio* dan *Latency Average*.

2.2.4.1 Delivery Probability

Delivery Probability adalah parameter yang digunakan untuk menunjukkan tingkat pencapaian bagaimana suatu *node* berhasil mengirimkan paket dari *node* sumber ke *node* tujuan. Nilai dari *delivery probability* didapatkan dengan membandingkan rasio jumlah paket yang terkirim dengan jumlah paket yang dibuat.

$$Delivery\ probability = \frac{D}{G} \quad (2.1)$$

Keterangan:

D = Jumlah paket yang sampai ke tujuan

G = Jumlah paket yang dibuat oleh sumber

2.2.4.2 Overhead Ratio

Banyaknya redundansi paket yang dilakukan dalam pengiriman paket ke tujuan disebut dengan *Overhead Ratio*. *Overhead ratio* dapat juga dikatakan sebagai biaya transmisi yang digunakan pada jaringan. Semakan rendah nilai *overhead ratio* menunjukkan semakin baiknya transmisi yang ada oada jaringan.

$$Overhead\ ratio = \frac{R-D}{D} \quad (2.2)$$

Keterangan:

R = Jumlah paket yang diteruskan *node relay*

D = Jumlah paket yang dikirim menuju *node* tujuan

2.2.4.3 Latency Average

Rata-rata total waktu *delay* saat paket dibuat dan paket diterima pada *node* tujuan disebut dengan *latency average*. Nilai *latency average* akan baik saat *node* sumber sering berjumpa dengan *node* tujuan. Namun, apabila yang terjadi adalah sebaliknya, maka nilai *latency average* akan tinggi. Semakin rendah nilai *latency average* menunjukkan semakin baiknya kinerja jaringan (Syafa, Primananda dan Siregar, 2019). Rumus untuk menghitung *latency average* adalah sebagai berikut:

$$Latency\ average = \frac{\sum_{i=1}^{n}(R_i - G_i)}{n} \quad (2,3)$$

Keterangan:

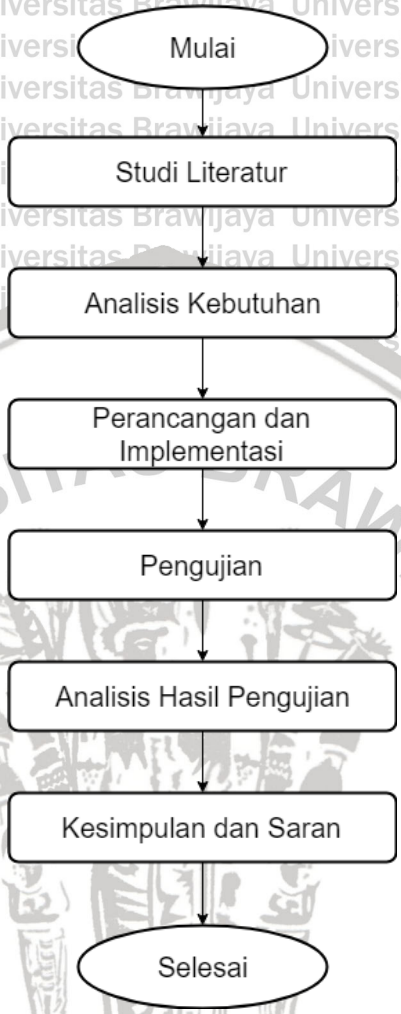
n = Jumlah paket terkirim ke tujuan

R_i = Waktu ketika paket i sampai ke tujuan

G_i = Waktu ketika paket i dibuat

2.2.4.4 METODOLOGI PENELITIAN

Metodologi penelitian menjelaskan alur yang dipakai pada saat melakukan penelitian. Gambar 3.1 menunjukkan diagram alir metodologi pada penelitian ini.



Gambar 2.6 Diagram alir proses metodologi penelitian

2.3 Landasan Kepustakaan

Penjelasan mengenai beberapa teori yang digunakan dalam penelitian dijelaskan pada landasan kepustakaan. Teori-teori tersebut didapatkan dari buku, situs web, jurnal, makalah ilmiah dan beberapa penelitian yang sudah dilakukan sebelumnya. Kajian pustaka yang dibahas pada landasan kepustakaan adalah DTN, *routing multi-copy*, manajemen *buffer* FIFO, manajemen *buffer* MOFO dan The ONE Simulator.

2.4 Analisis Kebutuhan

Sebelum melakukan perancangan maka dilakukan analisis mengenai kebutuhan yang diperlukan pada saat penelitian. Semua kebutuhan-kebutuhan yang diperlukan pada saat merancang sistem untuk penelitian dibahas pada

analisis kebutuhan. Kebutuhan pada penelitian ini dibagi menjadi dua, yaitu kebutuhan perangkat keras dan kebutuhan perangkat lunak.

2.4.1 Kebutuhan Perangkat Keras

Perangkat keras yang dipakai diharapkan manpu mendukung penelitian ini dan dapat melakukan *multitasking*. Berikut ini merupakan perangkat keras yang dipakai saat penelitian:

- Tipe Laptop : Asus X455D
- RAM : 4GB
- *Processor* : AMD A10-8700P

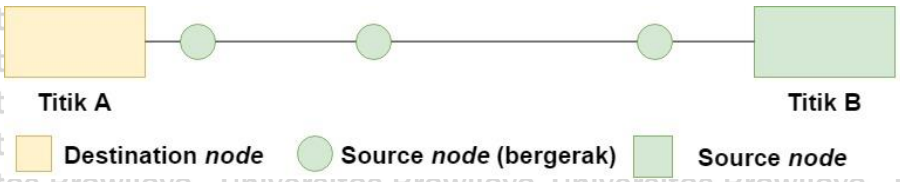
2.4.2 Kebutuhan Perangkat Lunak

Perangkat lunak yang dipakai untuk penelitian ini merupakan perangkat lunak yang mempunyai fitur yang mampu menunjang proses penelitian. Berikut ini merupakan perangkat lunak yang dipakai pada penelitian ini.

- Windows 10 Pro 64-bit
- Visual Studio Code
- The ONE Simulator
- OpenJump
- OSM2WKT
- OpenStreetMap

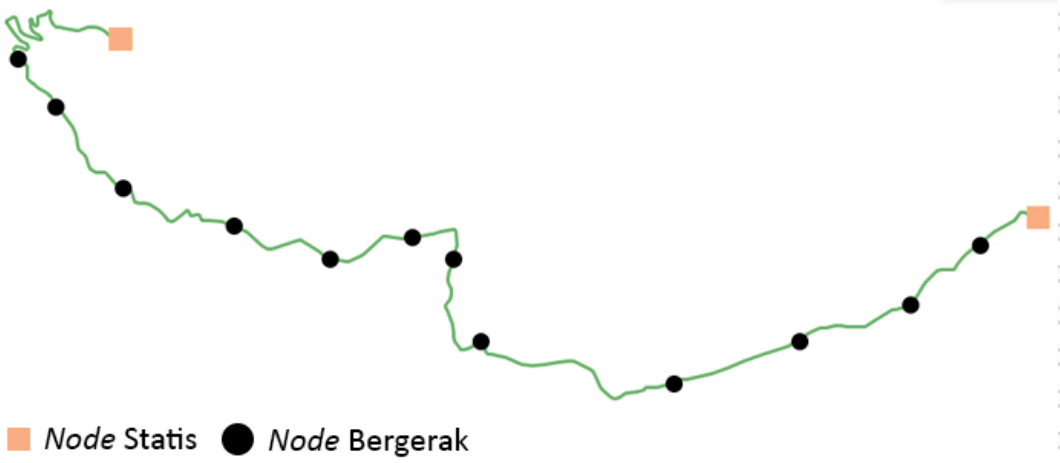
2.5 Perancangan

Bagian perancangan membahas tentang proses simulasi yang dilakukan saat penelitian secara umum. Peta yang dipakai pada penelitian ini merupakan jalur pendakian Gunung Sibayak. Daerah ini digunakan karena merupakan area pendakian dan rawan terjadi bencana. Jalur peta akan diimpor dari OpenStreetMap.org. Kemudian akan dirancang *routing* yang digunakan, yaitu *routing multi-copy*. Jaringan DTN menggunakan topologi yang dinamis. Oleh karena itu sulit memperkirakan arah gerak *node*. *Node* yang bergerak pada simulator disesuaikan dengan pergerakan dari manusia yang berjalan secara acak.



Gambar 2.7 Perancangan *node* DTN

Titik A dan Titik B merupakan *node* statis dan tidak bergerak. Sedangkan titik yang bulat merupakan *node* dinamis yang disesuaikan dengan pergerakan manusia. Jenis pergerakan yang dipakai pada penelitian ini merupakan *ShortestPathMapBasedMovement*.



Gambar 2.8 Topologi DTN

Gambar 3.3 merupakan gambaran dari topologi DTN yang dipakai pada saat proses penelitian. Terdapat dua *node* statis yang berada diujung jalur. Sedangkan bulatan yang berwarna hitam akan menjadi *node* bergerak. *Node* bergerak itu adalah pergerakan dari manusia yang akan disimulasikan.

2.6 Implementasi

Bagian Implementasi adalah tahapan dimana dilakukan proses implementasi berdasarkan hasil perancangan yang sudah ada sebelumnya. Penulis menggunakan dua manajemen *buffer* dan tiga *routing multi-copy* untuk membandingkan performa dari masing-masing *routing* yang digunakan, yaitu Epidemic, Spray-and-Wait dan MaxProp. *Routing* dan manajemen *buffer* disimulaksikan menggunakan aplikasi The ONE Simulator. Berikut ini adalah hal-hal yang diimplementasikan pada penelitian ini:

- Implementasi jaringan DTN.
- Implementasi *routing multi-copy* Epidemic, Spray-and-Wait dan MaxProp
- Implementasi konfigurasi parameter DTN.
- Implementasi manajemen *buffer* FIFO
- Implementasi manajemen *buffer* MOFO

2.7 Pengujian dan Analisis

Bagian pada bab ini dibagi menjadi dua, yaitu pengujian dan analisis. The ONE Simulator merupakan alat simulasi yang dilakukan pada saat pengujian. Parameter pengujian yang dipakai untuk menilai kinerja dari jaringan DTN pada penelitian ini

yaitu *delivery probability*, *overhead ratio* dan *latency average*. Proses simulasi berdurasi selama 12 jam.

Pengujian pada penelitian ini dibagi menjadi beberapa skenario. Berikut ini merupakan skenario yang digunakan pada penelitian ini:

1. Skenario 1
   Simulasi dengan *routing* Epidemic dengan mengimplementasikan manajemen *buffer* FIFO dan MOFO dengan ukuran *buffer* 5MB, 7MB, 10MB dan 12MB.
2. Skenario 2
   Simulasi dengan *routing* Spray-and-Wait dengan mengimplementasikan manajemen *buffer* FIFO dan MOFO dengan ukuran *buffer* 5MB, 7MB, 10MB dan 12MB.
3. Skenario 3
   Simulasi dengan *routing* MaxProp dengan mengimplementasikan manajemen *buffer* FIFO dan MOFO dengan ukuran *buffer* 5MB, 7MB, 10MB dan 12MB.

Selanjutnya adalah menganalisis hasil dari pengujian yang sudah didapatkan sehingga didapatkan pengaruh penggunaan manajemen *buffer* FIFO dan MOFO sesuai dengan ukuran *buffer* yang dipakai di setiap skenario yang digunakan. Bab ini juga memaparkan perbandingan kinerja tiap-tiap *routing multi-copy* yang digunakan pada DTN dengan mengimplementasikan manajemen *buffer* FIFO dan MOFO menggunakan parameter uji *delivery probability, overhead ratio* dan *latency average*.

2.8 Penutup

Bab ini berisi rangkuman mengenai hasil pengujian dan analisis yang sudah dilakukan pada bab sebelumnya yang akan dituliskan pada subbab kesimpulan. Kesimpulan akan menjawab permasalahan yang ada pada rumusan masalah. Setelah itu, penulis juga memberikan saran yang berisi ide atau pengembangan untuk penelitian selanjutnya yang terkait dengan penelitian ini.

BAB 3 PERANCANGAN DAN IMPLEMENTASI

Bab ini menjelaskan mengenai tahapan-tahapan perancangan yang dilakukan. Sebelum melakukan simulasi, tahapan pertama yang dilakukan adalah mengumpulkan kebutuhan yang diperlukan pada saat melakukan simulasi. Setelah kebutuhan terpenuhi, maka selanjutnya adalah melakukan simulasi. Pada tahapan simulasi, dilakukan perancangan topologi, perancangan *node* dan juga perancangan skenario. Kemudian, tahapan terakhir adalah melakukan konfigurasi pada aplikasi The ONE Simulator.

3.1 Analisis Kebutuhan

3.1.1 Kebutuhan Perangkat Keras

Dibutuhkan perangkat keras yang dapat menjalankan berbagai aplikasi pendukung untuk penelitian ini. Selain itu, hal penting lainnya adalah perangkat keras yang disediakan harus dapat melakukan *multitasking* dan dapat menjakan simulator yang dibutuhkan pada saat penelitian. Berikut ini adalah spesifikasi perangkat keras yang digunakan pada penelitian ini.

- Asus X455D
- *Processor* AMD A10-8700P Radeon R6, 10 *Compute Cores* 4C+6G
- RAM 4GB

3.1.2 Kebutuhan Perangkat Lunak

Beberapa perangkat lunak juga dibutuhkan untuk mendukung penelitian ini. Berikut ini merupakan perangkat lunak yang digunakan pada penelitian ini.

- Sistem Operasi Windows 10 Pro 64-bit, merupakan sistem operasi yang digunakan pada Asus X455D
- The Opportunistic Network Environment Simulator v1.6.0, merupakan aplikasi simulasi pada *Delay Tolerant Network*.
- Visual Studio Code, digunakan untuk mengolah file konfigurasi *default_settings* dari perangkat The ONE Simulator.
- OpenStreetMap.org, digunakan untuk *mapping area* dengan hasil ekstensi .osm yang menjadi lokasi dilakukannya simulasi.
- OpenJUMP 1.14.1.rev.6147 PLUS, merupakan aplikasi *Geographic Information System* (GIS) *open source* yang ditulis dalam Bahasa pemrograman Java yang digunakan untuk mengedit hasil pemetaan area yang sudah ditentukan.
- OSM2WKT, merupakan perangkat lunak untuk mengkonversi file yang berekstensi .osm menjadi format .wkt.

3.1.3 Kebutuhan Perancangan Skenario

Saat melakukan perancangan simulasi, ada beberapa kebutuhan skenario yang harus terpenuhi. Berikut merupakan perancangan skenario yang disimulasikan pada penelitian ini.

1. Lokasi pendakian Gunung Sibayak yang berada di Kabupaten Karo, Sumatera Utara digunakan sebagai peta lokasi untuk simulasi *Delay Tolerant Network*.
2. *Node* yang bergerak pada saat simulasi *Delay Tolerant Network* disesuaikan dengan orang yang melintasi lokasi yang telah ditentukan.
3. Kecepatan pada tiap *node* disesuaikan dengan kecepatan pada kondisi nyata.
4. *Routing* yang digunakan adalah *routing Multi-Copy* yaitu MaxProp, Spray-and-Wait dan Epidemic.
5. Penambahan manajemen *buffer* FIFO dan MOFO pada setiap *routing* untuk mengetahui pengaruh penambahan manajemen *buffer* tersebut pada kinerja setiap *routing*.

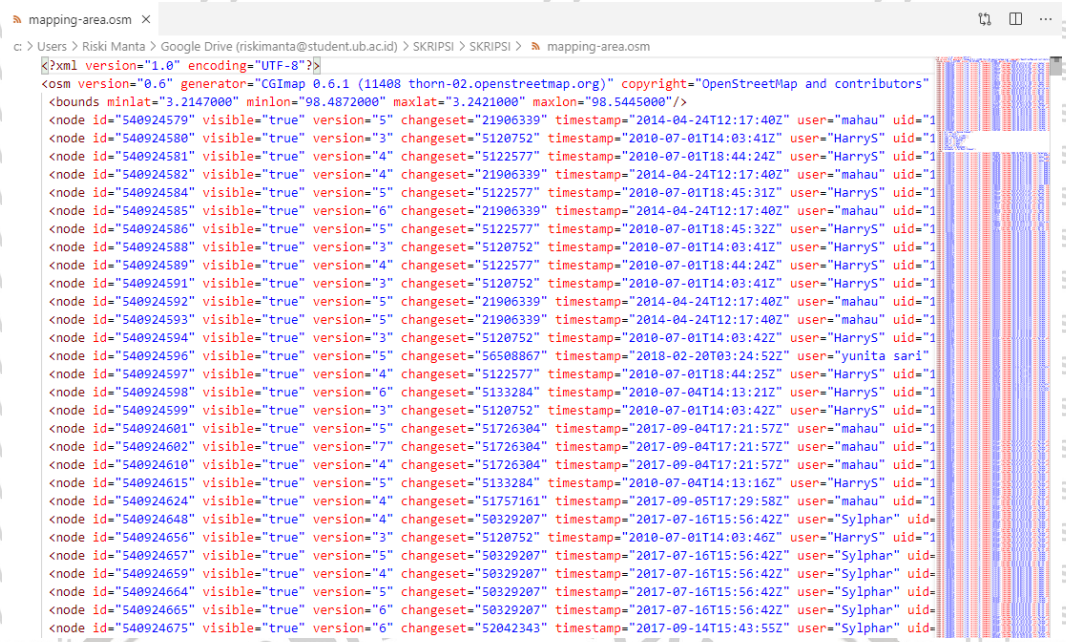
3.2 Pembuatan Jalur Peta Menggunakan OpenJump

Rute pendakian Gunung Sibayak yang tergambar pada Gambar 4.1 merupakan peta yang digunakan pada penelitian ini. Peta rute didapatkan dari website OpenStreetMap.org. Dari website tersebut, rute peta di *export* dengan format osm (*open street map*)

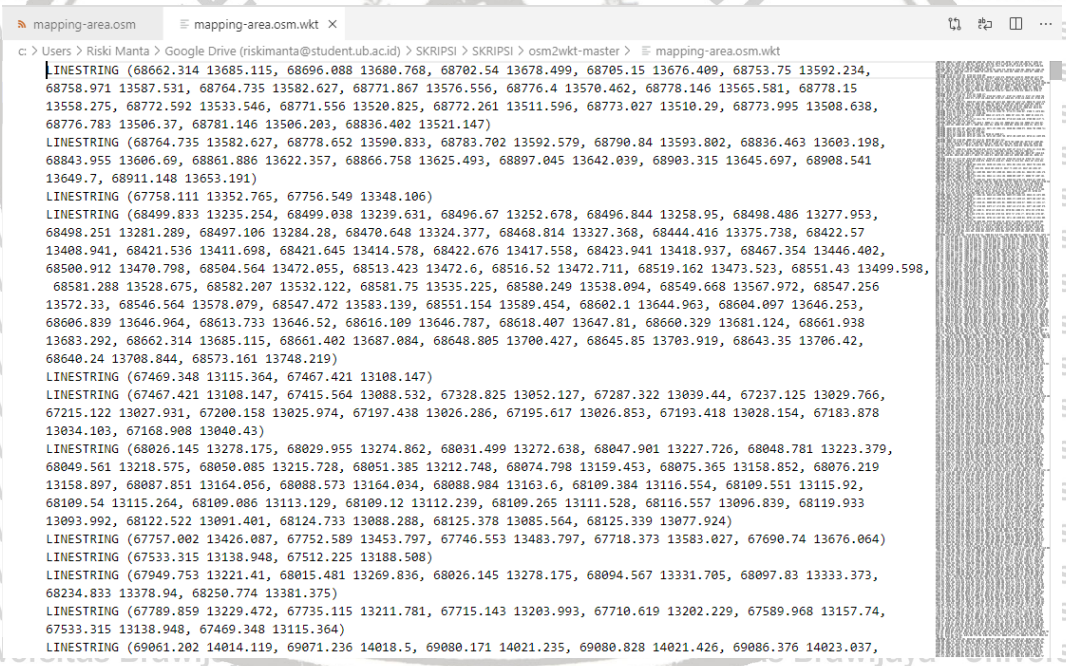
Gambar 3.1 Peta pendakian Gunung Sibayak

Sumber: openstreetmap.org

Data file osm dapat dilihat dengan menggunakan Visual Code Studio seperti yang ada pada Gambar 4.2. Namun, file osm tersebut harus dikonversi terlebih dahulu menjadi format wkt (*well known text*) agar file dapat diolah pada aplikasi OpenJUMP. Proses konversi dapat dilakukan dengan menggunakan OSM2WKT. Data file yang sudah dikonversi menjadi wkt dapat dilihat pada Gambar 4.3.



Gambar 3.2 Tampilan file osm peta pendakian Gunung Sibayak

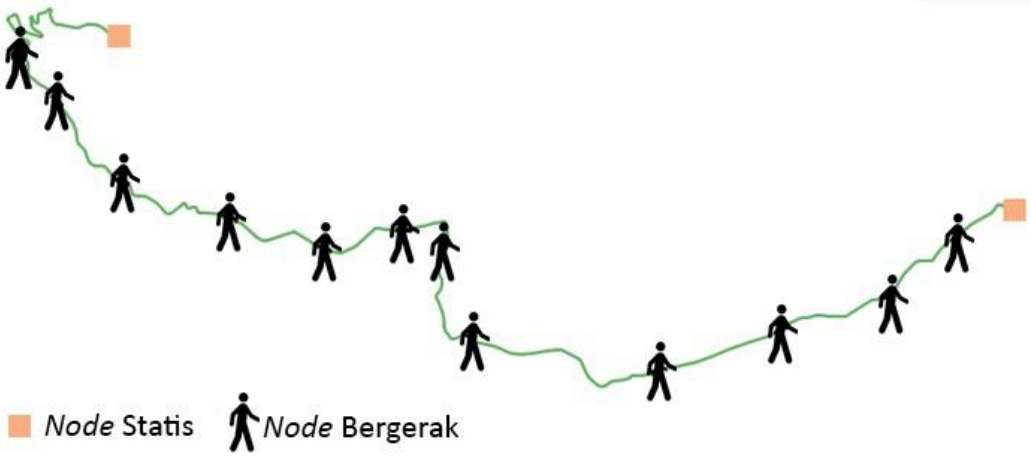


Gambar 3.3 Tampilan file peta pendakian Gunung Sibayak dengan format wkt

3.3 Perancangan Topologi

Pada penelitian ini, node dibagi menjadi dua, yaitu node statis dan node dinamis. Manusia yang bergerak, diasumsikan sebagai *node* dinamis. Sedangkan Gunung Sibayak dan desa Doulu diasumsikan sebagai *node* statis. Diasumsikan *node* yang bergarak ada sebanyak 100 *node*. *ShortestPathMapBasedMovement* menjadi model pergerakan yang digunakan pada simulasi ini. Kecepatan *node* bergerak yaitu diantara 1.5-2 Km/Jam. Gambar 4.4 menunjukkan jalur *node* yang

menjadi jalur pergerakan dari *node* dinamis. Sedangkan dikedua ujung jalur merupakan lokasi dari *node* statis.



Gambar 3.4 Perancangan *node*

Sebelum dilakukannya simulasi, terlebih dahulu harus dilakukan perancangan parameter yang digunakan pada saat simulasi. Parameter yang digunakan pada penelitian ini tidak berubah-ubah untuk setiap skenario. *Routing* yang digunakan adalah Epidemic, Spray-and-Wait, dan MaxProp yang termasuk ke dalam *routing Multi-Copy* dengan waktu simulasi selama 12 jam sesuai dengan waktu aktivitas pendakian. Jumlah *node* yang digunakan adalah 100 *node* dengan kecepatan pergerakan *node* 0.5-1.5 km/Jam sesuai dengan kecepatan dari pejalan kaki. TTL yang digunakan adalah 360 menit sesuai dengan kondisi *node* yang dibawa pejalan kaki yaitu *mobile* yang tidak bertahan terlalu lama. Untuk area peta yang digunakan adalah area pendakian Gunung Sibayak yang berada di Sumatera Utara. Ukuran *buffer* untuk pengujian pada penelitian ini adalah 5MB, 7MB, 10MB dan 12 MB dengan kecepatan pengiriman masing-masing *node* sebesar 500kB-1MB. *Movement Model* yang dipilih adalah *ShortestPathMapBased* karena memiliki hasil pengiriman paket yang lebih baik dari pada *MapBased*. *Bluetooth interface* adalah *network interface* yang digunakan pada penelitian ini karena mendukung penyebaran protokol DTN dan jarak transmisi yang digunakan sebesar 25 meter. Parameter simulasi ditunjukkan pada Tabel 4.1.

Tabel 3.1 Parameter Simulasi

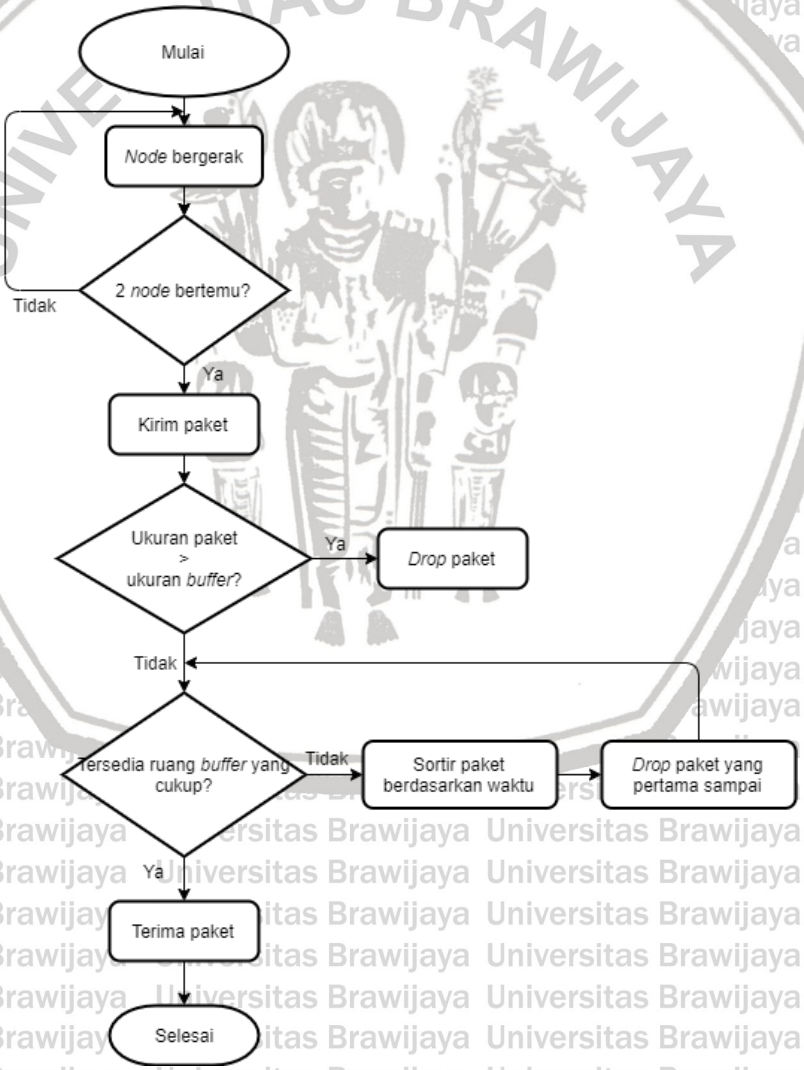
No.	Parameter	Skenario
1	Waktu simulasi	12 jam
2	Jumlah *node*	100 *node*
3	*Time To Live*	360 menit
4	Kecepatan rata-rata	0.5-1.5 km/jam
5	Lokasi	Jalur pendakian Gunung Sibayak

Tabel 3.1 Parameter Simulasi (lanjutan)

No.	Parameter	Skenario
6	Ukuran *buffer*	5MB, 7MB, 10MB, 12MB
7	*Routing*	Epidemic, Spray-and-Wait, MaxProp
8	*Movement Model*	*ShortestPathMap-Based*
9	Ukuran paket	500kB-1MB
10	Kecepatan pengiriman data	250KBps
11	*Network interface*	*Bluetooth*

3.4 Perancangan Manajemen *Buffer*

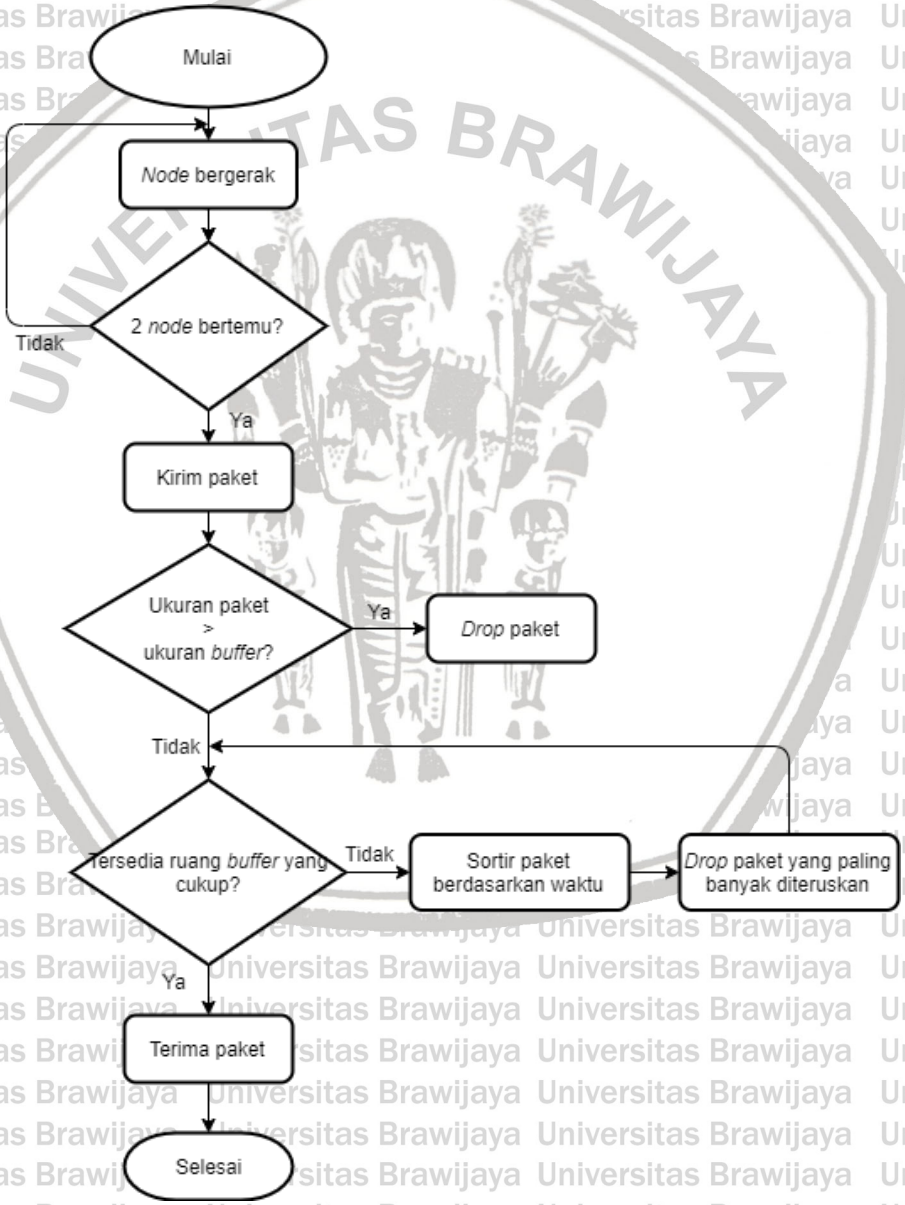
3.4.1 Perancangan Manajemen *Buffer* FIFO



Gambar 3.5 Diagram alir manajemen *buffer* FIFO

Gambar 4.5 merupakan diagram alir manajemen *buffer* yang digunakan pada penelitian ini. Proses pengiriman dan penerimaan paket dilakukan ketika dua *node* bertemu. Sebelum paket dikirimkan, *node* penerima terlebih dahulu memeriksa paket yang dimasukkan ke dalam *buffer*. Jika ukuran paket lebih besar dari pada ukuran *buffer*, maka paket akan langsung *drop*. Jika tidak, selanjutnya adalah memeriksa apakah tersedia *buffer* untuk menyimpan paket. Jika tidak tersedia, maka dilakukan penyortiran paket berdasarkan waktu paket sampai ke *buffer*. Manajemen *buffer* akan menjatuhkan paket yang pertama kali sampai. Setelah itu, dicek kembali apakah *buffer* dapat menerima paket yang akan dikirim. Jika *buffer* cukup, *node* akan menerima paket.

3.4.2 Perancangan Manajemen *Buffer* MOFO



Gambar 3.6 Diagram alir manajemen *buffer* MOFO

Gambar 4.6 merupakan diagram alir manajemen *buffer* yang digunakan pada penelitian ini. Proses pengiriman dan penerimaan paket dilakukan ketika dua *node* bertemu. Sebelum paket dikirimkan, *node* penerima terlebih dahulu memeriksa paket yang akan dimasukkan ke dalam *buffer*. Jika ukuran paket lebih besar dari pada ukuran *buffer*, maka paket akan langsung *drop*. Jika tidak, selanjutnya adalah memeriksa apakah tersedia *buffer* untuk menyimpan paket. Jika tidak tersedia, maka dilakukan penyortiran paket berdasarkan jumlah penerusan paket. Manajemen *buffer* akan menjatuhkan paket yang paling banyak diteruskan. Setelah itu, akan dicek kembali apakah *buffer* dapat menerima paket yang akan dikirim. Jika *buffer* cukup, *node* akan menerima paket.

3.5 Konfigurasi The ONE Simulator

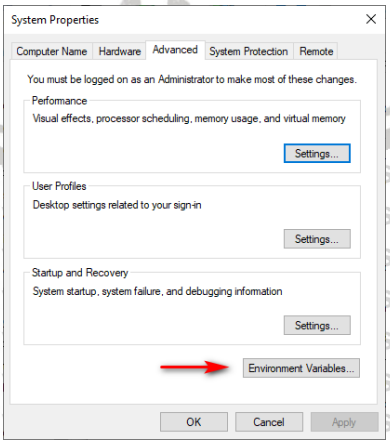
3.5.1 Instalasi The ONE Simulator

Aplikasi The ONE Simulator dapat diunduh melalui http://akeranen.github.io/the-one/. Terdapat beberapa versi yang tersedia pada web tersebut. Pada penelitian kali ini aplikasi yang digunakan adalah versi 1.6.0 karena merupakan versi terbaru ketika penelitian ini dimulai dan kompatibel dengan Windows 10. *File* yang udah di diunduh harus di ekstrak dulu sebelum digunakan.

3.5.2 *Setting* Java *Path Variable*

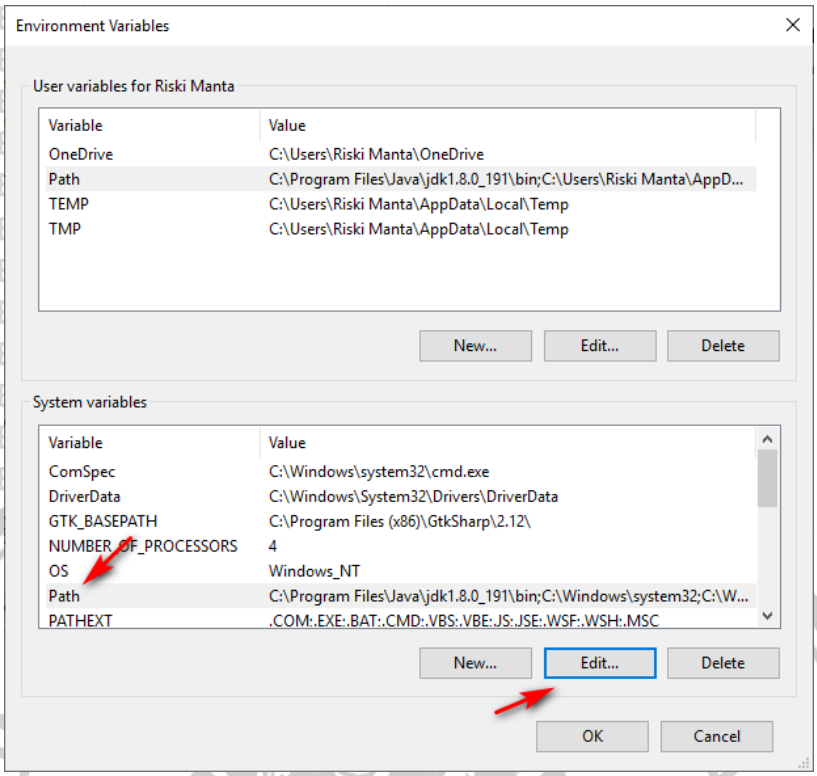
Sebelum melakukan simulasi, ada beberapa hal yang harus di-*install* dan di konfigurasikan terlebih dahulu. Pertama, JDK (*Java Development Kit*) harus sudah terpasang pada sistem operasi yang digunakan. Penulis menggunakan JDK versi 1.8.0_191 pada penelitian ini. Setelah itu, dilakukan konfigurasi Java *path variable* seperti berikut ini:

1. Klik kanan pada *My Computer* kemudian klik *Properties*.
2. Klik *Advanced System Settings* kemudian pada tab *Advanced* klik tombol *Environment Variables*.



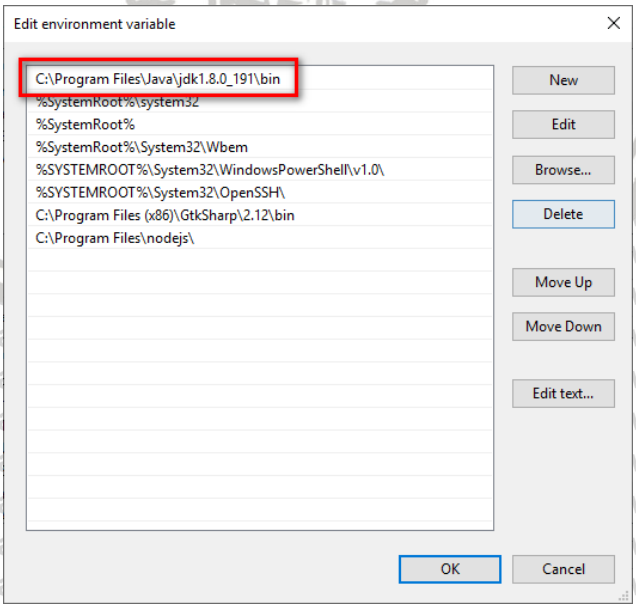
Gambar 3.7 Tampilan jendela *System Properties*

3. Setelah jendela *Environment Variables* terbuka, kemudian klik *variable path* pada *System Variables* dan klik tombol *Edit*.



Gambar 3.8 Tampilan jendela *Environment Variables*

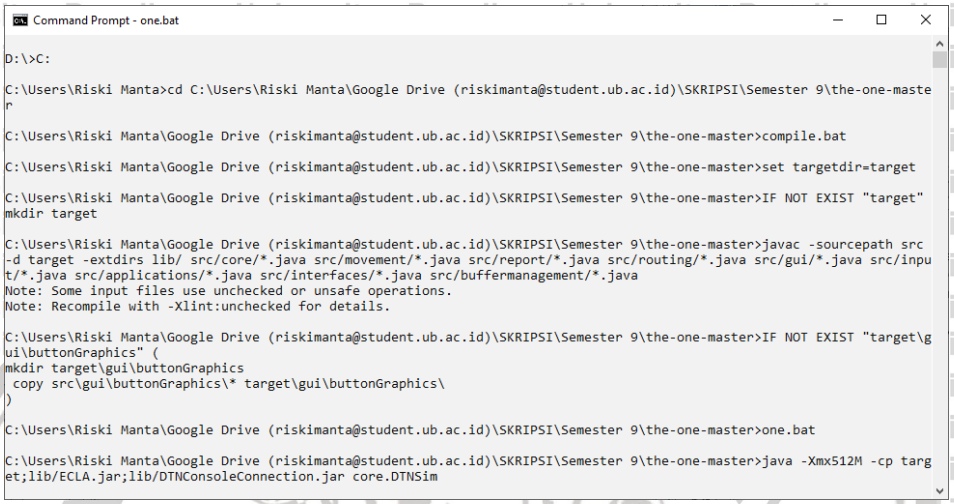
4. Akan muncul jendela *Edit environment variable*. Cara penulisan *Path* yang benar adalah dengan menuliskan *path jdk* yang sudah terpasang pada sistem operasi.



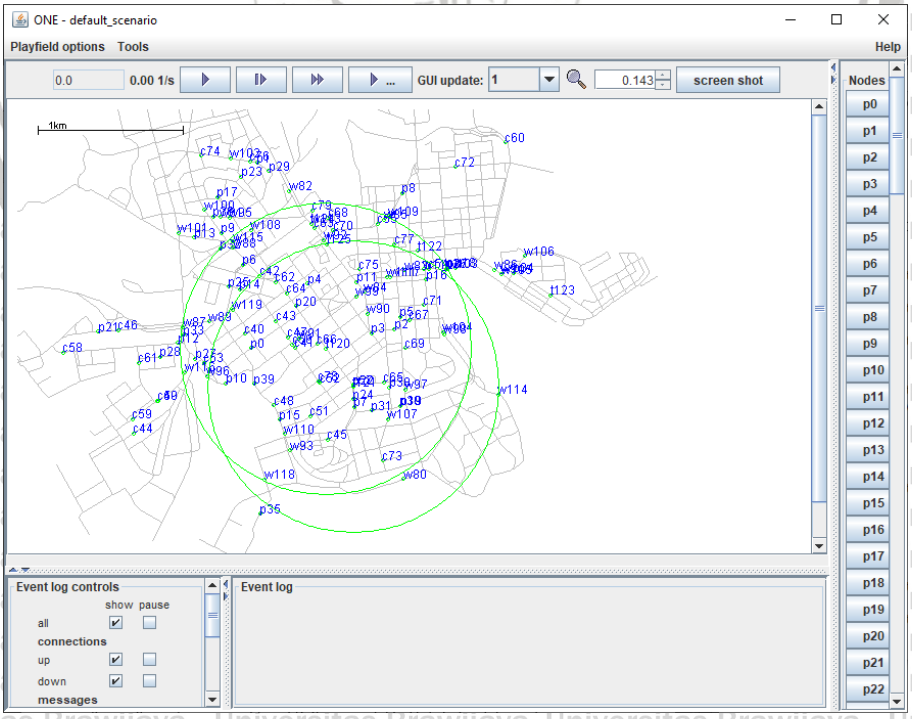
Gambar 3.9 Tampilan jendela *Edit environment variables*

3.5.3 *Compiling* dan Menjalankan The ONE Simulator

Setelah semua konfigurasi selesai dilakukan, maka selanjutnya adalah melakukan *compile* pada aplikasi The ONE Simulator. Proses *compile* dilakukan dengan melakukan *command line* (cmd). Pada cmd masuk ke direktori dimana aplikasi The ONE Simulator diekstrak. Kemudian jalankan compile.bat. Setelah selesai selanjutnya adalah membuka aplikasi The ONE Simulator dengan mengetikkan perintah one.bat.



Gambar 3.10 Tampilan *command line* saat melakukan perintah complile.bat dan one.bat



Gambar 3.11 Tampilan *default* The One Simulator

Gambar 4.11 merupakan tampilan *default* dari aplikasi The ONE Simulator saat pertama kali dijalankan. Saat pertama kali dibuka, terlihat peta dengan *node* yang dapat dijalankan dengan waktu 12 jam. Simulasi dapat dilakukan dengan cara klik tombol *Play* pada simulator.

3.6 Skenario Simulasi

Peta yang digunakan untuk melakukan simulasi pada penelitian ini adalah jalur pendakian Gunung Sibayak. Simulasi dilakukan selama 12 jam dan dapat dipercepat sesuai dengan percepatan yang ada pada simulator dengan menggunakan 100 *node*. Kecepatan tiap *node* nya adalah 0.5-1.5 km/jam sesuai dengan rata-rata kecepatan orang berjalan. Ukuran *buffer* yang digunakan adalah 5MB, 7MB, 10MB dan 12MB. *Routing* yang digunakan saat simulasi adalah Epidemic, Spray-and-Wait dan MaxProp dengan menggunakan manajemen *buffer* *First In-First Out* (FIFO) dan *Most Forwarded First* (MOFO).

Skenario simulasi yang pertama menggunakan *routing* Epidemic dengan menambahkan manajemen *buffer* FIFO dan MOFO dengan ukuran *buffer* 5MB, 7MB, 10MB dan 12MB. Skenario simulasi yang kedua adalah menggunakan *routing* Spray-and-Wait dengan menambahkan manajemen *buffer* FIFO dan MOFO dengan ukuran *buffer* 5MB, 7MB, 10MB dan 12 MB. Sedangkan skenario simulasi yang ketiga adalah menggunakan *routing* MaxProp dengan menambahkan manajemen *buffer* FIFO dan MOFO dengan ukuran *buffer* 5MB, 7MB, 10MB dan 12MB. Skenario pengujian dapat dilihat padaTabel 4.2.

Tabel 3.2 Skenario simulasi

No.	Skenario	Penjelasan	*Buffer*
1	Skenario 1	Simulasi dengan menggunakan *routing* Epidemic yang telah ditambahkan manajenemen *buffer* FIFO dan MOFO	5MB, 7MB, 10MB, 12MB
2	Skenario 2	Simulasi dengan menggunakan *routing* Spray-and-Wait yang telah ditambahkan manajemen *buffer* FIFO dan MOFO	5MB, 7MB, 10MB, 12MB
3	Skenario 3	Simulasi dengan menggunakan *routing* MaxProp yang telah ditambahkan manajemen *buffer* FIFO dan MOFO	5MB, 7MB, 10MB, 12MB

3.7 Konfigurasi *Default Settings* Skenario

Sebelum memulai simulasi dengan skenario yang sudah dibuat, diperlukan konfigurasi pada *default settings* pada aplikasi The ONE Simulator sesuai dengan parameter yang sudah ditetntukan. Berikut ini merupakan pengaturan-pengaturan yang digunakan pada penelitian ini:

A. Konfigurasi Waktu Simulasi

Lamanya waktu simulasi dapat diatur dengan mengganti nilai dari variabel Scenario.endTime pada *default settings* aplikasi The ONE Simulator. Satuan yang digunakan pada variabel ini adalah satuan detik. Pada penelitian ini, simulasi dijalankan selama 12 jam waktu simulator.

default_settings.txt: Konfigurasi waktu simulasi

```
1  # 43200s == 12h
2  Scenario.endTime = 43200
```

B. Konfigurasi Grup *Node*

Tanda pengenal dibutuhkan untuk membedakan antara ketiga grup *node* yang ada. Ketiga grup node tersebut adalah pendaki dan kedua ujung jalur pendakian yaitu Doulu dan Sibayak. Untuk mengganti identitas pada masing masing grup, dapat dilakukan dengan mengganti nilai dari variabel Group.groupID. Pada penelitian ini untuk pendaki diberikan nilai p, untuk lokasi Doulu diberikan nilai d dan untuk lokasi Sibayak diberikan nilai s.

default_settings.txt: Konfigurasi grup node

```
1  # group ID
2  Group.groupID = p
3  Group1.groupID = d
4  Group.groupID = s
```

C. Konfigurasi Jumlah *Node*

Jumlah *node* untuk melakukan simulasi pada penelitian ini adalah 100 *node*. Untuk memberikan jumlah *node* pada The ONE Simulator dapat dilakukan dengan memberikan nilai 100 pada variabel Group.nrofHosts pada setiap grup.

default_settings.txt: Konfigurasi jumlah node

```
1  # Jumlah Node
2  Group.nrofHosts = 100
3  Group1. nrofHosts = 1
4  Group2. nrofHosts = 1
```

D. Konfigurasi Kecepatan *Node*

Pada The ONE Simulator, kita juga dapat mengatur kecepatan dari *node*. Satuan yang digunakan pada The ONE Simulator satuan kecepatan yang digunakan adalah *meter per second* (m/s). Pada penelitian ini kecepatan yang digunakan adalah antara 0.5-1.5 km/jam.

default_settings.txt: Konfigurasi kecepatan node

```
1  # Kecepatan Pergerakan Node
2  # 1 kph = 0.277778 mps
3  Group.speed = 1.38889, 4.16667
```

E. Konfigurasi Pergerakan *Node*

Pergerakan *node* yang digunakan pada penelitian ini adalah *StationaryMovement* untuk kedua *node* statis dan *ShortestPathBasedMovement* untuk *node* yang bergerak. Model pergerakan dapat dikonfigurasikan dengan cara mengubah nilai pada variabel Group.movementModel.

default_settings.txt: Konfigurasi pergerakan node

```
1  # Model Pergerakan Node
2  Group.movementModel = ShortestPathBasedMovement
3  Group1.movementModel = StationaryMovement
4  Group2.movementModel = StationaryMovement
```

F. Konfigurasi TTL (*Time-to-live*)

Time-to-live yang digunakan pada penelitian ini adalah 360 menit. Satuan waktu yang digunakan aplikasi The ONE Simulator satuan menit. Untuk mengkonfigurasikan TTL dapat dilakukan dengan cara mengganti nilai pada variabel Group.msgTtl.

default_settings.txt: Konfigurasi TTL (Time-to-live)

```
1  # Time-to-live
2  Group.msgTtl = 360
```

G. Konfigurasi Peta Wilayah

Peta yang digunakan pada penelitian ini adalah jalur pendakian Gunung Sinabung yang berada di Sumatera Utara. Untuk mengkonfigurasikan peta pada penelitian ini, dapat dilakukan dengan cara mengubah nilai dari variabel MapBasedMovement.mapFile1.

default_settings.txt: Konfigurasi peta wilayah

```
1  # Peta Wilayah
2  MapBasedMovement.mapFile1 = data/mapping-area.wkt
```

H. Konfigurasi *Routing*

Routing yang digunakan pada penelitian ini adalah Epidemic, Spray-and-Wait dan MaxProp. Untuk mengkonfigurasikan *routing* pada penelitian ini dapat dilakukan dengan cara mengganti nilai dari variabel Group.router.

default_settings.txt: Konfigurasi Routing

```
1  Group.router = [EpidemicRouter; SprayAndWaitRouter; MaxPropRouter;]
```

I. Konfigurasi Ukuran *Buffer*

Ukuran *buffer* yang digunakan pada penelitian ini adalah 5MB, 7MB, 19MB dan 12MB. Untuk mengkonfigurasikan ukuran *buffer* dapat dilakukan dengan mengganti nilai dari variabel Group.bufferSize.

default_settings.txt: Konfigurasi Ukuran Buffer

```
1  Group.bufferSize = [5M; 7M, 10M, 12M]
```

J. Konfigurasi Ukuran Paket

Simulasi pada penelitian ini menggunakan paket yang berukuran 500kB-1MB yang dikirimkan dengan interval waktu pengiriman paket 25 – 35 detik. Untuk mengkonfigurasikan ukuran paket pada penelitian ini dapat dilakukan dengan mengganti variabel Event1.interval untuk interval waktu pengiriman dan Events1.size untuk ukuran paket.

default_settings.txt: Konfigurasi Ukuran Paket

```
1  Events.size = 500k,700k
2  Events.interval = 25,35
```

K. Konfigurasi Manajemen *Buffer*

Agar memudahkan proses simulasi, untuk memilih manajemen *buffer*, dapat dilakukan dengan mengganti nilai dari variabel Group.dropPolicy. Manajemen *buffer* yang digunakan pada penelitian ini adalah FIFO dan MOFO.

default_settings.txt: Manajemen Buffer

```
1  Group.dropPolicy = [MOFODropPolicy; FIFODropPolicy]
```

3.8 Implementasi Manajemen *Buffer*

Terdapat dua manajemen *buffer* yang diimplementasikan pada penelitian ini, yaitu manajemen *buffer* FIFO dan MOFO. Kedua manajemen *buffer* ini dibuat dalam kelas baru untuk memudahkan pemanggilan ketika melakukan simulasi. Pemanggilan fungsi manajemen *buffer* ini akan dipanggil melalui *default settings*.

3.8.1 Implementasi Manajemen *Buffer* FIFO

Manajemen *buffer* FIFO adalah teknik menjatuhkan paket berdasarkan urutan masuknya mereka ke dalam *buffer*, misalnya paket pertama yang masuk dalam antrian menjadi yang pertama kali dijatuhkan.

FIFODropPolicy.java

```
1                   if (size > router.getBufferSize()) {
2                       return false;
3                   }
4
5
6                   if (freeBuffer >= size) {
7                       return true;
8                   }
9
10                  ArrayList<Message>          messages          =          new
11  ArrayList<Message>(router.getMessageCollection());
12                  Collections.sort(messages, new FIFOComparator());
13
14                  while (freeBuffer < size) {
15
16                      if (messages.size() == 0) {
17                          return false;
18                      }
19
20                      Message msg = messages.remove(0);
21
22                      if          (this.dropMsgBeingSent          ||
23  !router.isSending(msg.getId())) {
24                          router.deleteMessage(msg.getId(), true);
25                          freeBuffer += msg.getSize();
26                      }
27                  }
28
29                  return true;
30          }
```

Kode diatas merupakan kelas untuk mengimplementasikan manajemen *buffer* FIFO yang dengan nama FIFODropPolicy.java. Baris 1-3 merupakan fungsi untuk memeriksa apakah paket yang masuk melebihi kapasitas *buffer*. Baris 6-8 adalah fungsi untuk memeriksa apakah ada ruang yang cukup untuk menerima paket sebelum dilakukannya sortir *buffer*. Baris 10-12 merupakan fungsi untuk sortir paket sesuai dengan waktu paket diterima. Baris 14-30 merupakan fungsi untuk menghapus paket dari *buffer* sampai ada cukup ruang pada *buffer*.

3.8.2 Implementasi Manajemen *Buffer* MOFO

Manajemen *buffer* MOFO adalah teknik yang mencoba untuk memaksimalkan penyebaran paket melalui jaringan dengan menjatuhkan paket yang telah diteruskan jumlah waktu maksimum.

MOFODropPolicy.java

```
1                   if (size > router.getBufferSize()) {
2                       return false;
3                   }
4
5                   if (freeBuffer >= size) {
6                       return true;
7                   }
8
```

```
9               ArrayList<Message>      messages      =      new
10  ArrayList<Message>(router.getMessageCollection());
11              Collections.sort(messages, new MOFOComparator());
12
13              while (freeBuffer < size) {
14
15                      if (messages.size() == 0) {
16                              return false;
17                      }
18
19                      Message msg = messages.remove(messages.size()-
20  1);
21
22                      if      (this.dropMsgBeingSent      ||
23  !router.isSending(msg.getId())) {
24                              router.deleteMessage(msg.getId(), true);
25                              freeBuffer += msg.getSize();
26                      }
27              }
28
```

Kode diatas merupakan kelas untuk mengimplementasikan manajemen *buffer* MOFO yang dengan nama MOFODropPolicy.java. Baris 1-3 merupakan fungsi untuk melihat apakah ukuran paket terlalu besar untuk *buffer*. Baris 5-7 merupakan fungsi untuk memeriksa apakah ada ruang yang cukup untuk menerima paket sebelum menyortir *buffer*. Baris 9-11 merupakan fungsi untuk menyortir paket berdasarkan jumlah penerusan. Baris 13-28 merupakan fungsi untuk menghapus paket dari *buffer* sampai ada cukup ruang pada *buffer*.

BAB 4 PENGUJIAN DAN ANALISIS

Pengujian dan analissi dari penelitian yang sudah dilakukan dibahas pada bab ini. Pengujian yang dilakukan adalah menguji kinerja dari *routing multi-copy* dengan menambahkan manajemen FIFO dan MOFO. *Routing multi-copy* yang digunakan adalah Epidemic, Spray-and-Wait dan MaxProp. Hasil pengujian yang sudah didapatkan dianalisis agar mengetahui kinerja dari *routing multi-copy* dengan adanya implementasi manajemen *buffer* FIFO dan MOFO. Parameter pengujian yang digunakan pada penelitian ini adalah *delivery probability, overhead ratio* dan *latency average*.

4.1 Pengujian *Delivery Probability*

Delivery probability merupakan parameter yang membandingkan rasio jumlah paket yang dikirim dengan jumlah paket yang dibuat. Nilai dari *delivery probability* menjadi ukuran tingkat keberhasilan dari *routing* dalam mendistribusikan paket dari pengirim ke penerima. Pada penelitian ini pengujian *delivery probability* bertujuan untuk menguji kinerja dari *routing multi-copy* Epidemic, Spray-and-Wait dan MaxProp dengan adanya implementasi manajemen *buffer* FIFO dan MOFO.

4.1.1 Pengujian *Delivery Probability* dengan *Routing* Epidemic

Pengujian ini dilakukan untuk mengetahui hasil pengujian dengan parameter *delivery probability* dengan menggunakan *routing* Epidemic. Ukuran paket yang digunakan adalah 500kB – 1MB dan ukuran *buffer* yang digunakan adalah 5MB, 7MB, 10MB dan 12MB. Skenario pengujian dapat dilihat pada Tabel 5.1.

Tabel 4.1 Skenario pengujian *delivery probability* dengan *routing* Epidemic

No.	Skenario	Penjelasan	*Buffer*
1	Skenario 1	Simulasi dilakukan dengan menggunakan *routing* Epidemic dengan mengimplementasikan manajemen *buffer* FIFO dan MOFO.	5MB, 7MB, 10MB, 12MB

4.1.2 Pengujian *Delivery Probability* dengan *Routing Spray-and-Wait*

Pengujian ini dilakukan untuk mengetahui hasil pengujian dengan parameter *delivery probability* dengan menggunakan *routing* Spray-and-Wait. Ukuran paket yang digunakan adalah 500kB – 1MB dan ukuran buffer yang digunakan adalah 5MB, 7MB, 10MB dan 12MB. Skenario pengujian dapat dilihat pada Tabel 5.2.

Tabel 4.2 Skenario pengujian *delivery probability* dengan *routing* Spray-and-Wait

No.	Skenario	Penjelasan	*Buffer*
2	Skenario 2	Simulasi dilakukan dengan menggunakan *routing* Spray-and-Wait	5MB, 7MB, 10MB, 12MB

		dengan mengimplementasikan manajemen *buffer* FIFO dan MOFO	

4.1.3 Pengujian *Delivery Probability* dengan *Routing* MaxProp

Pengujian ini dilakukan untuk mengetahui hasil pengujian dengan parameter *delivery probability* dengan menggunakan *routing* MaxProp. Ukuran paket yang digunakan adalah 500kB – 1MB dan ukuran *buffer* yang digunakan adalah 5MB, 7MB, 10MB dan 12MB. Skenario pengujian dapat dilihat pada Tabel 5.3.

Tabel 4.3 Skenario pengujian *delivery probability* dengan *routing* MaxProp

No.	Skenario	Penjelasan	*Buffer*
3	Skenario 3	Simulasi dilakukan dengan menggunakan *routing* MaxProp dengan mengimplementasikan manajemen *buffer* FIFO dan MOFO	5MB, 7MB, 10MB, 12MB

4.1.4 Hasil Pengujian *Delivery Probability*

Setelah dilakukannya pengujian, maka didapatkan nilai hasil dari tiap tiap *routing* yang diuji. Berikut ini merupakan hasil dari pengujian dengan menggunakan parameter *delivery probability* dengan *routing Multi-Copy* dengan mengimplementasikan manajemen *buffer* FIFO dan MOFO.

Tabel 4.4 Hasil pengujian *delivery probability* menggunakan manajemen *buffer* FIFO

Nilai *Delivery Probability*			
Ukuran *Buffer*	**Epidemic**	**Spray-and-Wait**	**MaxProp**
5MB	0.1152	0.2536	0.1616
7MB	0.1391	0.3286	0.2018
10MB	0.1854	0.4219	0.2686
12MB	0.1956	0.4799	0.3142

Tabel 4.44 menunjukkan hasil dari pengujian *routing muti-copy* Epidemic, Spray-and-Wait dan MaxProp dengan menggunakan parameter uji *delivery probability*. Manajemen *buffer* yang digunakan pada pengujian ini adalah manajemen *buffer* FIFO.

- Ukuran *buffer* 5MB dengan menggunakan manajemen *buffer* FIFO menghasilkan nilai *delivery probability* pada *routing* Epidemic adalah 0.1152, *routing* Spray-and-Wait sebesar 0.2536 dan *routing* MaxProp sebesar 0.1616.
- Ukuran *buffer* 7MB dengan menggunakan manajemen *buffer* FIFO menghasilkan nilai *delivery probability* pada *routing* Epidemic sebesar 0.1391, *routing* Spray-and-Wait sebesar 0.3286 dan *routing* MaxProp sebesar 0.2018.

- Ukuran *buffer* 10MB dengan menggunakan manajemen *buffer* FIFO menghasilkan nilai *delivery probability* pada *routing* Epidemic sebesar 0.1854, *routing* Spray-and-Wait sebesar 0.4219 dan *routing* MaxProp sebesar 0.2686.
- Ukuran *buffer* 12MB dengan menggunakan manajemen *buffer* FIFO menghasilkan nilai *delivery probability* pada *routing* Epidemic sebesar 0.1956, *routing* Spray-and-Wait sebesar 0.4799 dan *routing* MaxProp sebesar 0.3142.

Tabel 4.5 Hasil pengujian *delivery probability* menggunakan manajemen *buffer* MOFO

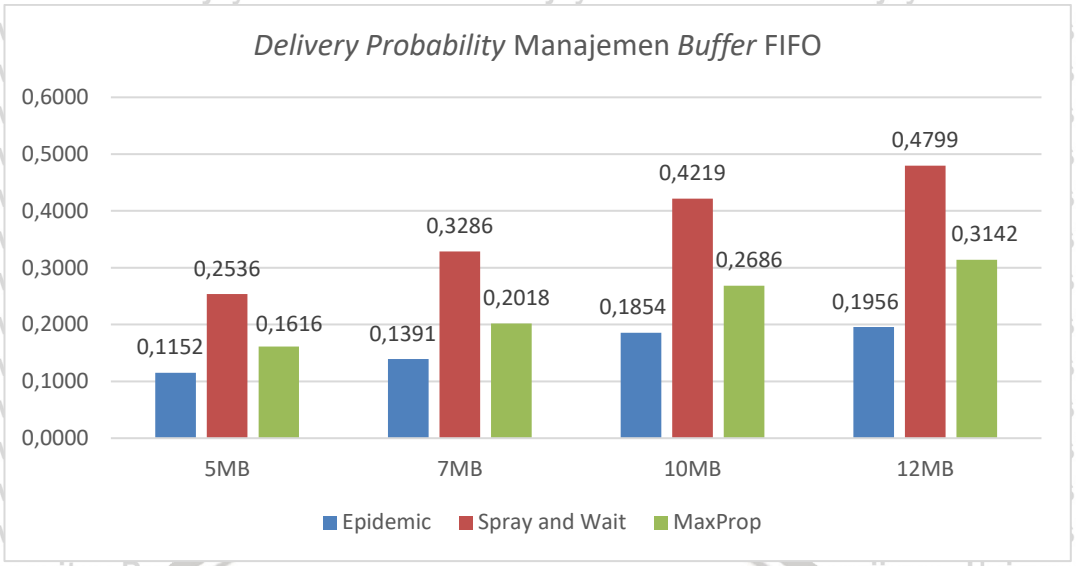
Nilai *Delivery Probability*			
Ukuran *Buffer*	**Epidemic**	**Spray-and-Wait**	**MaxProp**
5MB	0.0784	0.2338	0.0838
7MB	0.1002	0.2849	0.1043
10MB	0.1275	0.3838	0.1200
12MB	0.1404	0.4281	0.1411

Tabel 5.5 menunjukkan hasil dari pengujian *routing muti-copy* Epidemic, Spray-and-Wait dan MaxProp dengan menggunakan parameter uji *delivery probability*. Manajemen *buffer* yang digunakan pada pengujian ini adalah manajemen *buffer* MOFO.

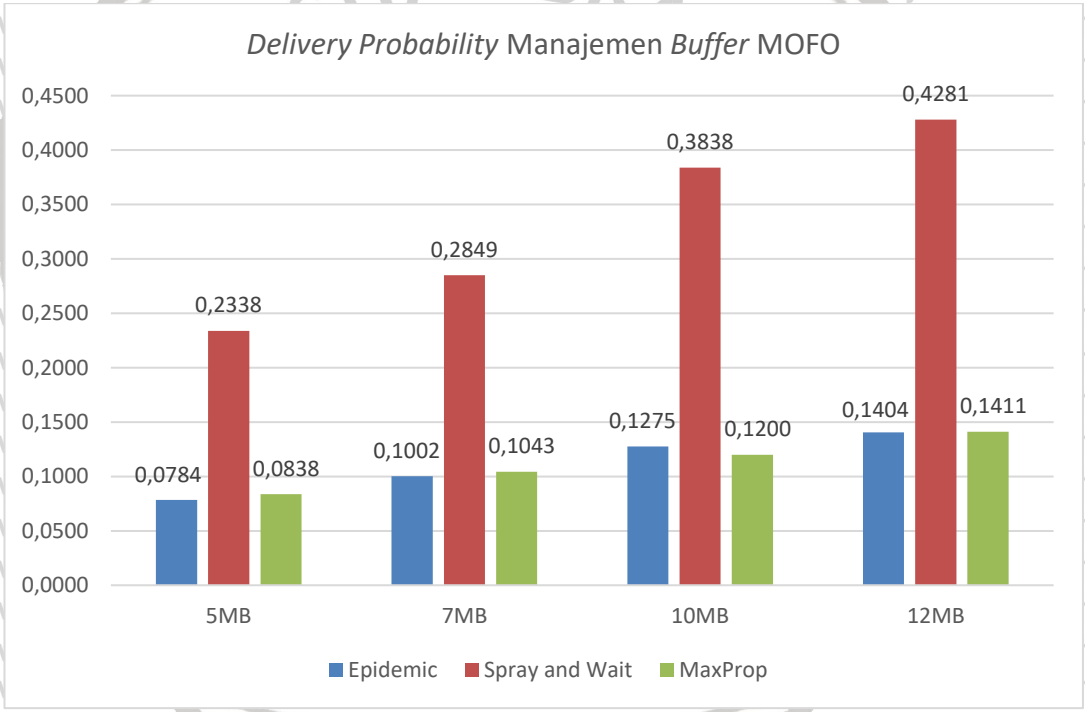
- Ukuran *buffer* 5MB dengan menggunakan manajemen *buffer* MOFO menghasilkan nilai *delivery probability* pada *routing* Epidemic sebesar 0.0784, *routing* Spray-and-Wait sebesar 0.2338 dan *routing* MaxProp sebesar 0.0838.
- Ukuran *buffer* 7MB dengan menggunakan manajemen *buffer* MOFO menghasilkan nilai *delivery probability* pada *routing* Epidemic sebesar 0.1002, *routing* Spray-and-Wait sebesar 0.2849 dan *routing* MaxProp sebesar 0.1043.
- Ukuran *buffer* 10MB dengan menggunakan manajemen *buffer* MOFO menghasilkan nilai *delivery probability* pada *routing* Epidemic sebesar 0.1275, *routing* Spray-and-Wait sebesar 0.3838 dan *routing* MaxProp sebesar 0.1200.
- Ukuran *buffer* 12MB dengan menggunakan manajemen *buffer* MOFO menghasilkan nilai *delivery probability* pada *routing* Epidemic sebesar 0.1404, *routing* Spray-and-Wait sebesar 0.4281 dan *routing* MaxProp sebesar 0.1411.

4.1.5 Analisis Hasil *Delivery Probability*

Setelah dilakukannya pengujian, selanjutnya adalah melakukan analisis hasil pengujian pada parameter *delivery probability*. Analisis dilakukan untuk melihat *routing* yang paling optimal dengan menggunakan manajemen *buffer* FIFO dan MOFO pada parameter *delivery probability.* Gambar 5.4 dan Gambar 5.5menunjukkan grafik dari hasil pengujian yang sudah dilakukan.



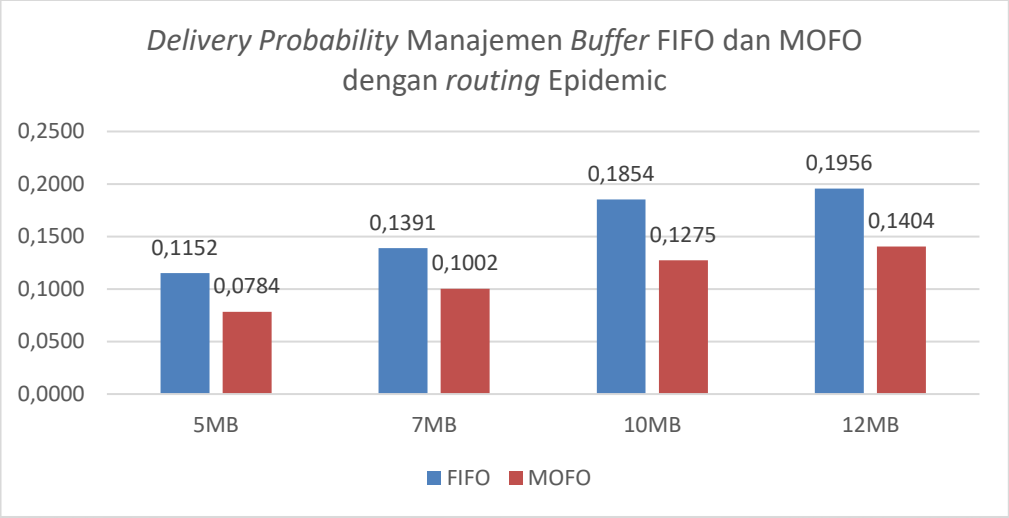
Gambar 4.1 Grafik *delivery probability* manajemen *buffer* FIFO



Gambar 4.2 Grafik *delivery probability* manajemen *buffer* MOFO

Berikut ini merupakan analisis dari hasil pengujian parameter uji *delivery probability* dengan menggunakan manajemen *buffer* FIFO dan MOFO pada *routing multi-copy*.

1. *Routing* Epidemic

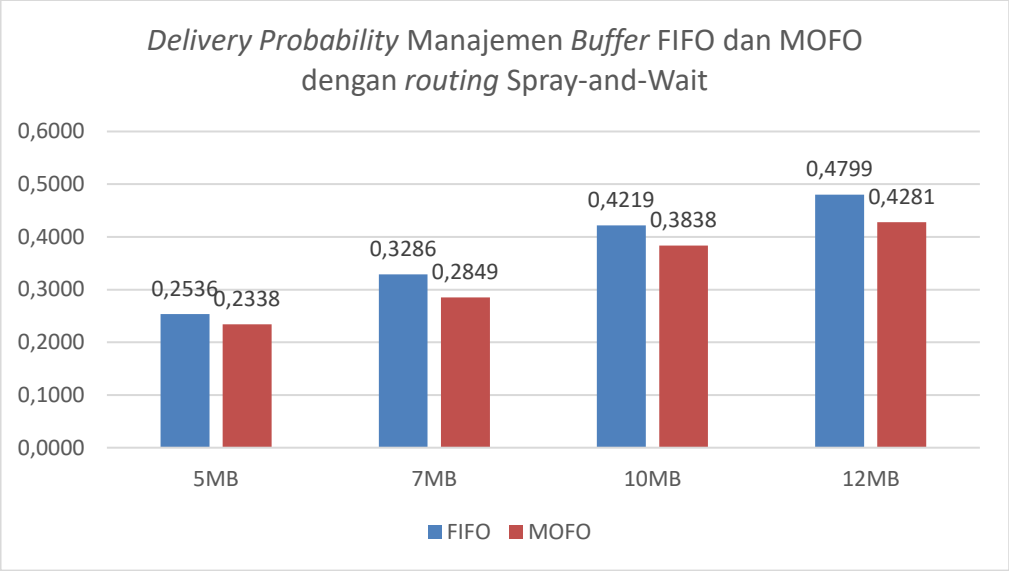


Gambar 4.3 Grafik *delivery probability* manajemen *buffer* FIFO dan MOFO dengan *routing* Epidemic

Gambar 5.6 menunjukkan grafik dari hasil pengujian dengan menggunakan parameter uji *delivery probability* dengan *routing* Epidemic. Nilai *delivery probability* menghasilkan nilai yang semakin tinggi apabila *buffer* yang disediakan juga semakin besar. Ini disebabkan karena *routing* Epidemic akan menyebarkan paket ke *node* yang dijumpainya yang tidak memiliki paket yang sama dan akan dilakukan sampai paket tersebut sampai di *node* tujuan. Semakin besar ukuran *buffer* yang disedakan untuk setiap *node* maka sekamin besar pula nilai *delivery probability* yang dihasilkan. Hal ini disebabkan karena jika ukuran *buffer* semakin besar, maka paket yang dapat ditampung juga semakin banyak yang dapat mempercepat penyebaran paket ke *node* lain. Sebaliknya, jika ukuran *buffer* yang disediakan kecil, maka paket yang dapat ditampung oleh setiap *node* menjadi terbatas.

Jika dilihat dari nilai *delivery probability,* penggunaan manajemen *buffer* FIFO lebih optimal dibandingkan dengan penggunaan manajemen *buffer* MOFO dengan *routing* Epidemic. Hal ini terjadi karena manajemen *buffer* MOFO menggunakan mekanisme menjatuhkan paket yang paling banyak diteruskan. Sedangkan *routing* Epidemic akan terus meneruskan paket ke *node* yang dijumpainya yang menyebabkan tingginya angka penerusan paket. Jika menggunakan manajemen *buffer* MOFO, paket akan lebih banyak di *drop*. Dengan menggunakan parameter uji *delivery probability* pada *routing* Epidemic, nilai yang paling tinggi pada manajemen *buffer* FIFO dengan ukuran *buffer* 12MB, yaitu sebesar 0,1956. Sedangkan nilai yang paling tinggi pada manajemen *buffer* MOFO adalah dengan menggunakan ukuran *buffer* 12MB, yaitu sebesar 0,1404. Dengan demikian penggunaan manajemen *buffer* yang optimal pada *routing* Epidemic adalah dengan manajemen *buffer* FIFO.

2. *Routing* Spray-and-Wait



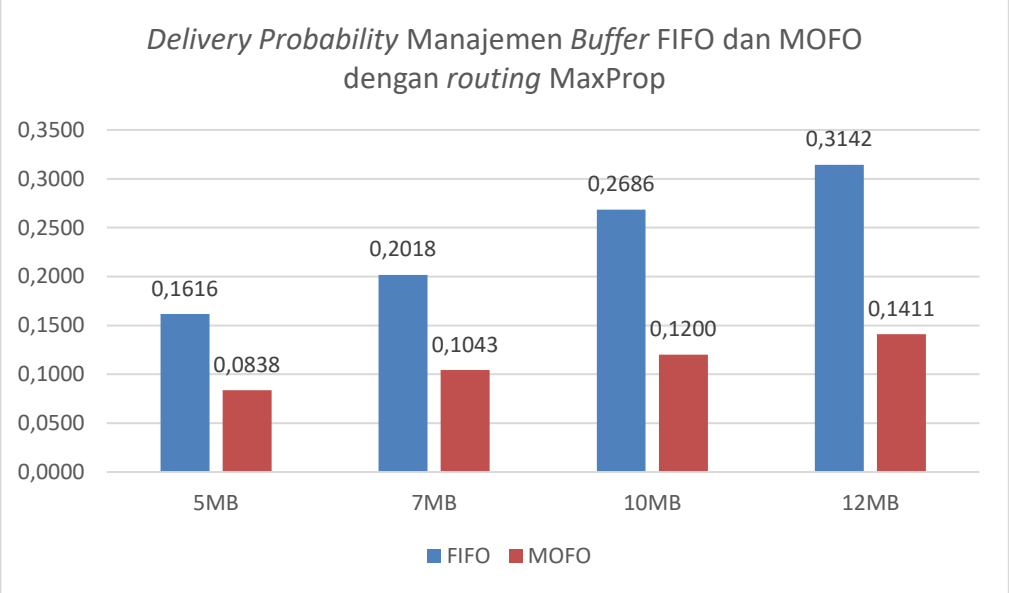
Gambar 4.4 Grafik *delivery probability* manajemen *buffer* FIFO dan MOFO dengan *routing* Spray-and-Wait

Gambar 5.7 menunjukkan grafik dari hasil pengujian dengan menggunakan parameter uji *delivery probability* dengan *routing* Spray-and-Wait. Sama seperti *routing* Epidemic, nilai *delivery probability* akan semakin tinggi jika ukuran *buffer* yang digunakan semakin besar. Hal ini terjadi karena paket yang dapat disimpan pada *buffer* juga semakin banyak untuk dapat disebar ke *node* lain atau *node* tujuan. Protokol ini mempunyai dua tahapan dalam mengirimkan paket. Tahapan pertama yaitu *Spray*. *Node* sumber akan meneruskan paket ke *node* lain yang ada pada jangkauannya. Proses penyebaran yang dilakukan sama seperti *routing* Epidemic. Namun, pada *routing* Spray-and-Wait, ada batasan penyebaran paket. Pengujian ini menggunakan batasan penyebaran sebanyak 10 salinan paket. Jika pada tahapan spray *node* tujuan tidak juga ditemukan dan jumlah penyebaran telah mencapai batasan, maka yang selanjutnya dilakukan adalah melakukan proses *Wait* sampai paket dikirim langsung ke *node* tujuan.

Gambar 5.5 dan Gambar 5.5 menunjukkan bahwa *routing* Spray-and-Wait menghasilkan nilai *delivery probability* yang paling tinggi dengan menggunakan manajemen *buffer* FIFO maupun manajemen *buffer* MOFO. Jika dilihat dari *delivery probability*, penggunaan manajemen *buffer* FIFO lebih optimal dibandingkan dengan penggunaan manajemen *buffer* MOFO pada *routing* Spray-and-Wait. Hal ini terjadi karena jumlah penerusan yang dilakukan menggunakan *routing* Spray-and-Wait dibatasi, sehingga penggunaan manajemen *buffer* MOFO akan berdampak kecil pada *routing* Spray-and-Wait. Rendahnya angka penerusan paket pada *routing* ini menyebabkan kecilnya dampak pengelolaan paket dengan menggunakan manajemen *buffer* MOFO. Dengan menggunakan *routing* Spray-and-Wait, paket lebih banyak terbuang pada proses *Spray* karena seringnya paket baru datang dari *node* lain sehingga harus menghapus paket lain karena terbatasnya *buffer* yang ada. Dengan menggunakan parameter uji *delivery*

probability pada *routing* Spray-and-Wait, nilai yang paling tinggi pada manajemen *buffer* FIFO adalah menggunakan *buffer* 12MB, yaitu sebesar 0,4799. Sedangkan nilai yang paling tinggi pada manajemen *buffer* MOFO adalah dengan menggunakan *buffer* 12MB, yaitu sebesar 0,4281. Dengan demikian pada parameter uji *delivery probability*, penggunaan manajemen *buffer* yang lebih optimal adalah dengan menggunakan manajemen FIFO.

3. *Routing* MaxProp



Gambar 4.5 Grafik *delivery probability* manajemen *buffer* FIFO dan MOFO dengan *routing* MaxProp

Gambar 5.8 menunjukkan grafik dari hasil pengujian dengan menggunakan parameter uji *delivery probability* pada *routing* MaxProp. Ukuran *buffer* yang disediakan akan berbanding lurus dengan nilai *delivery probability*. Semakin besar *buffer* yang digunakan pada *node* akan meningkatkan nilai dari *delivery probability*. Hal ini terjadi karena dengan besarnya ukuran *buffer* yang disediakan, maka paket yang dapat disimpan juga semakin banyak dibawa untuk disebarkan kembali ke *node* lain. *Routing* MaxProp menggunakan beberapa mekanisme untuk menentukan mana paket yang akan dihapus dan mana paket yang akan diterima. *Routing* MaxProp menyimpan *list cost* yang dibutuhkan untuk mengirimkan paket ke setiap *node* tujuan. Selain itu, *routing* MaxProp menerapkan *Acknowlegement* (ACK) ke setiap *node* untuk memberitahukan *node-node* lain tetang pengiriman paket.

Jika dilihat dari parameter uji *delivery* probability, penggunaan manajemen *buffer* FIFO lebih optimal dibandingkan dengan penggunaan manajemen *buffer* MOFO. Hal ini terjadi karena pada manajemen *buffer* MOFO, paket lebih banyak di-*drop* karena tingginya jumlah penerusan yang dilakukan. Banyak paket yang membutuhkan jumlah penerusan yang tinggi untuk sampai ke *node* tujuan, sehingga penggunaan manajemen *buffer* FIFO lebih optimal dalam pengiriman paket ke tujuan pada *routing* MaxProp yang tidak mementingkan jumlah

penerusan. Dengan menggunakan parameter uji *delivery probability*, nilai yang paling tinggi pada manajemen *buffer* FIFO adalah dengan ukuran *buffer* 12MB, yaitu sebesar 0,3142. Sedangkan nilai yang paling tinggi pada manajemen *buffer* MOFO adalah dengan ukuran *buffer* 12MB, yaitu sebesar 0,1411. Dengan demikian pada parameter uji *delivery probability*, penggunaan manajemen *buffer* yang lebih optimal adalah dengan menggunakan manajemen *buffer* FIFO.

4.2 Pengujian *Overhead Ratio*

Pengujian *overhead ratio* merupakan pengujian yang bertujuan untuk menghitung banyaknya redundansi paket yang dilakukan dalam pengiriman paket ke *node* tujuan. Nilai *overhead ratio* akan terus bertambah jika jumlah *node* yang digunakan semakin banyak. Nilai *overhead ratio* dikatakan optimal jika bernilai kecil. Pengujian *overhead ratio* pada penelitian ini bertujuan untuk menguji kinerja dari *routing multi-copy* Epidemic, Spray-and-Wait dan MaxProp tanpa menggunakan manajemen *buffer* dan dengan ditambahkannya manajemen *buffer* FIFO dan MOFO.

4.2.1 Pengujian *Overhead Ratio* dengan *Routing* Epidemic

Pengujian ini dilakukan untuk mengetahui hasil dari parameter *overhead ratio* dengan menggunakan *routing* Epidemic. Ukuran paket yang digunakan adalah 500kB – 1MB dan ukuran *buffer* yang digunakan adalah 5MB, 7MB, 10MB dan 12MB. Skenario pengujian dapat dilihat pada Tabel 5.6.

Tabel 4.6 Skenario pengujian *delivery probability* dengan *routing* Epidemic

No.	Skenario	Penjelasan	*Buffer*
1	Skenario 1	Simulasi dilakukan dengan menggunakan *routing* Epidemic dengan mengimplementasikan manajemen *buffer* FIFO dan MOFO	5MB, 7MB, 10MB, 12MB

4.2.2 Pengujian *Overhead Ratio* pada *Routing Spray-and-Wait*

Pengujian ini dilakukan untuk mengetahui hasil dari parameter *overhead ratio* dengan menggunakan *routing* Spray-and-Wait. Ukuran paket yang digunakan adalah 500kB – 1MB dan ukuran buffer yang digunakan adalah 5MB, 7MB, 10MB dan 12MB. Skenario pengujian dapat dilihat pada Tabel 5.7.

Tabel 4.7 Skenario pengujian *overhead ratio* pada *routing* Spray-and-Wait

No.	Skenario	Penjelasan	*Buffer*
2	Skenario 2	Simulasi dilakukan dengan menggunakan *routing* Spray-and-Wait dengan mengimplementasikan manajemen *buffer* FIFO dan MOFO	5MB, 7MB, 10MB, 12MB

4.2.3 Pengujian *Overhead Ratio* pada *Routing* MaxProp

Pengujian ini dilakukan untuk mengetahui hasil dari parameter *delivery probability* dengan menggunakan *routing* MaxProp. Ukuran paket yang digunakan adalah 500kB – 1MB dan ukuran buffer yang digunakan adalah 5MB, 7MB, 10MB dan 12MB. Skenario pengujian dapat dilihat pada Tabel 5.8.

Tabel 4.8 Skenario pengujian *overhead ratio* pada *routing* MaxProp

No.	Skenario	Penjelasan	*Buffer*
3	Skenario 3	Simulasi dilakukan dengan menggunakan *routing* MaxProp dengan mengimplementasikan manajemen *buffer* FIFO dan MOFO	5MB, 7MB, 10MB, 12MB

4.2.4 Hasil Pengujian *Overhead Ratio*

Setelah dilakukannya pengujian, maka didapatkan nilai hasil dari tiap *routing* yang diuji. Berikut ini merupakan hasil dari pengujian dengan menggunakan parameter *overhead ratio* dengan *routing multi-copy* dengan mengimplementasikan manajemen *buffer* FIFO dan MOFO.

Tabel 4.9 Hasil pengujian *overhead ratio* dengan menggunakan manajemen *buffer* FIFO

Nilai *Overhead Ratio*			
Ukuran *Buffer*	**Epidemic**	**Spray-and-Wait**	**MaxProp**
5MB	219.0592	32.5215	179.0127
7MB	266.5686	25.9232	203.9088
10MB	322.6287	20.4184	213.4137
12MB	358.5436	17.9844	206.0174

Tabel 5.9 menunjukkan hasil dari pengujian *routing muti-copy* Epidemic, Spray-and-Wait dan MaxProp dengan menggunakan parameter uji *overhead ratio*. Manajemen *buffer* yang digunakan pada pengujian ini adalah manajemen *buffer* FIFO.

- Ukuran *buffer* 5MB dengan menggunakan manajemen *buffer* FIFO menghasilkan nilai *overhead ratio* pada *routing* Epidemic sebesar 219.0592, *routing* Spray-and-Wait sebesar 32.5215 dan *routing* MaxProp sebesar 179.0127.
- Ukuran *buffer* 7MB dengan menggunakan manajemen *buffer* FIFO menghasilkan nilai *overhead ratio* pada *routing* Epidemic sebesar 266.5686, *routing* Spray-and-Wait sebesar 25.9232 dan *routing* MaxProp sebesar 203.9088.

- Ukuran *buffer* 10MB dengan menggunakan manajemen *buffer* FIFO menghasilkan nilai *overhead ratio* pada *routing* Epidemic sebesar 322.6287, *routing* Spray-and-Wait sebesar 20.4184 dan *routing* MaxProp sebesar 213.4137.
- Ukuran *buffer* 12MB dengan menggunakan manajemen *buffer* FIFO menghasilkan nilai *overhead ratio* pada *routing* Epidemic sebesar 358.5436, *routing* Spray-and-Wait sebesar 17.9844 dan *routing* MaxProp sebesar 206.0174.

Tabel 4.10 Hasil pengujian *overhead ratio* dengan menggunakan manajemen *buffer* MOFO

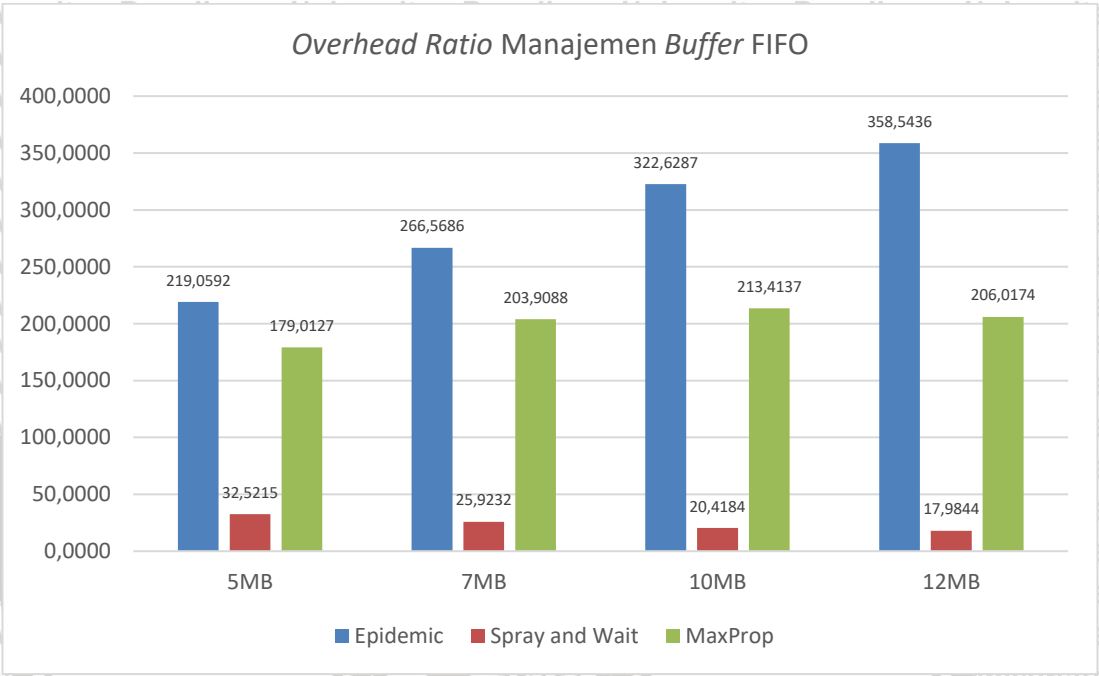
Nilai *Overhead Ratio*			
Ukuran *Buffer*	**Epidemic**	**Spray-and-Wait**	**MaxProp**
5MB	735.9652	20.9650	141.5366
7MB	582.2925	18.8589	154.5425
10MB	541.2727	15.4476	154.8295
12MB	520.9223	14.4873	143.4928

Tabel 5.10 menunjukkan hasil dari pengujian *routing muti-copy* Epidemic, Spray-and-Wait dan MaxProp dengan menggunakan parameter uji *overhead ratio*. Manajemen *buffer* yang digunakan pada pengujian ini adalah manajemen *buffer* MOFO.

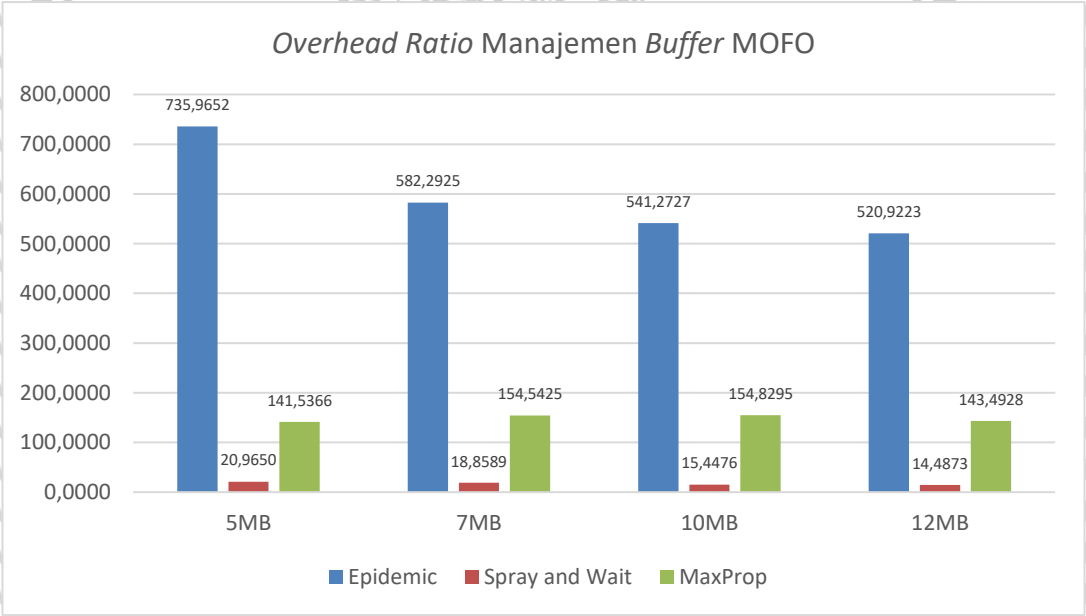
- Ukuran *buffer* 5MB dengan menggunakan manajemen *buffer* MOFO menghasilkan nilai *overhead ratio* pada *routing* Epidemic sebesar 735.9652, *routing* Spray-and-Wait sebesar 20.9650 dan *routing* MaxProp sebesar 141.5366.
- Ukuran *buffer* 7MB dengan menggunakan manajemen *buffer* MOFO menghasilkan nilai *overhead ratio* pada *routing* Epidemic sebesar 582.2925, *routing* Spray-and-Wait sebesar 18.8589 dan *routing* MaxProp sebesar 154.5425.
- Ukuran *buffer* 10MB dengan menggunakan manajemen *buffer* MOFO menghasilkan nilai *overhead ratio* pada *routing* Epidemic sebesar 541.2727, *routing* Spray-and-Wait sebesar 15.4476 dan *routing* MaxProp sebesar 154.8295.
- Ukuran *buffer* 12MB dengan menggunakan manajemen *buffer* MOFO menghasilkan nilai *overhead ratio* pada *routing* Epidemic sebesar 520.9223, *routing* Spray-and-Wait adalah 14.4873 dan *routing* MaxProp sebesar 143.4928.

4.2.5 Analisis Hasil *Overhead Ratio*

Setelah dilakukannya pengujian, selanjutnya adalah melakukan analisis hasil pengujian pada parameter *overhead ratio*. Analisis dilakukan untuk melihat *routing* yang paling optimal dengan menggunakan manajemen *buffer* FIFO dan MOFO pada parameter *overhead ratio.* Semakin rendah nilai dari *overhead ratio*, maka semakin optimal proses pengiriman pesan pada jaringan. Gambar 5.12 dan Gambar 5.13 menunjukkan grafik dari hasil pengujian yang sudah dilakukan.



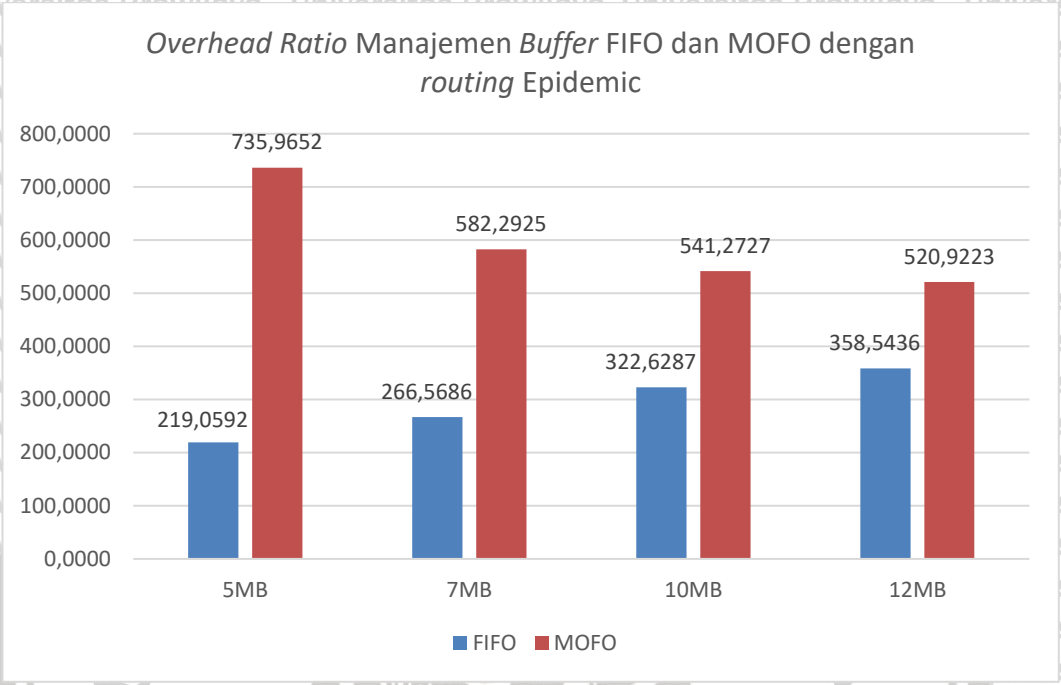
Gambar 4.6 Grafik nilai *overhead ratio* manajemen *buffer* FIFO



Gambar 4.7 Grafik nilai *overhead ratio* manajemen *buffer* MOFO

Berikut ini merupakan analisis dari hasil pengujian parameter uji *overhead ratio* dengan menggunakan manajemen *buffer* FIFO dan MOFO pada *routing multi-copy*.

1. *Routing* Epidemic



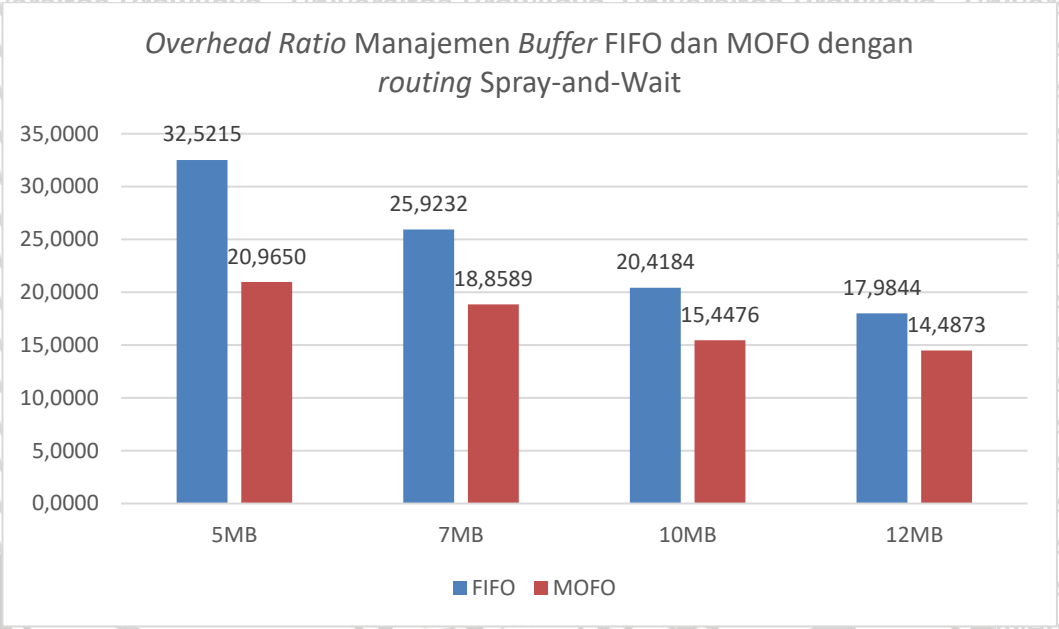
Gambar 4.8 Grafik *overhead ratio* manajemen *buffer* FIFO dan MOFO dengan *routing* Epidemic

Gambar 5.14 menunjukkan grafik dari hasil pengujian dengan menggunakan parameter uji *overhead ratio* pada *routing* Epidemic. Nilai *overhead ratio* pada manajemen *buffer* FIFO semakin meningkat seiring dengan semakin besarnya ukuran *buffer* yang disediakan. Hal ini disebabkan oleh *routing* Epidemic yang terus menyebarkan paket ke setiap *node* yang berada disekitarnya sampai paket tersebut terkirim ke *node* tujuan. Ketika *buffer* sudah penuh, manajemen *buffer* FIFO menjatuhkan paket yang pertama disimpan ke dalam *buffer* sehingga *node* dapat menyimpan paket yang baru. Sedikitnya redudansi yang dilakukan pada saat menggunakan manajemen *buffer* FIFO menyebabkan kecilnya nilai *overhead ratio* yang dihasilkan. Nilai *overhead ratio* yang paling kecil dengan manajemen *buffer* FIFO pada *routing* Epidemic ada pada ukuran *buffer* 5MB yang bernilai 219.0592.

Nilai *overhead ratio* dengan menggunakan manajemen *buffer* MOFO berbanding terbalik dengan ukuran *buffer* yang disediakan. Semakin besar ukuran *buffer* yang disediakan, maka nilai *overhead* ratio semakin kecil. Manajemen *buffer* MOFO akan menjatuhkan paket yang sudah mencapai batas maksimal untuk diteruskan ke *node* lainnya. Banyaknya paket yang harus di *drop* berdampak negative pada nilai *overhead ratio*. Penjatuhan pesan yang terlalu sedikit pada jaringan mengakibatkan nilai *overhead ratio* semakin meningkat. *Nilai overhead ratio* yang paling kecil dengan manajemen *buffer* MOFO pada *routing* Epidemic ada pada ukuran *buffer* 12MB yang bernilai 520.9223. Dengan demikian,

penggunaan manajemen *buffer* FIFO lebih optimal dibandingkan dengan manajemen *buffer* MOFO karena nilai *overhead ratio* terkecil ada pada ukuran buffer 5MB dengan menggunakan manajemen *buffer* FIFO.

2. *Routing* Spray-and-Wait



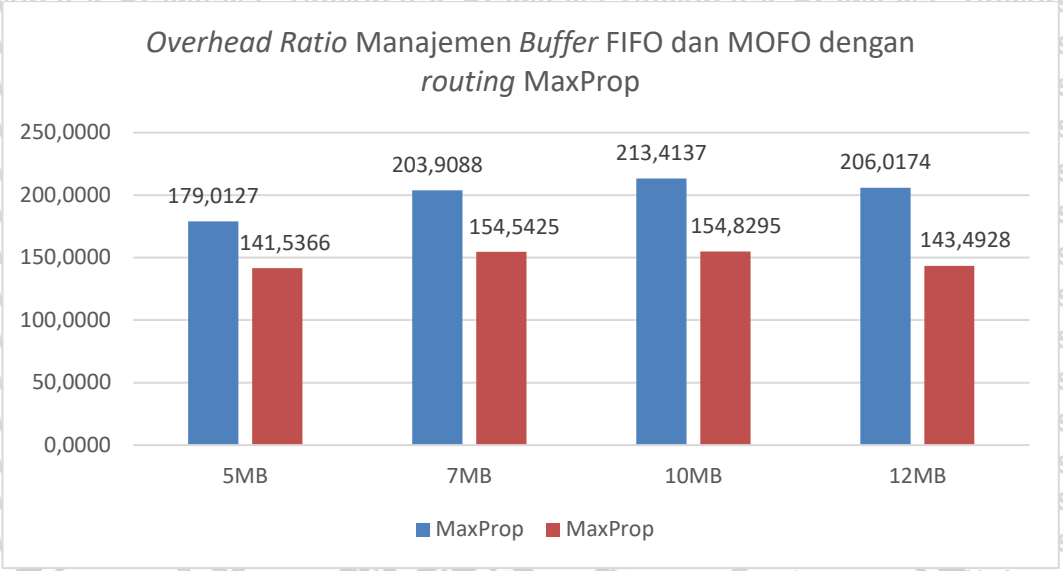
Gambar 4.9 Grafik *overhead ratio* manajemen *buffer* FIFO dan MOFO dengan *routing* Spray-and-Wait

Gambar 5.15 menunjukkan grafik dari hasil pengujian dengan menggunakan parameter uji *overhead ratio* pada *routing* Spray-and-Wait. Ukuran *buffer* yang ada pada *routing* Spray-and-Wait berbanding terbalik dengan nilai *overhead ratio*. Secara keseluruhan, nilai *overhead ratio* yang dihasilkan dengan menggunakan *routing* Spray-and-Wait lebih optimal dibandingkan dengan *routing* lainnya. Hal ini disebabkan oleh adanya fase *Spray* yang menyebabkan terbatasnya penyebaran paket pada jaringan. Pada penelitian ini, batasan salinan yang digunakan adalah 10 salinan. Jika sudah mencapai batas salinan, maka *node* tersebut akan mengirimkan paket yang ada langsung ke *node* tujuan. Dengan adanya batasan ini, transmisi yang ada pada jaringan akan berkurang. Selain itu, faktor lain yang mengurangi nilai *overhead ratio* adalah adanya manajemen *buffer*.

Jika dilihat dari nilai *overhead ratio*, penggunaan manajemen *buffer* MOFO lebih optimal dibandingkan dengan manajemen *buffer* FIFO pada *routing* Spray-and-Wait. Hal ini disebabkan oleh manajemen *buffer* MOFO yang menjatuhkan paket yang sudah diteruskan dengan jumlah maksimum. Hal ini berbanding lurus dengan mekanisme *routing* Spray-and-Wait yang membatasi jumlah salinan pada jaringan. Dengan demikian beban yang ada pada jaringan semakin berkurang. Nilai *overhead ratio* yang paling kecil dengan menggunakan manajemen *buffer* FIFO ada pada ukuran *buffer* 12MB, yaitu bernilai 17.9844. Sedangkan, untuk manajemen *buffer* MOFO juga ada pada ukuran *buffer* 12MB, yaitu bernilai 14,4873. Dari hasil

tersebut, penggunaan manajemen *buffer* MOFO lebih optimal pada *routing* Spray-and-Wait.

3. *Routing* MaxProp



Gambar 4.10 Grafik *overhead ratio* manajemen *buffer* FIFO dan MOFO dengan *routing* MaxProp

Gambar 5.16 menunjukkan grafik dari hasil pengujian dengan menggunakan parameter uji *overhead ratio* pada *routing* MaxProp. Nilai *overhead ratio* pada *routing* MaxProp dengan menggunakan manajemen *buffer* FIFO terus meningkat seiring dengan semakin besarnya ukuran *buffer* yang digunakan. Namun terjadi sedikit penurunan pada ukuran *buffer* 12MB. Manajemen *buffer* FIFO akan menjatuhkan paket yang pertama disimpan ke dalam *buffer.* Dengan demikian jika ukuran *buffer* kecil, maka paket yang sudah berada lama pada *buffer* akan langsung dijatuhkan agar paket lain dapat masuk ke dalam *buffer*. Mekanisme ini dapat mengoptimalkan nilai *overhead ratio* pada jaringan. Nilai *overhead ratio* pada *routing* Maxprop dengan menggunakan manajemen *buffer* MOFO juga naik jika buffer yang disediakan semakin. Namun, terjadi penurunan pada ukuran *buffer* 12MB. Perbedaan antara manajemen *buffer* FIFO dan MOFO pada *routing* ini adalah manajemen *buffer* FIFO lebih sering menjatuhkan paket sehingga mengingkatkan transmisi pada jaringan yang menyebabkan manajemen *buffer* MOFO lebih optimal dari pada manajemen *buffer* FIFO.

Nilai *overhead ratio* yang paling rendah dengan menggunakan manajemen *buffer* FIFO pada *routing* MaxProp ada pada ukuran *buffer* 5MB yang bernilai 179.0127. Sedangkan, nilai *overhead ratio* yang paling rendah dengan menggunakan manajemen *buffer* MOFO pada *routing* MaxProp ada pada ukuran *buffer* 5MB yang bernilai 141.5366. Dari hasil tersebut, penggunaan manajemen *buffer* MOFO lebih optimal pada *routing* MaxProp.

4.3 Pengujian *Latency Average*

Pengujian *latency average* merupakan pengujian yang bertujuan untuk menghitung nilai rata-rata waktu *delay* ketika paket dibuat dan paket diterima pada *node* tujuan. *Latency average* dikatakan semakin optimal jika nilainya semakin rendah. Pada penelitian ini pengujian *latency average* bertujuan untuk menguji kinerja dari *routing multi-copy* Epidemic, Spray-and-Wait dan MaxProp dengan mengimplementasikan manajemen *buffer* FIFO dan MOFO.

4.3.1 Pengujian *Latency Average* dengan *Routing* Epidemic

Pengujian ini dilakukan untuk mengetahui hasil pengujian dengan parameter *latency average* dengan menggunakan *routing* Epidemic. Ukuran paket yang digunakan adalah 500kB – 1MB dan ukuran *buffer* yang digunakan adalah 5MB, 7MB, 10MB dan 12MB. Skenario pengujian dapat dilihat pada Tabel 5.11.

Tabel 4.11 Skenario pengujian *latency average* dengan *routing* Epidemic

No.	Skenario	Penjelasan	*Buffer*
1	Skenario 1	Simulasi dilakukan dengan menggunakan *routing* Epidemic dengan mengimplementasikan manajemen *buffer* FIFO dan MOFO	5MB, 7MB, 10MB, 12MB

4.3.2 Pengujian *Latency Average* pada *Routing Spray-and-Wait*

Pengujian ini dilakukan untuk mengetahui hasil pengujian dengan parameter *latency average* dengan menggunakan *routing* Spray-and-Wait. Ukuran paket yang digunakan adalah 500kB – 1MB dan ukuran *buffer* yang digunakan adalah 5MB, 7MB, 10MB dan 12MB. Skenario pengujian dapat dilihat pada Tabel 5.12.

Tabel 4.12 Skenario pengujian *latency average* dengan *routing* Spray-and-Wait

No.	Skenario	Penjelasan	*Buffer*
2	Skenario 2	Simulasi dilakukan dengan menggunakan *routing* Spray-and-Wait tanpa adanya manajemen *buffer* dan dengan menambahkan manajemen *buffer* FIFO dan MOFO	5MB, 7MB, 10MB, 12MB

4.3.3 Pengujian *Latency Average* pada *Routing* MaxProp

Pengujian ini dilakukan untuk mengetahui hasil pengujian dengan parameter *latency average* dengan menggunakan *routing* MaxProp. Ukuran paket yang digunakan adalah 500kB – 1MB dan ukuran *buffer* yang digunakan adalah 5MB, 7MB, 10MB dan 12MB. Skenario pengujian dapat dilihat pada Tabel 5.13.

Tabel 4.13 Skenario pengujian *latency average* dengan *routing* MaxProp

No.	Skenario	Penjelasan	*Buffer*
3	Skenario 3	Simulasi dilakukan dengan menggunakan *routing* MaxProp dengan mengimplementasikan manajemen *buffer* FIFO dan MOFO	5MB, 7MB, 10MB, 12MB

4.3.4 Hasil Pengujian *Latency Average*

Setelah dilakukannya pengujian, maka didapatkan nilai hail dari tiap tiap *routing* yang diuji. Berikut ini merupakan hasil dari pengujian dengan menggunakan parameter *latency average* pada *routing multi-copy* dengan menggunakan manajemen *buffer* FIFO dan MOFO.

Tabel 4.14 Hasil pengujian *latency average* dengan menggunakan manajemen *buffer* FIFO

Nilai *Latency Average* (*second*)			
Ukuran *Buffer*	**Epidemic**	**Spray-and-Wait**	**MaxProp**
5MB	4771.0343s	2092.7180s	3967.4527s
7MB	5593.4309s	2280.9000s	4779.2669s
10MB	5405.8206s	3198.4384s	5389.4094s
12MB	5623.3143s	3594.8050s	5718.7310s

Tabel 5.14 menunjukkan hasil dari pengujian *routing muti-copy* Epidemic, Spray-and-Wait dan MaxProp dengan menggunakan parameter uji *latency average*. Manajemen *buffer* yang digunakan pada pengujian ini adalah manajemen *buffer* FIFO.

- Ukuran *buffer* 5MB dengan menggunakan manajemen *buffer* FIFO menghasilkan nilai *latency average* pada *routing* Epidemic sebesar 4771.0343s, *routing* Spray-and-Wait sebesar 2092.7180s dan *routing* MaxProp sebesar 3967.4527s.
- Ukuran *buffer* 7MB dengan menggunakan manajemen *buffer* FIFO menghasilkan nilai *latency average* pada *routing* Epidemic sebesar 5593.4309s, *routing* Spray-and-Wait sebesar 2280.9000s dan *routing* MaxProp sebesar 4779.2669s.
- Ukuran *buffer* 10MB dengan menggunakan manajemen *buffer* FIFO menghasilkan nilai *latency average* pada *routing* Epidemic sebesar 5405.8206s, *routing* Spray-and-Wait sebesar 3198.4384s dan *routing* MaxProp sebesar 5389.4094s.
- Ukuran *buffer* 12MB dengan menggunakan manajemen *buffer* FIFO menghasilkan nilai *latency average* pada *routing* Epidemic sebesar

5623.3143s, *routing* Spray-and-Wait sebear 3594.8050s dan *routing* MaxProp sebesar 5718.7310s.

Tabel 4.15 Hasil pengujian *latency average* dengan menggunakan manajemen *buffer* MOFO

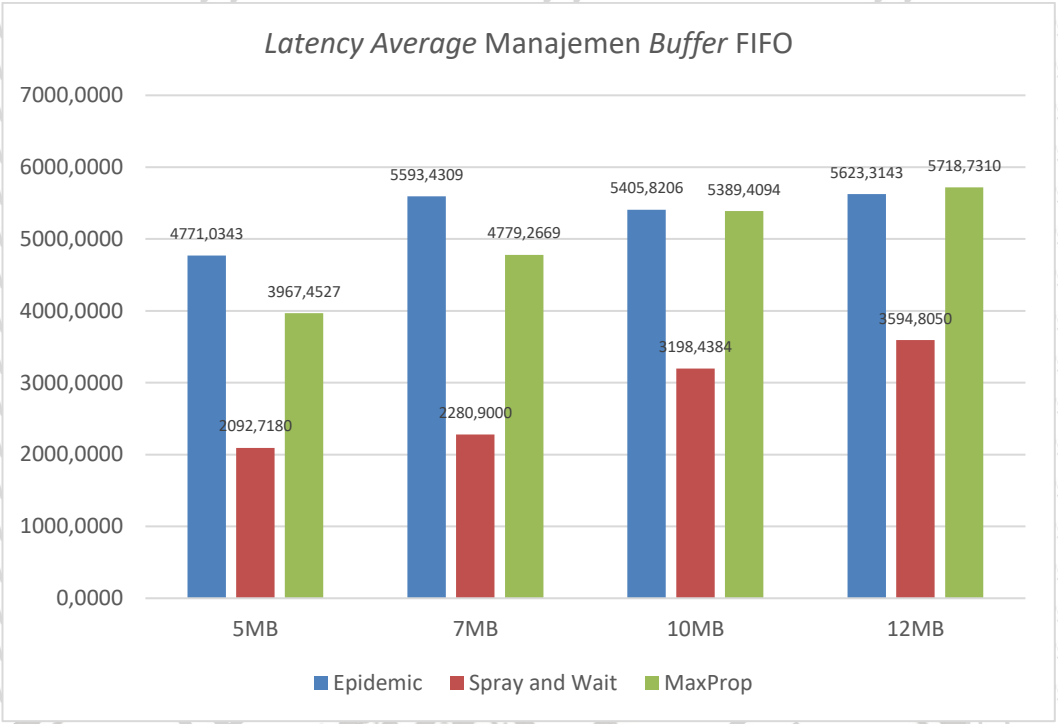
Nilai *Latency Average* (*second*)			
Ukuran *Buffer*	**Epidemic**	**Spray-and-Wait**	**MaxProp**
5MB	3638.1991s	4520.4426s	3935.5130s
7MB	3905.0721s	5039.4622s	4164.2673s
10MB	4339.2513s	5435.7863s	4462.8466s
12MB	4558.5209s	5594.9689s	4709.4097

Tabel 5.15 menunjukkan hasil dari pengujian *routing muti-copy* Epidemic, Spray-and-Wait dan MaxProp dengan menggunakan parameter uji *latency average*. Manajemen *buffer* yang digunakan pada pengujian ini adalah manajemen *buffer* MOFO.

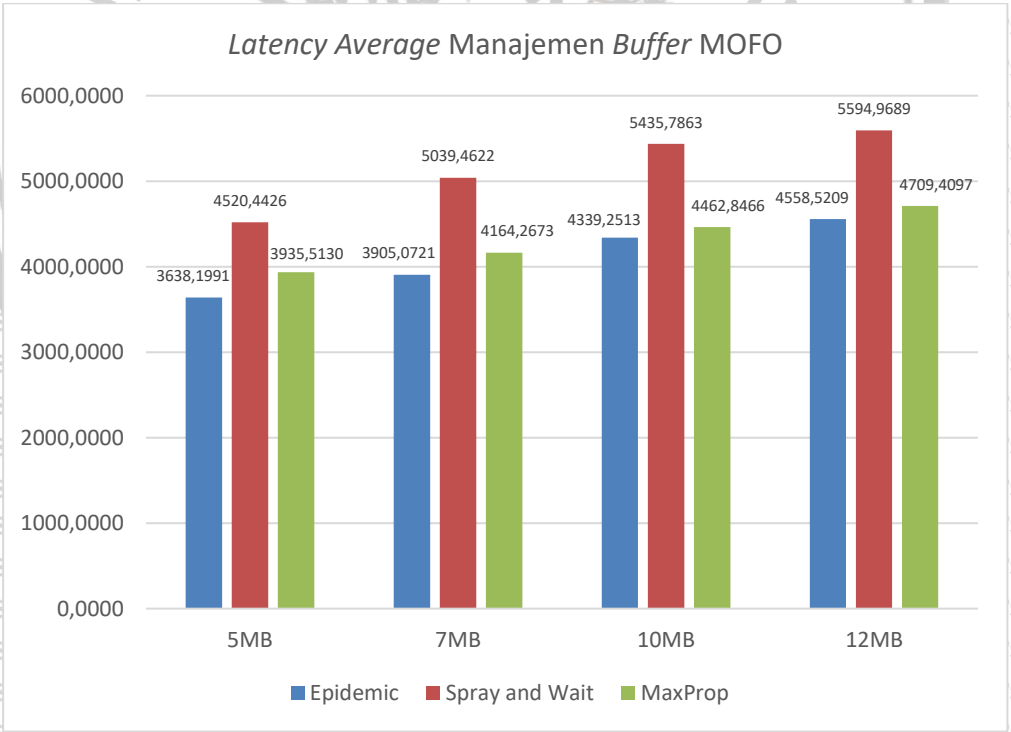
- Ukuran *buffer* 5MB dengan menggunakan manajemen *buffer* MOFO menghasilkan nilai *latency average* pada *routing* Epidemic sebesar 3638.1991s, *routing* Spray-and-Wait sebesar 4520.4426s dan *routing* Epidemic sebesar 3935.5130s.
- Ukuran *buffer* 7MB dengan menggunakan manajemen *buffer* MOFO menghasilkan nilai *latency average* pada *routing* Epidemic sebesar 3905.0721s, *routing* Spray-and-Wait sebesar 5039.4622s dan *routing* MaxProp sebesar 4164.2673s.
- Ukuran *buffer* 10MB dengan menggunakan manajemen *buffer* MOFO menghasilkan nilai *latency average* pada *routing* Epidemic sebesar 4339.2513s, *routing* Spray-and-Wait sebesar 5435.7863s dan *routing* MaxProp sebesar 4462.8466s.
- Ukuran *buffer* 12MB dengan menggunakan manajemen *buffer* MOFO menghasilkan nilai *latency average* pada *routing* Epidemic sebesar 4558.5209s, *routing* Spray-and-Wait sebesar 5594.9689s dan *routing* MaxProp sebesar 4709.4097s.

4.3.5 Analisis Hasil *Latency Average*

Setelah dilakukannya pengujian, selanjutnya adalah melakukan analisis hasil pengujian pada parameter *latency average*. Analisis dilakukan untuk melihat *routing* yang paling optimal dengan menggunakan manajemen *buffer* FIFO dan MOFO pada parameter *latency average*. Gambar 5.20 dan Gambar 5.21 menunjukkan grafik dari hasil pengujian yang sudah dilakukan.



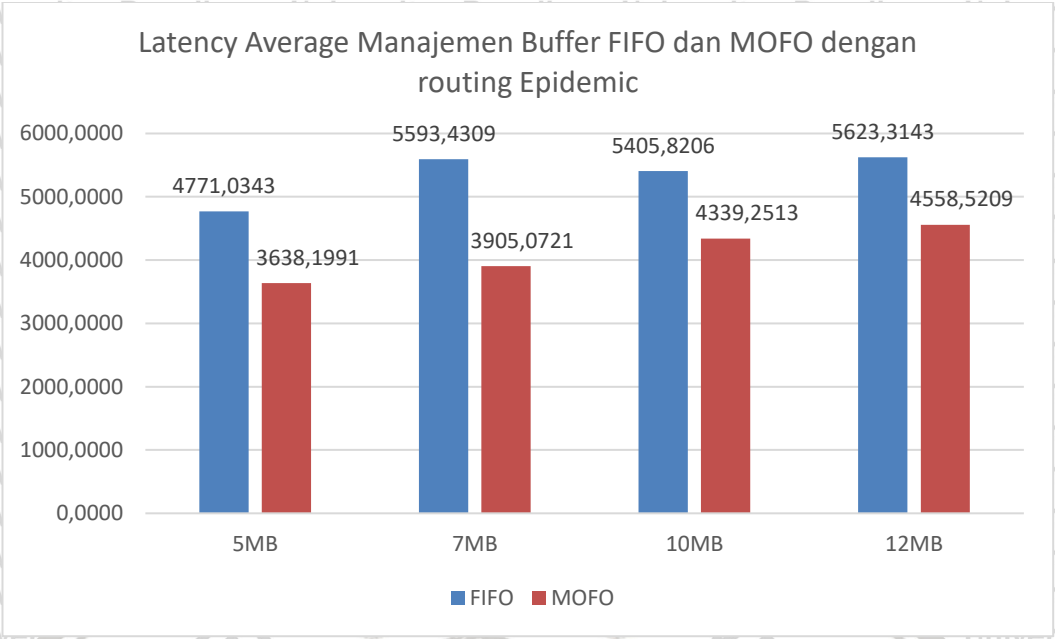
Gambar 4.11 Grafik nilai *latency average* manajemen *buffer* FIFO



Gambar 4.12 Grafik nilai *latency average* manajemen *buffer* MOFO

Berikut ini merupakan analisis dari hasil pengujian parameter uji *latency average* dengan menggunakan manajemen FIFO dan MOFO pada protocol *routing multi-copy*.

1. *Routing* Epidemic

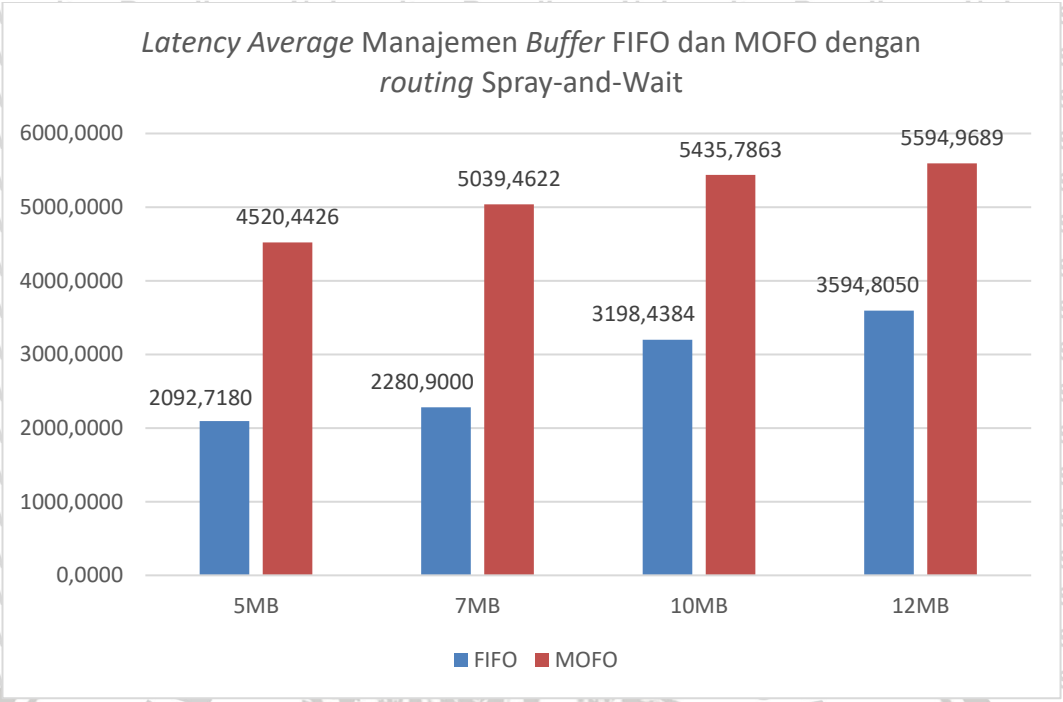


Gambar 4.13 Grafik *latency average* manajemen *buffer* FIFO dan MOFO dengan *routing* Epidemic

Gambar 5.22 menunjukkan grafik dari hasil pengujian dengan menggunakan parameter *latency average* pada protokol Epidemic. Nilai dari *latency average* terus meningkat seiring dengan semakin besarnya ukuran *buffer* yang digunakan. Namun, pada ukuran *buffer* 10MB dengan manajemen *buufer* FIFO, terjadi sedikit penurunan nilai *latency average*. Hal ini disebabkan oleh node dengan ukuran *buffer* yang lebih besar dapat menyimpan paket lebih lama di dalam. Dengan demikian paket akan berada pada *buffer* lebih lama sebelum sampai ke *node* tujuan. Adanya manajemen *buffer* FIFO dan MOFO dapat lebih mengoptimalkan *latency average* pada jaringan.

Manajemen *buffer* FIFO akan menjatuhkan paket yang pertama masuk kedalam *buffer* jika *buffer* sudah penuh. Mekanisme ini dapat mengoptimalkan *latency average* karena jika *buffer* penuh, paket yang paling lama berada pada *buffer* akan dijatuhkan terlebih dahulu. Sedangkan, manajemen *buffer* MOFO akan menjatuhkan paket yang paling sering diteruskan ke *node* lain. Dengan mekanisme ini, paket yang berulang ulang diteruskan namun tidak sampai ke *node* tujuan, akan diprioritaskan untuk dijatuhkan terlebih dahulu. Nilai *latency average* yang paling rendah menggunakan manajemen *buffer* FIFO pada *routing* Epidemic ada pada ukuran *buffer* 5MB yang bernilai 4771,0343s. Sedangkan nilai *latency average* yang paling rendah menggunakan manajemen *buffer* MOFO pada *routing* Epidemic ada pada ukuran *buffer* 5MB yang bernilai 3638.1991s. Dari hasil data yang ada, maka untuk *routing* Epidemic, manajemen *buffer* MOFO merupakan mekanisme yang lebih optimal pada *latency average*.

2. *Routing* Spray-and-Wait

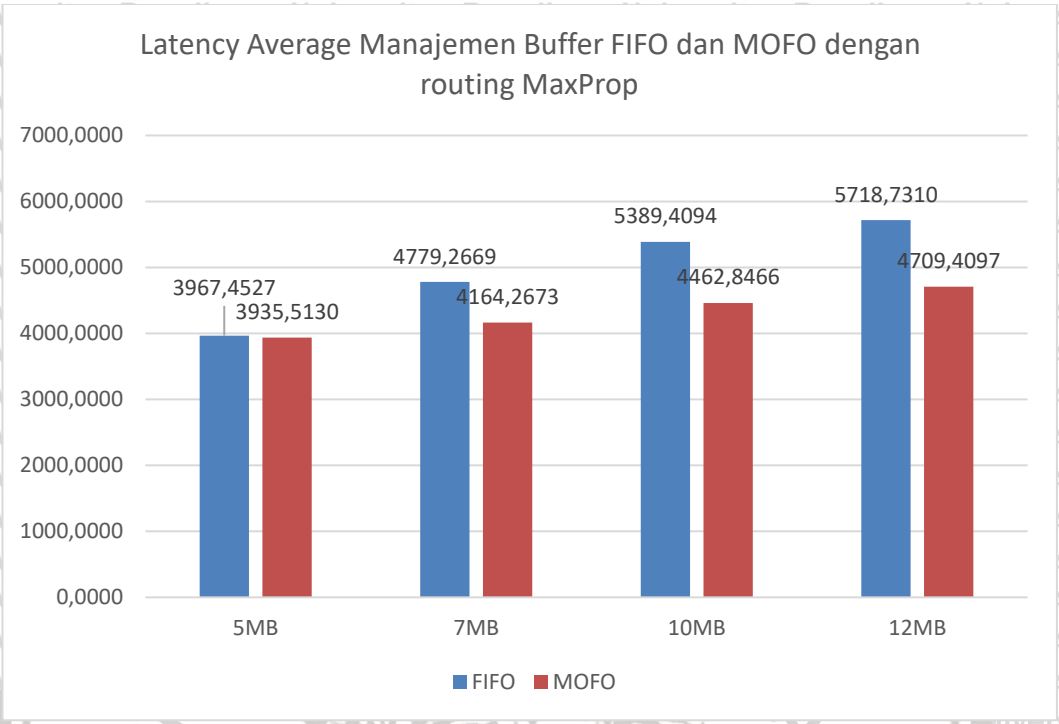


Gambar 4.14 Grafik nilai *latency average* manajemen *buffer* FIFO dan MOFO dengan *routing* Spray-and-Wait

Gambar 5.23 menunjukkan grafik dari hasil pengujian dengan menggunakan parameter *latency average* pada *routing* Spray-and-Wait. Nilai parameter *latency average* terus meningkat seiring dengan semakin besarnya ukuran *buffer* yang digunakan. Hal ini disebabkan karena *node* dengan ukuran *buffer* yang lebih besar akan menyimpan paket lebih lama di dalam *buffer*. Dengan demikian paket akan berada pada *buffer* lebih lama sebelum sampai ke *node* tujuan. Adanya pembatasan jumlah salinan pada *routing* Spray-and-Wait dapat mengurangi paket yang berlama-lama tersimpan pada *node* lain. Adanya manajemen *buffer* FIFO dan MOFO juga dapat lebih mengoptimalkan *latency average* pada jaringan. Secara keseluruhan *routing* Spray-and-Wait memiliki nilai *latency average* yang lebih optimal dibandingkan dengan *routing* lainnya.

Manajemen *buffer* FIFO akan menjatuhkan paket yang pertama masuk kedalam *buffer*. Dengan demikian, ketika *buffer* penuh, paket yang paling lama berada di dalam *buffer* akan diprioritaskan untuk dijatuhkan agar paket lain dapat masuk ke dalam *buffer*. Sedangkan, manajemen *buffer* MOFO lebih memprioritaskan menjatuhkan paket yang paling sering diteruskan jika *buffer* penuh. Nilai *latency average* yang paling rendah dengan menggunakan manajemen *buffer* FIFO pada *routing* Spray-and-Wait adalah pada ukuran *buffer* 5MB yang bernilai 2092.7180s. Sedangkan nilai *latency average* yang paling rendah dengan menggunakan manajemen *buffer* MOFO pada *routing* Spray-and-Wait adalah pada ukuran *buffer* 5MB yang bernilai 4520.4426s. Dari hasil data yang ada, maka untuk *routing* Spray-and-Wait, manajemen *buffer* FIFO merupakan mekanisme yang lebih optimal pada *latency average*.

3. *Routing* MaxProp



Gambar 4.15 Grafik nilai *latency average* manajemen *buffer* FIFO dan MOFO dengan *routing* MaxProp

Gambar 5.24 menunjukkan grafik dari hasil pengujian dengan menggunakan parameter *latency average* pada *routing* MaxProp. Nilai *latency average* pada manajemen *buffer* FIFO akan terus meningkat seiring dengan semakin besarnya ukuran *buffer* yang digunakan. Hal ini disebabkan karena ketika ukuran *buffer* besar, maka paket dapat berlama-lama terhenti pada *buffer* tersebut menunggu *buffer* penuh. Pengoptimalan *latency average* dapat dilakukan dengan menggunakan manajemen *buffer* FIFO. Mekanisme ini akan menjatuhkan paket yang paling lama berada pada *buffer* ketika *buffer* sudah penuh. Nilai *latency average* yang paling rendah dengan menggunakan manajemen *buffer* FIFO pada *routing* MaxProp adalah pada ukuran *buffer* 5MB yang bernilai 3967,4527s. Nilai ini merupakan nilai yang terkecil pada *routing* MaxProp.

Nilai *latency average* pada manajemen *buffer* MOFO juga akan terus meningkat seiring dengan semakin besarnya ukuran *buffer* yang digunakan. Dengan menggunakan manajemen *buffer* MOFO, paket yang sering diteruskan ke *node* lain akan diprioritaskan untuk dijatuhkan ketika *buffer* pada *node* sudah penuh. Dengan demikian, paket tidak akan bisa berlama-lama ada pada jaringan sehingga dapat mengoptimalkan *latency average*. Nilai *latency average* yang paling rendah dengan menggunakan manajemen *buffer* MOFO pada *routing* MaxProp adalah pada ukuran *buffer* 5MB yang bernilai 3935,5130s.

BAB 5 PENUTUP

5.1 Kesimpulan

Berikut ini merupakan kesimpulan yang diambil untuk menjawab permasalahan yang terdapat pada rumusan masalah:

1. Manajemen *buffer* FIFO dan MOFO diimplementasikan pada *routing multi-copy* Epidemic, Spray and Wait dan MaxProp dengan ukuran *buffer* 5MB, 7MB, 10MB dan 12MB. Penambahan manajemen *buffer* menghasilkan perbedaan nilai pada tiap-tiap parameter yang diuji dengan menggunakan *routing multi-copy*.
2. Nilai *delivery probability* tertinggi ada pada *routing* Spray-and-Wait dengan manajemen *buffer* FIFO dan MOFO. Nilai *delivery probability* tertinggi didapatkan sebesar 0.4799 yang terdapat pada manajemen *buffer* FIFO dengan ukuran *buffer* 12MB. Nilai *overhead ratio* terendah pada *routing* Spray-and-Wait dengan manajemen *buffer* FIFO dan MOFO. Nilai *overhead ratio* terendeh didapatkan sebesar 14.4873 yang terdapat pada manajemen *buffer* MOFO dengan ukuran *buffer* 12MB. Nilai *latency average* terendah pada *routing* Spray-and-Wait yaitu dengan menggunakan manajemen *buffer* FIFO dan *routing* Epidemic dengan manajemen *buffer* MOFO. Nilai *latency average* terendah didapatkan sebesar 2092.7180s yang terdapat pada *routing* Spray-and-Wait menggunakan manajemen *buffer* FIFO dengan ukuran *buffer* 5MB.

5.2 Saran

Saran untuk penelitian selanjutnya adalah melakukan penelitian dengan menggunakan *routing* yang lain, menambah parameter uji dan menambah manajemen *buffer* pada jaringan DTN.

DAFTAR REFERENSI

Benhamida, F.Z., Bouabdellah, A. dan Challal, Y., 2017. Using delay tolerant network for the Internet of Things: Opportunities and challenges. *2017 8th International Conference on Information and Communication Systems, ICICS 2017*, hal.252–257.

Dzurovcak, D.X. dan Yang, S., 2017. Performance Analysis of Routing Protocols in Delay Tolerant Networks. *Proceedings - 14th IEEE International Conference on Mobile Ad Hoc and Sensor Systems, MASS 2017*, hal.511–515.

Fall, K., 2003. A delay-tolerant network architecture for challenged internets. *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications - SIGCOMM '03*, [daring] hal.27. Tersedia pada: <http://portal.acm.org/citation.cfm?doid=863955.863960>.

Hutajulu, P.G., Yahya, W. dan Pramukantoro, E.S., 2018. Perbandingan Kinerja Routing Multi Copy Dan Routing First Contact Dengan Stationary Relay Node Pada Delay Tolerant Network (DTN). *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer*, 2(7), hal.2513–2522.

Keränen, A., 2019. *The Opportunistic Network Environment simulator*. [daring] Tersedia pada: <https://akeranen.github.io/the-one/> [Diakses 14 Feb 2019].

Li, F., 2008. Fairness analysis in competitive FIFO buffer management. *Conference Proceedings of the IEEE International Performance, Computing, and Communications Conference*, (July), hal.239–246.

Mcmahon, A., Farrell, S. dan College, T., 2009. Tolerant Networking. *IEEE Internet Computing*, [daring] 13, hal.82–87. Tersedia pada: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5318702>.

Putra, A.N., Vidya, Y.L. dan Tody, A.W., 2017. Performance analysis of Message Drop Control Source Relay (MDC-SR) in maxprop DTN routing. *ICCREC 2017 - 2017 International Conference on Control, Electronics, Renewable Energy, and Communications, Proceedings*, 2017-Janua, hal.217–220.

Rani, A., 2014. Performance Evaluation of Mofo Buffer Management Technique With Different Routing Protocols in Dtn Under Variable Message Buffer Size. *International Journal of Research in Engineering and Technology*, 03(03), hal.82–86.

Reza, M., Chrisdyan, W., Primananda, R. dan Siregar, R.A., 2019. Analisis Routing Multi Copy Dengan Stationary Relay Node Dan Management Buffer First In – First Out (FIFO) Pada Delay Tolerant Network (DTN). *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer*, 3(1), hal.1075–1084.

Ronaldo, D.M., 2019. *Analisis Kinerja Routing Multi - Copy Terhadap Message Drop Control Source Relay (MDC - SR) dan Stationary Relay Node Pada*

Delay Tolerant Network (DTN).

Samyal, V.K. dan Gupta, N., 2018. Comparision of MOFO Drop policy with New Efficient Buffer Comparison Management Policy. 8(Iv), hal.456–460.

Sazali, I. dan Basuki, A., 2018. Analisis Perbandingan Metode Routing Spray-and-Wait dengan Prophet untuk Daerah Terpencil. 2(4), hal.1709–1717.

Simangunsong, F.J.T.H., Nurwarsito, H. dan Siregar, R.A., 2018. Perbandingan Kinerja Pengiriman Data Skema Routing Single-Copy dan Multi-Copy pada Jaringan Delay Tolerant Network (DTN). 2(8), hal.2672–2681.

Sobin, C.C., 2016. An Efficient Buffer Management Policy for DTN. *Procedia Computer Science*, [daring] 93(September), hal.309–314. Tersedia pada: <http://dx.doi.org/10.1016/j.procs.2016.07.215>.

Spyropoulos, T., Psounis, K. dan Raghavendra, C.S., 2005. Spray-and-Wait: An Efficient Routing Scheme for Intermittently Connected Mobile Networks. *ACM SIGCOMM Workshop on Delay-Tolerant Networking*, [daring] hal.252–259. Tersedia pada: <http://portal.acm.org/citation.cfm?doid=1080139.1080143>.

Syafa, H., Primananda, R. dan Siregar, R.A., 2019. Implementasi Delay Tolerant Network (DTN) dengan Menggunakan Algoritme Hierarchical Token Bucket untuk Seleksi Node Routing Multicopy. 3(1), hal.7727–7735.

LAMPIRAN A KONFIGURASI THE ONE SIMULATOR

A.1 Konfigurasi *Default Settings*

```
#
# Default settings for the simulation
#

## Scenario settings
Scenario.name = default_scenario
Scenario.simulateConnections = true
Scenario.updateInterval = 0.1
# 43200s == 12h
Scenario.endTime = 43200

## Interface-specific settings:
# type : which interface class the interface belongs to
# For different types, the sub-parameters are interface-specific
# For SimpleBroadcastInterface, the parameters are:
# transmitSpeed : transmit speed of the interface (bytes per second)
# transmitRange : range of the interface (meters)

# "Bluetooth" interface for all nodes
btInterface.type = SimpleBroadcastInterface
# Transmit speed of 2 Mbps = 250kBps
btInterface.transmitSpeed = 250k
btInterface.transmitRange = 10

# High speed, long range, interface for group 4
highspeedInterface.type = SimpleBroadcastInterface
highspeedInterface.transmitSpeed = 10M
highspeedInterface.transmitRange = 1000

# Define 6 different node groups
Scenario.nrofHostGroups = 3

## Group-specific settings:
# groupID : Group's identifier. Used as the prefix of host names
# nrofHosts: number of hosts in the group
# movementModel: movement model of the hosts (valid class name from
movement package)
# waitTime: minimum and maximum wait times (seconds) after reaching
destination
# speed: minimum and maximum speeds (m/s) when moving on a path
# bufferSize: size of the message buffer (bytes)
# router: router used to route messages (valid class name from
routing package)
# activeTimes: Time intervals when the nodes in the group are active
(start1, end1, start2, end2, ...)
# msgTtl : TTL (minutes) of the messages created by this host group,
default=infinite
# dropPolicy: if specified a custom drop policy is used (see the
classes in the buffermanagement package).
# dropMsgBeingSent: define if the custom drop policy is allowed to
drop messages being sent.
Group.dropPolicy = FIFODropPolicy
Group.dropMsgBeingSent = false

## Group and movement model specific settings
# pois: Points Of Interest indexes and probabilities (poiIndex1,
poiProb1, poiIndex2, poiProb2, ... )
#       for ShortestPathMapBasedMovement

```

```
# okMaps : which map nodes are OK for the group (map file indexes),
default=all
#          for all MapBasedMovent models
# routeFile: route's file path - for MapRouteMovement
# routeType: route's type - for MapRouteMovement

# Common settings for all groups
Group.groupID = p
Group.movementModel = ShortestPathMapBasedMovement
Group.router = EpidemicRouter
Group.bufferSize =7M
Group.waitTime = 0, 120
# All nodes have the bluetooth interface
Group.nrofInterfaces = 1
Group.interface1 = btInterface
# Walking speeds
Group.speed = 0.138889, 0.416667
# Message TTL of 300 minutes (5 hours)
Group.msgTtl = 360

Group.nrofHosts = 100

# group1 (doulu) specific settings
Group1.groupID = d
Group1.nrofHosts = 1
Group1.movementModel = StationaryMovement
Group1.nodeLocation = 5651, 1119

# group2 (sibayak) specific settings
Group2.groupID = s
Group2.nrofHosts = 1
Group2.movementModel = StationaryMovement
Group2.nodeLocation = 588, 153

# group2 specific settings
#Group2.groupID = c
# cars can drive only on roads
#Group2.okMaps = 1
# 10-50 km/h
#Group2.speed = 2.7, 13.9

# another group of pedestrians
#Group3.groupID = p

# The Tram groups
#Group4.groupID = t
#Group4.bufferSize = 50M
#Group4.movementModel = MapRouteMovement
#Group4.routeFile = data/tram3.wkt
#Group4.routeType = 1
#Group4.waitTime = 10, 30
#Group4.speed = 7, 10
#Group4.nrofHosts = 2
#Group4.nrofInterfaces = 2
#Group4.interface1 = btInterface
#Group4.interface2 = highspeedInterface

#Group5.groupID = t
#Group5.bufferSize = 50M
#Group5.movementModel = MapRouteMovement
#Group5.routeFile = data/tram4.wkt
#Group5.routeType = 2
#Group5.waitTime = 10, 30
#Group5.speed = 7, 10
```

```
#Group5.nrofHosts = 2

#Group6.groupID = t
#Group6.bufferSize = 50M
#Group6.movementModel = MapRouteMovement
#Group6.routeFile = data/tram10.wkt
#Group6.routeType = 2
#Group6.waitTime = 10, 30
#Group6.speed = 7, 10
#Group6.nrofHosts = 2


## Message creation parameters
# How many event generators
Events.nrof = 1
Events.size = 500k,700k
Events.interval = 25,35
Events.size = 500k,700k
#Events.hosts = 0,126
Events.prefix = M
# Class of the first event generator
Events1.class = MessageEventGenerator
# (following settings are specific for the MessageEventGenerator
class)
# Creation interval in seconds (one new message every 25 to 35
seconds)
Events1.interval = 25,35
# Message sizes (500kB - 1MB)
Events1.size = 500k,700k
# range of message source/destination addresses
#Events1.hosts = 0,126
# Message ID prefix
Events1.prefix = M
Events1.hosts = 20, 101
Events1.tohosts = 0, 100


## Movement model settings
# seed for movement models' pseudo random number generator (default
= 0)
MovementModel.rngSeed = 1
# World's size for Movement Models without implicit size (width,
height; meters)
MovementModel.worldSize = 120000, 70000
# How long time to move hosts in the world before real simulation
MovementModel.warmup = 1000

## Map based movement -movement model specific settings
MapBasedMovement.nrofMapFiles = 1

MapBasedMovement.mapFile1 = data/mapping-area.wkt
#MapBasedMovement.mapFile2 = data/main_roads.wkt
#MapBasedMovement.mapFile3 = data/pedestrian_paths.wkt
#MapBasedMovement.mapFile4 = data/shops.wkt

## Reports - all report names have to be valid report classes

# how many reports to load
Report.nrofReports = 3
# length of the warm up period (simulated seconds)
Report.warmup = 0
# default directory of reports (can be overridden per Report with
output setting)
Report.reportDir = reports/skripsi
# Report classes to load
```

```
Report.report1 = MessageStatsReport
Report.report2 = DeliveredMessagesReport
Report.report3 = DistanceDelayReport
#Report.report2 = ContactTimesReport

## Default settings for some routers settings
ProphetRouter.secondsInTimeUnit = 30
SprayAndWaitRouter.nrofCopies = 10
SprayAndWaitRouter.binaryMode = true

## Optimization settings -- these affect the speed of the simulation
## see World class for details.
Optimization.cellSizeMult = 5
Optimization.randomizeUpdateOrder = true


## GUI settings

# GUI underlay image settings
#GUI.UnderlayImage.fileName = data/helsinki_underlay.png
# Image offset in pixels (x, y)
#GUI.UnderlayImage.offset = 64, 20
# Scaling factor for the image
#GUI.UnderlayImage.scale = 4.75
# Image rotation (radians)
#GUI.UnderlayImage.rotate = -0.015

# how many events to show in the log panel (default = 30)
#GUI.EventLogPanel.nrofEvents = 100
# Regular Expression log filter (see Pattern-class from the Java
API for RE-matching details)
#GUI.EventLogPanel.REfilter = .*p[1-9]<->p[1-9]$
```

A.2 *Pseudocode* Manajemen Buffer FIFO

```
package buffermanagement;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;

import core.Message;
import core.Settings;
import routing.ActiveRouter;

public class FIFODropPolicy extends DropPolicy{

        public FIFODropPolicy(Settings s) {
                super(s);
        }

        @Override
        public boolean makeRoomForMessage(ActiveRouter router,
Message incomingMessage) {

                int size = incomingMessage == null ? 0 :
incomingMessage.getSize();

                if (size > router.getBufferSize()) {
                        return false;
                }
                long freeBuffer = router.getFreeBufferSize();
```

```
29
30              if (freeBuffer >= size) {
31                  return true;
32              }
33
34              // Sort the messages by receive time
35              ArrayList<Message>      messages      =      new
36  ArrayList<Message>(router.getMessageCollection());
37              Collections.sort(messages, new FIFOComparator());
38
39              while (freeBuffer < size) {
40
41                  if (messages.size() == 0) {
42                  }
43                  Message msg = messages.remove(0);
44
45                  if          (this.dropMsgBeingSent          ||
46  !router.isSending(msg.getId())) {
47                      router.deleteMessage(msg.getId(), true);
48                      freeBuffer += msg.getSize();
49                  }
50              }
51
52              return true;
53          }
54
55          private class FIFOComparator implements Comparator<Message> {
56
57              @Override
58              public int compare(Message m1, Message m2) {
59                  return
60  ((Double)m1.getReceiveTime()).compareTo(m2.getReceiveTime());
61              }
62
63          }
64
65  }
```

A.3 *Pseudocode* Manajemen Buffer MOFO

```
1   package buffermanagement;
2
3   import java.util.ArrayList;
4   import java.util.Collections;
5   import java.util.Comparator;
6   import core.Message;
7   import core.Settings;
8   import routing.ActiveRouter;
9
10  public class MOFODropPolicy extends DropPolicy {
11
12
13          public MOFODropPolicy(Settings s) {
14              super(s);
15          }
16
17          @Override
18          public boolean makeRoomForMessage(ActiveRouter router, Message
19  incomingMessage) {
20
21              int    size    =    incomingMessage    ==    null    ?    0    :
22  incomingMessage.getSize();
23
24              if (size > router.getBufferSize()) {
25                  return false;
```

```
26                  }
27
28                  long freeBuffer = router.getFreeBufferSize();
29
30                  if (freeBuffer >= size) {
31                      return true;
32                  }
33
34                  ArrayList<Message>          messages          =          new
35  ArrayList<Message>(router.getMessageCollection());
36                  Collections.sort(messages, new MOFOComparator());
37
38                  while (freeBuffer < size) {
39
40                      if (messages.size() == 0) {
41                          return false;
42                      }
43
44                      Message  msg  =  messages.remove(messages.size()-
45  1);
46
47                      if          (this.dropMsgBeingSent                  ||
48  !router.isSending(msg.getId())) {
49                          router.deleteMessage(msg.getId(), true);
50                          freeBuffer += msg.getSize();
51                      }
52                  }
53
54                  return true;
55
56          }
57
58          private class MOFOComparator implements Comparator<Message> {
59
60                  @Override
61                  public int compare(Message msg0, Message msg1) {
62                      return
63  ((Integer)msg0.getForwardCount()).compareTo(msg1.getForwardCount());
64                  }
65
66          }
67
68  }
```