TESIS - KI092361

PEMILIHAN *BACKUP NODE* UNTUK REDUKSI *FEEDBACK IMPLOSION* PADA *RELIABLE MULTICAST PROTOCOL* DENGAN ESTIMASI *BANDWIDTH AVAILABILITY* DAN *PACKET LOSS*

MAHENDRA DATA
NRP. 5112201043

DOSEN PEMBIMBING
Waskitho Wibisono, S.Kom., M.Eng., PhD
NIP. 19741022 200003 1 001

PROGRAM MAGISTER
BIDANG KEAHLIAN KOMPUTASI BERBASIS JARINGAN
JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2014

TESIS KI092361

BACKUP NODE SELECTION FOR FEEDBACK IMPLOSION REDUCTION IN RELIABLE MULTICAST PROTOCOL USING BANDWIDTH AVAILABILITY AND PACKET LOSS

MAHENDRA DATA
NRP. 5112201043

SUPERVISOR
Waskitho Wibisono, S.Kom., M.Eng., PhD
NIP. 19741022 200003 1 001

MAGISTER PROGRAMME
INFORMATICS ENGINEERING DEPARTMENT
FACULTY OF INFORMATION TECHNOLOGY
SEPULUH NOPEMBER INSTITUTE OF TECHNOLOGY
SURABAYA
2014

PEMILIHAN *BACKUP NODE* UNTUK REDUKSI *FEEDBACK IMPLOSION* PADA *RELIABLE MULTICAST PROTOCOL* DENGAN ESTIMASI *BANDWIDTH AVAILABILITY* DAN *PACKET LOSS*

Nama Mahasiswa : Mahendra Data
NRP : 5112 201 043
Pembimbing : Waskitho Wibisono, S.Kom., M.Eng., PhD

ABSTRAK

Saat ini pengiriman data secara massal di internet semakin populer. Sebagai contoh adalah pendistribusian *update* program atau distribusi materi pembelajaran jarak jauh. Salah satu masalah yang ditimbulkannya adalah pemborosan *bandwidth*. *Multicast protocol* dapat dimanfaatkan untuk menyelesaikan masalah ini. Namun *multicast protocol* tidak *reliable*, sehingga mendorong banyak peneliti untuk merancang *reliable multicast protocol*. Permasalahan utama dalam merancang *reliable multicast protocol* adalah besarnya peluang *feedback implosion* ketika terjadi *packet loss*.

Dalam penelitian ini, penulis mengemukakan sebuah metode untuk meminimalisir *negative-acknowledgment* (NAK) agar dapat menekan *feedback implosion*. Pada metode ini, tiap *node* akan memetakan *node* lain yang berdasarkan tingkat reliabilitasnya dan menjadikannya sebagai *backup node*. Reliabilitas ini diukur berdasarkan *bandwidth availability* dan *packet loss*. *Node* yang mengalami *packet loss*, akan mengirimkan NAK ke *backup node* sesuai dengan urutan reliabilitasnya. *Backup node* akan membalas dengan memberikan paket perbaikan. Tujuannya agar tidak terjadi *bottle neck* di *node* pengirim, sehingga dapat menurunkan pengiriman NAK berulang kali dari *node* yang mengalami *packet loss*.

Dari hasil percobaan, terbukti metode ini dapat mengurangi peluang terjadinya *bottle neck* pada *node* pengirim sehingga dapat meningkatkan peluang terbalasnya NAK yang dikirim dan menurunkan jumlah pengiriman ulang NAK. Namun metode ini memerlukan *bandwidth* tambahan untuk proses pencarian *backup node*.

Kata kunci : *reliable multicast protocol, feedback implosion*, NAK, *backup node*.

BACKUP NODE SELECTION FOR FEEDBACK IMPLOSION REDUCTION IN RELIABLE MULTICAST PROTOCOL USING BANDWIDTH AVAILABILITY AND PACKET LOSS ESTIMATION

Student Name : Mahendra Data
NRP : 5112 201 043
Supervisor : Waskitho Wibisono, S.Kom., M.Eng., PhD

ABSTRACT

Nowadays, massive data transmission on the Internet, like the distribution of software updates, has been increasing. One of the impacts of this phenomenon is bandwidth wasting. Multicast protocols can be used to overcome this problem. But multicast protocol itself is not reliable. It‟s encouraging many researchers to design a reliable multicast protocol. The main problem in designing a reliable multicast protocol is the high possibility of feedback implosion due to packet loss.

This research is aimed to reduce Negative- Acknowledgment (NAK) in order to suppress feedback implosion. In this method, node will map other nodes based on the reliability of each node and make it as a backup node. This reliability is measured based on bandwidth availability and packet loss of each backup node. Nodes who experiencing packet loss, will send NAK to the backup node with highest reliability. Then backup node will reply by sending repair package. The goal of this method is to avoiding bottle neck at the sender node, which can avoid repetitive sending of NAK from nodes that experiencing packet loss.

Experimental results proved that this method can reduce the probability of bottle neck condition at the sender node and increase the successfull rate of NAK reply and decrease the chances of NAK retransmission. But this method need an additional bandwidth for backup node discovery process.

Keyword : *reliable multicast protocol, feedback implosion*, NAK, *backup node*.

LEMBAR PENGESAHAN
TESIS

Tesis ini disusun untuk memenuhi salah satu syarat memperoleh gelar
Magister Komputer (M.Kom)
di
Institut Teknologi Sepuluh November

Oleh:
MAHENDRA DATA
NRP. 5112201043

Tanggal Ujian : 2 Juli 2014
Periode Wisuda : September 2014

Disetujui oleh:

1. Waskitho Wibisono, S.Kom, M.Eng, Ph.D ..... (Pembimbing I)
   NIP. 19741022 200003 1 001
2. Tohari Ahmad, S.Kom., MIT, Ph.D ..... (Penguji)
   NIP. 19750525 200312 1 002
3. Hudan Studiawan, S.Kom., M.Kom. ..... (Penguji)
   NIP. 19870511 201212 1 003
4. Baskoro Adi Pratomo, S.Kom, M.Kom. ..... (Penguji)
   NIP. -

Direktur Program Pascasarjana,

Prof. Dr. Ir. Adi Soeprijanto, MT
NIP. 19640405 199002 1 001

KATA PENGANTAR

Segala puji bagi Allah ﷻ yang telah memberikan rahmatnya sehingga penulis dapat menyelesaikan tesis yang berjudul "Pemilihan *Backup node* untuk Reduksi *Feedback Implosion* pada *Reliable Multicast Protocol* dengan Estimasi *Bandwidth Availability* dan *Packet loss*". Tesis ini disusun untuk memenuhi sebagian persyaratan guna memperoleh gelar sarjana Magister Komputer di Institut Teknologi Sepuluh Nopember.

Penulis mengucapkan terima kasih dan penghargaan yang sebesar-besarnya kepada berbagai pihak yang telah memberikan bantuan dan dukungan sehingga tesis ini dapat terselesaikan dengan baik, khususnya kepada:

1. Bapak Waskitho Wibisono, S.Kom, M.Eng, Ph.D, selaku Ketua Program Studi Pascasarjana Teknik Informatika Institut Teknologi Sepuluh Nopember sekaligus sebagai pembimbing tesis yang telah membimbing penulis hingga tesis ini dapat terselesaikan dengan baik.
2. Orang tua penulis, yang telah memberikan dukungan mental dan spiritual yang sangat berharga selama penulis menuntut ilmu di Institut Teknologi Sepuluh Nopember ini.
3. Bapak Dr. Ir. Harry Soekotjo Dachlan, M.Sc, selaku Ketua Unit Teknologi Informasi dan Komunikasi Universitas Brawijaya, dan Bapak R. Arief Setyawan ST. MT., selaku Kepala Pengkajian dan Pengembangan Teknologi Informasi Universitas Brawijaya, yang telah menyediakan berbagai fasilitas penelitian yang penulis gunakan untuk melakukan percobaan penelitian tesis.
4. Bapak Ibu Dosen pengajar di Program Pascasarjana Teknik Informatika Institut Teknologi Sepuluh Nopember, yang telah mengajarkan banyak ilmunya.
5. Bapak Achmad Basuki, ST., MMG., Ph.D, guru sekaligus rekan kerja penulis yang telah banyak memberikan kritik, saran dan bantuan yang sangat membangun.
6. Ibu Dany Primanita Kartikasari, ST, guru sekaligus teman seperjuangan selama menempuh Pascasarjana Teknik Informatika ITS yang telah bersedia melakukan pengecekan penulisan dan tata bahasa dalam buku tesis ini.

7. Teman-teman Pascasarjana Teknik Informatika ITS, yang telah membantu dan berbagi informasi dengan penulis. Terutama Yuita Arum Sari dan Widhy Hayuhardhika Nugraha Putra yang sering penulis repotkan dengan berbagai permintaan bantuan selama di Surabaya. Serta tak lupa Bapak Johan Ericka Wahyu Prakasa, teman seperjuangan naik kereta api pulang pergi Malang Surabaya.
8. Segenap karyawan Tata Usaha Teknik Informatika yang telah membantu mengurus syarat-syarat administratif penyelesaian tesis.
9. Dita Oktaria, Gilang Ramadhan, Hariz Farisi, Arini Indah Permatasari dan Nurya Aghnia Farda, yang ruang kantornya penulis jadikan sebagai tempat penelitian tesis.
10. Sigit Adinugroho dan Danu Budi, rekan kerja penulis yang mendapat limpahan tugas dari Unit TIK selama penulis ditugaskan untuk melanjutkan sekolah, serta tak lupa Mas Rakhmadhany Primananda, yang selalu berlomba-lomba bersama penulis bersaing dalam menyelesaikan tesis. Penulis juga menyampaikan terima kasih kepada berbagai pihak yang tidak dapat penulis tulis satu-persatu yang telah membantu dalam telah penyelesaian tesis ini.

Apa yang penulis kerjakan dalam tesis ini hanya menyelesaikan sebagian permasalahan kecil yang ada, maka dari itu penulis mengharpkan penelitian ini dapat dikembangkan dan diperbaiki pada penelitian-penelitian selanjutnya sehinga hasilnya dapat bermanfaat bagi masyarakat.

Malang, 2 Juli 2014

Mahendra Data

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB 1
PENDAHULUAN

1.1 Latar Belakang

Pengiriman data secara massal, seperti pendistribusian *update* atau *patch* suatu program memiliki banyak tantangan. Salah satunya adalah besarnya kebutuhan *bandwidth* karena *update* atau *patch* program tersebut harus dikirimkan ke banyak *client*, padahal data yang dikirimkan sama (Gemmell dkk., 2000). *Multicast protocol* berpotensi untuk digunakan pada pendistribusian data semacam ini karena dapat menghemat *bandwidth* dan sumber daya jaringan. *Multicast protocol* dapat menghemat *bandwidth* karena data dikirimkan ke sebuah *multicast group*, tidak eksklusif untuk tiap *receiver* seperti halnya yang terjadi dalam *unicast*, sehingga penggunaan sumber daya pada *node* pengirim dapat dikurangi secara signifikan (Chawathe dkk., 1998). *Node* pengirim dalam sebuah jaringan *multicast* hanya perlu melakukan satu kali pengiriman untuk seluruh *node* penerima yang tergabung dalam *multicast group*.

Namun menggunakan *multicast protocol* bukan berarti bisa terbebas dari masalah. Pada awalnya *multicast protocol* awalnya didesain untuk digunakan pada pengiriman data yang tidak memprioritaskan reliabilitas seperti *audio video streaming*, dimana pada *audio video streaming* kehilangan beberapa paket data bukan merupakan masalah yang besar. Namun pada pengiriman data yang memerlukan reliabilitas tinggi, kehilangan satu paket saja merupakan masalah besar (Koutsonikolas dkk., 2012). Misalnya ketika kita mengirimkan sebuah *file*. *File* harus diterima oleh *node* penerima sesuai dengan *file* aslinya. Kehilangan satu paket dapat menyebabkan *file* menjadi tidak sama atau bahkan tidak terbaca.

Multicast protocol dirancang menggunakan UDP sebagai *transport protocol*, hal ini menyebabkan *multicast protocol* bersifat tidak *reliable* dan tidak menjamin paket data akan sampai ke tujuan dengan urutan yang sama dengan urutan pengiriman (Postel, 1980). Desain *multicast protocol* menyerahkan pengelolaan reliabilitas pengiriman ke *application layer*. Sejumlah penelitian telah dilakukan dengan menggunakan berbagai metode untuk mengatasi permassalahan ini, dengan tujuan agar *multicast* menjadi sebuah protokol pengiriman data yang *reliable*. Secara garis besar pendekatan yang

dilakukan dalam penelitian tersebut dapat dikelompokkan menjadi lima, yaitu ACK-*based protocols*, NAK-*based protocols*, *Ring-based protocols*, *Tree-based protocols* dan *FEC-based protocols* (Lane dkk., 2001) (Wu dkk., 2005).

ACK-based protocols dan *NAK-based protocols* menggunakan paket *acknowledgement* baik itu positif maupun negatif. Kelemahan dari penggunaan *acknowledgement* adalah peluang terjadinya *feedback implosion*, yaitu kondisi dimana *node* pengirim akan dibanjiri oleh *acknowledgement* dari *node* penerima (Lane dkk., 2001). *Feedback implosion* pada *ACK-based protocols* terjadi saat *node* penerima jumlahnya sangat banyak dan tiap-tiap *node* tersebut harus mengirimkan ACK ke *node* pengirim. Maka *node* pengirim akan mendapat dan harus memproses ACK sebanyak user penerima tersebut. Sedangkan *feedback implosion* pada *NAK-based protocols* terjadi saat terjadi *packet loss* yang besar pada banyak *node* penerima. Pada saat kondisi seperti itu maka *node* pengirim juga akan mengalami *feedback implosion* akibat banyaknya NAK yang datang dari *node* penerima (Lane dkk., 2001).

Ring-based protocols dan *Tree-based protocols* melakukan pendekatan struktur jaringan dalam menyelesaikan permasalahan ini. Pada *ring-based protocols* hanya *node-node* tertentu saja yang diharuskan membalas ACK atau NAK ke *node* pengirim (Lane dkk., 2001). Sedangkan pada *tree-based protocols*, *node* dihubungkan sedemikian rupa sehingga membentuk *tree*, dimana *node* pengirim menjadi *root* dari *tree* tersebut. Tiap *node leaf* pada *tree* tersebut akan mengirimkan paket ACK atau NAK ke *parent node*-nya. *Parent node* tersebut akan merangkum ACK atau NAK dari *leaf node* dibawahnya dan mengirimkan satu paket hasil rangkuman tersebut ke *parent node*-nya. Demikian seterusnya hingga ACK atau NAK tersebut sampai pada *node* pengirim. Kedua pendekatan ini berhasil menekan peluang terjadinya *feedback implosion* namun implementasinya cukup rumit (Lane dkk., 2001). Salah satu contoh penerapan *tree-based protocols* ini adalah *Reliable* Data Center *Multicast* (RDCM) yang dikembangkan oleh Dan Li dkk. (Li dkk., 2011).

Forward Error Correction (FEC) based protocols melakukan pendekatan yang berbeda. FEC bekerja dengan melakukan *encoding* dan menambahkan *parity* pada sejumlah paket yang dikirimkannya. Bila terjadi *packet loss* maka paket yang hilang tersebut dapat diganti dengan *parity* tanpa harus meminta kembali ke *node* pengirim. Tujuan dari *encoding* dan penambahan *parity* ini adalah untuk menekan atau bahkan

menghapuskan pengiriman *acknowledgement* sehingga *feedback implosion* dapat dihindari. Metode FEC inilah yang digunakan pada *Fcast Multicast File Distribution* (Gemmell dkk., 2000).

Fcast sukses diterapkan untuk mengatasi masalah *download* dalam jumlah yang sangat besar atau yang biasa dijuluki sebagai "*midnight madness*". *Midnight madness* adalah kondisi *download traffic* yang sangat besar ketika sebuah *software* baru saja diluncurkan saat malam hari. Penyebab besarnya *traffic* tersebut antara lain karena kepopuleran *software* tersebut, *software* tersebut merupakan *important update* atau merupakan *security bug fixes* sehingga akan sangat banyak orang ingin men-*download software* tersebut (Gemmell dkk., 2000).

Ide dari *Fcast* adalah mengkombinasikan pengiriman data menggunakan *multicast* protokol untuk mengurangi jumlah trafik secara signifikan dan teknik *forward error correction* untuk meningkatkan reliabilitas dari *multicast* protokol. Pada *Fcast*, data dipecah ke dalam beberapa blok data. Dalam setiap blok data terdapat *parity* (data *redundant*) yang dapat digunakan untuk memperbaiki data yang hilang akibat *packet loss* pada blok tersebut. Pengiriman blok data dan *parity* dilakukan secara acak untuk mengurangi peluang hilangnya data dalam sebuah blok dalam jumlah besar yang nantinya mengakibatkan blok tersebut tidak dapat diperbaiki menggunakan *parity* yang ada (Gemmell dkk., 2000).

Dalam penelitian ini penulis mengemukakan sebuah rancangan protokol untuk meminimalisir penggunaan paket *acknowledgement* (ACK) dan paket *negative-acknowledgment* (NAK) dengan tujuan mengurangi peluang terjadinya *feedback implosion* pada *multicast protocol*. Metode yang digunakan merupakan pengembangan dari *NAK-based protocol* yang dikombinasikan dengan sebuah mekanisme yang memungkinkan tiap *node* mengetahui tingkat reliabilitas *node* lain yang tergabung dalam *multicast* group tersebut. Berbeda dengan mekanisme NAK pada umumnya yang langsung meminta perbaikan paket ke *node* pengirim, maka dalam rancangan protokol baru ini *node* yang mengalami kegagalan penerimaan data akan meminta perbaikan dengan mengirimkan NAK ke *node* lain yang memiliki reliabilitas tertinggi. *Node* lain ini berperan sebagai *backup node* untuk perbaikan data.

Reliabilitas dari sebuah *backup node* diukur berdasarkan *bandwidth availability* dan *packet loss* yang terjadi pada *backup node*. *Bandwidth availability* yang

dimaksud disini adalah banyaknya *bandwidth* yang tersedia dalam sebuah *end-to-end network path*, dimana besarnya ditentukan oleh bagian yang mengalami *bottleneck*, yaitu bagian yang memiliki ketersediaan *bandwidth* yang paling kecil dalam *path* tersebut (Koitani dkk., 2012). Sedangkan *packet loss* adalah banyaknya paket yang tidak sampai pada *node* tujuan.

Packet loss dipilih sebagai salah satu parameter penilaian reliabilitas *backup node* karena walaupun *bandwidth availability* antara dua *node* besar, namun memiliki *packet loss* yang besar pula maka pengiriman data akan menjadi sia-sia. Maka dari itu kedua parameter ini dipilih untuk mengukur tingkat reliabilitas *backup node* karena bila dua faktor ini memiliki nilai yang baik, maka *backup node* tersebut merupakan *node* ideal bagi *node* lain untuk meminta perbaikan paket tanpa harus meminta langsung ke *node* pengirim.

Tujuan dari metode yang diusulkan dalam penelitian ini adalah untuk mempercepat proses permintaan kembali data yang mengalami kegagalan pengiriman dan meminimalisir peluang terjadinya *feedback implosion* pada *node* pengirim. Hal ini dimungkinkan karena nantinya tidak semua *node* akan meminta perbaikan paket ke *node* pengirim. Dengan kata lain beban yang diakibatkan pengiriman ulang paket yang hilang dapat dipecah dan tidak terpusat di *node* pengirim saja.

1.2 Perumusan Masalah

Rumusan masalah dalam penelitian ini dapat dijabarkan menjadi beberapa poin berikut ini:

1. Bagaimana cara menekan *feedback implosion* pada pengiriman data secara *multicast* yang memerlukan reliabilitas yang tinggi?
2. Bagaimana mekanisme pengukuran tingkat reliabilitas sebuah *backup node*?
3. Bagaimana mekanisme permintaan ulang paket yang hilang?
4. Bagaimana mekanisme *cache* pada tiap *backup node* sehingga pengiriman data menjadi optimal?

1.3 Tujuan

Tujuan dari tesis ini adalah mengembangkan sebuah protokol pengiriman data memanfaatkan *multicast protocol* yang dapat menghemat penggunaan sumber daya jaringan namun memiliki peluang terjadinya *feedback implosion* yang kecil.

1.4 Manfaat

Manfaat yang bisa didapatkan dari tesis ini nantinya adalah dapat digunakan untuk pengiriman data secara massal yang hemat sumber daya jaringan, seperti distribusi *program update* atau *patch*, sinkronisasi data pada banyak *server* dan pengiriman *file* dalam *chat group* dalam jumlah besar.

1.5 Kontribusi Penelitian

Kontribusi dalam penelitian ini adalah mengembangkan *reliable multicast protocol* dengan menambahkan mekanisme pemilihan *backup node* yang mampu meminimalisir peluang terjadinya *feedback implosion* sehingga dapat dimanfaatkan untuk pengiriman data ke banyak penerima namun hemat sumber daya jaringan.

1.6 Batasan Masalah

Batasan masalah dalam usulan tesis ini adalah:

1. Pengujian dilakukan pada lingkungan IPv4.
2. Pengujian dilakukan pada jaringan *Ethernet*, bukan *wireless*.
3. Pengujian dilakukan dalam program *emulator* komputer (VirtualBox).
4. Data yang dikirimkan dalam pengujian penelitian ini berupa *file*, bukan *stream* audio atau video.
5. Dalam setiap pengujian hanya terdapat satu *node* pengirim pada tiap pengujian.

1.7 Sistematika Penulisan

1. **Bab 1 Pendahuluan.**

   Bab ini berisi pendahuluan yang menjelaskan latar belakang permasalahan, perumusan masalah, tujuan, manfaat, batasan masalah serta sistematika penulisan penelitian ini.

2. **Bab 2 Kajian Pustaka dan Dasar Teori.**

   Berisi uraian tentang teori dasar dari *User Datagram Protocol* (UDP), *Multicast Protocol*, *Reliable Multicast Protocol*, *latency*, *bandwidth availability*, *packet loss*, VirtualBox dan *Network Emulation* pada sistem operasi Linux.

3. **Bab 3 Metodologi Penelitian.**

   Berisi tentang penjelasan langkah-langkah yang dilakukan dalam penelitian ini yang meliputi perumusan masalah, studi literatur, perancangan protokol, perancangan program, perancangan *testbed*, pengujian, analisis hasil pengujian dan penyusunan buku tesis.

4. **Bab 4 Hasil Uji Coba dan Pembahasan.**

   Berisi tentang penjelasan hasil uji coba yang telah dilakukan dan analisis hasil percobaan tersebut berdasarkan hipotesa yang telah disusun.

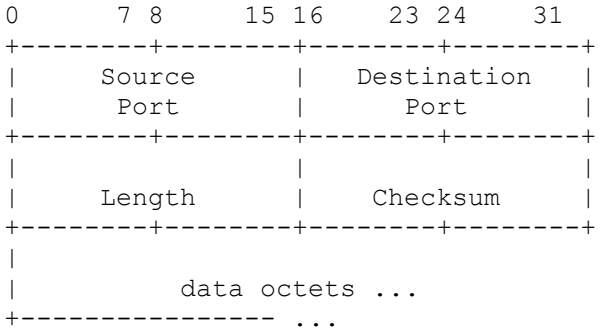
5. **Bab 5 Kesimpulan dan Saran.**

   Berisi tentang kesimpulan dari hasil percobaan dan analisis serta saran untuk penelitian selanjutnya.

BAB 2
KAJIAN PUSTAKA DAN DASAR TEORI

2.1 *User Datagram Protocol* (UDP)

User Datagram Protocol (UDP) adalah salah satu protokol *transport layer* pada TCP/IP. UDP bersifat tidak handal (*unreliable*) dan tidak menjamin urutan kedatangan data sama dengan urutan pengiriman data karena pada prosesnya UDP tidak membangun koneksi terlebih dahulu (*connectionless*). Namun karena sifatnya yang *connectionless* ini, UDP menjadi protokol yang mudah diimplementasikan dan memiliki kecepatan transmisi yang tinggi dibandingkan dengan *Transmission Control Protocol* (TCP) (Kurose and Ross, 2010).



Gambar 2.1 *User Datagram Header Format* (Postel, 1980)

Sesuai dengan standar yang telah ditetapkan oleh Postel (Postel, 1980) dalam RFC 768, UDP *header* memiliki 4 buah *field* dengan ukuran yang tetap, seperti pada Gambar 2.1. Fungsi dari masing-masing *field* tersebut adalah sebagai berikut:

1. *Source Port*

   Source port memiliki panjang 16 *bit* (2 *byte*). Bagian ini digunakan untuk mengidentifikasi alamat port yang digunakan oleh *node* pengirim untuk pengiriman pesan UDP. Penggunaan *field* ini bersifat opsional. Jika tidak digunakan, maka akan diisi dengan nilai 0. Beberapa *application layer protocol* menggunakan nilai *source port* dari pesan UDP yang masuk sebagai nilai *destination port* pada paket balasan.

2. *Destination Port*

   Destination port memiliki panjang 16 *bit* (2 *byte*). Bagian ini digunakan untuk mengidentifikasi alamat port yang digunakan oleh *node* penerima untuk menerima pesan UDP. Kombinasi antara alamat IP dengan nilai *destination port* ini adalah data yang digunakan untuk mengidentifikasi alamat *node* dan *application* yang dituju oleh sebuah pesan UDP.

3. *Length*

   Length memiliki panjang 16 *bit* (2 *byte*). Bagian ini digunakan untuk mengindikasikan panjang paket UDP (panjang *header* UDP ditambah dengan panjang pesan) dalam satuan *byte*. Ukuran paling kecil dari *filed lenght* adalah 8 *byte* (ukuran *header* UDP, ketika tidak pesan), dan ukuran paling besar adalah 65515 bytes. Panjang maksimum dari paket UDP akan disesuaikan dengan nilai *Maximum Transmission Unit* (MTU) dari jaringan di mana pesan UDP dikirimkan.

4. *Checksum*

   Checksum memiliki panjang 16 *bit* (2 *byte*). Bagian ini digunakan untuk pengecekan integritas dari paket UDP yang dikirimkan (*header* UDP dan pesan UDP). Penggunaan *field* ini bersifat opsional. Jika tidak digunakan, *field* ini akan bernilai 0.

Dari spesifikasi UDP yang dijabarkan oleh Postel (Postel, 1980), terdapat beberapa sifat dari UDP yang cocok digunakan untuk beberapa jenis *application* tertentu. Sifat-sifat tersebut antara lain:

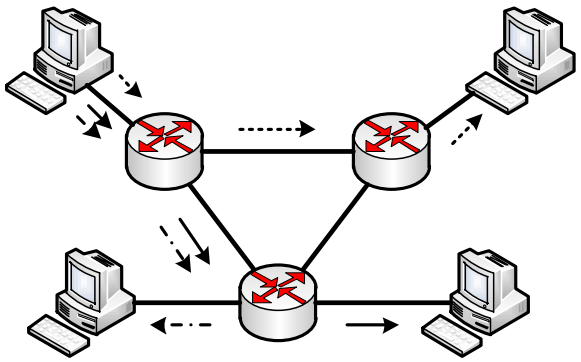
1. *Transaction oriented*, sehingga UDP cocok digunakan untuk *query-response protocol* yang sederhana seperti *Domain Name System* dan *Network Time Protocol*.
2. Memiliki *datagram* yang cocok digunakan untuk melakukan permodelan protokol lain seperti *IP Tunneling*, *Remote Procedure Call* dan *Network File System*.
3. Bersifat *stateless*, sehingga UDP cocok digunakan untuk *application* dengan jumlah *user* yang sangat banyak seperti *on demand video streaming* dan IPTV.
4. Memiliki *retransmission delays* yang kecil, sehingga cocok digunakan untuk *real-time application* seperti VoIP dan *teleconference*.

5. Bekerja dengan baik pada komunikasi *unidirectional*, sehingga cocok digunakan untuk pengiriman data secara *broadcast*.

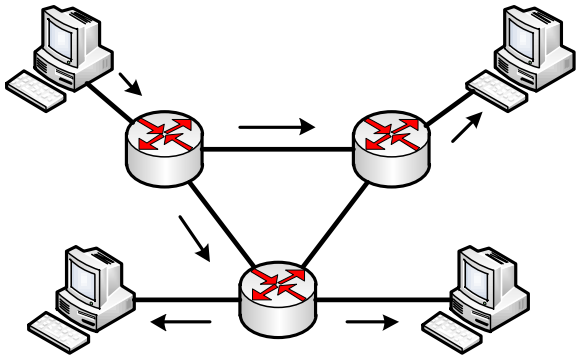
2.2 *Multicast Protocol*

Multicast protocol adalah sebuah teknik *routing* yang memungkinkan trafik IP dikirimkan dari satu atau lebih sumber ke banyak penerima. *Multicast* pertama kali distandarisasi dalam RFC 1112 (Deering, 1989), kemudian direvisi ke dalam RFC 4604 (Cain dkk., 2006) untuk menambahkan mekanisme manajemen *group* seperti *Internet Group Management Protocol Version 3* (IGMPv3) dan *Multicast Listener Discovery Protocol Version 2* (MLDv2) untuk *Source-Specific Multicast*. Revisi terbaru dari *multicast protocol* ini tertuang di dalam RFC 5771 (Vegoda dan Cotton, 2010) yang memasukkan *scope* alamat administrasi ke dalam *multicast protocol*.

Tidak seperti *unicast* yang mengirimkan tiap paket untuk tiap penerima, dalam *multicast protocol* sebuah paket dikirimkan ke alamat *multicast group*. Paket tersebut nantinya akan diduplikasi oleh *router* bila *router* mendeteksi terdapat *node* di bawah *router* tersebut yang bergabung ke dalam alamat *multicast group* tersebut. Mekanisme pendeteksian ini menggunakan *Internet Group Management Protocol* (IGMP). Sedangkan mekanisme komunikasi antar *router* diatur dalam *Multicast Listener Discovery Protocol* (MLD) (Deering, 1989).



Gambar 2.2 Pengiriman Data Menggunakan Protokol *Unicast*



Gambar 2.3 Pengiriman Data Menggunakan Protokol *Multicast*

Perbandingan antara protokol *unicast* dan *multicast* bisa dilihat pada Gambar 2.2 dan 2.3. Pada *unicast*, *node* pengirim akan mengirimkan data sebanyak jumlah *node* penerima, namun pada *multicast* hanya perlu mengirimkan data sekali saja, *router* akan menduplikasi data tersebut sebanyak *node* penerima yang terkoneksi dengannya.

Pada IPv4, salah satu komponen utama dari IP *multicast* adalah *Intenet Group-Membership Protocol* (IGMP). IGMP bergantung kepada kelas pengalamatan IP kelas D, yaitu 224.0.0.0/4 dengan range alamat 224.0.0.0 hingga 239.255.255.255, untuk membuat alamat *multicast group* seperti yang tercantum dalam RFC 1112. IGMP digunakan untuk mendaftarkan sebuah *host* ke dalam *multicast group* menggunakan alamat IP kelas D tersebut. Sebuah *host* mengidentifikasi keanggotaan dari sebuah grup dengan mengirimkan IGMP messages. Trafik IP *multicast* akan dikirimkan ke semua *host* yang tergabung ke dalam *multicast group* tesebut (Deering, 1989).

Ada beberapa *routing protocol* yang digunakan untuk menemukan *multicast group* dan membuat jalur *routing* untuk setiap grup. Tiga diantaranya adalah *Protocol Independent Multicast - Sparse Mode* (Fenner dkk., 2006), *Distance-Vector Multicast Routing Protocol* (Waitzman dkk., 1988) dan *Multicast Open Shortest Path First* (Moy, 1994).

2.3 *Reliable Multicast Protocol*

Ide awal munculnya *reliable multicast* protocol adalah akibat dari semakin banyaknya kebutuhan pengiriman data secara massal yang *reliable*, seperti sinkronisasi data pada banyak *server*, komunikasi *cluster workstation*, *live presentation* pada

kegiatan *teleconference* atau *massive peer to peer file exchange*. Selain itu kecenderungan pertumbuhan infrastruktur yang lebih lambat daripada pertumbuhan pengguna membuat kebutuhan akan protokol yang hemat *bandwidth* dan mampu digunakan secara massal semakin tinggi.

Sifat *multicast* yang hemat *bandwidth* dan *resource* berpotensi untuk dikembangkan guna menyelesaikan permasalahan ini. Lane dalam papernya (Lane dkk., 2001) menyebutkan bahwa tantangan yang harus dihadapi dalam usaha mengimplementasikan *reliable multicast* adalah:

1. Besarnya peluang terjadi *feedback implosion* yang akan membanjiri *node* pengirim (*sender*)
2. Sulitnya mengelola keanggotaan tiap *node* dan menerapkan semantik dari *reliable multicast* dalam satu *multicast group* yang dinamis (*node* bisa bergabung dan keluar *multicast group* dengan bebas)
3. Menerapkan *flow control* yang dapat diterima oleh semua *node* penerima (*receiver*) sangat rumit karena bisa jadi jenis receiver yang sangat heterogen

Saat ini, telah banyak pendekatan yang telah dikemukakan dalam mengimplementasikan *reliable multicast*. Masih dalam paper yang sama (Lane dkk., 2001), Lane membagi pendekatan *reliable multicast protocol* ke dalam empat kategori berikut:

1. ACK-*based protocols*
2. NAK-*based protocols*
3. *Ring-based protocols*
4. *Tree-based protocols*

Selain itu ada pendekatan lain yang menggunakan *Forward Error Correction* (Vicisano dkk., 2002) dan *Proxy-based* (Chawathe dkk., 1998).

2.3.1 ACK-*Based Protocols*

Pendekatan ACK-*based* pada dasarnya mengadopsi *reliable unicast protocol* seperti TCP. *Node* penerima mengirimkan data ke receiver secara *multicast*. *Node* penerima akan membalas dengan mengirimkan ACK secara *unicast* bila menerima data dari *node* penerima atau mengirimkan NAK bila tidak menerima data dari *node* penerima. *Node* penerima baru akan menghapus *cache* data tersebut bila sudah menerima ACK dari semua receiver (Lane dkk., 2001).

Kelebihan dari ACK-*based protocols* adalah sederhana, mudah diimplementasikan dan hemat penggunaan *cache*. Namun kelemahannya adalah kemungkinan terjadinya *feedback* ACK dan NAK *implosion* sangat besar. Protokol ini juga tidak dapat digunakan untuk *multicast* skala besar (*not scalable*) (Lane dkk., 2001).

2.3.2 NAK-*Based Protocols*

Pendekatan NAK-*based* memiliki perilaku yang berkebalikan dengan ACK-*based*. *Node* pengirim mengirimkan data ke receiver secara *multicast*. *Node* penerima akan membalas dengan mengirimkan NAK secara *unicast* hanya bila tidak menerima data dari *node* pengirim (Lane dkk., 2001).

Pada perkembangannya NAK-*based protocol* ada yang menggunakan metode *polling*. Bila menggunakan metode *polling*, maka *node* pengirim akan memberikan *flag* pada paket-paket tertentu secara periodik. *Node* penerima yang menerima paket dengan *flag* tersebut harus mengirimkan ACK kepada Sender secara *unicast* (Lane dkk., 2001).

Pengembangan lain NAK-*based protocol* selain menggunakan metode *polling* adalah dengan menggunakan metode NAK *suppression*, yaitu *node* penerima akan menunggu dalam periode acak tertentu sebelum mengirimkan NAK kepada *node* pengirim secara *multicast*. *Node* penerima yang menerima NAK untuk paket yang sama dari *receiver* lain akan berperilaku seakan telah mengirimkan NAK, sehingga penggiriman NAK dapat ditekan (Lane dkk., 2001).

Kelebihan dari *NAK-based protocols* adalah sederhana, mudah diimplementasikan dan terhindar dari kemungkinan terjadi *feedback* ACK *implosion*. Namun kelemahannya adalah boros penggunaan *cache*, peluang terjadinya *feedback* NAK *implosion* cukup besar bila terdapat banyak *node* yang tidak menerima data dan masih tidak dapat digunakan untuk *multicast* skala besar (*not scalable*) (Lane dkk., 2001).

2.3.3 *Ring-Based Protocols*

Sebelum pengiriman data dimulai, *node* pengatur melakukan pembagian tanggung jawab pengiriman ACK untuk tiap *node* penerima. Satu *node* penerima hanya bertanggung jawab pada paket-paket yang telah ditentukan (selanjutnya disebut sebagai *token receiver*). *Node* pengirim akan mengirimkan data ke *node* penerima secara

multicast. *Token receiver* akan membalas dengan mengirimkan ACK secara *multicast* ketika menerima data dari *node* pengirim. *Node* pengirim akan menunggu dalam periode tertentu dan akan melakukan transmisi ulang bila tidak menerima ACK dari *node* penerima. *Node* penerima akan mengirimkan NAK secara *unicast* kepada *token receiver* bila tidak menerima data. Lalu *token receiver* akan mengirimkan NAK secara *unicast* kepada *node* pengirim (Lane dkk., 2001).

Kelebihan dari *ring-based protocols* adalah terhindar dari kemungkinan terjadi *feedback* ACK dan NAK *implosion*. Selain itu pendekatan *ring-based* relatif dapat digunakan untuk *multicast* skala besar (*scalable*). Namun kelemahannya adalah protokol ini cukup kompleks dan sulit diimplementasikan. Selain itu pendekatan *ring-based protocols* boros *cache* dan tidak efisien untuk pengiriman data berukuran kecil (Lane dkk., 2001).

2.3.4 *Tree-Based Protocols*

Sebelum pengiriman data dimulai, dilakukan pembentukan *tree* dari *node* pengirim dan *fowarding router*. Tiap *fowarding router* bertangggung jawab terhadap beberapa *node* penerima atau *fowarding router* lain sehingga menghasilkan struktur berbentuk *tree* yang terdiri dari beberapa *group*. *Node* pengirim mengirimkan data ke *node* penerima secara *multicast*. *Group member* dari *multicast tree* mengirimkan ACK kepada *group leader* secara *unicast* ketika menerima data dari *node* pengirim. *Group leader* akan membuat *resume* ACK dari *group member* lalu mengirimkan satu *resume* ACK tersebut kepada *node* pengirim. Bila ada *group member* yang tidak menerima data, maka NAK akan dikirimkan ke *group leader*. *Group leader* akan membuat *resume* NAK dari *group member* lalu mengirimkan satu *resume* NAK tersebut kepada *node* pengirim (Lane dkk., 2001).

Kelebihan dari *tree-based protocols* adalah terhindar dari kemungkinan terjadi *feedback* ACK dan NAK *implosion*. Selain itu pendekatan *ring-based* relatif dapat digunakan untuk *multicast* skala besar (*scalable*). Namun kelemahannya adalah cukup kompleks dan sulit diimplementasikan. Selain itu pendekatan ini, peluang terjadinya *delay* cukup besar karena ACK dan NAK harus diresume terlebih dahulu dari banyak *group* serta pengelolaan bentuk *tree* tersebut harus dilakukan sepanjang waktu pengiriman (Lane dkk., 2001).

2.3.5 FEC-*Based Protocols*

Forward Error Correction (FEC) adalah mekanisme perbaikan data dengan menambahkan sejumlah *parity* yang dapat digunakan untuk perbaikan data yang rusak atau hilang (Vicisano dkk, 2002). Misalnya terdapat 10 paket data dan dibuat 2 paket *parity*, bila terjadi kegagalan pengiriman maksimal sejumlah 2 paket data maka *node* penerima akan mampu memperbaiki data tersebut. FEC-*based protocols* tidaklah berdiri sendiri, protokol ini merupakan tambahan protokol untuk keempat protokol sebelumnya. Tujuan dari penggunaan FEC adalah untuk meningkatkan reliabilitas pengiriman data tanpa harus melakukan proses pengiriman ulang.

Kelebihan dari *FEC-based protocols* adalah dapat meningkatkan reliabilitas pengiriman data dan dapat mengurangi frekuensi pengiriman NAK bila terjadi kegagalan pengiriman data. Namun kelemahannya adalah memerlukan *resource* yang lebih banyak serta memerlukan waktu tambahan untuk melakukan proses *encoding decoding* (Vicisano dkk, 2002).

2.3.6 *Proxy-Based Protocols*

Sama seperti FEC-*based protocols*, *Proxy-based protocols* bertujuan untuk meningkatkan reliabilitas pengiriman. Ketika *node* penerima mengalami kegagalan penerimaan data maka dia cukup mengirimkan NAK kepada *multicast proxy* (Chawathe dkk, 1998). *Multicast proxy* tersebut akan mengirimkan paket yang diminta tersebut. Hal ini dimungkinkan karena *multicast proxy* dapat menyimpan data *multicast* yang melaluinya untuk disimpan sementara waktu.

Kelebihan dari *proxy-based protocols* adalah dapat meningkatkan reliabilitas pengiriman data dan dapat mengurangi frekuensi pengiriman NAK bila terjadi kegagalan pengiriman data. Namun kelemahannya adalah memerlukan *resource* tampbahan berupa *node* yang berperan sebagai *multicast* proxy (Chawathe dkk, 1998).

2.4 *Latency*

Ada beberapa jenis *latency* dalam jaringan (Curtis dan Mcgregor, 2001). Yang pertama adalah *link latency*. *Link latency* adalah waktu yang dibutuhkan *byte* pertama dari sebuah paket mulai dari *byte* tersebut ditempatkan pertama kali dalam sebuah medium hingga *byte* tersebut keluar dari medium tersebut. Dengan kata lain *link latency*

adalah waktu tembuh sebuah *byte* data dalam sebuah medium pengiriman. Besarnya *link latency* ini bergantung pada jenis dan panjang dari mendium pengiriman yang digunakan tersebut (Curtis dan Mcgregor, 2001). Semakin panjang medium pengiriman maka semakin besar pula *link latency* yang terjadi.

Yang berikutnya adalah *transmission delay*. *Transmission delay* adalah waktu tunggu sebuah paket hingga paket tersebut dapat dikirimkan melalui media transmisi tertentu. *Transmission delay* dipengaruhi oleh *packet size* dan *bandwidth*. *Transmission delay* dihitung mulai dari *byte* pertama dikirimkan hingga *byte* terakhir dari sebuah paket dikirimkan (Curtis dan Mcgregor, 2001).

Transmission time adalah kombinasi dari *link latency* dan *transmission delay*. *Transmission time* adalah waktu yang dihitung mulai dari *byte* pertama dari sebuah paket dikirimkan di sebuah medium hingga *byte* terakhir dari sebuah paket tersebut keluar dari medium. Sehingga *transmission time* adalah penjumlahan dari *link latency* dan *transmission delay* (Curtis dan Mcgregor, 2001).

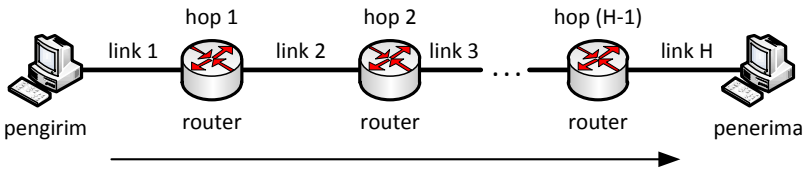
Path latency adalah total dari seluruh *transmission time* sebuah link yang ditempuh dan waktu antrian dalam tiap *router* (Curtis dan Mcgregor, 2001). *Path latency* ini cukup sulit untuk diukur karena diperlukan sinkronisasi *clock* pada tiap elemen dalam jaringan (*end point node* dan *router*) dengan sangat presisi (Curtis dan Mcgregor, 2001).

Cara alternatif untuk menghitung seluruh *latency* tersebut adalah dengan menghitung *round trip time* (RTT) *latency*. RTT adalah total dari *path latency* pengiriman dan *path latency* pengembalian. Untuk menghitung RTT dapat dilakukan dengan menghitung waktu mulai sebuah paket dikirimkan hingga paket tersebut sampai kembali. Namun yang perlu diperhatikan adalah *path* pengiriman bisa saja berbeda dari *path* pengembalian (Curtis dan Mcgregor, 2001). Sehingga kita tidak bisa menghitung *path latency* dengan cara membagi dua nilai RTT.

2.5 *Bandwidth Availability*

Bandwidth availability adalah banyaknya *bandwidth* yang tersedia dalam sebuah *end-to-end network path*, dimana besarnya ditentukan oleh bagian yang mengalami *bottleneck*, yaitu bagian yang memiliki ketersediaan *bandwidth* yang paling kecil dalam *path* tersebut (Koitani dkk., 2012). Prinsip dasar dari penghitungan

bandwidth availability ini adalah dengan membangkitkan sejumlah *probe packet* dan dikirim ke *node* penerima dalam *rate* tertentu. *Node* penerima akan mengobservasi interval waktu kedatangan dari *probe packet* tersebut dan memutuskan apakah *rate* pengiriman *proble packet* lebih besar dari *bandwidth* yang tersedia atau tidak. Proses pengiriman probe *packet* ini dilakukan beberapa kali dengan rate yang berbeda-beda agar bisa dapatkan nilai *bandwidth availability* yang akurat (Koitani dkk., 2012).



Gambar 2.4 Model Jaringan (Koitani dkk., 2012)

Misalkan terdapat sebuah *end-to-end network path* seperti Gambar 2.4. *Path* tersebut terdiri dari H *link* dan tiap *link* disebut sebagai *link i* dimana $(1 \leq i \leq H)$. *Physical bandwidth* dari *link i* dinotasikan dengan C_i dan *available bandwidth* dari *link i* dinotasikan dengan A_i. Nilai *physical bandwidth* dari *end-to-end network path* ekivalen dengan nilai *physical bandwidth* dari *link* dengan *physical bandwidth* terkecil atau dapat direpresentasikan dengan rumus 2.1 (Koitani dkk., 2012).

$$C \equiv \min_{i=1\ldots H} C_i \tag{2.1}$$

Jika *everange bandwidth utilization* dari *link i* pada waktu *t* dinotasikan dengan $u_i(t)$ dimana $(0 \leq u_i(t) \leq 1)$, maka *available bandwidth* dari *link i* dapat direpresentasikan dengan rumus 2.2 (Koitani dkk., 2012).

$$A_i(t) \equiv C_i\big(1 - u_i(t)\big) \tag{2.2}$$

Sedangkan nilai *available bandwidth* dari *end-to-end network path* ekivalen dengan nilai *available bandwidth* dari *link* dengan *available bandwidth* terkecil atau dapat direpresentasikan dengan rumus 2.3 (Koitani dkk., 2012).

$$A(t) \equiv \min_{i=1\ldots H} C_i\left(1 - u_i(t)\right) \quad (2.3)$$

Untuk mengukur *available bandwidth* dari *end-to-end network path*, *node* pengirim akan mengirimkan sejumlah *K probe packet* ke *node* penerima. Waktu pengiriman probe *packet* ke ke-*k* dinotasikan dengan t_k, dimana $(1 \leq k \leq K)$. Waktu kedatangan paket ke-*k* di *node* tujuan dinotasikan dengan t'_k, sehingga *one-way delay* dari paket ke-*k* direpresentasikan dengan $D_k = t'_k - t_k$. Selisih *one-way delay* dari paket ke-*k* dengan paket ke-(*k+1*) dapat dihitung dengan rumus 2.4 (Koitani dkk., 2012).

$$\begin{aligned} \Delta D_k &= \mathrm{D_{k+1}} - \mathrm{D_k} \\ &= (\mathrm{t'_{k+1}} - \mathrm{t_{k+1}}) - (\mathrm{t'_k} - \mathrm{t_k}) \\ &= (\mathrm{t'_{k+1}} - \mathrm{t'_k}) - (\mathrm{t_{k+1}} - \mathrm{t_k}) \\ &= \Delta \mathrm{t'_k} - \Delta \mathrm{t_k} \end{aligned} \quad (2.4)$$

$\Delta \mathrm{t'_k}$ adalah selisih waktu kedatangan paket ke-*k* dan paket ke-(*k+1*) di *node* penerima, sedangkan $\Delta \mathrm{t_k}$ adalah selisih waktu keberangkatan paket ke-*k* dan paket ke-(*k+1*) di *node* pengirim. Dalam melakukan perhitungan rumus 2.4 ini tidak diperlukan sinkronisasi *clock* antar *node* pengirim dengan *node* penerima karena yang dihitung hanyalah selisih waktu antar *probe packet* (Koitani dkk., 2012).

Ketika *rate* pengiriman sama dengan atau lebih kecil dari *available bandwidth* dari *end-to-end network path* maka hasil dari perhitungan rumus 2.4 akan menghasilkan nilai mendekati 0 karena interval waktu antar *probe packet* tidak berubah selama pengiriman melalui jaringan tersebut. Namun bila *rate* pengiriman lebih besar dari *available bandwidth* dari *end-to-end network path* maka hasil dari perhitungan rumus 2.4 akan bernilai positif karena interval waktu penerimaannya akan menjadi semakin besar dibandingkan interval waktu pengirimannya. Dengan mengirimkan *probe packet* dalam *rate* tertentu dalam beberapa percobaan yang berbeda, maka dapat dihitung berapakah *available bandwidth* dari *end-to-end network path* tersebut (Koitani dkk., 2012).

2.6 *Packet loss*

Packet loss terjadi ketika satu atau lebih paket yang melewati jaringan komputer mengalami kegagalan sehingga tidak dapat sampai ke tujuan. *Packet loss* ini dapat disebabkan karena banyak hal, sebagai contoh karena degradasi sinyal dalam medium pengiriman di jaringan, *channel congestion*, *corrupted paket* sehingga ditolak oleh mesim tujuan, dan disebabkan karena kerusakan *hardware* jaringan (Kurose dan Ross, 2010).

Network transport protocol yang *reliable* seperti TCP memiliki mekanisme untuk menangani *packet loss*. Ketika terjadi *packet loss*, penerima akan meminta pengiriman ulang paket atau *segment* yang hilang tersebut kepada pengirim secara otomatis. Walaupun TCP dapat menangani permasalahan *packet loss* ini, namun *packet loss* tepat dapat menurunkan *throughput* dalam jaringan (Kurose dan Ross, 2010).

Berbeda dengan TCP, UDP yang juga digunakan dalam *multicast protocol* sifatnya tidak *reliable*. UDP tidak memiliki mekanisme perbaikan paket. Perbaikan ini harus ditangani sendiri oleh *application layer protocol* (Kurose dan Ross, 2010).

2.7 VirtualBox [1]

VirtualBox adalah aplikasi *cross-platform virtualization* yang bisa berjalan di atas komputer berbasis Intel atau AMD. VirtualBox tidak terikat pada sistem operasi tertentu, sehingga dapat berbaik pada sistem operasi Windows, Mac, Linux atau Solaris. VirtualBox dapat mengoptimalkan kemampuan komputer, sehingga kita dapat menjalankan banyak sistem operasi menggunakan mesin virtual dalam satu komputer yang sama. Jumlah maksimal mesin virtual yang bisa berjalan bergantung pada besarnya *free space* dan *memory* dari komputer yang menjadi induk dari mesin virtual tersebut (*host*).

Ada beberapa terminologi dalam VirtualBox, yaitu:

1. *Host operating system* (*host* OS)

   Adalah sistem operasi yang berjalan di komputer dimana VirtualBox di *install*. Sistem operasi yang didukung oleh VirtualBox antara lain Windows, Mac OS X, Linux dan Solaris.

[1] https://www.virtualbox.org/

2. *Guest operating system* (*guest* OS).

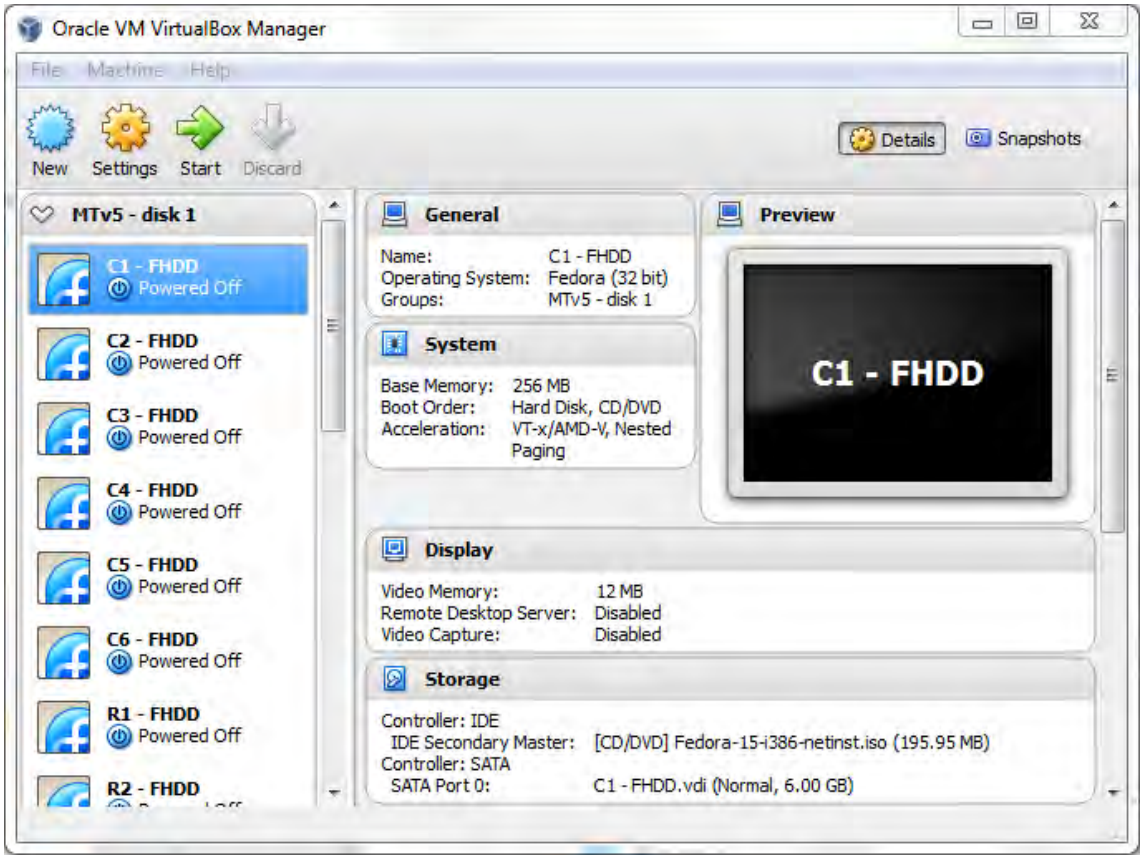
   Adalah sistem operasi yang berjalan di komputer atau mesin virtual. Secara teori, VirtualBox dapat menjalankan semua jenis sistem operasi beraksitektur x86, tapi hanya beberapa yang bisa berjalan nyaris menyamai perfoma sistem operasi non virtual.

3. *Virtual machine* (VM)

   Adalah *special environment* yang dibuat oleh VirtualBox untuk menjalankan guest OS. Virtual Machine ini memiliki beberapa parameter seperti jumlah CPU, ukuran primary memory, ukuran secondary memory, jaringan, dll. Parameter ini dapat diset sesuai dengan kebutuhan.

4. *Guest Additions*.

   Adalah *software* tambahan yang disediakan oleh VirtualBox. *Software* ini dapat di *install* ke dalam guest OS dengan tujuan untuk meningkatkan perfoma dari guest OS dan memberikan *feature* pendukung tambahan.



Gambar 2.5 VirtualBox pada Sistem Operasi Windows 7

2.8 *Network Emulation*

Network emulation adalah tenik memanipulasi jaringan komputer sehingga dapat mengemulasikan kondisi jaringan yang diinginkan untuk menilai kinerja, memprediksi dampak perubahan atau mengoptimalkan jaringan yang ada. Sistem operasi linux memiliki fungsi *network emulation* yang disebut dengan netem ("netem," 2009). Netem yang ada pada linux saat ini dapat mengemulasikan variable *delay*, *loss*, *duplication* dan *re-ordering*.

Netem pada linux dikontrol menggunakan perintah `tc` yang merupakan bagian dari paket `iproute2`. Berikut adalah contoh-contoh perintah *network emulation* menggunakan perintah `tc`.

2.8.1 Emulasi *Delay*

Perintah di bawah ini adalah contoh paling sederhana untuk memberikan *delay* sebesar 100 ms untuk seluruh paket yang dikirim menggunakan *interface* eth0 dari local ethernet.

```
tc qdisc add dev eth0 root netem delay 100ms
```

Berikut adalah penjelasan tiap perintah tersebut yang diambil dari *Linux manual page* (Hubert, 2001) (Ludovici, 2011):

1. **`tc`** adalah perintah yang digunakan untuk memanipulasi *traffic* di linux.
2. **`qdisc`** (singkatan dari „*queueing discipline*") merupakan tempat antrian paket sebelum paket tersebut dikirimkan ke *interface* yang dituju. **`qdisc`** akan berbeda-beda untuk tiap *interface*.
3. **`add`** adalah lanjutan dari perintah **`qdisc`** yang bertujuan untuk menambahkan **`qdisc`** ke dalam *interface* yang dituju (**`eth0`**). Bila pada *interface* tersebut sudah pernah ditambahkan **`qdisc`** maka perintah **`add`** tidak bisa digunakan, harus diganti dengan perintah **`change`**, **`replace`** atau **`link`**.
4. **`root`** menandakan bahwa **`qdics`** yang digunakan berjenis *classless*.
5. **`netem`** adalah perintah untuk memanipulasi traffik jaringan.
6. **`delay`** adalah lanjutan dari perintah **`netem`** yang menciptakan *delay* pada jaringan sebesar nilai yang di tentukan. Nilai pada perintah ini bisa berupa satu

nilai (misal **100ms**) atau nilai random dalam range tertentu (misal **100ms 10ms** akan membuat *delay* random antara 90ms - 110ms)

2.8.2 Emulasi *Packet Loss*

Berikut adalah contoh perintah untuk mengemulasikan *packet loss*:

```
tc qdisc add dev eth0 root netem loss 0.1%
```

Perintah tersebut akan menciptakan *packet loss* sebesar 0.1% dari total keseluruhan paket yang dikirimkan.

2.8.3 Emulasi *Packet Duplication*

Berikut adalah contoh perintah untuk mengemulasikan paket yang terduplikasi:

```
tc qdisc add dev eth0 root netem duplicate 1%
```

Perintah tersebut akan menciptakan duplikasi paket sebesar 1% dari total keseluruhan paket yang dikirimkan.

2.8.4 Emulasi *Packet Corruption*

Berikut adalah contoh perintah untuk mengemulasikan kerusakan paket:

```
tc qdisc add dev eth0 root netem corrupt 0.1%
```

Perintah tersebut akan membuat 1 *bit* kerusakan pada paket sebesar 0.1% dari total keseluruhan paket yang dikirimkan.

2.8.5 Emulasi *Packet Re-ordering*

Ada dua cara untuk melakukan *packet re-ordering*. Metode pertama adalah dengan menggunakan perintah **`gap`**, dimana perintah tersebut akan memberikan *delay* pada semua paket kecuali paket dengan urutan kelipatan tertentu. Misalnya:

```
tc qdisc add dev eth0 root netem gap 5 delay 10ms
```

Perintah tersebut akan memberikan *delay* sebesar 10ms pada tiap paket, kecuali pada paket ke-5 dan kelipatannya (ke-10, ke-15, dst), sehingga paket dengan urutan kelipatan 5 tersebut akan dikirimkan langsung tanpa *delay*.

Metode kedua adalah dengan menggunakan perintah **`reorder`**. Perintah **`reorder`** akan mebuat sejumlah paket mengalami *mis-ordered* dengan persentase tertentu.

`tc qdisc add dev eth0 root netem delay 10ms reorder 25% 50%`

Perintah ini akan memberikan *delay* pengiriman sebesar 10ms dan 25% (dengan korelasi 50%) akan dikirimkan langsung tanpa *delay*. Perintah ini akan memberikan efek *re-ordering* yang mirip dengan kondisi di jaringan sebenarnya.

BAB 3
METODOLOGI PENELITIAN

Langkah-langkah yang akan dilakukan pada penelitian ini adalah sebagai berikut:

1. Perumusan Masalah
2. Studi Literatur
3. Perancangan Protokol
4. Perancangan Program
5. Perancangan *Tested*
6. Pengujian
7. Analisis Hasil Pengujian
8. Penyusunan Buku Tesis

3.1 Perumusan Masalah

Masalah utama yang dihadapi dalam penelitian ini adalah peluang terjadinya *feedback implosion* yang besar ketika menggunakan protokol *multicast* untuk pengiriman data yang memerlukan reliabilitas tinggi. Berangkat dari permasalahan ini, penulis mengemukakan sebuah ide untuk mengatasi permasalahan tersebut dengan cara menggunakan *backup node* sehingga paket *acknowledge* tidak menumpuk di *node* pengirim.

Dari sini permasalahan berkembang ke teknik pemilihan pemilihan dan pengukuran tingkat reliabilitas *backup node*, mekanisme permintaaan ulang paket yang hilang serta teknik *caching* pada tiap *backup node* sehingga dapat mempercepat proses perbaikan paket yang hilang.

3.2 Studi Literatur

Dalam tahap studi literatur, dikaji berbagai referensi yang berkaitan dengan *User Datagram Protocol* (UDP), *Multicast Protocol, Reliable Multicast Protocol* yang sudah dikembangkan saat ini, teori *latency*, *bandwidth availability* dan *packet loss*. Selain itu dikaji pula hal-hal pendukung penelitian ini seperti VirtualBox yang berperan

sebagai *testbed* percobaan dan *Network Emulation Tools* yang berguna untuk mengemulasikan *delay, packet loss* dan beberapa perilaku pengiriman data lainnya di internet.

3.3 Perancangan Protokol

Protokol yang dirancang dalam penelitian ini akan berjalan pada *application layer* di sisi *end node*, tidak perlu mengganti *hardware* atau merubah protokol *multicast*, sehingga dapat diimplementasikan dalam jaringan nyata yang telah berjalan saat ini dengan cepat. Ada 2 protokol yang dirancang dalam penelitian ini, yaitu protokol *discovery* dan protokol pengiriman *file*.

3.3.1 Protokol Discovery

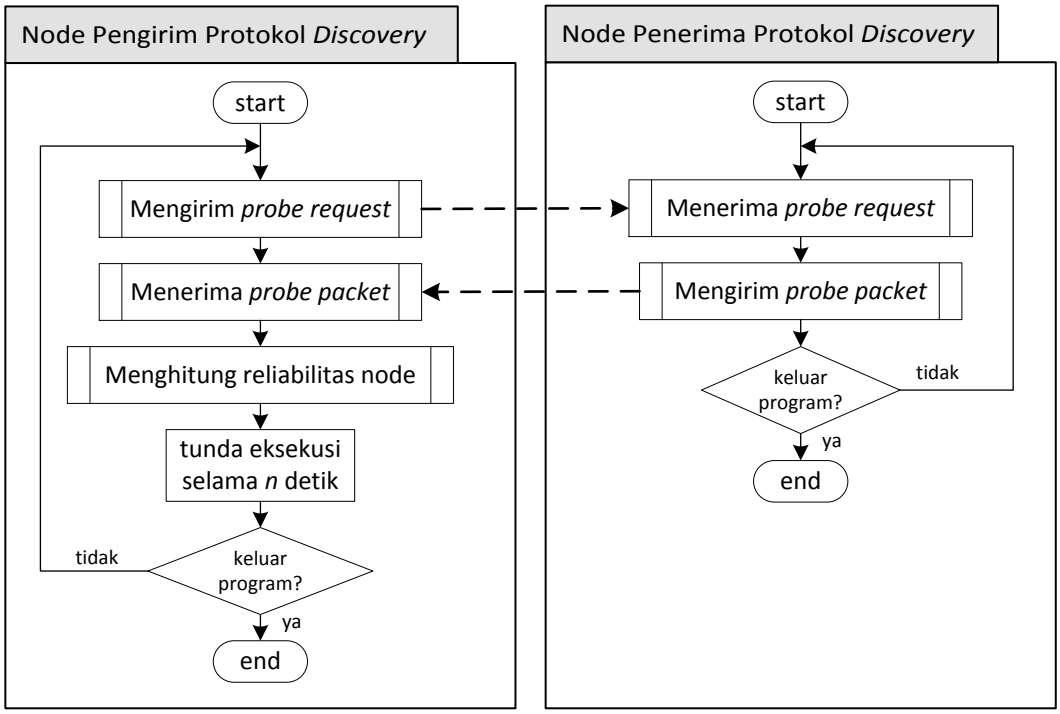
Protokol *discovery* adalah protokol yang digunakan untuk menemukan *node* lain yang tergabung dalam *multicast group* yang sama. Tujuan dari protokol ini adalah untuk mendapatkan data tingkat reliabilitas *node* lain yang nantinya akan dijadikan sebagai *backup node* saat penerimaan data. Protokol *discovery* akan berjalan terus-menerus dengan periode eksekusi tertentu dan akan berhenti ketika pengiriman data sudah berakhir. Alur dari protokol ini dapat dilihat pada Gambar 3.1.

Protokol ini diawali dengan pengiriman secara *multicast* sebuah *probe request*, yaitu sebuah paket yang berisi permintaan pengujian *bandwidth* ke setiap *node* dalam *multicast group*. *Node* yang menerima paket *probe request* akan membalas dengan mengirimkan paket *probe* dalam jumlah tertentu secara *unicast*. Paket *probe* inilah yang nantinya akan digunakan untuk menghitung reliabilitas dari *node* lain yang akan dijadikan sebagai *backup node*. Dari proses ini nantinya akan didapatkan sejumlah *N backup node*. Ilustrasi proses ini dapat dilihat pada Gambar 3.2.

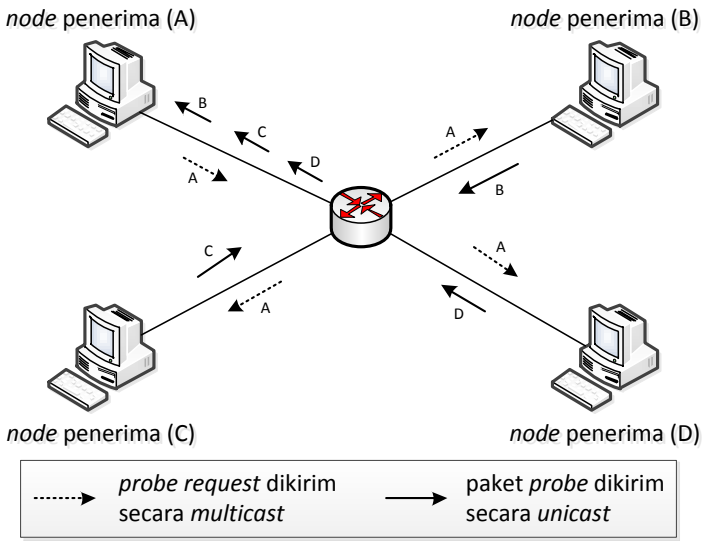
Semua *node* yang tergabung dalam *multicast group* menjalankan protokol *discovery*, kecuali *node* pengirim. *Node* pengirim tidak melakukan pengiriman *probe request*. *Node* pengirim hanya akan merespon *probe request* dari *node* lain. Hal ini dikarenakan *node pengirim* tidak akan pernah meminta perbaikan paket sehingga tidak perlu mengukur reliabilitas dari *node* lain.

Proses *discovery* dilakukan dengan mengirimkan sejumlah *P probe packet* ke alamat *multicast group*. Paket tersebut berisi *probe id*, *total sequence, sequence number*,

waktu pengiriman dari *node* pengirim dalam detik sejak *epoch time*[2] dan *dummy payload* yang sengaja ditambahkan untuk menguji *bandwidth availability* pada jaringan.



Gambar 3.1 *Flowchart* Protokol *Discovery*



Gambar 3.2 Ilustrasi Protokol *Discovery*

[2] *Epoch* adalah titik dimana perhitungan waktu dimulai. Pada Unix, *epoch* dimulai tanggal 1 Januari 1970 pukul 00.00. http://docs.python.org/2/library/time.html

Node penerima akan menghitung selisih antar *probe packet* menggunakan rumus 2.4 untuk memperkiraan *bandwidth availability* pada jaringan. Selain itu *node* penerima akan menghitung persentase *packet loss* dari *probe packet* tersebut. Proses ini akan dilakukan secara periodik dalam rentang waktu tertentu.

Tingkat reliabilitas *node* ke-*n* dari sejumlah N *backup node* dinotasikan dengan R_n dengan $(1 \leq n \leq N)$. R_n bernilai antara [0 – 1] yang nilainya ditentukan berdasarkan rumusan 3.1.

$$R_n = \left(b_1 \times \left(1 - \left(\frac{\Delta D_n}{\max\limits_{i=1\ldots N} \Delta D_i} \right) \right) \right) + \left(b_2 \times \frac{(P - l_n)}{P} \right) \tag{3.1}$$

ΔD adalah selisih *probe packet* pada tiap *node* yang dihasilkan dari rumus 2.4. P adalah jumlah total *probe packet* yang dikirimkan dan l_n adalah jumlah *probe packet* yang hilang (*loss*). b_1 adalah bobot nilai untuk persentase selisih waktu antar *probe packet* sedangkan b_2 adalah bobot nilai untuk persentase *packet loss* yang terjadi, dimana $(b_1 + b_2 = 1)$.

Paket *probe request* dan paket *probe* yang digunakan pada protokol ini memiliki format seperti pada Gambar 3.3 dan Gambar 3.4.

value	*packet* type
C data types	unsigned char
length (bytes)	1

Gambar 3.3 Format Paket *Probe Request*

value	*packet* type	probe id	total seq	seq number	timestamp	dummy payload
C data types	unsigned char	unsigned short	unsigned short	unsigned short	double	char[]
length (bytes)	1	2	2	2	8	(max) 1457

Gambar 3.4 Format Paket *Probe*

Paket *probe request* hanya berisi 1 *byte* data yang berisi tipe paket. Paket *probe* berisi beberapa data, yaitu *packet type, probe id, total seq, seq number, timestamp* dan *dummy payload*. Kegunaan data-data tersebut adalah sebagai berikut:

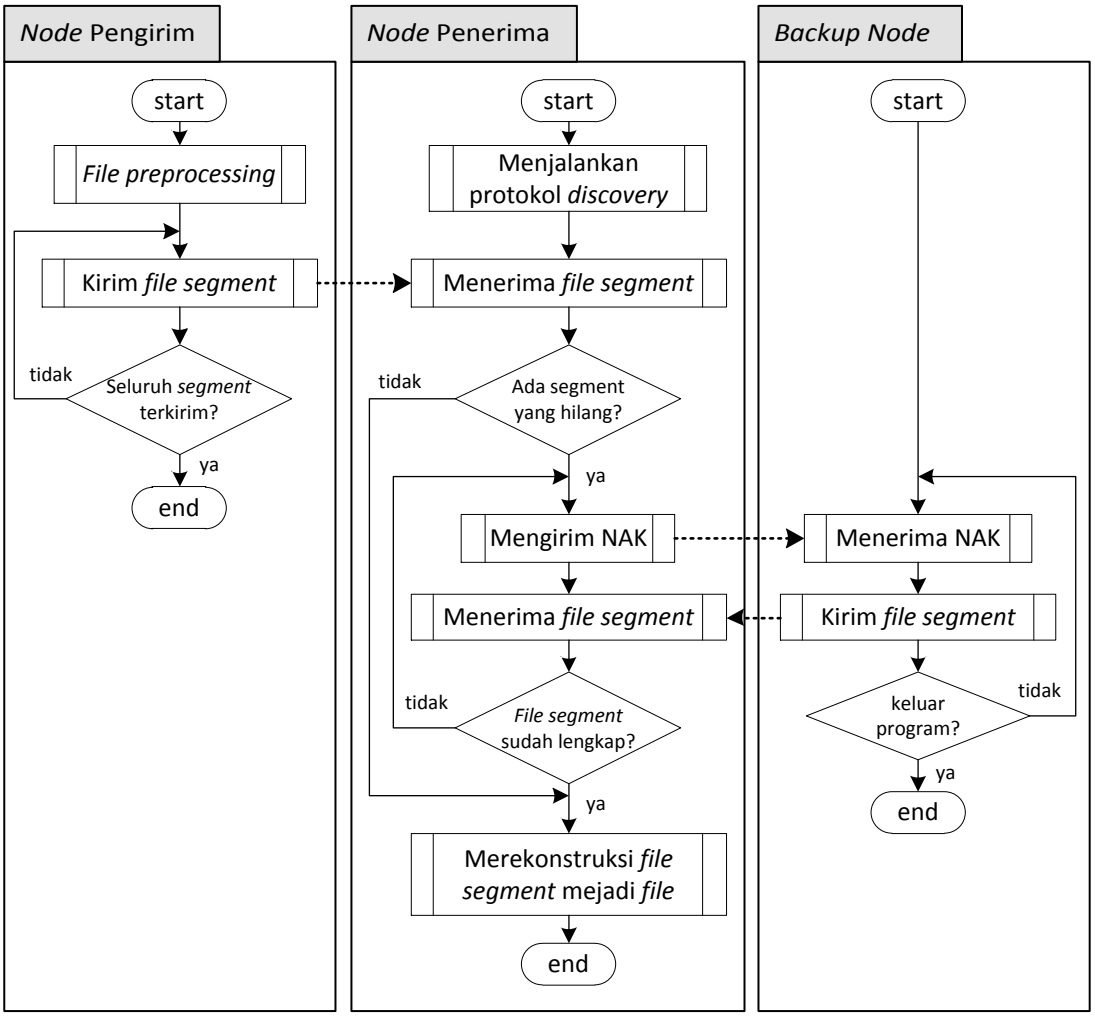
1. *probe id* adalah kode identitas yang digunakan untuk membedakan antara paket *probe* tiap periode pengujian *bandwidth*.
2. *total seq* adalah jumlah paket *probe* yang akan dikirim untuk satu periode pengujian.
3. *seq number* adalah kode urutan paket berupa angka untuk tiap paket dalam satu periode pengujian.
4. *timestamp* adalah waktu saat paket tersebut dikirimkan.
5. *dummy payload* adalah data *dummy* yang digunakan untuk mengukur *bandwidth availability* dalam jaringan.

3.3.2 Protokol Pengiriman *File*

Protokol pengiriman *file* adalah protokol yang digunakan untuk mengirim data dari *node* pengirim ke *node-node* penerima. Protokol ini melibatkan tiga jenis *node*, yaitu *node* pengirim, *node* penerima dan *backup node*. Tujuan dari protokol ini adalah mengirimkan *file* dari *node* pengirim ke *node* penerima serta mendistribusikan NAK ke *backup node* sehingga tidak terjadi *feedback implosion* pada *node* pengirim. Secara garis besar alur protokol ini dapat dilihat pada Gambar 3.5.

Node pengirim akan melakukan *preprocessing* terhadap *file* yang akan dikirimkan. *File* akan dipecah menjadi segmen-segmen kecil dengan ukuran yang sesuai dengan MTU pada jaringan untuk mencegah agar tidak terjadi segmentasi saat proses pengiriman. Setiap segmen akan ditambahkan header sesuai dengan format paket pada Gambar 3.6. Kegunaan tiap data pada format paket tersebut adalah sebagai berikut:

1. *packet type* digunakan sebagai kode untuk mengidentifikasi jenis paket yang dikirimkan.
2. *sha* digunakan sebagai kode *hash sha1* yang digunakan untuk mengidentifkasi *file* yang sedang dikirimkan.
3. *total sequence* digunakan sebagai total paket yang akan dikirimkan untuk satu *file* yang sama.
4. *sequence number* digunakan sebagai kode angka penanda urutan segmen-segmen yang dikirimkan.
5. *file segment* adalah potongan atau segmen dari *file* yang akan dikirimkan.



Gambar 3.5 *Flowchart* Protokol Pengiriman *File*

value	*packet* type	*sha*	total sequence	sequence number	*file segment*
C data types	unsigned char	char[]	unsigned long	unsigned long	char[]
length (bytes)	1	20	4	4	(max) 1443

Gambar 3.6 Format Paket *Segment*

Tiap segmen hasil *preprocessing* ini akan dikirimkan menggunakan protokol *multicast* dan akan diterima oleh seluruh *node* penerima yang tergabung ke dalam *multicast group* tersebut. Proses ini akan dilanjutkan hingga seluruh segmen terkirim.

Tiap *node* penerima akan menjalankan protokol *discovery* agar dapat mengetahui *node-node* lain yang *reliable* untuk dijadikan sebagai *backup node*. Yang perlu dicatat disini adalah bisa jadi urutan *backup node* bisa berbeda-beda untuk tiap-

tiap *node* penerima. Hal ini disebabkan karena hasil perhitungan pada protokol *discovery* akan bervariasi untuk tiap *node* penerima.

Setelah protokol *discovery* dijalankan menggunakan *thread* yang terpisah, *node* penerima akan bersiap untuk menerima seluruh *segment file* yang dikirimkan oleh *node* pengirim. Bila *node* penerima mendeteksi adalah paket yang hilang saat pengiriman ini, maka *node* penerima akan mengirimkan NAK ke *backup node* secara *unicast*, dimulai dari *backup node* dengan tingkat reliabilitas tertinggi. Format paket NAK ini dapat dilihat pada Gambar 3.7.

value	*packet* type	*sha*	sequence number
C data types	unsigned char	char[]	unsigned long
length (bytes)	1	20	4

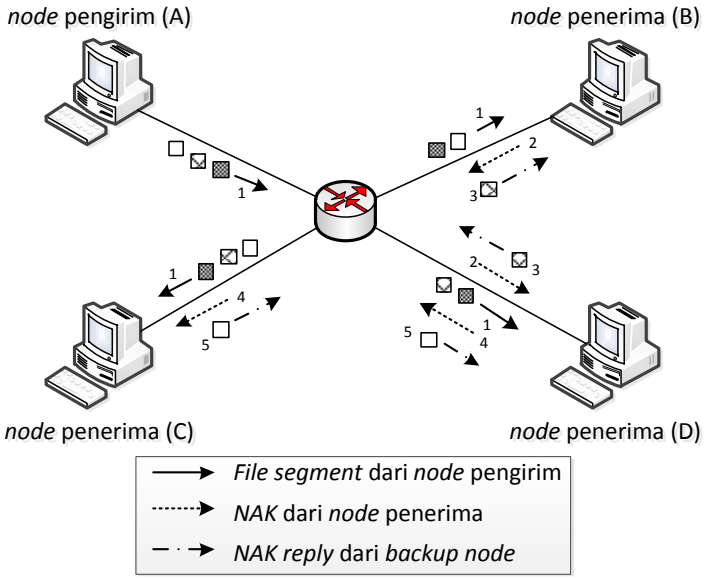
Gambar 3.7 Format Paket NAK

Kegunaan dari tiap data pada paket tersebut adalah sebagai berikut:

1. *packet type* adalah kode untuk mengidentifikasi jenis paket.
2. *sha* adalah kode hash *sha1* dari *file* yang mengalami *packet loss*.
3. *sequence number* adalah kode segmen yang hilang.

Backup node yang menerima NAK akan membalasnya dengan mengirimkan segmen yang diminta tersebut secara *unicast* dengan format sesuai Gambar 3.5. Namun bila tidak memiliki *segment* yang diminta maka *backup node* tidak akan membalas NAK tersebut.

Node penerima akan menerima balasan NAK dari *backup node* dan mengecek apakah masih terdapat segmen yang hilang. Bila masih ada segmen yang hilang maka *node* penerima akan mengirimkan NAK lagi ke *backup node* lain dengan reliabilitas yang lebih rendah. Proses ini akan berlangsung hingga seluruh segmen diterima atau tidak ada lagi *backup node* yang dapat memberikan segmen yang hilang tersebut. Ilustrasi dari proses ini dapat dilihat pada Gambar 3.8.



Gambar 3.8 Ilustrasi Protokol Pengiriman *File*

3.4 Perancangan Program

Protokol yang diusulkan dalam penelitian ini akan diimpelentasikan dalam bahasa pemrograman Python dan akan dijalankan pada sistem operasi Linux Fedora 15[3]. Program akan dibagi ke dalam beberapa modul sesuai dengan fungsinya masing-masing. Modul utama dari program ini adalah modul *rmcast*. Modul *rmcast* tidak memiliki fungsi spesifik, hanya berperan sebagai kontainer untuk modul-modul lain. Modul lain yang ada di bawah rmcast *adalah configuration*, *file*, *netiface*, *packet*, *protocol*, *sock*. Skema dari modul tersebut dapat dilihat pada Gambar 3.9. Pada bab 3.4.1 hingga bab 3.4.6 akan dijelaskan lebih detail mengenai fungsi dari tiap-tiap modul.

3.4.1 Modul *configuration*

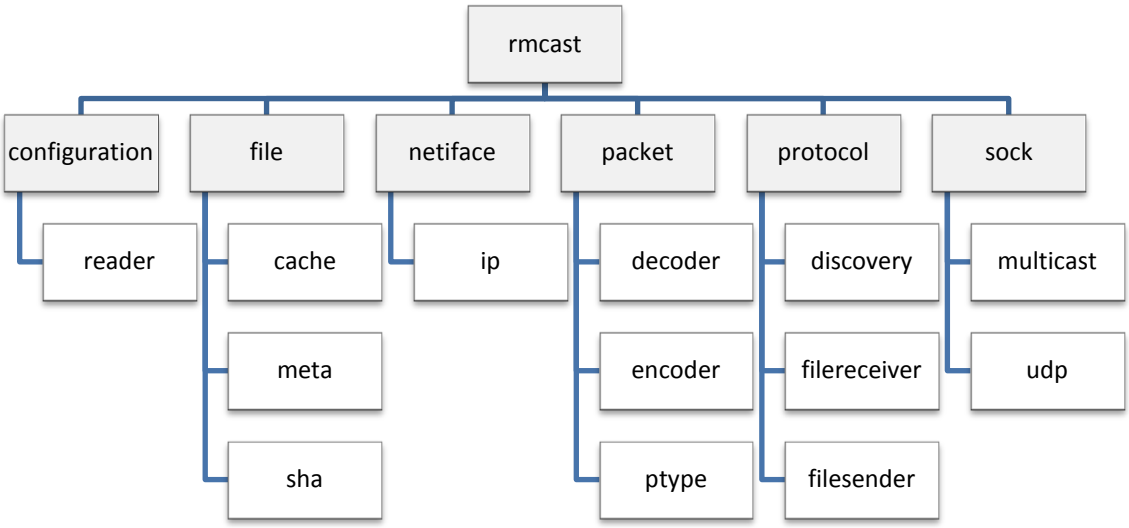
Modul *configuration* berisi fungsi-fungsi yang berkaitan dengan pembacaan dan penyimpanan data konfigurasi. Dalam modul *configuration* terdapat satu *sub*, yaitu modul *reader*.

1. Modul *reader*

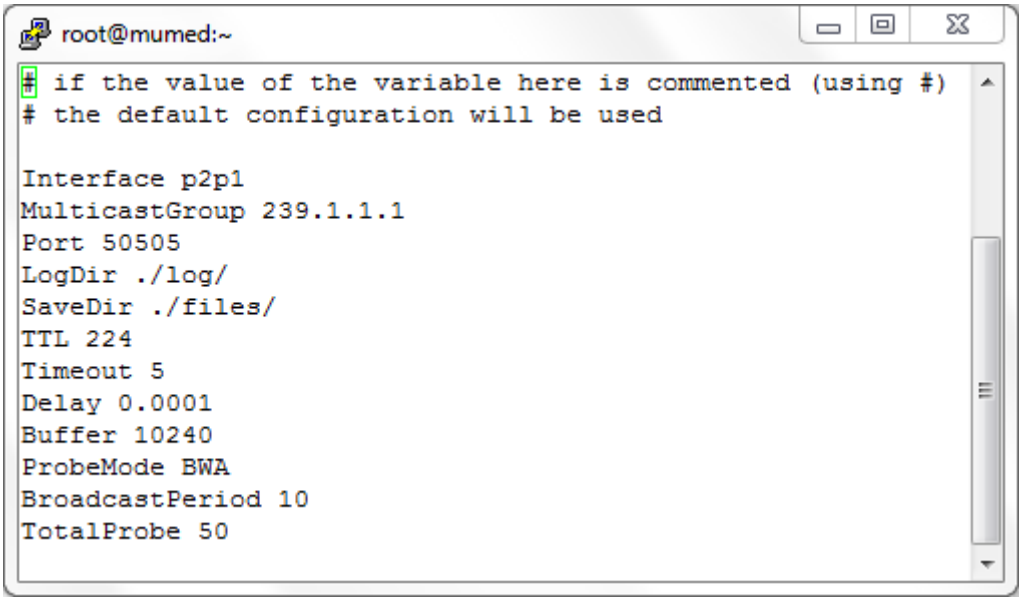
   Modul *reader* berisi fungsi-fungsi yang berkaitan dengan pembacaan sekaligus penyimpanan data konfigurasi dari program. Contoh *file* konfigurasi yang dibaca

[3] https://fedoraproject.org/

oleh modul ini dapat dilihat pada gambar 3.10. Dalam *file* konfigurasi tersebut terdapat beberapa parameter yang dapat dirubah. Parameter tersebut adalah *Interface, MulticastGroup, Port, LogDir, SaveDir, TTL, Timeout, Delay, Buffer, ProbeMode, BroadcastPeriod* dan *TotalProbe.*



Gambar 3.9 Modul Program



Gambar 3.10 Contoh *Configuration File*

3.4.2 Modul *file*

Modul *file* berisi fungsi-fungsi yang berkaitan dengan pengelolaan *file*. Di dalam *sub* modul *file* terdapat modul *cache*, *meta*, *sha*.

1. Modul *cache*

   Modul *cache* digunakan untuk melakukan berbagai proses *caching*, baik *cache* pengiriman maupun penerimaan. *Cache* bermanfaat untuk menyimpan data sementara dalam RAM sebelum data tersebut dikirimkan atau disimpan ke dalam HDD. Tujuan dari penggunaan cache ini adalah untuk mengurangi frekuensi pembacaan dan penulisan data ke dalam HDD, sehingga diharapkan dapat mempercepat proses pengiriman *file*.

2. Modul *meta*

   Modul *meta* digunakan untuk membaca metadata dari *file* yang akan dikirim. Metadata yang dimaksud adalah nilai SHA-1[4] dan jumlah segmen data yang terbentuk dari *file* yang akan dikirimkan. Nilai SHA-1 akan didapatkan dari hasil kalkulasi pada modul *sha*.

3. Modul *sha*

   Modul *sha* digunakan untuk melakukan kalkulasi nilai SHA-1 dari *file* yang akan dikirim. Nilai SHA-1 yang didapatkan memiliki panjang 20 *byte string* yang terdiri dari karakter ASCII dan non-ASCII. Namun bila kita mengubah nilai SHA-1 ke dalam bentuk *hexadecimal*, maka nilai SHA-1 yang didapatkan memiliki panjang 40 *byte*

3.4.3 Modul *netiface*

Modul *netiface* berisi fungsi-fungsi yang berkaitan dengan *network interface* dari komputer. Di dalamnya terdapat modul *ip*.

1. Modul *ip*

   Modul *ip* digunakan untuk fungsi-fungsi yang berkaitan dengan IP dalam sebuah *interface*. Di dalam modul ini terdapat fungsi untuk mendapatkan IP dari *interface* tertentu dan fungsi untuk mendapatkan IP seluruh interface yang terpasang dalam komputer.

---

[4] https://docs.python.org/2/library/sha.html

3.4.4 Modul *packet*

Modul *packet* berisi fungsi-fungsi yang berkaitan dengan pemrosesan paket data, baik pemrosesan paket sebelum data dikirimkan maupun pemrosesan paket setelah data dikirimkan. Didalamnya terdapat modul *encoder*, *decoder* dan *ptype*.

1. Modul *encoder*

   Modul *encoder* digunakan untuk melakukan proses *encoding* terhadap paket sebelum paket tersebut dikirimkan. Proses *encoding* yang dimaksud disini adalah proses konversi dari dari tipe data Python menjadi C *structs*. Python merepresentasikan hasil konversi ini dalam bentuk Python *string*. Data hasil konversi inilah yang nantinya akan dikirimkan melalui jaringan.

2. Modul *decoder*

   Modul *decoder* digunakan untuk melakukan proses *decoding* terhadap paket ketika paket tersebut diterima. Proses *decoding* yang dimaksud disini adalah proses kofersi dari C *structs* menjadi tipe data Python. Tujuan dari konversi ini adalah agar data yang diterima dapat dipergunakan dalam program dengan karakteristik yang sama seperti tipe data Python.

3. Modul *ptype*

   Modul ip digunakan untuk mendefinisikan tipe paket yang dikirimkan. Tujuan dari pemberikan tipe paket yang berbeda adalah agar dapat membedakan satu jenis paket dengan jenis lainnya bila paket dikirimkan pada alamat port yang sama.

3.4.5 Modul *protocol*

Modul *protocol* berisi fungsi-fungsi yang berkaitan dengan protokol yang digunakan dalam penelitian ini. Didalamnya terdapat modul *discovery*, *filesender*, dan *filereceiver*.

1. Modul *discovery*

   Modul ip digunakan untuk menjalankan proses *discovery*. Proses *discovery* pada modul ini mencakup proses pengiriman paket *probe*, proses perhitungan nilai reliabilitas *node* berdasarkan data yang diperoleh dari paket probe dan pembandingan tingkat reliabilitas antar *backup node* hingga didapatkan hasil akhir berupa daftar *backup node* yang diurutkan mulai dari backup *node* dengan

tingkat reliabilitas tertinggi hinga backup *node* dengan tingkat reliabilitas terendah.

2. Modul *filesender*

   Modul *filesender* digunakan untuk menjalankan proses pengiriman *file* dan pengaturan kecepatan pengiriman paket data. Dalam modul ini juga terjadi proses konversi dari tipe data Python menjadi paket data menggunakan program pada modul *decoder*.

3. Modul *filereceiver*

   Modul *filereceiver* digunakan untuk menjalankan proses penerimaan segmen *file* dan pengaturan penyimpanan sementara segmen dalam *cache* sebelum ditulis ke dalam HDD. Dalam modul ini juga terjadi proses konversi dari paket data menjadi tipe data Python menggunakan program pada modul *decoder*.

3.4.6 Modul *sock*

Modul *sock* berisi fungsi-fungsi yang berkaitan dengan socket. Didalamnya terdapat modul *multicast* dan *udp*.

1. Modul *multicast*

   Modul *multicast* berisi fungsi-fungsi yang berkaitan dengan pendefinisian *socket multicast*, proses *join* ke dalam *multicast group* dan prosen *leave* dari *multicast group*.

2. Modul *udp*

   Modul *udp* berisi fungsi-fungsi yang berkaitan dengan pendefinisian *socket* udp yang bersifat *unicast*.

3.5 Perancangan *Testbed*

Desain *testbed* pada penelitian ini bisa dilihat pada Gambar 3.10. *Testbed* ini nantinya akan dibuat menggunakan applikasi VirtualBox. VirtualBox adalah perangkat lunak untuk membuat sebuah komputer *virtual*, yaitu perangkat lunak yang digunakan untuk menjalankan *software* dan *hardware* komputer secara *virtual* (*virtual computer*) diatas *hardware* dan *software non virtual* yang berjalan (*physical computer*).

Spesifikasi perangkat keras (*physical computer*) yang digunakan pada penelitian ini adalah sebagai berikut:

- *Processor* : Intel Core i7-3770, CPU @ 3.40 GHz
- RAM : 8 GB
- *Hard disk* : 2 x 1 TB
- Sistem Operasi : Windows 7 Home Premium 64-*bit*

Adapun setiap *node* pada *testbed* ini (*virtual computer*) memiliki spesifikasi sebagai berikut:

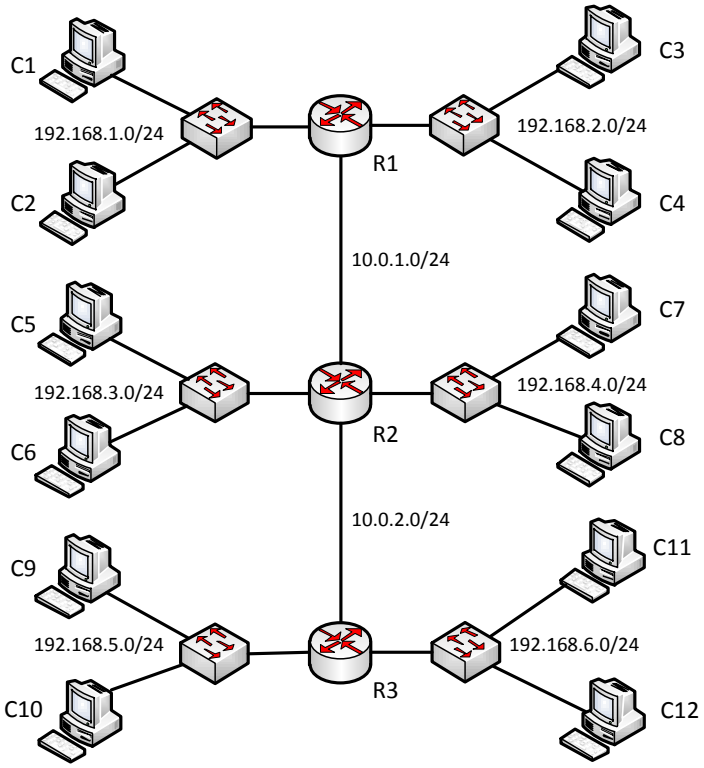
- *Processor* : Virtual Processor
- RAM : 256 MB
- *Hard disk* : 6 GB
- Sistem Operasi : Fedora Linux 15, 32-*bit*

Tiap *router* pada *testbed* ini dilengkapi dengan aplikasi XORP versi 1.8.5[5] yang digunakan untul melakukan proses *routing* baik *unicast routing* maupun *multicast routing*. Protokol yang digunakan untuk *unicast routing* adalah *Open Shortest Path First* (OSPF). Sedangkan protokol yang digunakan untuk *multicast routing* adalah *Internet Group Management Protocol* (IGMP) dan *Protocol Independent Multicast Sparse Mode* (PIM-SM). Semua protokol tersebut telah disediakan oleh XORP.

Alasan penggunaan *routing protocol* dinamis pada percobaan ini adalah untuk mensimulasikan kondisi jaringan *multicast* di lapangan yang lebih banyak menggunakan *routing protocol* dinamis. Dimana pada *routing protocol* dinamis terdapat trafik-trafik antara *end node* dan *router* yang bisa jadi mempengaruhi proses penyeleksian *backup node*. Sehingga diharapkan hasil dari percobaan semakin mencerminkan kondisi jaringan *multicast* di lapangan.

Untuk mensimulasikan *delay*, *packet loss*, dan beberapa perilaku jaringan lainnya digunakan *tools netem* (Network Emulation) yang sudah tertanam dalam kernel linux versi 2.6 keatas (Linux, 2009). *Tools netem* tersebut akan dipasang pada setiap *router* tersebut. Beberapa link pada *testbed* tersebut akan diberikan *delay* dan *packet loss* yang berbeda.

[5] http://www.xorp.org/




```
Konfigurasi Interface Router:

R1 - p2p1 - 192.168.1.1/24
   - p7p1 - 192.168.2.1/24
   - p8p1 - 10.0.1.1/24

R2 - p2p1 - 192.168.3.2/24
   - p7p1 - 192.168.4.2/24
   - p8p1 - 10.0.1.2/24

R3 - p2p1 - 192.168.5.3/24
   - p7p1 - 192.168.6.3/24
   - p8p1 - 10.0.2.3/24
```

```
Konfigurasi Interface Client:

C1  - p2p1 - 192.168.1.101/24
C2  - p2p1 - 192.168.1.102/24
C3  - p2p1 - 192.168.2.103/24
C4  - p2p1 - 192.168.2.104/24
C5  - p2p1 - 192.168.3.105/24
C6  - p2p1 - 192.168.3.106/24
C7  - p2p1 - 192.168.4.107/24
C8  - p2p1 - 192.168.4.108/24
C9  - p2p1 - 192.168.5.109/24
C10 - p2p1 - 192.168.5.110/24
C11 - p2p1 - 192.168.6.111/24
C12 - p2p1 - 192.168.6.112/24
```

Gambar 3.11 *Topology* Jaringan *Testbed*

3.6 Pengujian

Terdapat beberapa skenario pengujian yang bertujuan untuk membandingkan protokol yang diusulkan dengan protokol NAK yang mengirimkan NAK langsung ke *node* pengirim. Untuk selanjutnya, protokol yang disusulkan dalam penelitian ini disebut dengan metode *Multi-Nodes-Based Loss-Recovery*, disingkat dengan nama metode MNB, dan protokol NAK yang mengirimkan NAK langsung ke *node* pengirim disebut dengan metode *Sender-Based Loss-Recovery*, disingkat dengan nama metode SB,. Skenario-skenario yang dipergunakan dalam penelitian ini akan dijelaskan pada

sub bab 3.6.1 hingga 3.6.4. Nantinya tiap skenario akan diuji sebanyak 10 kali untuk meningkatkan akurasi dari hasil penelitian ini.

3.6.1 Pengujian Berdasarkan Perbedaan Ukuran *File*

Pada skenario pengujian ini akan digunakan beberapa ukuran *file* yang bervariasi antara 64 MB hingga 512 MB. Tujuan dari variasi ukuran *file* ini adalah untuk mengetahui kinerja dari protokol saat diuji menggunakan *file* berukuran kecil hingga besar. Terdapat 10 skenario percobaan yang secara detail dapat dilihat pada Tabel 3.1.

Tabel 3.1 Skenario Perbedaan Ukuran *File*

Kode	**Metode**	**Ukuran *File* (MB)**	**Ukuran *Cache* (MB)**	***Packet loss* (%)**
1.1	SB	32	14	0
1.2	MNB			
1.3	SB	64		
1.4	MNB			
1.5	SB	128		
1.6	MNB			
1.7	SB	256		
1.8	MNB			
1.9	SB	512		
1.10	MNB			

3.6.2 Pengujian Berdasarkan Perbedaan Ukuran Cache

Ukuran *Cache* yang dimaksud dalam dalam skenario ini adalah besar *primary memory* (RAM) maksimal yang boleh dipergunakan untuk menyimpan segmen *file* yang dikirim atau diterima sebelum dituliskan ke dalam *secondary memory* (HDD). Tujuan dari penggunaan *cache* yang bervariasi ini adalah untuk mencari tahu pengaruh ukuran *cache* terhadap kinerja dari protokol yang diusulkan. Terdapat 6 skenario percobaan yang secara detail dapat dilihat pada Tabel 3.2.

Tabel 3.2 Skenario Perbedaan Ukuran *Cache*

Kode	**Metode**	**Ukuran *Cache* (MB)**	**Ukuran *File* (MB)**	***Packet loss* (%)**
2.1	SB	3.5	128	0
2.2	MNB			
2.3	SB	7		
2.4	MNB			
2.5	SB	14		
2.6	MNB			

3.6.3 Pengujian Berdasarkan Perbedaan Jumlah Backup Node

Tujuan dari jumlah *backup node* yang bervariasi skenario ini adalah untuk mencari tahu pengaruh jumlah *backup node* terhadap kinerja dari protokol yang diusulkan. Terdapat 4 skenario percobaan yang secara detail dapat dilihat pada Tabel 3.3.

Tabel 3.3 Skenario Perbedaan Jumlah *Backup Node*

Kode	**Jumlah *Backup node***	**Ukuran *Cache* (MB)**	**Ukuran *File* (MB)**	***Packet loss* (%)**
3.1	1	14	128	0
3.2	6			
3.3	9			
3.4	11			

Jumlah *backup node* adalah banyaknya *backup node* yang digunakan dalam satu percobaan. *Backup node* berjumlah 1 menandakan bahwa metode yang digunakan adalah metode *Sender-Based Loss-Recovery* (SB). Sedangkan *backup node* dengan jumlah lebih dari 1 menandakan bahwa metode yang digunakan adalah metode *Multi-Nodes-Based Loss-Recovery* (MNB).

3.6.4 Pengujian Berdasarkan Kemampuan Menangani Packet Loss

Nilai *packet lost* adalah besarnya *paket loss* yang dibangkitkan menggunakan *network emulator* (netem) pada linux. Terdapat 2 *subnet* yang akan dibangkitkan *packet loss*-nya yaitu *subnet* 192.168.3.0/24 dan 192.168.5.0/25. *Packet loss* sengaja tidak

dibangkitkan pada seluruh *subnet* karena tujuan dari pembangkitan ini adalah untuk mencari dan membandingkan dampak dari adanya *packet loss* pada tiap *node* penerima. Terdapat 6 skenario percobaan yang secara detail dapat dilihat pada Tabel 3.4.

Tabel 3.4 Skenario Kemampuan Menangani *Packet loss*

Kode	**Metode**	***Packet loss* (%)**	**Ukuran *Cache* (MB)**	**Ukuran *File* (MB)**
4.1	SB	0	14	128
4.2	MNB			
4.3	SB	10		
4.4	MNB			
4.5	SB	20		
4.6	MNB			

3.7 Analisis Hasil Pengujian

Untuk membandingkan kinerja antara metode *Multi-Nodes-Based Loss-Recovery* (MNB) dengan metode *Sender-Based Loss-Recovery* (SB), ada beberapa parameter hasil pengujian yang dibandingkan dan dianalisis, yaitu:

1. Rata-rata NAK yang seharusnya dikirim dan rata-rata pengiriman ulang NAK pada tiap *node*, yang didapatkan dari dua rumus berikut:

$$Nak_{rate} = \frac{\sum_{i=1}^{N} Nak_i}{N} \quad (3.2)$$

$$Err_{rate} = \frac{\sum_{i=1}^{N} Err_i}{N} \quad (3.3)$$

dimana:

Nak_{rate} = Rata-rata NAK yang seharusnya dikirim

Nak_i = Jumlah NAK yang seharusnya dikirim pada *backup node* ke-i

Err_{rate} = Rata-rata pengiriman ulang NAK

Err_i = Jumlah pengiriman ulang NAK pada *backup node* ke-i

N = Jumlah *backup node*

Semakin kecil nilai Nak_{rate} dan Err_{rate} maka semakin baik kinerja dari protokol yang digunakan.

2. Rata-rata total waktu *download*, yang didapatkan dari rumus:

$$Time_{rate} = \frac{\sum_{i=1}^{N} Time_i}{N} \quad (3.4)$$

dimana:

$Time_{rate}$ = Rata-rata total waktu *download*

$Time_i$ = Total waktu *download* pada *backup node* ke-i

N = Jumlah *backup node*

Semakin kecil nilai $Time_{rate}$ maka maka semakin baik kinerja dari protokol yang digunakan.

3. Rata-rata *bandwidth* yang terpakai pada tiap *node*, yang didapatkan dari rumus:

$$Bw_{rate} = \frac{\sum_{i=1}^{N} Bw_i}{N} \quad (3.5)$$

dimana:

Bw_{rate} = Rata-rata *bandwidth* yang terpakai

Bw_i = Total *bandwidth* yang terpakai pada *backup node* ke-i

N = Jumlah *backup node*

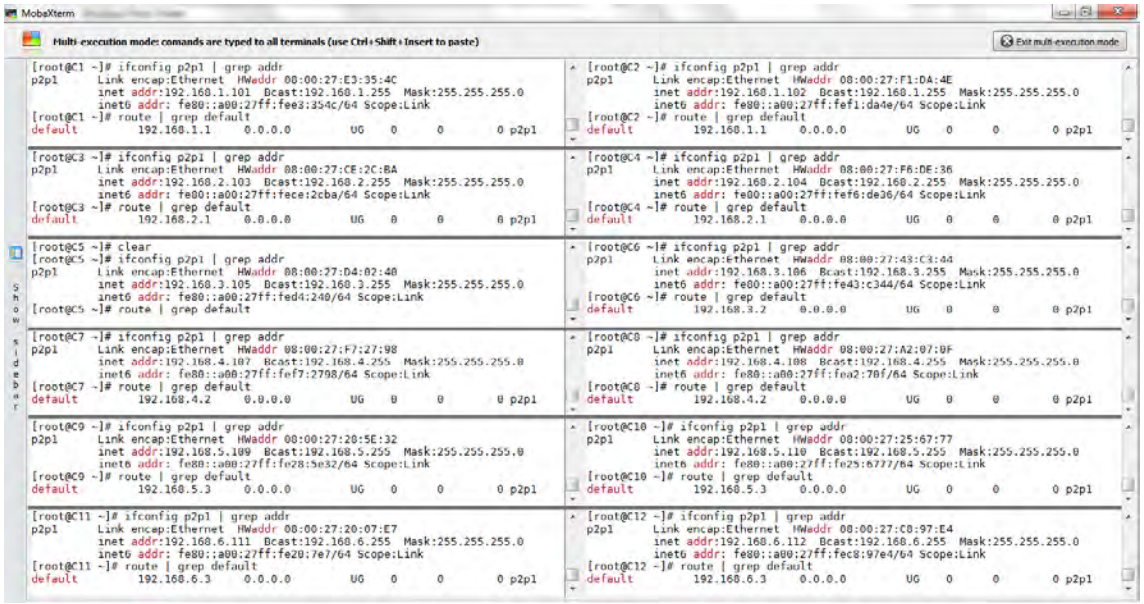
Semakin kecil Bw_{rate} yang digunakan maka semakin baik kinerja dari protkol yang digunakan. Bw_{rate} yang akan diukur dalam penelitian ini adalah total dari:

a. Bw_{rate} untuk pengiriman *file*
b. Bw_{rate} untuk pengiriman NAK
c. Bw_{rate} untuk melakukan proses pencarian *backup node*.

BAB 4
HASIL DAN PEMBAHASAN

4.1 Hasil Uji Coba

Percobaan dilakukan menggunakan 12 *node*, dimana salah satu *node* berperan sebagai *node* pengirim dan *node* lainnya berperan sebagai *node* penerima. Gambar 4.1 adalah konfigurasi *network interface* pada 12 *node* tersebut. 12 *node* tersebut berada dalam 6 *subnet* yang berbeda. 6 *subnet* tersebut terhubung dengan 3 *router* sesuai dengan *topology* pada Gambar 3.11.

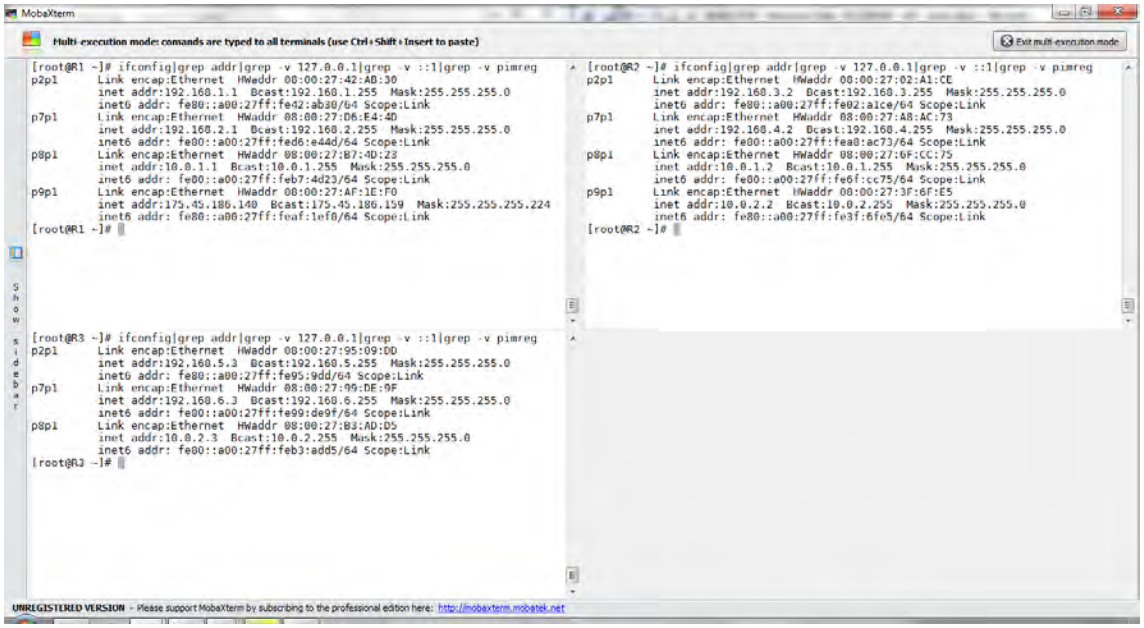


Gambar 4.1 Konfigurasi *Network Interface* Pada Tiap *Client*

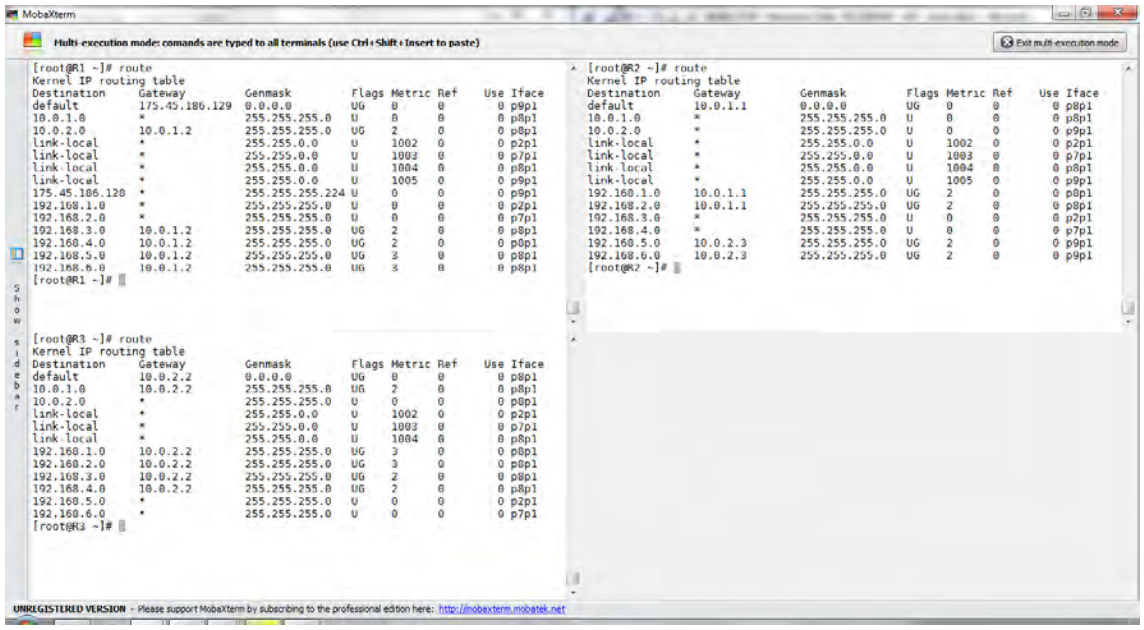
Router pada percobaan ini memiliki beberapa *network interface* yang terhubung dengan *subnet* tertentu. Konfigurasi *network interface* pada tiap *router* pada percobaan ini dapat dilihat pada Gambar 4.2. Pada tiap *router* telah dilengkapi dengan program XORP yang digunakan untuk melakukan *routing unicast* dan *multicast* secara dinamis.

Gambar 4.3 menunjukkan tabel *unicast routing* yang dihasilkan oleh program XORP pada tiap *router*. Tabel *unicast routing* relatif tidak banyak mengalami

perubahan. Tabel *unicast routing* hanya akan berubah ketika terjadi perubahan *topology* jaringan, seperti terdapat *router* yang *down* atau terdapat jaringan yang putus.

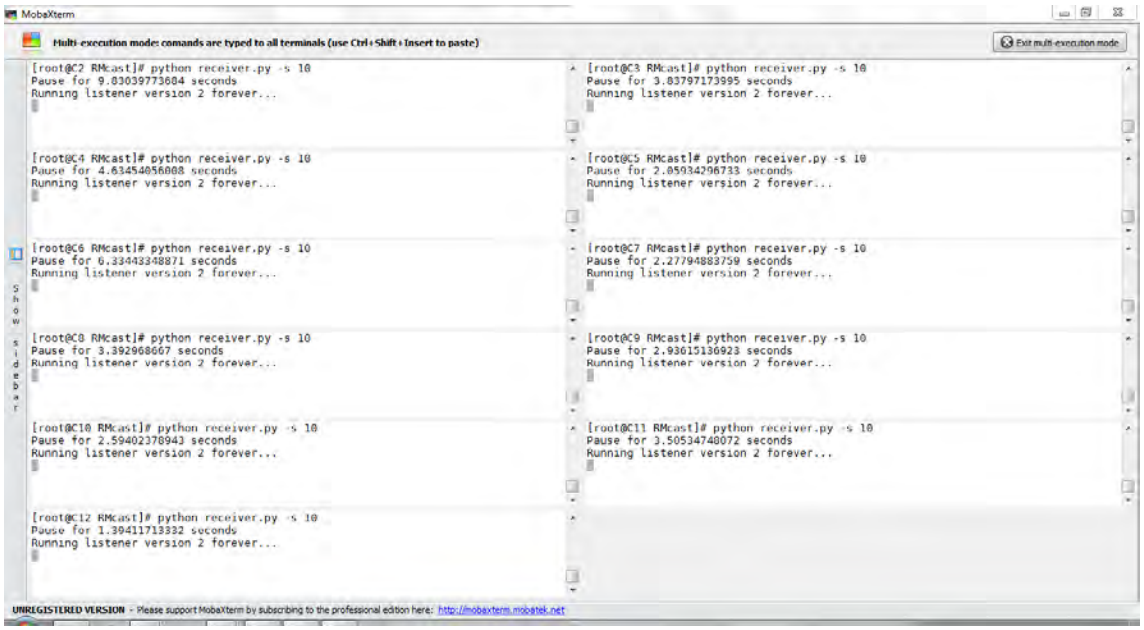


Gambar 4.2 Konfigurasi *Network Interface* Pada Tiap *Router*



Gambar 4.3 *Unicast Routing Table* Pada Tiap *Router*

Berbeda dengan tabel *unicast routing*, tabel *multicast routing* mengalami perubahan secara dinamis menyesuaikan dengan sumber dan tujuan pengiriman paket *multicast*. Ketika terdapat *node receiver* dalam sebuah *subnet*, *router* akan memodifikasi tabel *multicast routing* agar paket *multicast* dapat sampai ke *node receiver*.

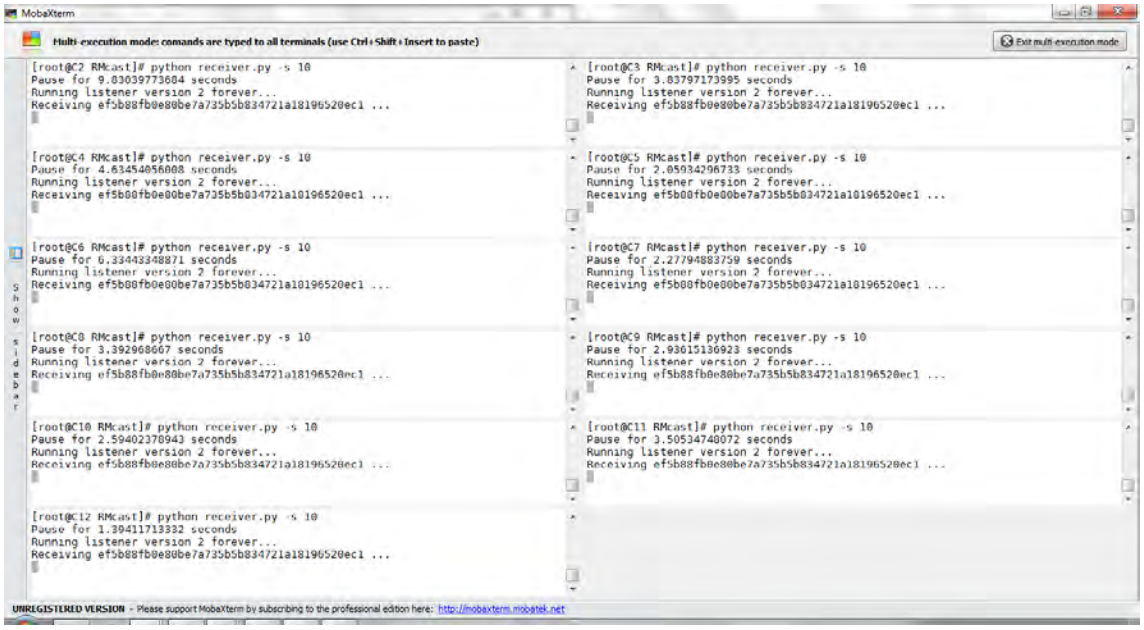


Gambar 4.4 Program *Multicast Receiver* Pada Tiap *Node* Penerima

Gambar 4.4 adalah tampilan ketika program dijalankan pada sisi *node* penerima. Pada tahap ini, tiap *node* penerima akan menunggu kiriman paket data *multicast* pada alamat *multicast gorup* yang telah ditentukan.

Gambar 4.5 Program *Multicast Sender* Pada *Node* Pengirim

Gambar 4.5 adalah tampilan ketika program dijalankan pada sisi *node* pengirim. *Node* pengirim akan mengirimkan *file* ke alamat *multicast group*. Pengiriman ini akan berjalan hingga seluruh segmen *file* berhasil terkirim dan tidak ada lagi permintaan pengiriman ulang dari *node* penerima dalam jangka waktu tertentu.



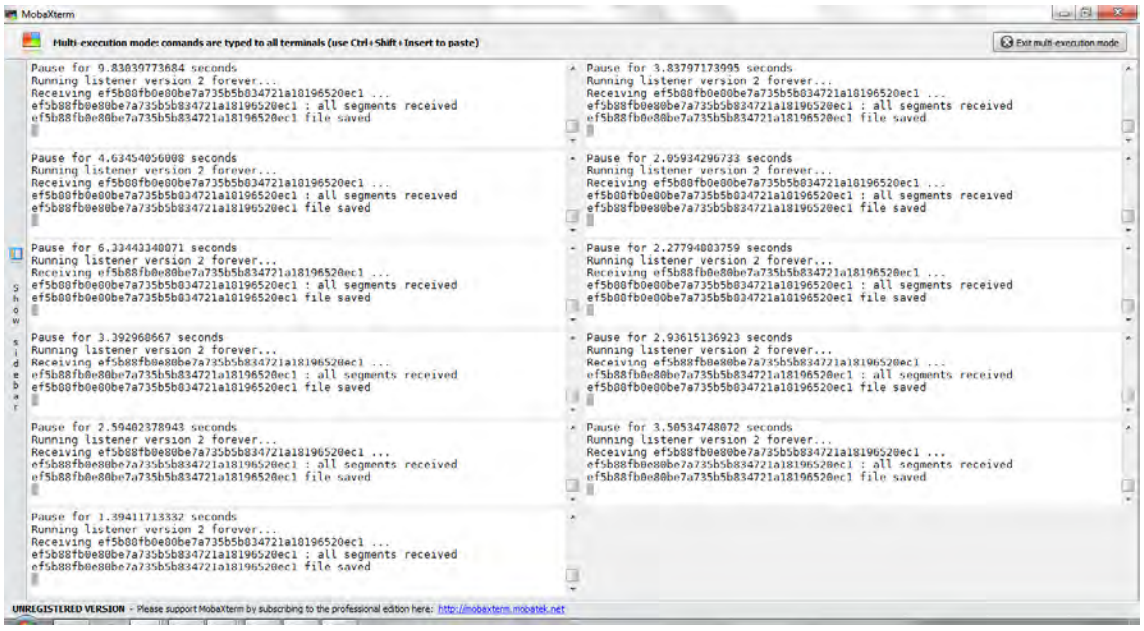
Gambar 4.6 Proses Penerimaan *File* Pada Tiap *Node* Penerima

Gambar 4.6 adalah tampilan program saat menerima segmen *file*. Program akan terus dalam kondisi ini hingga seluruh segmen *file* diterima. Bila dalam waktu tertentu program tidak menerima seluruh segmen *file* maka program akan dihentikan dan pengiriman dianggap gagal. Dalam proses ini juga terjadi permintaan ulang segmen yang hilang ke backup *node* yang memiliki nilai reliabilitas terbaik.

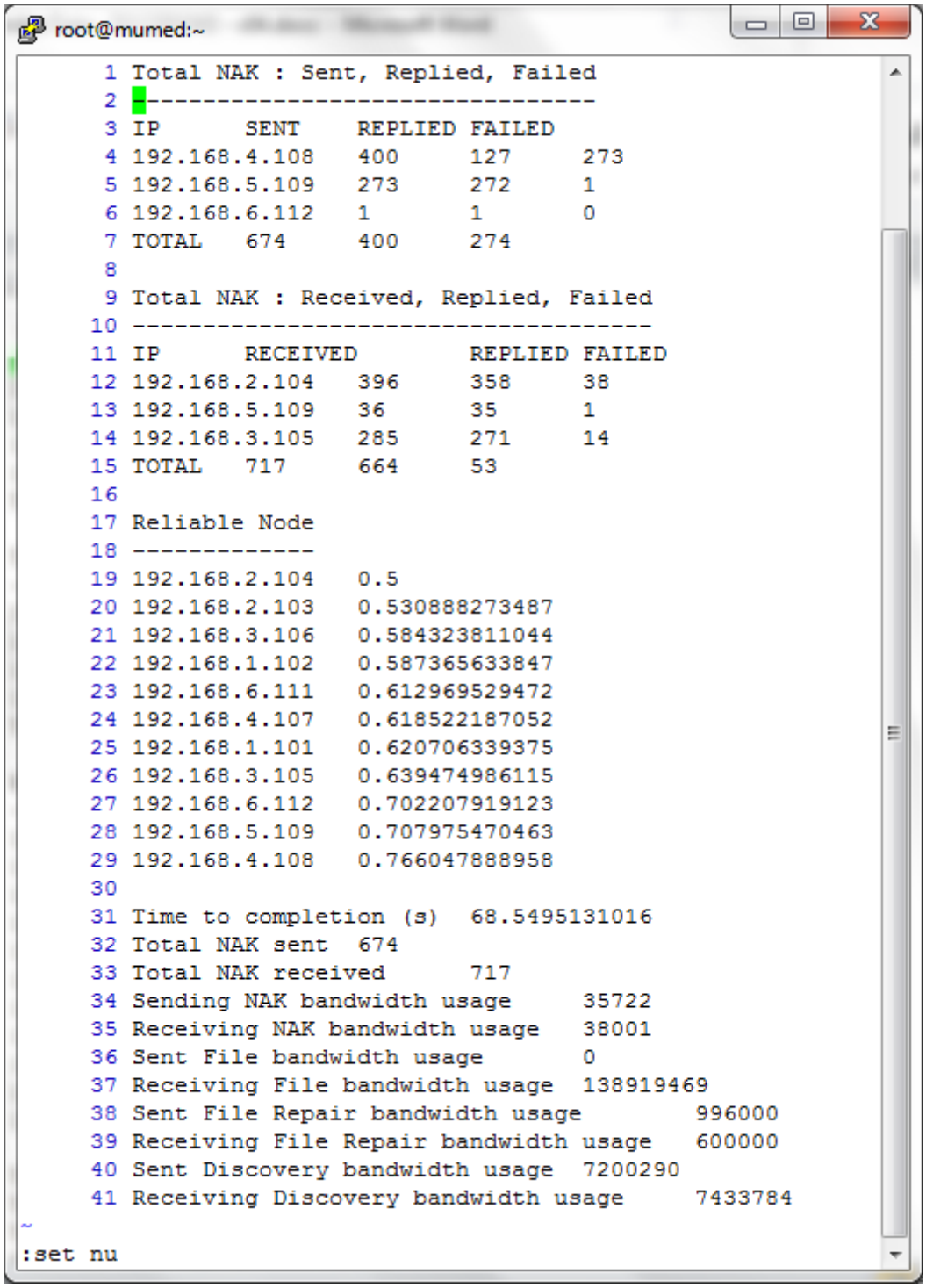
Gambar 4.7 adalah tampilan program saat *node* pengirim selesai mengirimkan segmen *file*. Program tidak akan langsung diakhiri karena masih menunggu NAK yang mungkin saja dikirimkan oleh *node* penerima. Lama waktu menunggu ini tergantung dari banyaknya permintaan perbaikan segmen *file* yang masuk dan lama waktu tunggu yang ditentukan pada *file* konfigurasi program.

Gambar 4.7 Proses Pengiriman *File* Selesai

Gambar 4.8 adalah tampilan program pada *node* penerima ketika seluruh segmen telah diterima dan proses rekonstruksi *file* selesai dilakukan. Ketika *node* penerima telah menerima seluruh segmen *file* maka program akan merekonstruksi segmen *file* tersebut dan menyimpannya ke dalam *file* yang telah ditentukan. Bila terdapat segmen *file* yang tidak terkirim dan hingga jangka waktu yang telah ditentukan maka pengiriman dianggap gagal.



Gambar 4.8 Seluruh *Node* Penerima Berhasil Menerima *File*



Gambar 4.9 Contoh *File Log* Pada *Node* 10, Skenario 1.6

Uji coba dilakukan dengan melakukan percobaan sesuai dengan skenario pada Tabel 3.1 hingga 3.4. Untuk tiap skenario, uji coba dilakukan sebanyak 10 kali untuk melihat pola data yang dihasilkan sekaligus menilai keakuratan data yang dihasilkan.

Dari hasil uji coba ini, tiap *node* akan menghasilkan *file log* berupa text yang kemudian diolah menjadi tabel dan grafik untuk dianalisis. Gambar 4.9 adalah contoh *file log* yang dihasilkan oleh program pada *node* 10, dengan IP 192.168.5.110, saat percobaan ke-5 menggunakan skenario dengan kode 1.6 (lihat Tabel 3.1).

Bagian pertama dari *file log* ini adalah data mengenai jumlah dan *node* tujuan pengiriman NAK (lihat baris ke-1 hingga 7). Pada contoh *file log* Gambar 4.9, *node* 5 mula-mula mengirimkan NAK sebanyak 400 kali ke *backup node* dengan IP 192.168.4.108 (pada rumus 3.2, nilai ini disimbolkan dengan Nak_i), namun hanya mendapat balasan sebanyak 127 kali, sedangkan sisanya sebanyak 273 tidak terbalas. Sisa NAK yang tidak terbalas ini kemudian dikimkan ke *backup node* lain dengan IP 192.168.5.109 dan mendapat balasan sebanyak 272 kali. Sisa 1 NAK yang tidak terbalas dikirimkan ke *backup node* selanjutnya dengan IP 192.168.6.112 dan sukses mendapat balasan. Sehingga dapat ditarik kesimpulan bahwa *node* 10 mengalami *packet loss* sebanyak 400 *packet*, lalu mengirimkan NAK sebanyak 400 kali. Dari pengiriman NAK tersebut, terdapat 247 NAK yang tidak mendapat balasan sehingga mengirimkan NAK ulang sebanyak 247 kali (pada rumus 3.3, nilai ini disimbolkan dengan Err_i). Bila dijumlahkan, total NAK yang dikirimkan sebanyak 674 kali.

Bagian kedua adalah data mengenai jumlah dan pengirim NAK yang diterima oleh *node* 10 (lihat baris ke-9 hingga 15). *Node* 10 menerima NAK dari beberapa *node*. Dari *node* 192.168.2.104 menerima sebanyak 396 kali, sukses membalas sebanyak 358 kali, gagal membalas (karena segmen yang diminta tidak ada) sebanyak 38. Dari *node* 192.168.2.109 menerima sebanyak 36 kali, sukses membalas sebanyak 35 kali, gagal membalas sebanyak 1 kali. Dari *node* 192.168.2.105 menerima sebanyak sebanyak 285 kali, sukses membalas sebanyak 271 kali, gagal membalas sebanyak 14 kali. Sehingga dapat ditarik kesimpulan bahwa *node* 10 menerima NAK sebanyak 717 kali, sukses membalasan sebanyak 664 kali, dan gagal membalasan sebanyak 53 kali.

Bagian ketiga adalah data mengenai *reliable node* beserta nilai R_n yang dihasilkan dari rumus 3.1. Pada contoh *log* pada Gambar 4.9 terdapat 11 *backup node* yang diurutkan dari *backup node* dengan nilai R_n terendah hingga tertinggi (lihat baris ke-17 hingga 29). *Node* dengan nilai R_n tertinggi akan dipilih sebagai target pengiriman NAK terlebih dahulu.

Bagian keempat (lihat baris ke-31 hingga 41) adalah data yang berisi total waktu *download*, jumlah *bandwidth* yang dihabiskan untuk pengiriman dan penerimaan NAK, *file*, perbaikan *file* dan proses *discovery reliable node*. Waktu *download* ditampilkan dalam satuan detik, sedangkan penggunaan *bandwidth* disimpan dalam satuan *byte*.

Dari seluruh percobaan pada penelitian ini menghasilkan *file log* sebanyak 3120. *File log* ini dihasilkan dari 12 *node* yang diuji menggunakan 26 skenario pengujian yang berbeda dimana masih-masing skenario diulang sebanyak 10 kali. *File log* ini kemudian diolah menggunakan bahasa pemrograman *Python* sehingga menghasilkan data sesuai dengan rumus 3.2, 3.3, 3.4 dan 3.5.

Ada 4 parameter yang dibandingkan pada tiap uji coba, yaitu jumlah NAK yang dihasilkan dan jumlah NAK yang tidak mendapat balasan (rumus 3.2 dan rumus 3.3), total waktu *download* (rumus 3.3) dan rata-rata penggunaan *bandwidth* (rumus 3.4). Data selengkapnya dari hasil percobaan ini dapat dilihat pada Lampiran 1, sedangkan analisis hasil uji coba tersebut akan dijabarkan pada sub bab 4.2.

4.2 Analisis Hasil Uji Coba

4.2.1 Analisis Berdasarkan Perbedaan Ukuran *File*

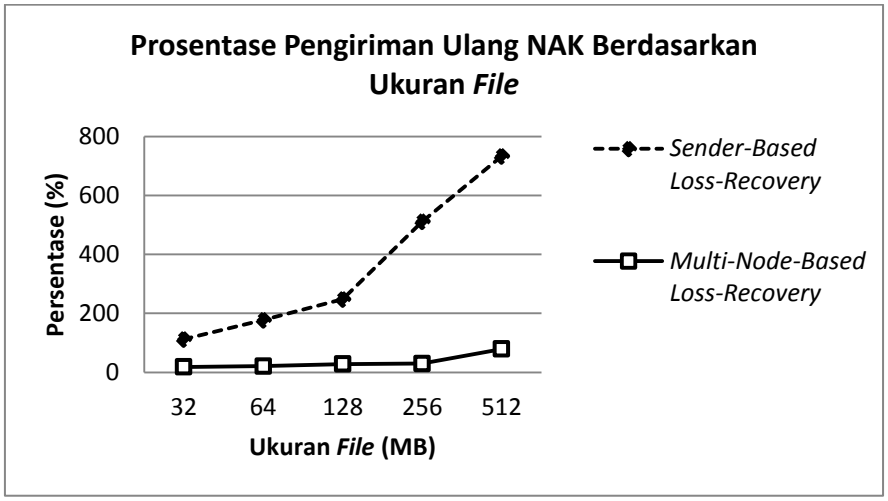
Percobaan dilakukan sesuai dengan skenario pada Tabel 3.1. Dari hasil pengolahan *log* hasil percobaan didapatkan beberapa nilai yang menggambarkan perbandingan nilai antara metode *Sender-Based Loss-Recovery* dengan metode *Multi-Nodes-Based Loss-Recovery* ditinjau dari ukuran *file* yang dikirimkan.

Tabel 4.1 Perbandingan Rata-rata NAK Berdasarkan Ukuran *File*

Ukuran *File* (MB)	*Sender-Based Loss-Recovery*			*Multi-Nodes-Based Loss-Recovery*		
	Nak_{rate}	Err_{rate}	persentase Err_{rate}	Nak_{rate}	Err_{rate}	persentase Err_{rate}
32	147.4	165.7	112.4%	168.4	31.4	18.6%
64	273.2	484.5	177.4%	316.7	66.1	20.9%
128	469.2	1164.1	248.1%	328.2	92.5	28.2%
256	893.2	4565.3	511.1%	187.0	55.4	29.7%
512	1797.3	13182.1	733.4%	345.7	273.9	79.2%

Tabel 4.1 merupakan perbandingan rata-rata yang terjadi pada kedua metode. Nak_{rate} adalah rata-rata NAK yang seharusnya dikirim dan Err_{rate} adalah rata-rata pengiriman ulang NAK. *Persentase* Nak_{rate} adalah persentase yang didapatkan dari hasil perbandingan nilai Nak_{rate} dan Nak_{rate}. Nilai *persentase* Nak_{rate} ini kemudian divisualisasikan dalam grafik pada Gambar 4.10.

Table 4.1 menunjukkan bahwa metode *Multi-Nodes-Based Loss-Recovery* mampu menekan jumlah pengiriman ulang NAK dengan signifikan bila dibandingkan dengan metode *Sender-Based Loss-Recovery*. Pada *file* berukuran 32 MB, metode *Multi-Nodes-Based Loss-Recovery* melakukan pengiriman ulang NAK sebanyak 18.6% dari jumlah NAK yang seharusnya dikirimkan, sedangkan metode *Sender-Based Loss-Recovery* melakukan pengiriman ulang NAK sebanyak 112.4%. Persentase ini semakin meningkat seiring dengan peningkatan ukuran *file* yang dikirimkan seperti yang terlihat pada grafik di Gambar 4.10.

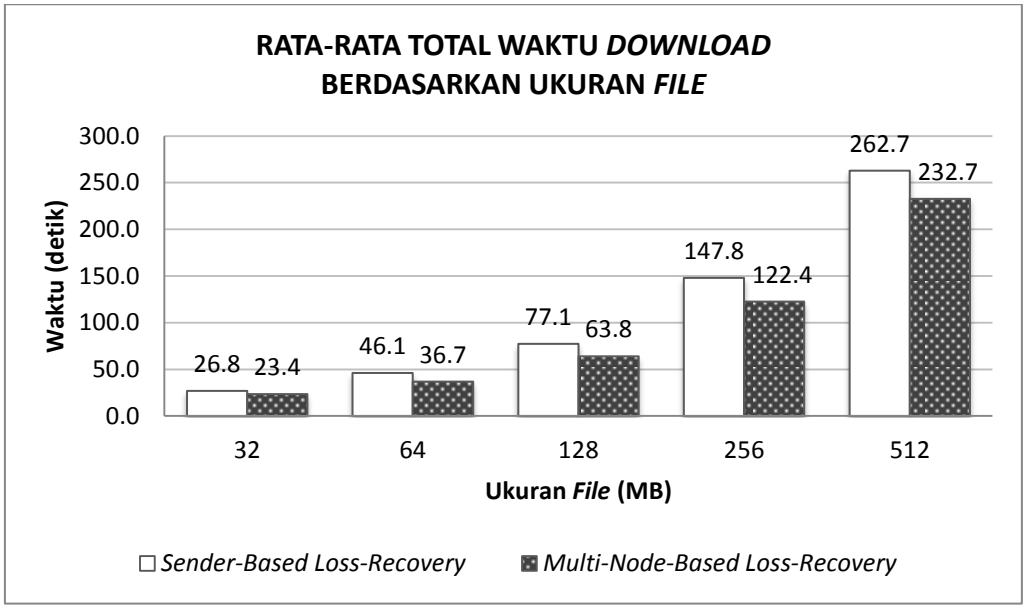


Gambar 4.10 Persentase Pengiriman Ulang NAK Berdasarkan Ukuran *File*

Dari data ini dapat disimpulkan bahwa dengan mendistribusikan pengiriman NAK terbukti dapat meningkatkan peluang terbalasnya paket NAK yang dikirim dengan cara mengurangi peluang "*bottle neck*" pada *node* pengirim. Pada metode *Sender-Based Loss-Recovery*, semakin besar *file* yang dikirimkan semakin besar pula *bottle neck* yang terjadi pada *node* pengirim, sehingga jumlah pengiriman ulang NAK semakin meningkat tajam. Namun hal ini tidak terjadi pada metode *Multi-Nodes-Based Loss-*

Recovery, dimana NAK didistribusikan ke *backup node* yang berbeda antara satu *node* dengan *node* yang lain.

Gambar 4.11 merupakan grafik rata-rata total waktu *download*. Dari grafik tersebut terlihat bahwa metode *Multi-Nodes-Based Loss-Recovery* memiliki total waktu *download* yang lebih sedikit bila dibandingkan dengan metode *Sender-Based Loss-Recovery*. Ketika ukuran *file* semakin besar, selisih waktu dari kedua metode ini juga semakin membesar. Waktu *download* pada *file* berukuran 32 MB, selisih waktu *download* adalah 3.4 detik sedangkan waktu *download file* berukuran 512 MB adalah 30.0 detik. Sehingga dapat disimpulkan bahwa dengan meningkatkan peluang terbalasnya paket NAK yang dikirim dapat menurunkan waktu *download* dari sebuah *file*.

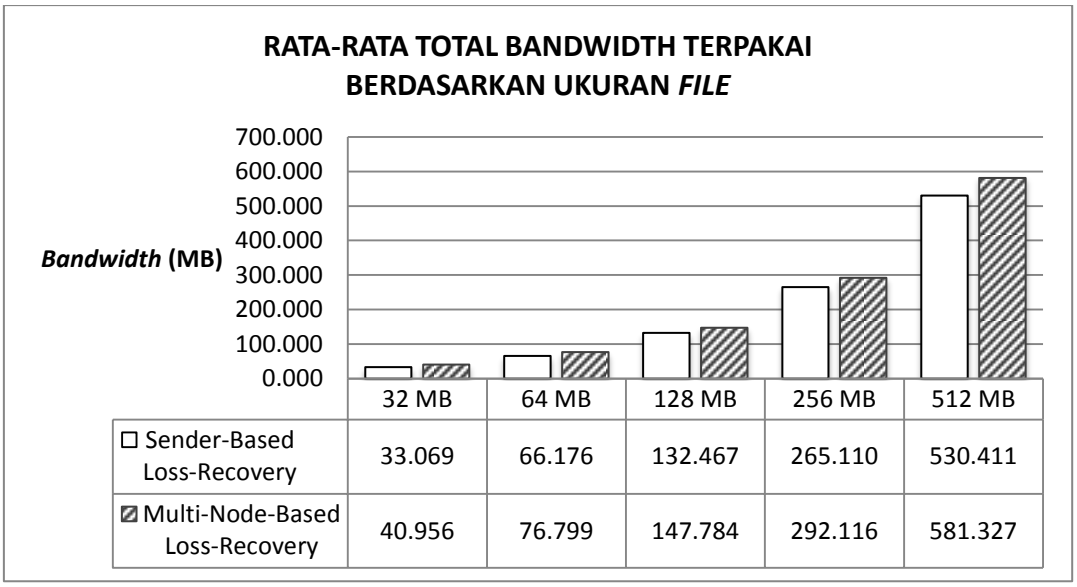


Gambar 4.11 Rata-rata Total Waktu *Donwload* Berdasarkan Ukuran *File*

Tabel 4.2 menunjukkan data penggunaan *bandwidth* untuk tiap jenis paket yang dikirimkan dengan ukuran *file* yang bervariasi. Data tersebut kemudian dijumlahkan sehingga menghasilkan Gambar 4.12.

Tabel 4.2 Rata-rata Penggunaan *Bandwidth* Berdasarkan Ukuran *File*

Ukuran *File* (MB)	Pengiriman Segmen *File*		Pengiriman NAK		Pengiriman Paket *Discovery*	
	Sender-Based Loss-Recovery	*Multi-Nodes-Based Loss-Recovery*	*Sender-Based Loss-Recovery*	*Multi-Nodes-Based Loss-Recovery*	*Sender-Based Loss-Recovery*	*Multi-Nodes-Based Loss-Recovery*
32	33.053	33.232	0.016	0.018	0	7.705
64	66.137	66.506	0.038	0.037	0	10.256
128	132.385	133.033	0.083	0.039	0	14.712
256	264.835	266.157	0.276	0.023	0	25.936
512	529.653	532.333	0.757	0.052	0	48.942



Gambar 4.12 Rata-rata *Bandwidth* Terpakai Berdasarkan Ukuran *File*

Gambar 4.12 menunjukkan bahwa metode *Multi-Nodes-Based Loss-Recovery* menggunakan *bandwidth* yang lebih banyak daripada metode *Sender-Based Loss-Recovery*. Seperti yang terlihat pada Tabel 4.2, perbedaan yang paling mencolok terletak pada pengiriman paket *discovery*. Metode *Multi-Nodes-Based Loss-Recovery* menggunakan *bandwidth* yang cukup banyak dalam melakukan proses pencarian *reliable node*, sedangkan metode *Sender-Based Loss-Recovery* tidak memerlukan proses ini sehingga tidak ada *bandwidth* yang terpakai untuk proses tersebut.

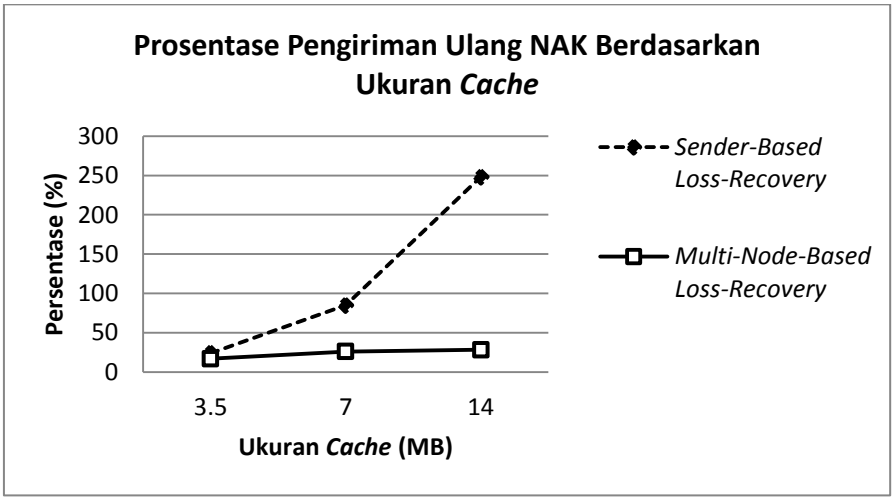
4.2.2 Analisis Berdasarkan Perbedaan Ukuran *Cache*

Percobaan dilakukan sesuai dengan skenario pada Tabel 3.2. Dari hasil pengolahan log hasil percobaan didapatkan beberapa nilai yang menggambarkan perbandingan nilai antara metode *Sender-Based Loss-Recovery* dengan metode *Multi-Nodes-Based Loss-Recovery* ditinjau dari ukuran *cache* yang digunakan.

Tabel 4.3 dan grafik pada Gambar 4.13 menunjukkan bahwa metode *Multi-Nodes-Based Loss-Recovery* memiliki persentase pengiriman ulang NAK yang lebih kecil dibandingkan dengan metode *Sender-Based Loss-Recovery* berapapun ukuran *cache* yang digunakan. Selain itu penggunaan *cache* berukuran 3.5 MB juga memiliki persentase pengiriman ulang NAK yang lebih kecil dibandingkan dengan penggunaan *cache* berukuran lebih besar, baik menggunakan metode *Multi-Nodes-Based Loss-Recovery* maupun menggunakan metode *Sender-Based Loss-Recovery*.

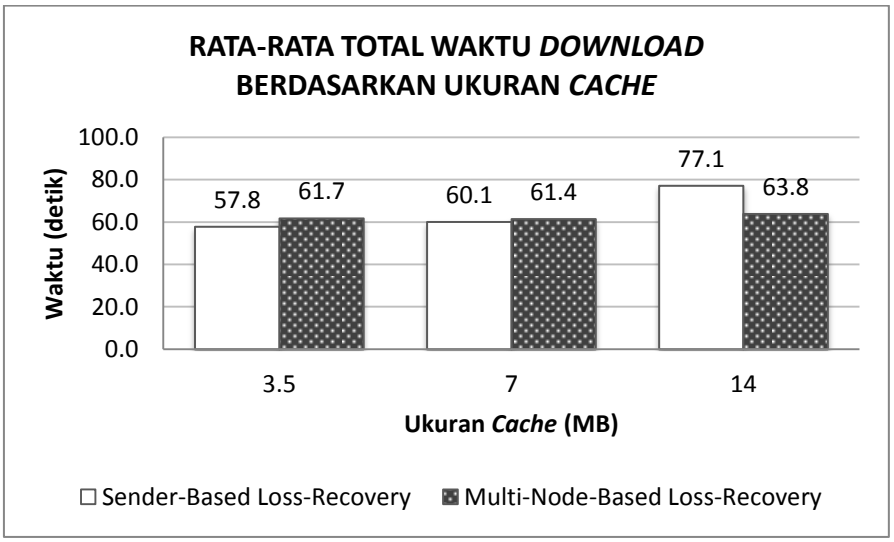
Tabel 4.3 Perbandingan Rata-rata NAK Berdasarkan Ukuran *Cache*

Ukuran *Cache* (MB)	***Sender-Based Loss-Recovery***			***Multi-Nodes-Based Loss-Recovery***		
	Nak_{rate}	Err_{rate}	persentase Err_{rate}	Nak_{rate}	Err_{rate}	persentase Err_{rate}
3.5	35.1	8.455	24.1%	49.736	8.436	17.0%
7	69.3	58.8	84.8%	76.4	19.864	26.0%
14	469.227	1164.055	248.1%	328.182	92.473	28.2%



Gambar 4.13 Persentase Pengiriman Ulang NAK Berdasarkan Ukuran *Cache*

Gambar 4.14 merupakan grafik rata-rata total waktu *download* berdasarkan ukuran *cache*. Dari grafik tersebut terlihat bahwa penggunaan *cache* berukuran 3.5 MB pada metode *Sender-Based Loss-Recovery* memiliki rata-rata total waktu terkecil. Penggunaan *cache* yang lebih besar cenderung meningkatkan rata-rata total waktu *download*. Pada metode *Multi-Nodes-Based Loss-Recovery*, penggunaan *cache* berukuran 7 MB memiliki rata-rata total waktu terkecil bila dibandingkan dengan penggunaan *cache* berukuran 3.5 MB dan 14 MB walaupun selisihnya tidak terlalu besar.



Gambar 4.14 Rata-rata Total Waktu *Donwload* Berdasarkan Ukuran *Cache*

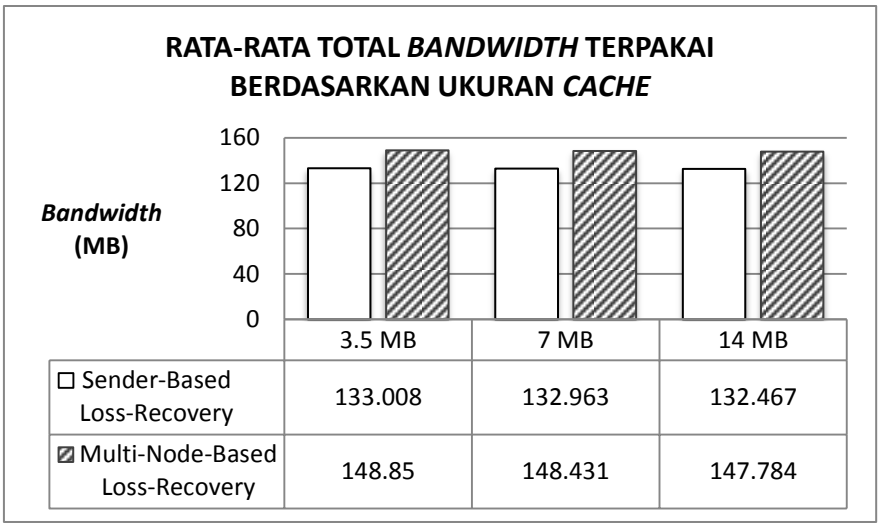
Tabel 4.4 menunjukkan data penggunaan *bandwidth* untuk tiap jenis paket yang dikirimkan dengan ukuran *cache* yang bervariasi. Data tersebut kemudian dijumlahkan sehingga menghasilkan Gambar 4.7.

\

Tabel 4.4 Rata-rata Penggunaan *Bandwidth* Berdasarkan Ukuran *Cache*

Ukuran *Cache* (MB)	**Pengiriman Segmen *File***		**Pengiriman NAK**		**Pengiriman Paket *Discovery***	
	Sender-Based Loss-Recovery	*Multi-Nodes-Based Loss-Recovery*	*Sender-Based Loss-Recovery*	*Multi-Nodes-Based Loss-Recovery*	*Sender-Based Loss-Recovery*	*Multi-Nodes-Based Loss-Recovery*
3.5	133.006	133.067	0.002	0.006	0	15.778
7	132.957	133.056	0.006	0.009	0	15.366
14	132.385	133.033	0.083	0.039	0	14.712

Gambar 4.15 menunjukkan bahwa metode *Multi-Nodes-Based Loss-Recovery* menggunakan *bandwidth* yang lebih banyak daripada metode *Sender-Based Loss-Recovery* dan ukuran *cache* tidak berpengaruh terlalu signifikan. Penggunaan *cache* sebesar 14 MB hanya menurunkan penggunaan *bandwidth* sekitar 0.647 MB bila dibandingkan dengan saat menggunakan *cache* berukuran 7 MB.



Gambar 4.15 Rata-rata *Bandwidth* Terpakai Berdasarkan Ukuran *Cache*

4.2.3 Analisis Berdasarkan Perbedaan Jumlah *Backup Node*

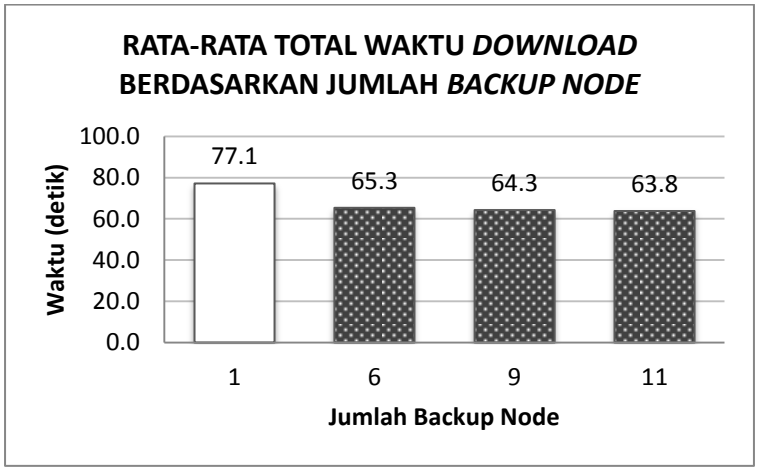
Percobaan dilakukan sesuai dengan skenario pada Tabel 3.3. Dari hasil pengolahan log hasil percobaan didapatkan beberapa nilai yang menggambarkan perbandingan nilai antara metode *Sender-Based Loss-Recovery* dengan metode *Multi-Nodes-Based Loss-Recovery* ditinjau dari jumlah *backup node* yang digunakan. *Backup node* berjumlah satu menandakan bahwa metode yang digunakan adalah metode *Sender-Based Loss-Recovery*, sedangkan *backup node* yang berjumlah lebih dari satu menandakan bahwa metode yang digunakan adalah metode *Multi-Nodes-Based Loss-Recovery* dengan jumlah *backup node* tertentu.

Tabel 4.5 menunjukkan bahwa metode *Multi-Nodes-Based Loss-Recovery* akan lebih optimal bila menggunakan 6 *backup node*. Hal ini dibuktikan pada penggunaan 6 *backup node*, metode *Multi-Nodes-Based Loss-Recovery* memiliki persentase pengiriman ulang NAK sebesar 10.8% dan memiliki kecenderungan peningkatan seiring dengan penambahan jumlah *backup node*. Di sisi lain, penggunaan metode *Multi-*

Nodes-Based Loss-Recovery, dengan berbagai jumlah *backup node*, masih memiliki persentase pengiriman ulang NAK yang lebih kecil dibandingkan dengan metode *Sender-Based Loss-Recovery*. Metode *Multi-Nodes-Based Loss-Recovery* paling besar menghasilkan pengiriman ulang NAK sebesar 28.2% sedangkan *Sender-Based Loss-Recovery* menghasilkan pengiriman ulang NAK sebesar 248.1%.

Tabel 4.5 Perbandingan Rata-rata NAK Berdasarkan Jumlah *Backup Node*

Jumlah *Backup Node*	*Sender-Based Loss-Recovery*			*Multi-Nodes-Based Loss-Recovery*		
	Nak_{rate}	Err_{rate}	persentase Err_{rate}	Nak_{rate}	Err_{rate}	persentase Err_{rate}
1	469.2	1164.1	248.1%	-	-	-
6	-	-	-	225.8	24.4	10.8%
9	-	-	-	226.1	33.0	14.6%
11	-	-	-	328.2	92.5	28.2%



Gambar 4.16 Rata-rata Penggunaan *Bandwidth* Berdasarkan Jumlah *Backup Node*

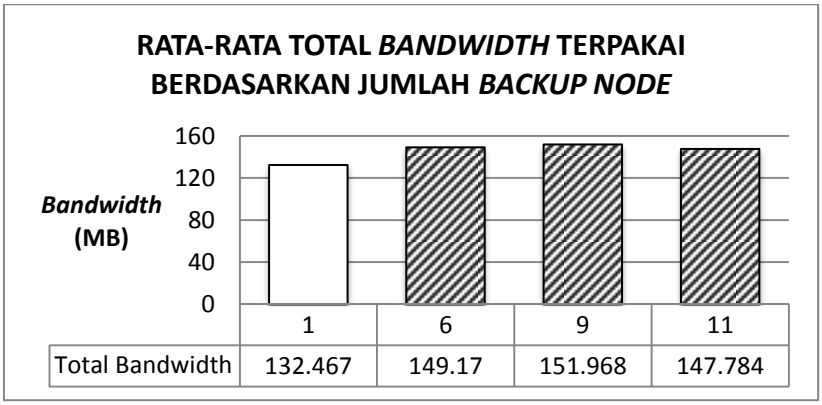
Gambar 4.16 merupakan grafik rata-rata total waktu *download* berdasarkan jumlah *backup node* yang digunakan. Terlihat bahwa pada metode *Multi-Nodes-Based Loss-Recovery* perbedaan jumlah *node* tidak berpengaruh terlalu signifikan terhadap rata-rata waktu *download*. Penggunaan 11 *backup node* hanya mampu menurunkan waktu *download* sebesar 1.5 detik lebih cepat bila dibandingkan saat menggunakan 6 *backup node*. Dari grafik pada Gambar 4.8 juga terlihat bahwa penggunaan metode *Multi-Nodes-Based Loss-Recovery*, dengan jumlah *backup node* yang berbeda-beda,

memiliki rata-rata waktu *download* yang lebih cepat daripada metode *Sender-Based Loss-Recovery*.

Tabel 4.6 Rata-rata Penggunaan *Bandwidth* Berdasarkan Jumlah *Backup Node*

Jumlah *Backup Node*	Pengiriman Segmen *File*	Pengiriman NAK	Pengiriman Paket *Discovery*
1	132.385	0.083	0
6	133.061	0.024	16.084
9	133.026	0.024	18.918
11	133.033	0.039	14.712

Tabel 4.6 menunjukkan data penggunaan *bandwidth* untuk tiap jenis paket yang dikirimkan dengan jumlah *backup node* yang bervariasi. Data tersebut kemudian dijumlahkan sehingga menghasilkan Gambar 4.17.



Gambar 4.17 Rata-rata *Bandwidth* Terpakai Berdasarkan Jumlah *Backup Node*

Gambar 4.17 menunjukkan bahwa metode *Multi-Nodes-Based Loss-Recovery*, dengan berbagai jumlah *backup node*, menggunakan *bandwidth* yang lebih banyak daripada metode *Sender-Based Loss-Recovery*, dimana perbedaan yang signifikan terdapat pada *bandwidth* untuk proses discovery.

4.2.4 Analisis Berdasarkan Kemampuan Menangani *Packet Loss*

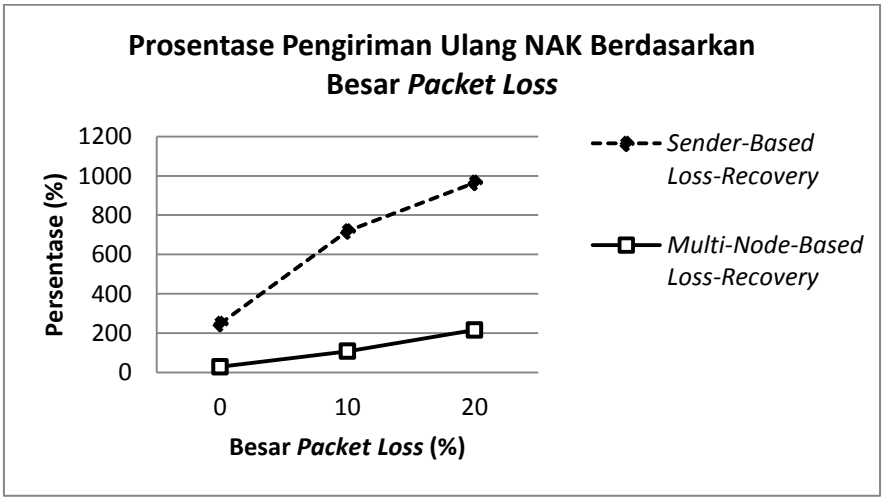
Percobaan dilakukan sesuai dengan skenario pada Tabel 3.4. Dari hasil pengolahan log hasil percobaan didapatkan beberapa nilai yang menggambarkan

perbandingan nilai antara metode *Sender-Based Loss-Recovery* dengan metode *Multi-Nodes-Based Loss-Recovery* ditinjau dari kemampuan menangani *packet loss*.

Tabel 4.7 Jumlah NAK Yang Tidak Terbalas Besar *Packet Loss*

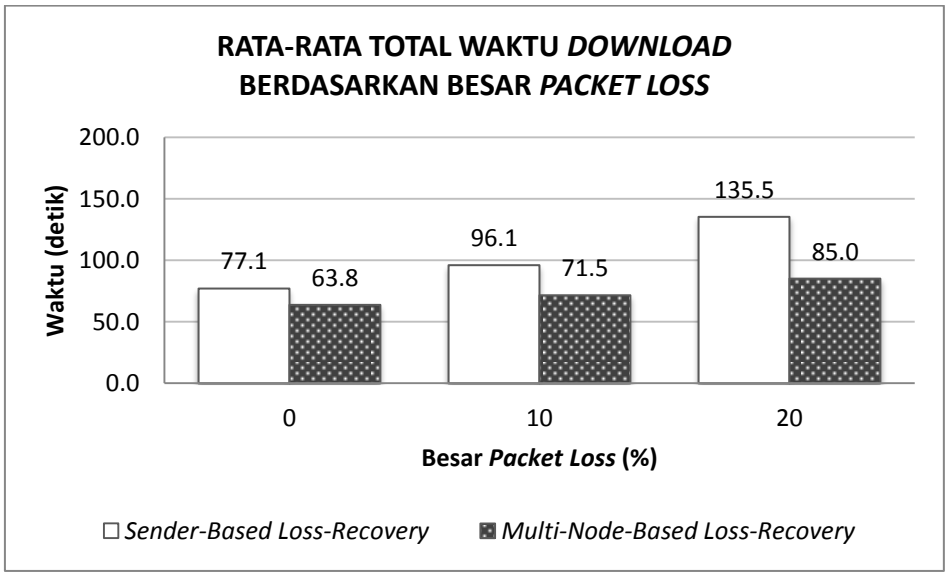
Besar *Packet Loss* (%)	*Sender-Based Loss-Recovery*			*Multi-Nodes-Based Loss-Recovery*		
	Nak_{rate}	Err_{rate}	persentase Err_{rate}	Nak_{rate}	Err_{rate}	persentase Err_{rate}
0	469.2	1164.1	248.1%	328.2	92.5	28.2%
10	1945.1	13977.1	718.6%	1943.5	2090.5	107.6%
20	5699.7	55019.9	965.3%	5717.7	12330.9	215.7%

Tabel 4.7 menunjukkan bahwa dalam berbagai ukuran *packet loss* secara umum metode *Multi-Nodes-Based Loss-Recovery* memiliki persentase pengiriman ulang NAK yang lebih kecil daripada metode *Sender-Based Loss-Recovery*. Pada pengujian menggunakan *packet loss* sebesar 20%, metode *Multi-Nodes-Based Loss-Recovery* menghasilkan pengiriman ulang NAK sebesar 215.7%, sedangkan pada metode *Sender-Based Loss-Recovery* menghasilkan pengiriman ulang NAK sebesar 965.3%. Dari data ini dapat disimpulkan bahwa metode *Multi-Nodes-Based Loss-Recovery* memiliki kemampuan dalam menangani *packet loss* yang lebih baik daripada metode *Sender-Based Loss-Recovery*.



Gambar 4.18 Persentase Pengiriman Ulang NAK Berdasarkan Besar *Packet Loss*

Gambar 4.19 merupakan grafik rata-rata total waktu *download* berdasarkan kemampuan menangani *packet loss*. Dari grafik tersebut terlihat bahwa metode *Multi-Nodes-Based Loss-Recovery* memiliki total waktu *download* yang lebih sedikit bila dibandingkan dengan metode *Sender-Based Loss-Recovery*. Selain itu, rata-rata waktu *download* pada metode *Sender-Based Loss-Recovery* memiliki kecenderungan meningkat yang cukup tajam seiring dengan peningkatan jumlah *packet loss*, sedangkan pada metode *Multi-Nodes-Based Loss-Recovery* peningkatan rata-rata waktu *download* tidak terlalu tinggi.



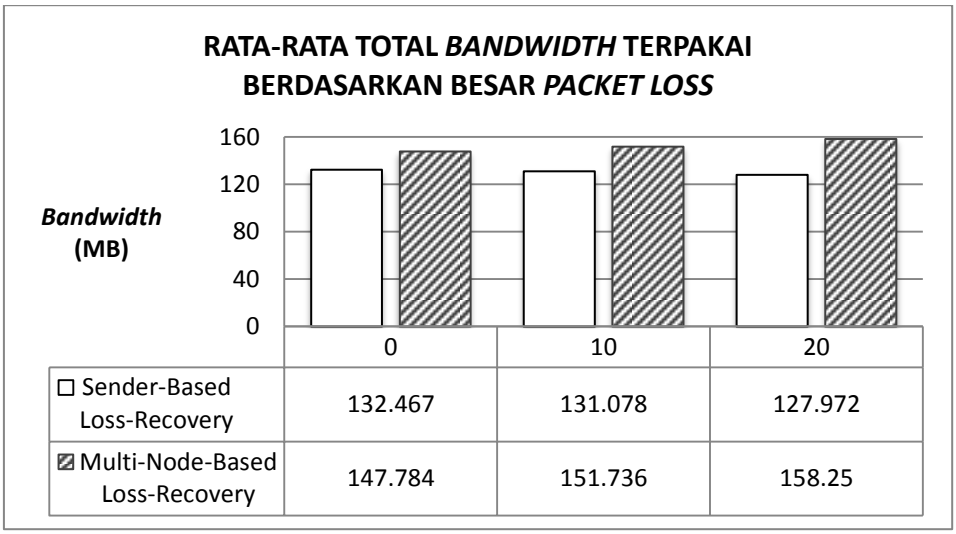
Gambar 4.19 Rata-rata Total Waktu *Donwload* Berdasarkan Besar *Packet Loss*

Tabel 4.8 menunjukkan data penggunaan *bandwidth* untuk tiap jenis paket yang dikirimkan dengan jumlah *packet loss* yang bervariasi. Data tersebut kemudian dijumlahkan sehingga menghasilkan Gambar 4.12.

Tabel 4.8 Rata-rata Penggunaan *Bandwidth* Berdasarkan Besar *Packet Loss*

Besar *Packet Loss* (%)	**Pengiriman Segmen *File***		**Pengiriman NAK**		**Pengiriman Paket *Discovery***	
	Sender-Based Loss-Recovery	*Multi-Nodes-Based Loss-Recovery*	*Sender-Based Loss-Recovery*	*Multi-Nodes-Based Loss-Recovery*	*Sender-Based Loss-Recovery*	*Multi-Nodes-Based Loss-Recovery*
0	132.385	133.033	0.083	0.039	0	14.712
10	130.274	133.299	0.805	0.335	0	18.102
20	124.903	134.449	3.069	1.4	0	22.402

Gambar 4.20 menunjukkan bahwa metode *Multi-Nodes-Based Loss-Recovery* menggunakan *bandwidth* yang lebih banyak daripada metode *Sender-Based Loss-Recovery*, dan penggunaan *bandwidth* ini cederung meningkat seiring dengan peningkatan jumlah *packet loss*. Tercatat bahwa pada *packet loss* sebesar 10% rata-rata *bandwidth* yang terpakai sebesar 151.736 MB dan saat *packet loss* sebesar 20% rata-rata *bandwidth* yang terpakai meningkat menjadi 158.25 MB.



Gambar 4.20 Rata-rata *Bandwidth* Terpakai Berdasarkan Besar *Packet Loss*

BAB 5
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Dari hasil percobaan dan analisis yang telah dilakukan, dapat ditarik kesimpulan sebagai berikut:

1. Metode yang diusulkan dalam penelitian ini, yaitu metode *Multi-Nodes-Based Loss-Recovery*, mampu mereduksi *feedback implosion* pada pengiriman data menggunakan *reliable multicast protocol*.

   Bila dibandingkan dengan metode *Sender-Based Loss-Recovery*, metode *Multi-Nodes-Based Loss-Recovery* mampu mereduksi peluang *bottle neck* pada *node* pengirim sehingga menurunkan jumlah NAK yang dikirim ulang. Hal ini dibuktikan dari data percobaan skenario pada Tabel 3.1. Data hasil percobaan, menunjukkan bahwa pada pengiriman file sebesar 32 MB, metode *Sender-Based Loss-Recovery* melakukan pengiriman NAK ulang sebanyak 112.4% dari jumlah NAK yang seharusnya, sedangkan metode *Multi-Nodes-Based Loss-Recovery* hanya melakukan pengiriman ulang NAK sebanyak 18.6%.

2. Pemilihan *backup node* menggunakan *bandwidth availability* dan *packet loss* sebagai parameter pemilihan dapat menurunkan pengiriman ulang NAK lebih optimal ketika jumlah *backup node* dibatasi dalam jumlah tertentu.

   Berdasarkan hasil pengujian pada skenario Tabel 3.3, penggunaan 6 *backup node* pada metode *Multi-Nodes-Based Loss-Recovery* memiliki rata-rata jumlah pengiriman ulang NAK sebanyak 10.8% dari jumlah NAK yang seharusnya. Namun nilai ini memiliki kecenderungan peningkatan ketika jumlah *backup node* ditambah. Ketika pengujian dilakukan menggunakan 11 *backup node*, rata-rata jumlah pengiriman ulang NAK meningkat menjadi 28.2%. Sehingga dapat disimpulkan bahwa penambahan jumlah *backup node* tidak dapat meningkatkan perfoma metode *Multi-Nodes-Based Loss-Recovery*. Namun bila backup node yang digunakan hanya satu maka peluang *bottle neck* akan sangat besar sehingga menurunkan perfoma dari metode ini.

3. Penggunaan *bandwidth availability* dan *packet loss* sebagai parameter pemilihan *backup node* memerlukan *bandwidth* tambahan yang cukup besar.

   Kesimpulan ini didasarkan pada hasil percobaan pada berbagai skenario pengujian yang telah dilakukan. Pada hasil pengujian skenario 3.1, metode *Multi-Nodes-Based Loss-Recovery* rata-rata memerlukan *bandwidth* tambahan sebesar 21.5 MB untuk proses pemilihan *backup node*. Hal ini disebabkan karena untuk menentukan *bandwidth availability* tiap *node* akan melakukan *flooding probe packet* ke jaringan.

4. Penggunaan *cache* yang besar pada metode *Multi-Nodes-Based Loss-Recovery* tidak menurunkan pengiriman ulang NAK.

   Kesimpulan ini didasarkan pada hasil percobaan pada skenario 3.2. Pada hasil pengujian skenario 3.2 penggunaan cache sebesar 3.5 MB menghasilkan pengiriman ulang NAK sebesar 17%, namun pada penggunaan cache sebesar 14 MB pengiriman ulang NAK meningkat menjadi 28.2%.

5. Penggunaan metode *Multi-Nodes-Based Loss-Recovery* dapat menurunkan waktu pengiriman.

   Kesimpulan ini didasarkan pada hasil percobaan pada berbagai skenario pengujian yang telah dilakukan. Sebagai contoh pada hasil pengujian skenario 3.1, metode *Multi-Nodes-Based Loss-Recovery* memilki rata-rata waktu *download* 15.8% lebih cepat daripada metode *Sender-Based Loss-Recovery*.

6. Metode *Multi-Nodes-Based Loss-Recovery* memiliki kemampuan menangani *packet loss* yang lebih baik daripada metode *Sender-Based Loss-Recovery*.

   Kesimpulan ini didasarkan pada hasil percobaan pada skenario 3.4. Hasil pengujian pada metode *Multi-Nodes-Based Loss-Recovery* menunjukkan bahwa pada saat *packet loss* yang semakin besar, persentase NAK yang dibangkitkan tidak langsung meningkat secara tajam. Hal ini sangat berbeda dengan metode *Sender-Based Loss-Recovery* yang peningkatan NAK-nya melonjak tajam seiring dengan semakin besarnya *packet loss* yang terjadi.

5.2 Saran

Pemanfaatan *multicast* untuk pengiriman data yang *reliable* secara massal masih mejadi topik yang menarik untuk dikembangkan. Dari sekian banyak metode

yang telah diusulkan oleh para peneliti, tentu masih banyak aspek yang bisa dikembangkan. Salah satunya adalah *semi reliable multicast protocol*, yaitu suatu pengiriman data menggunakan *multicast protocol* dengan memberikan prioritaskan pada beberapa jenis paket tertentu agar paket tersebut dikirimkan secara *reliable*. Metode *Multi-Nodes-Based Loss-Recovery* dapat dikembangkan untuk pengiriman data semacam ini.

DAFTAR PUSTAKA

Cain, B., Haberman, B., Holbrook, H., 2006. *Using Internet Group Management Protocol Version 3 (IGMPv3) and Multicast Listener Discovery Protocol Version 2 (MLDv2) for Source-Specific Multicast* [WWW Document]. URL http://tools.ietf.org/html/rfc4604 (diakses September 2013).

Chawathe, Y., Fink, S.A., McCanne, S., Brewer, E.A., 1998. A proxy architecture for reliable multicast in heterogeneous environments, dalam: *Proceedings of the Sixth ACM International Conference on Multimedia*, MULTIMEDIA '98. ACM, New York, NY, USA, hal. 151–159.

Curtis, J., Mcgregor, T., 2001. Review of Bandwidth Estimation Techniques, dalam: *New Zealand Computer Science Research Students' Conference*.

Deering, S.E., 1989. *Host extensions for IP multicasting* [WWW Document]. URL http://tools.ietf.org/html/rfc1112 (diakses September 2013).

Fenner, B., Handley, M., Kouvelas, I., Holbrook, H., 2006. *Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification (Revised)* [WWW Document]. URL http://tools.ietf.org/html/rfc4601 (diakses September 2013).

Gemmell, J., Gray, J., Schooler, E., 2000. Fcast multicast *file* distribution. *IEEE Network 14*, hal. 58–68.

Hubert, B., 2001. tc(8) - *Linux man page* [WWW Document]. URL http://lartc.org/manpages/tc.txt (diakses September 2013).

Koitani, K., Hasegawa, G., Murata, M., 2012. Measuring available bandwidth of multiple parts on end-to-end network path, dalam: *2012 IEEE International Workshop Technical Committee on Communications Quality and Reliability (CQR)*. Dipresentasikan di 2012 IEEE International Workshop Technical Committee on Communications Quality and Reliability (CQR), hal. 1–6.

Koutsonikolas, D., Hu, Y.C., Wang, C.-C., 2012. Pacifier: high-throughput, reliable multicast without "Crying babies" in *wireless* mesh networks. *IEEE/ACM Trans*. Netw. 20, hal. 1375–1388.

Kurose, J.F., Ross, K.W., 2010. *Computer networking: a top-down approach. Addison-Wesley*, Boston.

Lane, R.G., Daniels, S., Yuan, X., 2001. An empirical study of reliable multicast protocols over *Ethernet*-connected networks, dalam: *International Conference on Parallel Processing*, 2001. Presented at the International Conference on Parallel Processing, 2001, hal. 553–560.

Li, D., Xu, M., Zhao, M., Guo, C., Zhang, Y., Wu, M.-Y., 2011. RDCM: Reliable data center multicast, dalam: *2011 Proceedings IEEE INFOCOM. Presented at the 2011 Proceedings IEEE INFOCOM*, hal. 56–60.

Liu, D.-K., Hwang, R.-H., 2003. A P2P hierarchical clustering live video streaming system, dalam : *The 12th International Conference on Computer Communications and Networks*, 2003. ICCCN 2003. Proceedings. Dipresentasikan pada The 12th International Conference on Computer Communications and Networks, 2003. ICCCN 2003. Proceedings, hal. 115–120.

Ludovici, F., 2011. *NetEm - Network Emulator - Linux man page* [WWW Document]. URL http://stuff.onse.fi/man?program=tc-netem&section=8 (diakse September 2013).

Moy, J., 1994. *Multicast Extensions to OSPF* [WWW Document]. URL http://tools.ietf.org/html/rfc1584 (diakses September 2013).

Network Emulation [WWW Document], 2009. The Linux Foundation. URL http://www.linuxfoundation.org/collaborate/workgroups/networking/netem (diakses September 2013).

Postel, J., 1980. *User Datagram Protocol* [WWW Document]. URL http://tools.ietf.org/html/rfc768 (diakses September 2013).

Vegoda, L., Cotton, M., 2010. *IANA Guidelines for IPv4 Multicast Address Assignments* [WWW Document]. URL http://tools.ietf.org/html/rfc5771 (diakses September 2013).

Vicisano, L., Luby, M., Handley, M., Gemmell, J., Crowcroft, J., Rizzo, L., 2002. *The Use of Forward Error Correction (FEC) in Reliable Multicast* [WWW Document]. URL https://tools.ietf.org/html/rfc3453 (diakses September 2013).

Waitzman, D., Deering, S.E., Partridge, C., 1988. *Distance Vector Multicast Routing Protocol* [WWW Document]. URL http://tools.ietf.org/html/rfc1075 (diakses September 2013).

Wu, M., Karande, S.S., Radha, H., 2005. Network-embedded FEC for optimum throughput of multicast packet video. *Signal Processing: Image Communication 20*, hal. 728–742.

BIOGRAFI PENULIS

Penulis adalah anak bungsu dari dua bersaudara yang lahir di Malang pada tanggal 17 November 1986. Pendidikan sekolah dasarnya ditempuh di SD Kartika V-1 Malang, kemudian dilanjutkan ke SLTPN 3 Malang dan SMAN 1 Malang. Pada tahun 2010 penulis berhasil mendapatkan gelar sarjana di bidang komputer (S.Kom) dari Universitas Brawijaya. Setelah mendapatkan gerlar sarjana, penulis mendedikasikan dirinya untuk bekerja sebagai pengajar di kampus almamaternya, Universitas Brawijaya hingga saat ini. Pendidikan pascasarja penulis ditempuh pada tahun 2012 di jurusan Teknik Informatika Institut Teknologi Sepuluh Nopember Suarabaya dan berhasil mendapatkan gelar magister di bidang komputer (M.Kom) pada tahun 2014.

Selain di bidang akademis, penulis juga aktif di beberapa organisasi dan kegiatan internasional pada bidang pendidikan dan teknologi informasi. Ketika masih menjadi mahasiswa, penulis bergabung pada School On Internet Asia Project (http://www.soi.asia/) hingga pada tahun 2009 penulis diberangkatkan ke Universitas Keio di Jepang untuk melakukan penelitian pada bidang jaringan komputer selama 3 bulan. Pada tahun 2010 juga pernah ditugaskan oleh UNESCO Jakarta sebagai asisten pada pemasangan instalasi jaringan internet via satelit di Universitas Nasional Timor Leste (http://www.untl.edu.tl/) .

Untuk korespondensi dengan penulis dapat dilakukan melalui email dengan alamat mahendra.data@ub.ac.id.

“Manusia sangat membutuhkan ilmu dari pada (mereka) membutuhkan makanan dan minuman, karena makanan dan minuman hanya dibutuhkan sehari sekali atau dua kali, sementara ilmu dibutuhkan sepanjang nafasnya.”
– Imam Ahmad bin Hambal *t dalam Thabaqat Al-Hanabilah (I/146)*

LAMPIRAN 1

Lampiran 1 berisi data-data hasil percobaan pada tiap skenario pengujian. Pada tiap skenario dilakukan percobaan sebanyak 10 kali, sehingga pada tiap tabel skenario akan terdapat 10 baris data.

Keterangan untuk masing-masing kolom adalah sebagai berikut:

- $\boldsymbol{Nak}$ Rata-rata NAK yang seharusnya dikirimkan
- $\boldsymbol{Err}$ Rata-rata pengiriman ulang NAK
- $\boldsymbol{Time}$ Rata-rata total waktu *download* dalam satuan waktu (detik)
- $\boldsymbol{Bw_{file}}$ Rata-rata *bandwidth* terpakai untuk pengiriman *file* dalam satuan MB
- $\boldsymbol{Bw_{NAK}}$ Rata-rata *bandwidth* yang terpakai untuk NAK dalam satuan MB
- $\boldsymbol{Bw_{disc}}$ Rata-rata *bandwidth* yang terpakai untuk *discovery* dalam satuan MB

Rata-rata nilai **a**, **e**, **t**, **ba**, **bb** dan **bc** didapatkan dari hasil perhitungan *file log* menggunakan rumus 3.2, 3.3, 3.4 dan 3.5.

TABEL HASIL PERCOBAAN SKENARIO 1.1

Percobaan Ke-	$\boldsymbol{Nak}$	$\boldsymbol{Err}$	$\boldsymbol{Time}$	$\boldsymbol{Bw_{file}}$	$\boldsymbol{Bw_{NAK}}$	$\boldsymbol{Bw_{disc}}$
1	46.273	14.182	23.731	33.198	0.003	0.000
2	252.727	338.364	31.588	32.903	0.030	0.000
3	236.182	347.273	28.491	32.926	0.029	0.000
4	101.091	92.545	24.695	33.119	0.010	0.000
5	111.727	63.818	25.428	33.104	0.009	0.000
6	150.091	158.636	24.270	33.049	0.016	0.000
7	198.455	276.273	32.911	32.980	0.024	0.000
8	120.000	119.545	25.082	33.092	0.012	0.000
9	115.000	99.000	26.456	33.100	0.011	0.000
10	142.091	147.364	25.337	33.061	0.015	0.000
Rata-rata	147.364	165.700	26.799	33.053	0.016	0.000

TABEL HASIL PERCOBAAN SKENARIO 1.2

Percobaan Ke-	Nak	Err	$Time$	Bw_{file}	Bw_{NAK}	Bw_{disc}
1	57.091	0.909	21.178	33.264	0.006	8.600
2	133.182	17.636	27.777	33.260	0.015	8.299
3	142.818	23.727	21.999	33.226	0.015	6.792
4	216.000	36.818	25.361	33.263	0.026	7.304
5	199.364	51.091	21.341	33.191	0.021	7.062
6	246.545	45.818	23.204	33.224	0.028	6.773
7	191.727	9.455	23.357	33.232	0.019	8.043
8	184.818	72.182	23.556	33.203	0.021	9.224
9	177.909	47.727	25.500	33.205	0.020	7.619
10	134.091	8.182	20.824	33.248	0.014	7.337
Rata-rata	168.355	31.355	23.410	33.232	0.018	7.705

TABEL HASIL PERCOBAAN SKENARIO 1.3

Percobaan Ke-	Nak	Err	$Time$	Bw_{file}	Bw_{NAK}	Bw_{disc}
1	207.818	486.091	50.956	66.231	0.035	0.000
2	221.182	295.273	44.038	66.212	0.026	0.000
3	299.455	418.273	43.315	66.100	0.036	0.000
4	230.273	344.273	44.737	66.199	0.029	0.000
5	267.545	436.636	46.283	66.145	0.036	0.000
6	264.273	407.182	44.076	66.150	0.034	0.000
7	312.818	666.273	49.210	66.081	0.049	0.000
8	330.818	756.545	46.242	66.055	0.055	0.000
9	314.455	561.182	46.254	66.078	0.044	0.000
10	283.091	473.636	45.672	66.123	0.038	0.000
Rata-rata	273.173	484.536	46.078	66.137	0.038	0.000

TABEL HASIL PERCOBAAN SKENARIO 1.4

Percobaan Ke-	Nak	Err	$Time$	Bw_{file}	Bw_{NAK}	Bw_{disc}
1	148.727	12.909	35.452	66.485	0.015	10.953
2	280.273	91.091	38.576	66.541	0.037	10.928
3	476.000	155.636	35.641	66.513	0.061	9.658
4	281.182	95.909	36.333	66.527	0.035	10.148
5	412.364	41.636	37.434	66.464	0.043	9.336
6	341.545	40.636	34.830	66.526	0.038	10.821
7	248.818	35.455	34.464	66.527	0.028	10.551
8	392.727	59.818	35.042	66.516	0.045	9.037
9	279.455	81.364	33.738	66.448	0.030	8.195
10	305.818	46.455	45.132	66.511	0.035	12.938
Rata-rata	316.691	66.091	36.664	66.506	0.037	10.256

TABEL HASIL PERCOBAAN SKENARIO 1.5

Percobaan Ke-	Nak	Err	$Time$	Bw_{file}	Bw_{NAK}	Bw_{disc}
1	209.182	365.091	71.144	132.757	0.029	0.000
2	529.818	1388.091	80.087	132.298	0.097	0.000
3	525.364	1201.727	72.689	132.305	0.087	0.000
4	587.091	1646.182	81.422	132.216	0.113	0.000
5	627.545	1923.091	87.995	132.158	0.129	0.000
6	435.182	956.727	73.423	132.434	0.070	0.000
7	525.455	1337.545	80.220	132.304	0.094	0.000
8	449.182	1049.909	76.231	132.414	0.076	0.000
9	330.909	621.091	69.993	132.583	0.048	0.000
10	472.545	1151.091	78.127	132.380	0.082	0.000
Rata-rata	469.227	1164.055	77.133	132.385	0.083	0.000

TABEL HASIL PERCOBAAN SKENARIO 1.6

Percobaan Ke-	Nak	Err	$Time$	Bw_{file}	Bw_{NAK}	Bw_{disc}
1	587.545	246.909	67.002	133.023	0.073	18.818
2	457.545	114.091	60.038	133.051	0.054	16.737
3	587.909	122.818	62.307	132.978	0.064	15.569
4	436.636	126.818	62.405	133.049	0.055	15.486
5	420.818	112.364	61.958	133.020	0.048	13.734
6	470.545	91.545	62.659	133.043	0.054	14.355
7	87.545	66.818	72.007	133.037	0.014	13.929
8	43.091	1.545	60.384	133.056	0.004	12.160
9	44.364	0.182	61.772	133.052	0.004	13.301
10	145.818	41.636	67.957	133.023	0.017	13.034
Rata-rata	328.182	92.473	63.849	133.033	0.039	14.712

TABEL HASIL PERCOBAAN SKENARIO 1.7

Percobaan Ke-	Nak	Err	$Time$	Bw_{file}	Bw_{NAK}	Bw_{disc}
1	727.636	5658.091	177.688	265.071	0.323	0.000
2	876.727	4204.000	141.157	264.858	0.257	0.000
3	845.545	3659.636	139.760	264.903	0.228	0.000
4	841.091	2953.545	134.229	264.909	0.192	0.000
5	766.636	3040.091	137.108	265.016	0.192	0.000
6	898.909	5039.091	150.758	264.826	0.300	0.000
7	962.909	5406.636	150.711	264.735	0.322	0.000
8	967.182	4872.818	149.352	264.729	0.295	0.000
9	1060.364	5546.364	152.760	264.595	0.334	0.000
10	985.000	5273.091	144.308	264.703	0.316	0.000
Rata-rata	893.200	4565.336	147.783	264.835	0.276	0.000

TABEL HASIL PERCOBAAN SKENARIO 1.8

Percobaan Ke-	Nak	Err	$Time$	Bw_{file}	Bw_{NAK}	Bw_{disc}
1	557.364	315.909	130.020	266.297	0.075	32.006
2	169.000	18.000	121.359	266.124	0.019	24.125
3	176.818	38.364	125.041	266.127	0.021	23.613
4	44.909	2.636	119.637	266.112	0.005	25.197
5	117.545	8.636	118.192	266.115	0.013	26.344
6	269.818	80.273	123.802	266.189	0.033	23.688
7	98.182	39.273	122.428	266.147	0.013	27.116
8	97.818	10.636	120.976	266.104	0.011	26.189
9	144.091	27.455	119.202	266.132	0.017	27.002
10	194.000	13.273	123.521	266.223	0.020	24.084
Rata-rata	186.955	55.445	122.418	266.157	0.023	25.936

TABEL HASIL PERCOBAAN SKENARIO 1.9

Percobaan Ke-	Nak	Err	$Time$	Bw_{file}	Bw_{NAK}	Bw_{disc}
1	1407.364	15827.182	306.288	530.211	0.871	0.000
2	1982.000	14146.909	256.256	529.389	0.815	0.000
3	1849.818	11520.182	249.254	529.578	0.676	0.000
4	1787.636	13830.182	262.165	529.667	0.789	0.000
5	1885.909	11479.364	249.768	529.527	0.676	0.000
6	1626.818	11670.909	253.965	529.897	0.672	0.000
7	1645.545	10872.091	256.595	529.871	0.633	0.000
8	2036.091	13985.818	261.170	529.312	0.810	0.000
9	2045.636	14609.273	265.760	529.298	0.842	0.000
10	1706.273	13879.273	266.167	529.784	0.788	0.000
Rata-rata	1797.309	13182.118	262.739	529.653	0.757	0.000

TABEL HASIL PERCOBAAN SKENARIO 1.10

Percobaan Ke-	Nak	Err	$Time$	Bw_{file}	Bw_{NAK}	Bw_{disc}
1	923.818	1295.727	245.705	532.539	0.169	59.830
2	282.000	146.909	230.082	532.299	0.036	46.413
3	232.182	70.727	226.735	532.267	0.029	46.506
4	296.636	125.364	226.621	532.328	0.037	47.381
5	227.909	64.455	225.635	532.285	0.027	45.130
6	231.636	98.000	227.506	532.206	0.028	47.330
7	456.545	414.364	236.385	532.461	0.073	53.930
8	337.364	275.091	244.867	532.351	0.053	49.922
9	297.091	166.273	236.081	532.332	0.042	45.011
10	171.818	82.000	227.319	532.266	0.023	47.972
Rata-rata	345.700	273.891	232.694	532.333	0.052	48.942

TABEL HASIL PERCOBAAN SKENARIO 2.1

Percobaan Ke-	Nak	Err	$Time$	Bw_{file}	Bw_{NAK}	Bw_{disc}
1	104.909	54.727	65.566	132.906	0.008	0.000
2	48.727	11.182	61.074	132.986	0.003	0.000
3	21.636	0.000	54.653	133.025	0.001	0.000
4	3.182	0.000	53.990	133.052	0.000	0.000
5	19.273	0.000	56.891	133.029	0.001	0.000
6	49.727	10.545	57.935	132.985	0.003	0.000
7	5.091	0.000	54.029	133.049	0.000	0.000
8	31.727	4.091	57.603	133.011	0.002	0.000
9	28.455	0.000	56.331	133.015	0.001	0.000
10	38.273	4.000	59.637	133.001	0.002	0.000
Rata-rata	35.100	8.455	57.771	133.006	0.002	0.000

TABEL HASIL PERCOBAAN SKENARIO 2.2

Percobaan Ke-	Nak	Err	$Time$	Bw_{file}	Bw_{NAK}	Bw_{disc}
1	158.091	19.091	67.723	133.051	0.017	17.647
2	43.636	18.091	70.492	133.056	0.006	16.407
3	71.000	19.909	59.701	133.060	0.008	16.223
4	25.818	0.091	60.019	133.056	0.003	15.262
5	46.818	19.545	66.252	133.080	0.006	15.988
6	10.091	0.273	59.188	133.063	0.001	14.932
7	22.000	0.000	57.169	133.072	0.002	15.353
8	15.818	0.000	57.781	133.067	0.002	14.423
9	44.909	1.909	59.727	133.080	0.005	15.537
10	59.182	5.455	59.065	133.084	0.006	16.006
Rata-rata	49.736	8.436	61.712	133.067	0.006	15.778

TABEL HASIL PERCOBAAN SKENARIO 2.3

Percobaan Ke-	Nak	Err	$Time$	Bw_{file}	Bw_{NAK}	Bw_{disc}
1	215.455	401.818	75.029	132.748	0.031	0.000
2	37.273	0.182	58.220	133.003	0.002	0.000
3	54.273	17.545	55.979	132.978	0.004	0.000
4	134.000	108.091	63.769	132.864	0.012	0.000
5	26.455	0.000	56.407	133.018	0.001	0.000
6	37.727	6.091	59.761	133.002	0.002	0.000
7	29.273	0.000	56.729	133.014	0.001	0.000
8	26.273	0.000	56.572	133.019	0.001	0.000
9	72.909	26.636	59.176	132.952	0.005	0.000
10	59.364	27.636	59.358	132.971	0.004	0.000
Rata-rata	69.300	58.800	60.100	132.957	0.006	0.000

TABEL HASIL PERCOBAAN SKENARIO 2.4

Percobaan Ke-	Nak	Err	$Time$	Bw_{file}	Bw_{NAK}	Bw_{disc}
1	427.818	165.909	64.693	133.135	0.056	17.288
2	64.909	24.364	67.022	133.041	0.008	22.862
3	44.273	0.000	58.967	133.031	0.004	17.016
4	21.000	0.727	61.045	133.058	0.002	11.813
5	49.182	2.000	62.120	133.049	0.005	15.847
6	30.000	0.182	58.784	133.053	0.003	12.787
7	37.000	0.000	58.489	133.045	0.003	12.837
8	25.364	1.364	59.963	133.054	0.002	11.780
9	43.545	1.182	62.780	133.041	0.004	18.031
10	20.909	2.909	60.181	133.055	0.002	13.394
Rata-rata	76.400	19.864	61.404	133.056	0.009	15.366

TABEL HASIL PERCOBAAN SKENARIO 2.5

Percobaan Ke-	Nak	Err	$Time$	Bw_{file}	Bw_{NAK}	Bw_{disc}
1	209.182	365.091	71.144	132.757	0.029	0.000
2	529.818	1388.091	80.087	132.298	0.097	0.000
3	525.364	1201.727	72.689	132.305	0.087	0.000
4	587.091	1646.182	81.422	132.216	0.113	0.000
5	627.545	1923.091	87.995	132.158	0.129	0.000
6	435.182	956.727	73.423	132.434	0.070	0.000
7	525.455	1337.545	80.220	132.304	0.094	0.000
8	449.182	1049.909	76.231	132.414	0.076	0.000
9	330.909	621.091	69.993	132.583	0.048	0.000
10	472.545	1151.091	78.127	132.380	0.082	0.000
Rata-rata	469.227	1164.055	77.133	132.385	0.083	0.000

TABEL HASIL PERCOBAAN SKENARIO 2.6

Percobaan Ke-	Nak	Err	$Time$	Bw_{file}	Bw_{NAK}	Bw_{disc}
1	587.545	246.909	67.002	133.023	0.073	18.818
2	457.545	114.091	60.038	133.051	0.054	16.737
3	587.909	122.818	62.307	132.978	0.064	15.569
4	436.636	126.818	62.405	133.049	0.055	15.486
5	420.818	112.364	61.958	133.020	0.048	13.734
6	470.545	91.545	62.659	133.043	0.054	14.355
7	87.545	66.818	72.007	133.037	0.014	13.929
8	43.091	1.545	60.384	133.056	0.004	12.160
9	44.364	0.182	61.772	133.052	0.004	13.301
10	145.818	41.636	67.957	133.023	0.017	13.034
Rata-rata	328.182	92.473	63.849	133.033	0.039	14.712

TABEL HASIL PERCOBAAN SKENARIO 3.1

Percobaan Ke-	Nak	Err	$Time$	Bw_{file}	Bw_{NAK}	Bw_{disc}
1	209.182	365.091	71.144	132.757	0.029	0.000
2	529.818	1388.091	80.087	132.298	0.097	0.000
3	525.364	1201.727	72.689	132.305	0.087	0.000
4	587.091	1646.182	81.422	132.216	0.113	0.000
5	627.545	1923.091	87.995	132.158	0.129	0.000
6	435.182	956.727	73.423	132.434	0.070	0.000
7	525.455	1337.545	80.220	132.304	0.094	0.000
8	449.182	1049.909	76.231	132.414	0.076	0.000
9	330.909	621.091	69.993	132.583	0.048	0.000
10	472.545	1151.091	78.127	132.380	0.082	0.000
Rata-rata	469.227	1164.055	77.133	132.385	0.083	0.000

TABEL HASIL PERCOBAAN SKENARIO 3.2

Percobaan Ke-	Nak	Err	$Time$	Bw_{file}	Bw_{NAK}	Bw_{disc}
1	287.273	31.182	70.432	133.026	0.030	18.278
2	290.455	26.091	61.660	133.018	0.031	15.003
3	251.727	36.818	67.318	133.069	0.028	16.507
4	218.727	8.818	62.402	133.054	0.022	15.183
5	218.636	22.091	62.854	133.079	0.023	15.235
6	181.909	12.636	64.040	133.064	0.020	15.635
7	201.727	18.636	63.757	133.087	0.022	16.770
8	348.273	57.000	67.225	133.096	0.038	16.456
9	169.000	19.273	64.953	133.067	0.019	14.394
10	90.182	11.636	68.353	133.054	0.010	17.378
Rata-rata	225.791	24.418	65.299	133.061	0.024	16.084

TABEL HASIL PERCOBAAN SKENARIO 3.3

Percobaan Ke-	Nak	Err	$Time$	Bw_{file}	Bw_{NAK}	Bw_{disc}
1	141.545	20.636	66.197	133.056	0.016	16.314
2	266.818	57.727	63.522	133.008	0.030	18.045
3	265.636	53.545	68.425	132.997	0.030	16.507
4	181.636	7.364	62.419	133.016	0.018	15.984
5	239.273	29.636	67.156	133.088	0.027	18.420
6	226.727	26.364	62.169	133.003	0.023	17.418
7	160.182	29.545	61.764	133.058	0.019	28.891
8	292.636	64.909	64.118	133.004	0.032	19.889
9	230.182	23.727	63.652	132.987	0.023	18.410
10	256.182	16.818	63.242	133.040	0.027	19.300
Rata-rata	226.082	33.027	64.266	133.026	0.024	18.918

TABEL HASIL PERCOBAAN SKENARIO 3.4

Percobaan Ke-	Nak	Err	$Time$	Bw_{file}	Bw_{NAK}	Bw_{disc}
1	587.545	246.909	67.002	133.023	0.073	18.818
2	457.545	114.091	60.038	133.051	0.054	16.737
3	587.909	122.818	62.307	132.978	0.064	15.569
4	436.636	126.818	62.405	133.049	0.055	15.486
5	420.818	112.364	61.958	133.020	0.048	13.734
6	470.545	91.545	62.659	133.043	0.054	14.355
7	87.545	66.818	72.007	133.037	0.014	13.929
8	43.091	1.545	60.384	133.056	0.004	12.160
9	44.364	0.182	61.772	133.052	0.004	13.301
10	145.818	41.636	67.957	133.023	0.017	13.034
Rata-rata	328.182	92.473	63.849	133.033	0.039	14.712

TABEL HASIL PERCOBAAN SKENARIO 4.1

Percobaan Ke-	Nak	Err	$Time$	Bw_{file}	Bw_{NAK}	Bw_{disc}
1	209.182	365.091	71.144	132.757	0.029	0.000
2	529.818	1388.091	80.087	132.298	0.097	0.000
3	525.364	1201.727	72.689	132.305	0.087	0.000
4	587.091	1646.182	81.422	132.216	0.113	0.000
5	627.545	1923.091	87.995	132.158	0.129	0.000
6	435.182	956.727	73.423	132.434	0.070	0.000
7	525.455	1337.545	80.220	132.304	0.094	0.000
8	449.182	1049.909	76.231	132.414	0.076	0.000
9	330.909	621.091	69.993	132.583	0.048	0.000
10	472.545	1151.091	78.127	132.380	0.082	0.000
Rata-rata	469.227	1164.055	77.133	132.385	0.083	0.000

TABEL HASIL PERCOBAAN SKENARIO 4.2

Percobaan Ke-	Nak	Err	$Time$	Bw_{file}	Bw_{NAK}	Bw_{disc}
1	587.545	246.909	67.002	133.023	0.073	18.818
2	457.545	114.091	60.038	133.051	0.054	16.737
3	587.909	122.818	62.307	132.978	0.064	15.569
4	436.636	126.818	62.405	133.049	0.055	15.486
5	420.818	112.364	61.958	133.020	0.048	13.734
6	470.545	91.545	62.659	133.043	0.054	14.355
7	87.545	66.818	72.007	133.037	0.014	13.929
8	43.091	1.545	60.384	133.056	0.004	12.160
9	44.364	0.182	61.772	133.052	0.004	13.301
10	145.818	41.636	67.957	133.023	0.017	13.034
Rata-rata	328.182	92.473	63.849	133.033	0.039	14.712

TABEL HASIL PERCOBAAN SKENARIO 4.3

Percobaan Ke-	Nak	Err	$Time$	Bw_{file}	Bw_{NAK}	Bw_{disc}
1	2106.000	17693.636	112.331	130.043	1.001	0.000
2	1845.727	11971.273	88.767	130.416	0.698	0.000
3	1959.364	14818.091	100.172	130.253	0.848	0.000
4	1931.909	14422.545	94.983	130.293	0.827	0.000
5	1882.818	12621.455	90.501	130.363	0.733	0.000
6	1961.000	14484.273	96.010	130.251	0.831	0.000
7	1964.636	13359.273	95.147	130.246	0.775	0.000
8	1930.909	13667.909	94.180	130.294	0.788	0.000
9	1932.455	13258.909	96.304	130.292	0.768	0.000
10	1936.455	13473.545	93.009	130.286	0.779	0.000
Rata-rata	1945.127	13977.091	96.141	130.274	0.805	0.000

TABEL HASIL PERCOBAAN SKENARIO 4.4

Percobaan Ke-	Nak	Err	$Time$	Bw_{file}	Bw_{NAK}	Bw_{disc}
1	2058.091	798.909	71.348	133.465	0.267	16.156
2	1931.000	2705.000	78.661	133.199	0.366	18.635
3	1945.273	1401.909	69.879	133.102	0.294	18.117
4	2008.727	2971.273	75.537	133.269	0.413	17.890
5	1961.909	1973.000	71.740	133.503	0.342	18.816
6	1931.636	1657.818	71.238	133.471	0.303	17.954
7	1905.727	2269.909	68.796	133.029	0.336	19.218
8	1866.364	3562.182	70.042	133.507	0.423	17.794
9	1914.636	1581.182	68.454	133.321	0.295	20.013
10	1911.909	1984.182	69.274	133.127	0.310	16.429
Rata-rata	1943.527	2090.536	71.497	133.299	0.335	18.102

TABEL HASIL PERCOBAAN SKENARIO 4.5

Percobaan Ke-	Nak	Err	$Time$	Bw_{file}	Bw_{NAK}	Bw_{disc}
1	5771.182	74694.909	151.191	124.800	4.067	0.000
2	5698.545	46994.455	126.708	124.904	2.663	0.000
3	5658.364	55889.455	135.299	124.962	3.111	0.000
4	5655.636	55949.182	139.441	124.966	3.114	0.000
5	5700.909	50246.455	127.274	124.901	2.828	0.000
6	5804.182	53685.273	141.198	124.753	3.007	0.000
7	5636.091	56411.545	133.505	124.994	3.136	0.000
8	5699.909	49405.364	127.248	124.902	2.785	0.000
9	5727.455	51772.545	132.962	124.863	2.906	0.000
10	5644.909	55149.818	139.732	124.981	3.073	0.000
Rata-rata	5699.718	55019.900	135.456	124.903	3.069	0.000

TABEL HASIL PERCOBAAN SKENARIO 4.6

Percobaan Ke-	Nak	Err	$Time$	Bw_{file}	Bw_{NAK}	Bw_{disc}
1	5885.545	5143.909	76.488	131.913	0.975	18.160
2	5757.545	14830.636	86.465	136.053	1.497	22.530
3	5661.000	12153.636	82.505	134.019	1.420	22.797
4	5707.091	10991.545	89.573	134.292	1.273	22.627
5	5744.273	16937.455	89.521	135.526	1.619	24.817
6	5665.000	10618.364	82.092	134.227	1.313	22.076
7	5648.909	13470.091	82.363	134.478	1.459	21.108
8	5706.000	13938.545	86.917	135.235	1.496	25.479
9	5696.182	13873.273	85.895	134.251	1.584	21.780
10	5705.091	11352.000	88.356	134.492	1.363	22.646
Rata-rata	5717.664	12330.945	85.017	134.449	1.400	22.402