PEMBANGUNAN APLIKASI KOMPRESI DATA MENGGUNAKAN ALGORITMA LEMPLE ZIV MARKOV CHAIN (LZMA) DAN ALGORITMA LEMPLE ZIV STORER SYMANSKI (LZSS) PADA JARINGAN INTRANET

Subangkit Ramadhan[1]

[1] Teknik Informatika – Universitas Komputer Indonesia
Jl. Dipatiukur 112-114 Bandung
Email : subangkit.ramadhan@gmail.com[1]

ABSTRAK

Pada umumnya ukuran data yang berada pada saat ini memiliki ukuran yang sangat besar. Sehingga pada saat mengirim data yang berukuran besar melalui jaringan tentu akan mengakibatkan pemborosan pada penggunaan *resource* jaringan ketika data tersebut diakses oleh pengguna lain dan membuat media penyimpanan kurang efisien dalam menyimpan data-data tersebut.
Metode pemampatan *file* yang digunakan adalah Algoritma LZSS (*Lemple Ziv Storer Symanski*) serta dikombinasikan dengan metode LZMA yang bersifat *lossless*. Diharapkan dengan kedua algoritma ini dapat mengurangi waktu yang diperlukan dalam proses pengiriman data, akan menghasilkan nilai rasio kompresi data yang optimal sehingga ukuran data yang telah mengalami proses kompresi akan memiliki ukuran yang relatif lebih kecil dibandingkan dengan ukuran data sebelum terjadinya kompresi.
Dari hasil pengujian *white box* dan pengujian *black box* dapat diambil kesimpulan bahwa fungsional dari setiap aplikasi berjalan dengan baik. Serta proses kompresi dapat mempersingkat waktu dalam proses *transfer file*. Proses kompresi pun dapat menghemat penggunaan media penyimpanan.

Kata Kunci : Kompresi data, Algoritma, LZSS, LZMA, *lossless*, Intranet.

1. PENDAHULUAN

Data merupakan hal yang penting dan memiliki peranan yang sangat penting dalam kegiatan penggunaan komputer. Pada umumnya ukuran data yang berada pada saat ini memiliki ukuran yang sangat besar. Sehingga pada saat mengirim data yang berukuran besar melalui jaringan tentu akan mengakibatkan pemborosan pada penggunaan *resource* jaringan ketika data tersebut diakses oleh pengguna lain dan membuat media penyimpanan lebih kurang efisien dalam menyimpan data-data tersebut.
Terdapat penelitian yang pernah dilakukan sebelumnya berkaitan dengan jaringan intranet, diantaranya adalah penelitian yang dilakukan oleh Muhammad Nur Kamid dengan judul penelitian "*Analisa Keefektifan Jaringan Intranet Universitas Muria Kudus*"[1]. Pada penelitian tersebut dijelaskan bahwa adanya sebuah kondisi *delay* ketika terjadinya *traffic* yang cenderung tinggi pada saat penggunaan jaringan meningkat, sehingga kebutuhan waktu dan penggunaan *bandwith* pada saat proses transmisi akan meningkat.
Kompresi adalah proses pengubahan sekumpulan data menjadi bentuk kode dengan tujuan untuk menghemat kebutuhan tempat penyimpanan dan waktu untuk melakukan transmisi data. Kompresi data dilakukan dengan mengkodekan setiap karakter di dalam pesan atau data dengan mengkodekan setiap karakter di dalam pesan atau data dengan kode yang lebih pendek[2].
Terdapat penelitian yang pernah dilakukan sebelumnya berkaitan dengan kompresi pada jaringan berbasis LAN, diantaranya adalah penelitian yang dilakukan oleh ningning Hu dengan judul penelitian "*Network Aware Data Transmission With Compression*"[3]. Hasil penelitian dapat disimpulkan, bahwa proses kompresi dapat mengurangi waktu yang dibutuhkan untuk melakukan proses pengiriman sebuah data. Hanya saja pada kesimpulan penelitian aplikasi yang dihasilkan memisahkan antara proses kompresi dan *transfer* sehingga proses kompresi terlebih dahulu dilakukan secara sendiri lalu data hasil dari proses kompresi tersebut dikirimkan. Sehingga memungkinkan terjadinya penambahan waktu dikarenakan proses yang terjadi secara terpisah.
Algoritma *Lemple Ziv Markov Chain* (LZMA) adalah salah satu teknik dalam kompresi data yang merupakan pembaruan dari LZ77. Sedangkan Algoritma *Lemple Ziv Storer Symanski* (LZSS) adalah algoritma kompresi data *lossless* yang dimodifikasi dari LZ77, dinamai penciptanya James Storer dan Thomas Szymanski (yang dibangun di atas karya abraham lempel dan jacob ziv).
Mereferensi dari masalah tersebut serta penelitian yang telah dilakukan sebelumnya, maka penelitian yang akan dilakukan adalah melakukan pembangunan aplikasi kompresi data, dengan

menggunakan kombinasi algoritma *Lemple Ziv Markov Chain* (LZMA) dan algoritma *Lemple Ziv Storer Symanski*(LZSS) pada saat pengiriman data terjadi pada jaringan LAN. Diharapkan dengan kedua algoritma ini dapat mengurangi waktu yang diperlukan dalam proses pengiriman data, akan menghasilkan nilai rasio kompresi data yang optimal sehingga ukuran data yang telah mengalami proses kompresi akan memiliki ukuran yang relatif lebih kecil dibandingkan dengan ukuran data sebelum terjadinya kompresi serta pada penggunaan media penyimpanan data dapat lebih efisien.

1.1 Kompresi Data

Kompresi berarti memampatkan/mengecilkan ukuran. Kompresi data adalah proses mengkodekan informasi menggunakan bit atau *information-bearing unit* yang lain yang lebih rendah daripada representasi data yang tidak terkodekan dengan suatu sistem *encoding* tertentu[4]. Contoh kompresi sederhana yang biasa kita lakukan dalam kehidupan sehari-hari adalah misalnya kita menyingkatkan kata-kata yang sering kita gunakan tapi kata tersebut telah dipahami oleh setiap orang atau biasa digunakan secara umum. Misalnya : kata "yang" dikompres atau disingkat menjadi kata "yg".
Secara mendasar standar-standar teknik kompresi data memiliki 2 teknik kompresi yaitu teknik kompresi *lossy* dan teknik kompresi *loseless*. Untuk teknik kompresi secara *lossy* ada sebagian informasi yang dihilangkan di dalam data tersebut pada saat proses kompresi terjadi. Sedangkan untuk teknik kompresi secara *loseless* kebalikan dari teknik kompresi secara *lossy* yaitu tidak adanya informasi yang hilang pada saat proses kompresi terjadi.

1.1.1 Faktor Penting Kompresi Data

Dalam kompresi data, terdapat 4 (empat) faktor penting yang perlu diperhatikan, yaitu :

a. *Time Process* (waktu yang dibutuhkan dalam menjalankan proses).
b. *Completeness* (kelengkapan data setelah terjadinya proses kompresi).
c. *Ratio Compress* (ukuran data setelah dilakukannya proses kompresi) Berikut ini adalah persamaan menentukan nilai rasio kompresi[5] :

$$Ratio\ Compress = \frac{ukuran\ data\ setelah\ dikompresi}{ukuran\ data\ sebelum\ dikompresi} x\ 100\ \% \quad (2,1)$$

d. *Optimaly* (perbandingan apakah ukuran *file* setelah dilakukannya proses kompresi sama atau tidak sama dengan *file* sebelum dilakukannya proses kompresi).

1.1.2 Tipe Kompresi Data

Pada proses kompresi data, secara umum proses kompresi ini memiliki dua tipe, yaitu :

1. *Lossy Compression* adalah tipe kompresi data yang memiliki sifat data sumber (asli) dan data setelah dikompresi apabila dilakukannya proses dekompresi kembali hasilnya tidak sesuai dengan data sumbernya. Umumnya tipe kompresi data ini digunakan untuk proses kompresi pada data bertipe *image*[6].
Keuntungan pada tipe kompresi ini adalah membuat nilai rasio kompresi (perbandingan antara ukuran *file* asli dengan *file* hasil proses kompresi) cukup tinggi. Hal ini dikarenakan cara kerja *lossy* adalah dengan mengeliminasikan beberapa data dari suatu *file*, sehingga menyebabkan data pada suatu *file* yang telah melewati proses kompresi hilang ketika proses dekompresi. Namun data yang dieliminasikan biasanya adalah data yang kurang diperhatikan atau di luar jangkauan manusia, sehingga pengeliminasian data tersebut kemungkinan besar tidak akan mempengaruhi proses interaksi manusia dengan *file* tersebut. Contohnya pada pengkompresian *file* teks, kompresi *lossy* akan mengeliminasi data dari *file* teks yang memiliki frekuensi sangat tinggi/rendah sehingga banyak menghilangkan informasi dalam data yang dianggap penting. Beberapa jenis data yang masih bisa mentoleransi tipe kompresi ini adalah citra, audio dan video.
2.Loseless Compression adalah tipe kompresi data yang memiliki sifat data sumber (asli) dan data setelah dikompresi lalu dilakukannya proses dekompresi kembali akan memiliki struktur *file* yang sama[6].

1.2 Algoritma Lemple Ziv Storer Symanski (LZSS)

Algoritma *Lemple Ziv Storer Symanski* (LZSS) adalah algoritma kompresi data yang bertipe *loseless* yang dimodifikasi dari algoritma LZ77, dinamai penciptanya James Storer dan Thomas Symanski (yang dibangun di atas karya Abraham Lempel dan Jacob Ziv). Algoritma *Lemple Ziv Storer Symanski* (LZSS) salah satu kompresi urutan simbol data (misalnya, *byte-data*) dengan mengidentifikasi urutan simbol yang berulang dalam masukan, dan menggantikan urutan-urutan simbol yang lebih kecil. Implementasi algoritma *Lemple Ziv Storer Symanski* (LZSS) dapat menyesuaikan jumlah bit yang dialokasikan dengan mengganti panjang ukuran *byte* (antara parameter lain) untuk mendapatkan kinerja kompresi yang cukup baik[4].

1.2.1 Cara Kerja Algoritma LZSS

Cara kerja algoritma *Lemple Ziv Storer Symaski* (LZSS) terdiri dari sebuah *literal bytes* atau pasangan (*offset,match length*). *Literal bytes* merupakan *input byte* sederhana yang dikopikan secara langsung ke *output*. pasangan (*offset,match length*) menjelaskan proses kembali ke jarak *offset* suatu *input* data yang nantinya akan ditemukan sebuah *match length* dari data tersebut, algoritma ini menggunakan dua buah *flag* pada *output*, yaitu :

1. Jika *flag* "0", *byte* berikutnya adalah *literal*.
2. Jika *flag* "1", merupakan pasangan (*offset,match length*).

Sebagai Contohnya :

1. *Input* data = "ABCDABCA"

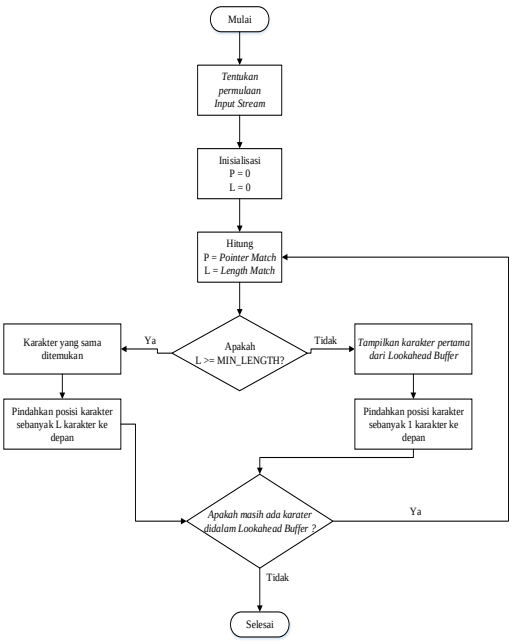
Mulai dengan byte "A", apakah telah ada sebelumnya? "TIDAK". Enkodenya adalah sebuah *literal byte* "0" diikuti data "A". *Output* Data = 0"A". Proses terus berlangsung sampai byte berikutnya sama dengan byte sebelumnya. Dengan kata lain *output* data menjadi : 0 "A " 0 " B " 0" C " 0 " D".

2. *Input* data = "ABCDABCA"

Byte berikutnya adalah "A", apakah telah ada sebelumnya ? "YA" 4 *bytes* sebelumnya (*offset* = 4). Ada berapa bytes yang sama? 3 *byte* yang sama "ABC", maka akan dikodekan "1" (untuk mengidentifikasi bahwa ada data yang sama) diikuti (4,3). *Output* Data= 0 "A" 0 "B" 0 "C" 0 "D" 1 (4,3).

3. *Input* data = "ABCDABCA"

Byte berikutnya adalah "A", apakah telah ada sebelumnya ? "YA" 3 *bytes* sebelumnya (*offset* = 3). Ada berapa *bytes* yang sama? 1 *byte* yang sama "A", maka akan dikodekan "1" diikuti (3,1). *Output* Data = 0 "A" 0 "B" 0 "C" 0 "D" 1 (4,3) 1 (3,1).



Gambar 1 Diagram Alir Algoritma LZSS

1.3 Algoritma Lemple Ziv Markov Chain (LZMA)

LZMA adalah pembaharuan dari algoritma LZ77, dengan menggunakan *markov chain* dan *entropy coding*[4]. Metode algoritma ini menggunakan bagian *stream input* sebagai kamus yang akan digunakan dalam kompresi data. *Encoder* menerima *input* dari kanan ke kiri bersamaan dengan keseluruhan data yang akan melalui proses kompresi. Metode ini juga disebut dengan istilah *sliding windows*. Pada *windows* ini terbagi menjadi dua bagian, bagian sebelah kiri dinamakan *search buffer* dan sebelah kanan dinamakan *look-ahead buffer*, yang berisi teks yang akan dikompresi. Seperti halnya LZ77, LZMA juga menghasilkan suatu keluaran yang menyimpan *distance*, *length*, dan *next symbol* sebagai suatu kesatuan berkas hasil kompresi. *Distance* merupakan jarak antara karakter yang dicari pada *look-ahead buffer* yang sama dengan karakter pada *search buffer*. *Length* adalah panjang dari rangkaian karakter terpanjang pada *search buffer* yang cocok pada *look-ahead buffer*. *Next symbol* adalah satu karakter pada *look-ahead buffer* setelah rangkaian karakter terpanjang terpilih.

Prinsip kompresi pada LZMA sama dengan teknik kompresi pada deflate, tetapi menggunakan proses *range coding* sebagai pengganti *Huffman coding*. Pada kasus nyata, hasil dari proses kompresi menggunakan algoritma LZMA berupa suatu *file* yang berisi rangkaian karakter representasi dari satu kesatuan berkas hasil kompresi yang terdiri dari *distance*, *length*, dan *next symbol*.

1.3.1 Cara Kerja Algoritma LZMA

Cara kerja algoritma *Lemple Ziv Markov Chain* (LZMA) pada proses kompresi adalah dimulai dengan memilih *file* yang akan dikompresi. *File* yang dipilih akan dikenali sebagai *input stream* yaitu deretan simbol/karakter yang ukuran panjangnya sesuai dengan besar *file*. Algoritma ini akan membaca *input stream* dengan menggunakan *search buffer* dengan ukuran 2^w bit dan *lookahead buffer*. Dimana w adalah nilai pangkat yang dibatasi oleh jumlah *memory* dalam perangkat keras.

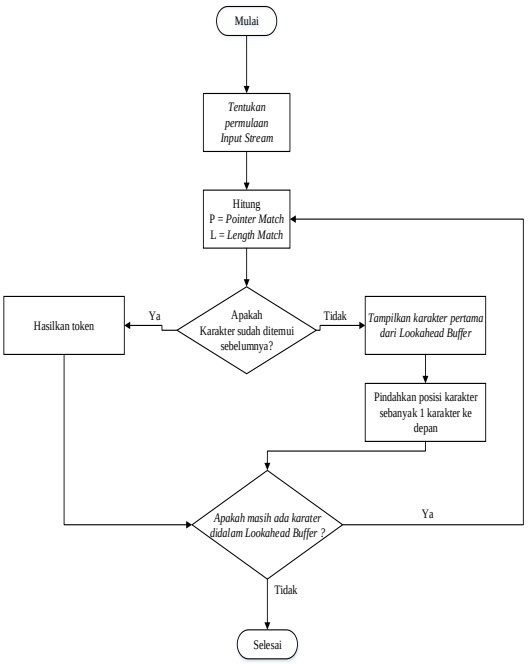
Proses kompresi diawali dengan membaca *input stream* dalam satuan *bytes* dan mencatat n *bytes* terakhir yang dibaca, jika menemukan susunan kata yang sama maka akan memberikan *output* nilai yang disebut dengan *token*. Dimana *token* memiliki struktur sebagai berikut :

T = (x,y,Z)
x = *offset* (2 bit)
y = *length* (2 bit)
z = *next symbol* (1 *byte*)
Selama proses *encode* ini, program juga merekam probabilitas setiap karakter yang bersebelahan serta membentuk *markov model* dalam susunan *directed graph*.
Sebagai contoh, data yang akan dikompresi sebagai berikut : AABBCBBAABC.
Di bawah ini adalah tabel untuk proses *Encoding* yang disajikan pada tabel 1.

Tabel 1 hasil proses *encoding* algoritma LZMA

Step	Pos	Match	Token
1	1	--	(0,0,"A")
2	3	A	(1,1,"B")
3	5	B	(1,1,"C")
4	6	BBB	(3,2,"A")
5	11	ABA	(7,2,"C")

Berikut ini adalah keterangan dari tabel 1 :
Proses *Encoding* : proses pengkodean input *Stream*
Pos : posisi dari setiap karakter pada *input stream*
Step : langkah proses pengkodean;
Match : *karakter yang sama yang ditemukan*
Token : barisan *simbol* hasil proses e*ncoding* yang terdiri dari nilai *offset*, *length* dan karakter selanjutnya
Dari uraian di atas, maka diagram alir untuk proses *encode* dengan menggunakan algoritma LZMA dapat disajikan sebagai berikut :

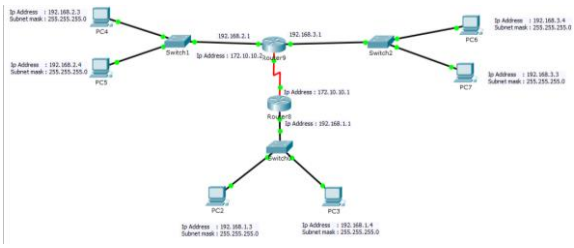


Gambar 2 Diagram Alir Algoritma LZMA

2. ISI PENELITIAN

2.1 Skema Jaringan

Untuk mensimulasikan perangkat lunak ini, dibutuhkan jaringan intranet yang digunakan dalam proses *transfer file* dan sudah mengimplementasikan *routing static* pada jaringan intranet tersebut. Berikut adalah arsitektur jaringan yang akan digunakan dalam pengujian perangkat lunak ini :



Gambar 3 Diagram blok skema jaringan

Tabel 2 Tabel Routing Static

Router	Tujuan	Mask	Next Hop	Keterangan
Router8	192.168.1.0	255.255.255.0	-	*Direct Connection*
	192.168.2.0	255.255.255.0	172.10.10.2	*Indirect Connection*
	192.168.3.0	255.255.255.0	172.10.10.2	*Indirect Connection*
Router9	192.168.1.0	255.255.255.0	172.10.10.1	*Indirect Connection*
	192.168.2.0	255.255.255.0	-	*Direct Connection*
	192.168.3.0	255.255.255.0	-	*Direct Connection*

Pada kasus dalam gambar dan tabel di atas adalah jaringan intranet yang memiliki 3 *network* yang berbeda yang telah dihubungkan dengan 2 buah *router* dan mengimplementasikan sistem *routing static* yang dapat dilihat pada tabel 2 di atas. Sehingga apabila suatu PC akan melakukan proses *transfer* data, maka PC tersebut harus menulusiri terlebih dahulu jalur yang telah ditentukan pada tabel *routing* tersebut.

Maka berdasarkan gambar 3 dan tabel 2 yang telah dipaparkan, sebagai contoh skenario untuk pengiriman data pada penumpukan proses pengiriman dari PC2 dengan ip 192.168.1.2 ke PC4 dengan ip 192.168.2.3 menggunakan perangkat lunak yang akan dibangun dapat dilihat sebagai berikut :

1. PC2 akan melakukan penetapan terhadap tujuan dan melakukan koneksi kepada tujuan,

di dalam contoh skenario ini tujuannya yaitu PC4 dengan ip 192.168.2.3.

2. Setelah PC2 terhubung dengan tujuan yaitu PC4, maka selanjutnya PC2 akan melakukan proses *transfer* data, pada perangkat lunak ini proses *transfer* akan dimulai dengan melakukan proses kompresi dengan menggunakan algoritma LZSS yang akan berlangsung pada PC2.
3. Setelah proses kompresi pada PC2 selesai dilakukan, maka data tersebut akan diteruskan kepada *gateway* terlebih dahulu. Setelah paket tersebut diterima oleh *gateway*, maka paket itu akan diperiksa oleh *gateway* untuk ditentukan alamat tujuannya dan *gateway* akan memeriksa jalur yang akan digunakan dalam pengiriman paket tersebut.
4. Setelah alamat dan jalur dalam pengiriman paket tersebut didapat, maka paket tersebut akan dikirim kepada alamat tujuan dengan menggunakan jalur yang telah ditentukan oleh *gateway* sebelumnya.
5. Sebelum paket data tersebut diterima oleh tujuan, proses kompresi dengan menggunakan algoritma LZMA pada paket tersebut akan dilakukan.
6. Setelah data tersebut diterima oleh tujuan, maka aplikasi ini akan menutup koneksi yang digunakan pada proses *transfer* tersebut.

Sehingga berdasarkan contoh skenario proses pengiriman di atas maka diharapkan apabila terjadi penumpukan paket data pada proses pengiriman seperti pada gambar 3, diharapkan akan mengurangi beban kerja dari router sehingga dapat mengurangi waktu *delay* yang terjadi akibat penumpukan paket data di router itu sendiri.

2.2 Proses Encoding

Cara kerja proses *encoding* ini terdiri dari proses *encoding* menggunakan algoritma LZSS terlebih dahulu sesudah menghasilkan *output* hasil proses *encoding* menggunakan algoritma tersebut, diteruskan proses *encoding* menggunakan algoritma *Lemple Ziv Markov Chain* (LZMA) terhadap hasil atau *output* dari proses *encoding* algoritma LZSS.

Video pada dasarnya tersusun atas serangkaian *frame*. Masing-masing *frame* merupakan citra. Suatu citra direpresentasikan dengan sebuah bidang yang masing-masing elemennya merepresentasikan nilai intensitas. Titik-titik dimana citra disampling disebut sebagai *picture elements* atau sering dikenal dengan pixel.

Langkah melakukan proses *encoding* pada *file* sumber adalah sebagai berikut :

1. Pertama proses *encoding* dimulai dengan membaca *file* video.
2. Setelah itu *file* tersebut akan diubah ke dalam bentuk *array bytes* yang berasal dari kumpulan *pixel file* sumber tersebut.
3. Setelah mendapatkan *array bytes* dari *file* tersebut, kedua algoritma ini mencari kesamaan antara tiap *bytes* yang berada pada *array bytes* tersebut.
4. Apabila ditemukan kesamaan maka *bytes* tersebut akan dikodekan dengan ketentuan masing-masing algoritma.
5. Setelah seluruh *array bytes* selesai ditelusuri, maka hasil dari penelusuran tersebut disimpan ke dalam *array bytes* baru, kemudian *array bytes* baru tersebut disusun ulang dan mengubahnya menjadi *file* hasil *encoding* dengan format lzsa

Pada proses *encoding* ini dimulai dengan proses *encoding* menggunakan algoritma LZSS terlebih dahulu. Berikut ini adalah tahapan dalam proses *encoding* dengan menggunakan LZSS terhadap contoh *input stream* yang didapat dari *file* sumber.

1. *Input* data = “7575759596”

Mulai dengan *byte* “7”, apakah telah ada sebelumnya? “TIDAK”. Maka akan didahului dengan sebuah *literal byte* “0” dan diikuti dengan *byte* “7”. Maka *output* data = 0 7.

2. *Input* data = “7575759596”

Dilanjutkan dengan *byte* “5”, apakah telah ada sebelumnya? “TIDAK”. Maka akan didahului dengan sebuah *literal byte* “0” dan diikuti dengan *byte* “5”. Maka *output* data = 07 05.

3. *Input* data = “7575759596”

Dilanjutkan dengan *byte* “7”, apakah telah ada sebelumnya? “YA”. Dikarenakan telah ditemukan sebelumnya maka akan dihitung berapa jarak antara *byte* yang sedang ditunjuk oleh *pointer* dengan *byte* yang telah ada sebelumnya. Pada contoh ini jarak antara *byte* yang sedang ditunjuk dengan *byte* yang sudah ada adalah 2.Maka nilai *offset*=2, dan setelah nilai *offset* ditentukan akan ada pengecekan terhadap *byte* berikutnya apakah sudah ada sebelumnya atau tidak. Dikarenakan *byte* selanjutnya memiliki kesamaan maka penulusuran dilanjutkan sampai ditemukan *byte* yang tidak memiliki kesamaan sebelumnya. Sehingga *output* data akan didahului dengan sebuah *literal byte* “1” lalu diikuti dengan pasangan (*offset,length match*) yaitu dalam contoh ini pasangan tersebut adalah (2,3). Maka *output* data = 07 05 1(2,3).

4. *Input* data = “7575759596”

Dilanjutkan dengan *byte* “9”, apakah telah ada sebelumnya? “TIDAK”. Maka akan didahului dengan sebuah *literal byte* “0” dan diikuti dengan *byte* “9”. maka *Output* data = 07 05 1(2,3) 09.

5. *Input* data = “7575759596”

Dilanjutkan dengan *byte* “5”, apakah telah ada sebelumnya? “YA”. Dikarenakan telah ditemukan sebelumnya maka akan dihitung berapa jarak antara *byte* yang sedang ditunjuk oleh *pointer* dengan *byte* yang telah ada sebelumnya. Pada contoh ini jarak antara *byte* yang sedang ditunjuk dengan *byte* yang

sudah ada adalah 6.Maka nilai *offset*=6, dan setelah nilai *offset* ditentukan akan ada pengecekan terhadap *byte* berikutnya apakah sudah ada sebelumnya atau tidak. Dikarenakan *byte* selanjutnya memiliki kesamaan maka penulusuran dilanjutkan sampai ditemukan *byte* yang tidak memiliki kesamaan sebelumnya. Sehingga *output* data akan didahului dengan sebuah *literal byte* "1" lalu diikuti dengan pasangan (*offset,length match*) yaitu dalam contoh ini pasangan tersebut adalah (2,2). Maka *output* data =07 05 1(2,3) 09 1(2,2).

6. *Input* data = "7575759596"

Dilanjutkan dengan *byte* "6", apakah telah ada sebelumnya? "TIDAK". Maka akan didahului dengan sebuah *literal byte* "0" dan diikuti dengan *byte* "6". Maka *output* data = 07 05 1(2,3) 09 1(2,2) 06.

Setelah *input* data selesai melalui proses *encoding* menggunakan algoritma LZSS, hasil proses tersebut akan disusun kembali menjadi sebuah *file temporary* yang akan digunakan oleh proses *encoding* selanjutnya. Maka hasil dari proses *encoding* menggunakan algoritma LZSS terhadap contoh *input* data. dapat dilihat pada tabel 3.

Tabel 3 Hasil *Encoding* Pada Proses LZSS

Output
0 7
0 5
1 2 3
0 9
1 2 2
0 6

Setelah mendapatkan hasil dari proses *encoding* menggunakan algoritma LZSS tersebut, hasil proses *encoding* menggunakan LZSS akan ditampung kedalam sebuah *file* sementara yang selanjutnya akan digunakan dalam proses *encoding* menggunakan algoritma LZMA.

Berikut ini adalah tahapan dalam proses *encoding* dengan menggunakan LZMA terhadap contoh *input stream* yang didapat dari hasil *encoding* menggunakan LZSS yang telah disajikan pada tabel 3.

1. *Input* data = "07051230912206"

Mulai dengan *byte* "0", apakah telah ada sebelumnya? "TIDAK". Maka *output* data = <0,0,"0">.

2. *Input* data = "07051230912206"

Dilanjutkan dengan *byte* "7", apakah telah ada sebelumnya? "TIDAK". Maka *output* data = <0,0,"0"> <0,0,"7">.

3. *Input* data = "07051230912206"

Dilanjutkan dengan *byte* "0", apakah telah ada sebelumnya? "YA". Dikarenakan telah ditemukan sebelumnya maka akan dihitung berapa jarak antara *byte* yang sedang ditunjuk oleh *pointer* dengan *byte* yang telah ada sebelumnya. Pada contoh ini jarak antara *byte* yang sedang ditunjuk dengan *byte* yang sudah ada adalah 2. Maka nilai *offset*=2, dan setelah nilai *offset* ditentukan akan ada pengecekan terhadap *byte* berikutnya apakah sudah ada sebelumnya atau tidak. Dikarenakan *byte* selanjutnya tidak memiliki kesamaan maka penulusuran dihentikan. Sehingga *output token* yang dihasilkan adalah (2,1,"5"). Maka *output* data = <0,0,"0"> <0,0,"7"> <2,1,"5">.

4. *Input* data = "07051230912206"

Dilanjutkan dengan *byte* "1", apakah telah ada sebelumnya? "TIDAK". Maka *output* data = <0,0,"0"> <0,0,"7"> <2,1,"5"> <0,0,"1">.

5. *Input* data = "07051230912206"

Dilanjutkan dengan *byte* "2", apakah telah ada sebelumnya? "TIDAK". Maka *output* data = <0,0,"0"> <0,0,"7"> <2,1,"5"> <0,0,"1"> <0,0,"2">.

6. *Input* data = "07051230912206"

Dilanjutkan dengan *byte* "3", apakah telah ada sebelumnya? "TIDAK". Maka *output* data = <0,0,"0"> <0,0,"7"> <2,1,"5"> <0,0,"1"> <0,0,"2"><0,0,"3">.

7. *Input* data = "07051230912206"

Dilanjutkan dengan *byte* "0", apakah telah ada sebelumnya? "YA". Dikarenakan telah ditemukan sebelumnya maka akan dihitung berapa jarak antara *byte* yang sedang ditunjuk oleh *pointer* dengan *byte* yang telah ada sebelumnya. Pada contoh ini jarak antara *byte* yang sedang ditunjuk dengan *byte* yang sudah ada adalah 5. Maka nilai *offset*=5, dan setelah nilai *offset* ditentukan akan ada pengecekan terhadap *byte* berikutnya apakah sudah ada sebelumnya atau tidak. Dikarenakan *byte* selanjutnya tidak memiliki kesamaan maka penulusuran dihentikan. Sehingga *output token* yang dihasilkan adalah (5,1,"9"). Maka *output* data = <0,0,"0"> <0,0,"7"> <2,1,"5"> <0,0,"1"> <0,0,"2> <0,0,"3><5,1,"9">.

8. *Input* data = "07051230912206"

Dilanjutkan dengan *byte* "1", apakah telah ada sebelumnya? "YA". Dikarenakan telah ditemukan sebelumnya maka akan dihitung berapa jarak antara *byte* yang sedang ditunjuk oleh *pointer* dengan *byte* yang telah ada sebelumnya. Pada contoh ini jarak antara *byte* yang sedang ditunjuk dengan *byte* yang sudah ada adalah 5. Maka nilai *offset*=5, dan setelah nilai *offset* ditentukan akan ada pengecekan terhadap *byte* berikutnya apakah sudah ada sebelumnya atau tidak. Dikarenakan *byte* selanjutnya sudah memiliki kesamaan maka penulusuran dilanjutkan sampai *byte* yang tidak memiliki kesamaan atau sampai akhir panjang *input stream*. Sehingga *output token* yang dihasilkan adalah <5,2,"2">. Maka *output* data = <0,0,"0"> <0,0,"7"> <2,1,"5"> <0,0,"1"> <0,0,"2> <0,0,"3"> <5,1,"9"> <5,2,"2">.

9. *Input* data = "07051230912206"

Dilanjutkan dengan *byte* "0", apakah telah ada sebelumnya? "YA". Dikarenakan telah ditemukan sebelumnya maka akan dihitung berapa jarak antara *byte* yang sedang ditunjuk oleh *pointer* dengan *byte* yang telah ada sebelumnya. Pada contoh ini jarak antara *byte* yang sedang ditunjuk dengan *byte* yang sudah ada adalah 5. Maka nilai *offset*=5, dan setelah nilai *offset* ditentukan akan ada pengecekan terhadap *byte* berikutnya apakah sudah ada sebelumnya atau tidak. Dikarenakan *byte* selanjutnya sudah memiliki kesamaan maka penulusuran dilanjutkan sampai *byte* yang tidak memiliki kesamaan atau sampai akhir panjang *input stream*. Sehingga *output token* yang dihasilkan adalah <5,1,"6">. Maka *output* data = <0,0,"0"> <0,0,"7"> <2,1,"5"> <0,0,"1"> <0,0,"2> <0,0,"3"> <8,1,"9"> <5,2,"2"><5,1,"6">

Token-token yang dihasilkan dari proses *encoding* dapat dilihat pada tabel 4.

Tabel 4 Hasil *Encoding* Proses LZMA

Output
<0,0,"0">
<0,0,"7">
<2,1,"5">
<0,0,"1">
<0,0,"2">
<0,0,"3">
<8,1,"9">
<5,2,"2">
<5,1,"6">

Pada tabel 4 telah diuraikan hasil dari contoh proses *encoding* yang setiap keluarannya berupa *token-token* sebagai pengganti dari setiap *byte* yang terdapat pada *array bytes* yang dijadikan sebagai *file* sumber. Setelah proses *encoding* selesai dilakukan, selanjutnya adalah pembentukan atau penulisan kembali *token-token* yang dihasilkan ke dalam sebuah *file* baru hasil *encoding* tersebut dengan berformat ekstensi baru yaitu lzsa.

2.3 Pengujian Perangkat Lunak

Pengujian perangkat lunak ini, dilakukan dengan lingkungan sistem berdasarkan dengan gambar 3 dan konfigurasi masing-masing tabel *routing* yang sesuai dengan tabel 2.

2.3.1 Pengujian *Black Box*

Berdasarkan skenario pengujian *Black Box*, maka hasil dari pengujian dapat dilihat sebagai berikut :

1. Pengujian proses kompres

Pengujian *Black Box* untuk proses kompresi untuk data benar dan data salah dapat dilihat pada tabel 5 dan tabel 6.

Tabel 5 Pengujian Proses Kompres Data Normal

Kasus dan Hasil Uji (Data Benar)			
Data Masukkan	Yang Diharapkan	Pengamatan	Kesimpulan
File Input big_buck_bunny_360p214.avi	Melakukan proses kompresi pada *file*	Menampilkan pesan proses kompresi berhasil	Diterima

Tabel 6 Pegujian Proses Kompres Data Salah

Kasus dan Hasil Uji (Data Salah)			
Data Masukkan	Yang Diharapkan	Pengamatan	Kesimpulan
File Input big_buck_bunny_360p214.mkv	Melakukan proses kompresi pada *file*	Tidak menampilkan pesan proses kompresi berhasil	Ditolak

2. Pengujian proses dekompres

Pengujian *Black Box* untuk proses dekompresi untuk data benar dan data salah dapat dilihat pada tabel 7 dan tabel 8.

Tabel 7 Pengujian Proses Dekompres Data Normal

Kasus dan Hasil Uji (Data Benar)			
Data Masukkan	Yang Diharapkan	Pengamatan	Kesimpulan
File Input big_buck_bunny_360p214.lzsa	Melakukan proses dekompresi pada *file*	Menampilkan pesan proses dekompresi berhasil	Diterima

Tabel 6 Pegujian Proses Dekompres Data Salah

Kasus dan Hasil Uji (Data Salah)			
Data Masukkan	Yang Diharapkan	Pengamatan	Kesimpulan
File Input big_buck_bunny_360p214.rar	Melakukan proses dekompresi pada *file*	Tidak menampilkan pesan proses dekompresi berhasil	Ditolak

3. Pengujian proses *transfer*

Pengujian *Black Box* untuk proses *transfer* untuk data benar dan data salah dapat dilihat pada tabel 9 dan tabel 10.

Tabel 9 Pengujian Proses *Transfer* Data Normal

Kasus dan Hasil Uji (Data Benar)			
Data Masukkan	Yang Diharapkan	Pengamatan	Kesimpulan
File Input big_buck_bunny_360p214.avi	Melakukan proses kompresi dan *transfer file* secara bersamaan dengan tujuan yang dikehendaki	Menampilkan pesan proses *transfer* berhasil	Diterima

Tabel 10 Pegujian Proses *Transfer* Data Salah

Kasus dan Hasil Uji (Data Salah)			
Data Masukkan	Yang Diharapkan	Pengamatan	Kesimpulan
File Input big_buck_bunny_360p214.rar	Melakukan proses kompresi dan *transfer file* seccara bersamaan dengan tujuan yang dikehendaki	TIdak menampilkan pesan proses *transfer* berhasil	Ditolak

4. Pengujian proses piih *file* masukkan

Pengujian *Black Box* untuk proses pilih *file* masukkan untuk data benar dan data salah dapat dilihat pada tabel 11 dan tabel 12.

Tabel 11 Pengujian Proses Piih F*ile M*asukkan Data Normal

Kasus dan Hasil Uji (Data Benar)			
Data Masukkan	Yang Diharapkan	Pengamatan	Kesimpulan
File Input big_buck_bunny_360p214.avi	Mendeteksi alamat *file* sumber sebagai data masukkan	Menampilkan alamat *file* sumber yang dipilih	Diterima

Tabel 12 Pengujian Proses Piih F*ile M*asukkan Data Salah

Kasus dan Hasil Uji (Data Salah)			
Data Masukkan	Yang Diharapkan	Pengamatan	Kesimpulan
File Input big_buck_bunny_360p214.rar	Mendeteksi alamat *file* sumber sebagai data masukkan	Tidak Menampilkan alamat *file* sumber yang dipilih	Ditolak

5. Pengujian proses piih lokasi keluaran

Pengujian *Black Box* untuk proses piih lokasi keluaran untuk data benar dan data salah dapat dilihat pada tabel 13 dan tabel 14.

Tabel 13 Pengujian Proses Piih Lokasi Keluaran Data Normal

Kasus dan Hasil Uji (Data Benar)			
Data Masukkan	Yang Diharapkan	Pengamatan	Kesimpulan
Alamat Lokasi Penyimpanan C:\Users \Documents	Mendeteksi lokasi penyimpan an *file* hasil sebagai data keluaran	Menampilkan alamat lokasi *file* keluaran yang dipilih	Diterima

Tabel 14 Pengujian Proses Piih Lokasi Keluaran Data Salah

Kasus dan Hasil Uji (Data Salah)			
Data Masukkan	Yang Diharapkan	Pengamatan	Kesimpulan
Alamat Lokasi Penyimpanan C:\Users \Documents	Mendeteksi lokasi penyimpan an *file* hasil sebagai data keluaran	Tidak Menampilkan alamat lokasi *file* keluaran yang dipilih	Ditolak

6. Pengujian proses koneksi

Pengujian *Black Box* untuk proses koneksi untuk data benar dan data salah dapat dilihat pada tabel 15 dan tabel 16.

Tabel 15 Pengujian Proses koneksi Data Normal

Kasus dan Hasil Uji (Data Benar)			
Data Masukkan	Yang Diharapkan	Pengamatan	Kesimpulan
Alamat IP Adrdress tujuan 192.168.1.5	Menghubun gkan dengan tujuan yang dikehendak i	Menampilkan pesan berhasil terhubung dengan tujuan	Diterima

Tabel 16 Pengujian Proses Koneksi Data Salah

Kasus dan Hasil Uji (Data Salah)			
Data Masukkan	Yang Diharapkan	Pengamatan	Kesimpulan
Alamat IP Adrdress tujuan 192.168.1.5	Menghubun gkan dengan tujuan yang dikehendak i	Tidak Menampilkan pesan berhasil terhubung dengan tujuan	Ditolak

2.3.2 Pengujian *White Box*

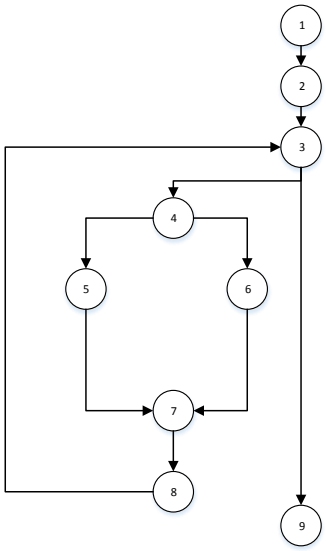
Pengujian *White Box* digunakan untuk meyakinkan semua perintah dan kondisi pada aplikasi dieksekusi secara minimal, pengujian *white box* menggunakan dua *tools* yaitu *flow graph* yang digunakan untuk menggambarkan alur dari algoritma dan *graph matrix* yang digunakan untuk menggenerasi *flow graph*. Adapun pengujian *white box* adalah sebagai berikut :

2.3.2.1 Pengujian Algoritma LZSS

Berikut ini adalah *pseudocode* algoritma LZSS :

```
1: baca permulaan input stream
2: inisialisasi p = 0, l=0
3: Do while lookahead buffer masih ada karakter
       Baca input stream
       Hitung p dan l
4:     if ( l >= Min_Length)
5:       simpan posisi karakter pada p
         simpan  p pada record
                 pindahkan karakter ke depan
         sebanyak L
6:     else
         simpan karakter pada record
         bergerak ke karakter selanjutnya
7:     endif
8: endDo
9:end
```

Berdasarkan *pseudocode* di atas maka *flow graph* algoritma LZSS yang digunakan dapat dilihat pada gambar 4.



Gambar 4 Flow Graph Algoritma LZSS

Dari gambar 4 yang telah dipaparkan di atas, dapat dihitung *cyclomatic complexity* sebagai berikut :

V(G) = E – N +2
V(G) = 10 – 9 +2
V(G) = 3

Dimana :
E = jumlah *edge* pada *flow graph*
N = jumlah *node* pada *flow graph*

Jadi, *cyclomatic complexity* untuk gambar 4.7 adalah 3. Berdasarkan *cyclomatic complexity* tersebut, maka terpilih 3 *path* yang terdiri dari

Path 1 = 1-2-3-9
Path 2 = 1-2-3-4-5-7-8-3-9
Path 3 = 1-2-3-4-6-7-8-3-9

Untuk melakukan uji coba basis *path* di atas digunakan *graph matrix*. *Graph matrix* merupakan *matrix* empat persegi yang mempunyai ukuran yang sama dengan jumlah *node* pada *flow graph*. Adapun *graph matrix* pada *flow graph* Algoritma LZSS di atas adalah sebagai berikut :

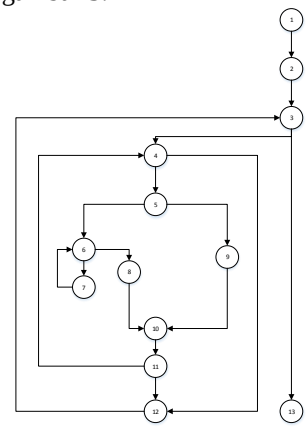
N	1	2	3	4	5	6	7	8	9	Sum-1
1		1								0
2			1							0
3				1					1	1
4					1	1				1
5							1			0
6							1			0
7								1		0
8			1							0
9										0
Jumlah +1										3

2.3.2.2 Pengujian Algoritma LZMA

Berikut ini adalah *pseudocode* algoritma lZMA :

```
1: baca permulaan input stream
2: inisialisasi p = 0, l=0
3: Do while lookahead buffer masih ada karakter
       baca input stream
4:     Do while search buffer masih ada karakter
       baca search buffer
5:     if (byte di search buffer == byte di lookahead
buffer)
6:     do while byte selanjutnya di lookahead buffer
== byte di search buffer
         Hitung jumlah byte yang sama
7:     endDo
8:       simpan jumlah byte yang sama ke dalam l
         simpan nilai offset posisi byte yang sama
pada search buffer hingga posisi byte
sekarang ke dalam p
         hasilkan token
9:     else
         hasilkan token
10:    endIf
11:    endDo
12: endDO
13: end
```

Berdasarkan *pseudocode* di atas maka *flow graph* algoritma LZMA yang digunakan dapat dilihat pada gambar 5.



Gambar 5 Flow Graph Algoritma LZMA

Dari gambar 5 yang telah dipaparkan di atas, dapat dihitung *cyclomatic complexity* sebagai berikut :
V(G) = E – N +2
V(G) = 17 –13 +2
V(G) = 6
Dimana :
E = jumlah *edge* pada *flow graph*
N = jumlah *node* pada *flow graph*
Jadi, *cyclomatic complexity* untuk gambar 4.8 adalah 6. Berdasarkan *cyclomatic complexity* tersebut, maka terpilih 6 *path* yang terdiri dari
Path 1 = 1-2-3-13
Path 2 = 1-2-3-4-12-3-13
Path 3 = 1-2-3-4-5-6-7-6-8-10-11-4-12-3-13
Path 4 = 1-2-3-4-5-6-8-10-11-4-12-3-13
Path 5 = 1-2-3-4-5-9-10-11-4-12-3-13
Path 6 = 1-2-3-4-5-9-10-11-12-3-13
Untuk melakukan uji coba basis *path* di atas digunakan *graph matrix*. *Graph matrix* merupakan *matrix* empat persegi yang mempunyai ukuran yang sama dengan jumlah *node* pada *flow graph*. Adapun *graph matrix* pada *flow graph* Algoritma LZMA di atas adalah sebagai berikut :

N	1	2	3	4	5	6	7	8	9	10	11	12	13	Sum-1
1		1												0
2			1											0
3				1									1	1
4					1							1		1
5						1			1					1
6							1	1						1
7						1								0
8										1				0
9										1				0
10											1			0
11				1								1		1
12			1											0
13														0
													Jumlah +1	6

2.3.3 Pengujian Data Yang Diuji

Pada proses pengujian data, proses pengujian yang akan dilakukan melalui dua bagian pengujian, yaitu :
Melakukan pengujian hanya dengan melakukan proses kompresi dan melakukan proses dekompresi tanpa adanya proses *transfer file*.
Melakukan pengujian dengan melakukan proses *transfer file*. Dimana dalam tahap *transfer file* ini akan dilakukannya proses kompresi pada saat data dikirimkan.
Proses pengujian ini akan dilakukan terhadap *file* video bertipe avi. Dari hasil pengujian ini akan diambil data berupa ukuran *file* hasil kompresi, waktu kompresi dan dekompresi, serta waktu dan ukuran *bandwith* yang dibutuhkan aplikasi untuk melakukan proses kompresi pada saat melakukan proses *transfer* terjadi.
Dalam melakukan pengujian pada *file* video, aplikasi ini diuji dengan *file* video yang berukuran 673 MB dan 4,94 GB. Pengukuran ini dilakukan untuk mengetahui berapa lama waktu yang dibutuhkan aplikasi untuk dapat melakukan proses kompresi dan proses dekompresi serta nilai rasio yang didapatkan dari hasil proses kompresi. Perhitungan rasio kompresi dapat dilakukan dengan membandingkan ukuran *file* hasil kompresi dengan ukuran semula. Maka perhitungan rasio terhadap setiap *file* yang diuji dapat dilihat sebagai berikut :

Rasio = $\frac{250812134}{673430026}$ x 100% = 37,24%

Rasio = $\frac{1324443054}{4947349933}$ x 100% = 26,77%

Berikut ini merupakan hasil pengujian data dengan melakukan proses kompresi dan dekompresi tanpa adanya proses *transfer file* dengan aplikasi yang dibangun. Hasil dapat dilihat pada Tabel 17.

Tabel 17 Hasil Pengujian Proses Kompresi Dan Dekompresi Pada file

No	Nama File	Tipe File	Ukuran (byte)		Waktu (ms)		Rasio
			File Asli	*File* hasil kompresi	kompresi	dekompresi	
1	big_buck_bunny_360p	avi	673430026	250812134	00:20:08	00:01:07	37,24%
2	Elephants_Dream_360p	avi	4947349933	1324443054	02:05:29	00:07:08	26,77%

Pada tabel 17 telah dijelaskan hasil dari pengujian menggunakan 2 data dengan kapasitas yang berbeda sehingga dihasilkan sebuah nilai rasio yang berbeda. Untuk data uji pertama didapatkan nilai rasio 37,24% dari hasil pengujian, yang berarti setelah dikompresi ukuran data menjadi 37,24% dari data sebelum dilakukannya kompresi. Sedangkan untuk data uji kedua didapatkan nilai rasio 26,77% dari hasil pengujian, yang berarti setelah dikompresi ukuran data menjadi 26,77% dari data sebelum dilakukannya kompresi.
Selain pengujian terhadap proses kompresi dan dekompresi, pengujian akan dilakukan terhadap proses *transfer* . Berikut ini merupakan hasil pengujian data dengan melakukan proses *transfer file* serta membandingkan waktu proses *transfer file* yang dibutuhkan dalam proses *transfer* menggunakan dengan waktu yang dibutuhkan dalam proses *transfer file* hasil kompresi tanpa aplikasi dan proses *transfer file* asli. Hasil dapat dilihat pada tabel 18.

Tabel 18 Hasil Pengujian Dan Perbandingan Waktu yang dibutuhkan dalam proses Transfer File

No	Nama File	Ukuran (byte)	Waktu (ms)	
		File Asli	Kirim setelah terjadinya kompresi	Kirim Tanpa Terjadinya Kompresi
1	big_buck_bunny_360p	673430026	00:00:25	00:00:59
2	Elephants_Dream_360p	4947349933	00:05:53	00:08:27

Pada tabel 18 telah dijelaskan hasil dari pengujian proses *transfer* dan menunjukkan perbedaan waktu yang diperlukan dalam melakukan proses *transfer* tersebut. Dapat disimpulkan bahwa penggunaan waktu yang lebih cepat pada proses *transfer file* yang telah melalui proses kompresi dibandingkan pada proses *transfer file* tanpa terjadinya kompresi terlebih dahulu.

3. KESIMPULAN

Setelah melakukan analisis serta pengujian pada sistem yang dibangun, maka dapat diperoleh kesimpulan sebagai berikut :

Proses *transfer file* hasil kompresi lebih cepat dibandingkan pada proses *transfer file* tanpa terjadinya proses kompresi terlebih dahulu.

Proses kompresi dengan menggunakan algoritma LZSS dan algoritma LZMA terhadap *file* video bertipe avi ini menghasilkan *file* dengan ukuran kurang dari setengah ukuran *file* asli, sehingga dapat menghemat penggunaan media penyimpanan.

4. DAFTAR PUSTAKA

[1] Nurkamid, Mukhamad, *Analisa Keefektifan Jaringan Local Area Network (Intranet) Universitas Muria Kudus*, Universitas Muria Kudus, Kudus, 2011.

[2] Bell, T.C., Cleary, J.G and Witten, I.H , *Text Compression*, Prentice-Hall. Englewood, USA, 1990.

[3] Hu, Ningning, *Network Aware Data Transmission With Compression*, In The 4th Annual CMU Symposium on Computer System, 2001.

[4] Solomon, David., *Data Compression, The Complete Reference. 4th edition*, Springer, Inc : United States of America, 2004.

[5] Salomon, D. & Motta, G., *Handbook Of Data Compression*. 5th *edition*, Springer, Inc : United States Of America, 2010.

[6] Nelson, Mark., *The Data Compression Book*, IDG Books Worldwide, Inc : United States of America, 2000.