ANALISIS TEORI DAN PENGUKURAN KINERJA CRC

Oleh:

Brahmanti Prameswari

G64101040

DEPARTEMEN ILMU KOMPUTER
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
INSTITUT PERTANIAN BOGOR

2005

ANALISIS TEORI DAN PENGUKURAN KINERJA CRC

Skripsi
sebagai salah satu syarat untuk memperoleh gelar
Sarjana Komputer
pada
Fakultas Matematika dan Ilmu Pengetahuan Alam
Institut Pertanian Bogor

Oleh:

Brahmanti Prameswari

G64101040

DEPARTEMEN ILMU KOMPUTER
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
INSTITUT PERTANIAN BOGOR

2005

ABSTRAK

BRAHMANTI PRAMESWARI. Analisis Teori dan Pengukuran Kinerja CRC. Dibimbing oleh SUGI GURITMAN dan IRMAN HERMADI.

Cyclic redundancy codes (CRC) adalah salah satu kode pendeteksi *error* yang mampu memberikan perlindungan yang baik terhadap integritas data. Mengingat arti penting tugas yang diemban oleh CRC ini, maka menjadi suatu kesadaran penuh bagi para pelaku industri untuk terus melakukan penelitian-penelitian yang dapat meningkatkan kinerja CRC. Penelitian kali ini mencoba untuk memberikan suatu ulasan mengenai CRC secara lebih mendalam dimulai dengan menganalisis teori pembuatannya sampai dengan menganalisis teori pengukuran kinerjanya. Dalam penelitian ini CRC diimplementasikan dalam bentuk simulasi yang dicobakan pada contoh kasus CRC-32.

Hasil dari penelitian ini menunjukkan bahwa pemilihan CRC yang tepat sebagai salah satu kode pendeteksi *error* telah terbukti mampu memberikan perlindungan yang baik terhadap integritas data. Namun untuk percobaan pengukuran kinerja CRC, ternyata pada prakteknya sangat sulit dicobakan pada CRC-32 karena melibatkan begitu banyak komputasi. Oleh karena itu, agar tetap bisa memberikan suatu gambaran yang jelas mengenai mekanisme pengukuran kinerja CRC tersbut maka akhirnya pengukuran kinerja CRC pada penelitian kali ini dicobakan pada CRC-8.

Kata Kunci: CRC, *Polynomial.*

Judul : **Analisis Teori dan Pengukuran Kinerja CRC**
Nama : **Brahmanti Prameswari**
NRP : **G64101040**

Menyetujui,

Pembimbing I

Dr. Sugi Guritman
NIP. 131 999 582

Pembimbing II

Irman Hermadi, S. Kom, MS.

Mengetahui,
Dekan Fakultas Matematika dan Ilmu Pengetahuan Alam

Dr. Ir. H. Yonny Koesmaryono, M.S.
NIP. 131 473 999

Tanggal Lulus :

RIWAYAT HIDUP

Penulis dilahirkan di Jakarta pada tanggal 10 Juli 1984 sebagai anak kedua dari pasangan Hartono dan Kustiningsih. Pada tahun 1995 penulis menyelesaikan pendidikan sekolah dasar di SDN Cawang 01 Pagi, tahun 1998 menyelesaikan pendidikan sekolah menengah pertama di SMPN 216 Jakarta Pusat, dan tahun 2001 menyelesaikan pendidikan sekolah menengah umum di SMUN 68 Jakarta Pusat. Pada tahun yang sama (2001) penulis memasuki pendidikan tinggi di Departemen Ilmu Komputer Institut Pertanian Bogor melalui jalur USMI.

Selama perkuliahan, penulis aktif di berbagai kegiatan mahasiswa. Pada tingkat akhir perkuliahan, penulis berkesempatan untuk melakukan kerja praktek di *Center for International Forestry Research* (CIFOR) untuk mengembangkan suatu aplikasi e-tutorial, dan kemudian pada periode berikutnya penulis terlibat sebagai *consultant* dalam suatu proyek publikasi untuk lembaga tersebut.

PRAKATA

Tiada kata yang pantas diucapkan selain puja dan puji syukur kepada Allah SWT, yang telah memberikan karunia yang luar biasa kepada penulis. Sholawat dan salam semoga selalu terarah kepada baginda Rasulullah SAW beserta keluarga, sahabat & pengikutnya dan semoga kita semua termasuk di dalamnya.

Merupakan suatu kebahagian tak terkira akhirnya penulis dapat menyelesaikan penelitian ini sebagai salah satu syarat kelulusan studi di Ilmu Komputer IPB. Karya akhir ini merupakan suatu momentum indah akhir dari usaha pencapaian yang sangat panjang, walaupun sangat disadari ini bukanlah puncak dari segala-galanya. Oleh karena itu tidaklah berlebihan jikalau penulis merasa keberhasilan yang ada saat ini bukanlah semata merupakan hasil jerih payah individu, melainkan merupakan hasil kerjasama segenap pihak yang terus mendukung penulis, baik secara moril maupun materiil. *Matur nuwun sanget* untuk orang-orang yang telah berpasrtisipasi didalamnya.

Walaupun tidak bisa dituliskan satu per satu, namun penulis hendak menghaturkan banyak terima kasih kepada pihak-pihak yang disebutkan dibawah ini:

1. Bapak Sugi Guritman dan Bapak Irman Hermadi, terimakasih atas kerjasama dan dukungan penuhnya selama penelitian ini.
2. Bapak Heru Sukoco selaku dosen penguji yang telah memberikan saran-saran yang membangun.
3. *My beloved family*; Bapak, Mama, Mba Anggi, Dimas Adi dan De' Dita. *I love u and I really love u*. *I am nothing without you*.
4. Da' Ridha, terimakasih untuk paket (dukungan, kritik, saran, kebersamaan) indahnya.
5. Hendra Saputra dan Khamamudin, terimakasih atas kesediaannya menjadi pembimbing ketiga ☺. Juga untuk para pembahas dalam seminar penulis; Hamzah Agung, Ratih Ramadhini, dan Sifilia.
6. Sahabat – sahabat: Putri, Elmi, Yani, Fajar dan semua M5 *crew* (2001-2003), terimakasih buat *never ending support* nya.
7. Seluruh teman-teman ILKOMERZ 38 dan juga teman-teman penulis di tempat lain yang telah memberikan warna warni dalam kehidupan Penulis.
8. Pak Tisna, Pak Fathur, Pak Ruhyan, Pak Pendi, Pak Soleh dan Mas Irfan, terimakasih atas bantuan-bantuannya.
9. Yang terakhir terimakasih yang sebesar-besarnya untuk semua guru-guru (dalam semua artian) penulis, tiada hal lain yang dapat dipersembahkan melainkan hanya sebuah karya kecil ini.

Akhir kata semoga penelitian ini dapat bermanfaat sehingga menjadi suatu amalan yang tiada terputus bagi diri penulis dan semua pihak yang telah turut membantu. Amin.

Jakarta, Oktober 2005

Brahmanti Prameswari

DAFTAR ISI

Halaman

DAFTAR GAMBAR

Halaman

DAFTAR LAMPIRAN

Halaman

PENDAHULUAN

Latar Belakang

Integritas data dalam komunikasi digital merupakan hal yang sangat penting karena apabila terjadi kesalahan dalam penerimaan data dapat mengakibatkan hal yang sangat fatal. Ada dua macam serangan yang perlu diwaspadai dalam menjaga integritas data, yaitu serangan secara *cryptographic* dan *non cryptographic*.

Perlindungan data terhadap serangan *cryptographic* bertujuan untuk melindungi data dari serangan-serangan musuh, baik musuh aktif maupun pasif, yang berkeinginan untuk dapat mengetahui ataupun mengubah isi dari suatu data. Oleh karena itu perlindungan terhadap serangan ini sangat memperhatikan aspek-aspek keamanan seperti kerahasiaan, autentikasi, dll.

Sedangkan perlindungan data terhadap serangan *non cryptographic* bertujuan untuk melindungi data dari kerusakan teknis yang mungkin terjadi selama berlangsungnya proses pengiriman data. Seperti misalnya interfensi gelombang elektro magnetik, intensitas cahaya, dll. Perlindungan terhadap serangan ini dapat dilakukan dengan menggunakan salah satu teknik dari *error control codes*, yaitu *error detection codes* atau *error correction codes*.

Cyclic redundancy codes (CRC) merupakan salah satu contoh algoritme dari *error detection codes* yang memiliki kinerja yang cukup baik. Beberapa CRC yang telah banyak dijadikan standar industri antara lain CRC-12, CRC-16, dan yang sekarang banyak digunakan adalah CRC-32.

Penelitian kali ini mencoba untuk memaparkan kembali teori-teori pembentukan CRC yang telah ada dan kemudian merekonstrusinya dalam sebuah implementasi yang mampu mensimulasikan kerja dari CRC tersebut. Tidak hanya itu, pada penelitian kali ini juga dilengkapi dengan paparan teori-teori mengenai pengukuran kinerja CRC yang juga akan direkonstruksi dalam sebuah implementasi yang mampu mensimulasikan pengukuran kinerja untuk berbagai jenis CRC. Sehingga dengan begitu diharapkan penelitian kali ini mampu memberikan suatu uraian tuntas mengenai CRC dari awal pembuatannya hingga pengukuran kinerjanya.

Tujuan

Tujuan penelitian ini adalah :

1. Mempelajari dan memahami teknik kode pendeteksi *error* dengan menggunakan algoritme CRC dan mengimplementasikannya.
2. Mempelajari teknik pengukuran kinerja CRC dan mengimplementasikannya.
3. Menganalisis hasil yang diperoleh dari masing-masing implementasi.

Ruang Lingkup

Seluruh implementasi yang akan dihasilkan pada penelitian kali ini dibuat khusus untuk CRC-32 saja. Untuk implementasi pertama, yaitu simulasi kerja CRC-32, implementasi akan dibuat untuk dapat mensimulasikan semua kemungkinan jenis dari CRC-32. Namun untuk implementasi kedua, yaitu pengukuran kinerja CRC, CRC-32 yang akan dianalisa hanya dibatasi pada tiga jenis saja, kemudian tiap jenis ini akan dicobakan pada 12 panjang data yang berbeda. Untuk masing-masing percobaan akan dilengkapi dengan total waktu pemrosesannya.

TINJAUAN PUSTAKA

Brute Force Search

Brute force search atau *exhaustive key search* adalah suatu teknik dasar pencarian kunci yang dilakukan dengan mencoba semua kemungkinan kombinasi kunci.

Burst Error

Burst error dengan panjang *x bit* adalah urutan *bit-bit* sepanjang *x* dimana *bit* pertama, *bit* terakhir, dan sebagian besar atau semua *bit-bit* diantaranya terkena *error* (Stalling 2004).

Check Bit

Check bit adalah *bit-bit* yang sengaja dibuat berdasarkan suatu algoritme tertentu yang berfungsi untuk memberikan jaminan terhadap integritas suatu data.

Codeword

Codeword adalah sebuah data sepanjang *n bit* yang terdiri dari *k bit* data awal dan *n-k bit check bit*.

Field

Field ***F*** adalah himpunan dari elemen-elemen dimana dengan dua operasi penjumlahan (+) dan perkalian (•) dapat memenuhi aksioma-aksioma sebagai berikut:

(i) *F* tertutup pada + dan • , sehingga jika *a* dan *b* ∈ *F*, maka *a* + *b* dan *a* • *b* juga ∈ *F*.

Untuk semua *a, b, c* dalam *F*, akan berlaku:

(ii) Hukum komutatif: $a + b = b + a$, $a \bullet b = b \bullet a$

(iii) Hukum asosiatif: $(a + b) + c = a + (b + c)$, $a \bullet (b \bullet c) = (a \bullet b) \bullet c$

(iv) Hukum distributif: $a \bullet (b + c) = a \bullet b + a \bullet c$

Lebih lanjut, elemen identitas yaitu 0 dan 1 harus ada dalam *F* memenuhi:

(v) $a + 0 = a$

(vi) $a \bullet 1 = a$

(vii) Untuk setiap *a* dalam *F*, akan mempunyai sebuah *additive inverse* dimana $a + (-a) = 0$

(viii) Untuk setiap a dalam *F*, akan mempunyai sebuah *multiplicative inverse* dimana $a \bullet a^{-1} = 1$.

(Bose 2003).

***Galois Field* (GF)**

Galois field adalah *field* yang memiliki jumlah elemen yang berhingga, misal *q*, yang disebut orde dan dinotasikan sebagai GF (*q*) (Bose 2003).

***Hamming Distance* (HD)**

Hamming distance antara dua *codeword* (atau vektor lainnya) adalah banyaknya jumlah bit yang berbeda dari masing-masing *codeword* tersebut (Bose 2003).

Integritas Data

Integritas data adalah suatu keadaan dimana data tidak mengalami perubahan yang disebabkan oleh hal-hal yang tidak diinginkan sejak data itu dibuat, ditransmisikan atau disimpan oleh pihak yang berwenang (Menezes *et al.* 1996).

Polynomial

Polynomial adalah sebuah ekspresi matematika:

$$f(x) = f_m x^m + f_{m-1} x^{m-1} + \ldots + f_1 x + f_0$$

dengan *x* adalah *dummy variable* dan koefisien $f_0, f_1, \ldots, f_m$ adalah elemen dari GF (*q*). Koefisien f_m dinamakan koefisien utama. Jika $f_m \neq 0$, maka *m* adalah derajat dari *polynomial* tersebut, yang kemudian akan dilambangkan dengan deg *f*(*x*) (Bose 2003).

Shift Register

Shift register adalah sebuah *string* yang berisi kumpulan tempat penyimpanan sebesar 1 *bit* (1 *bit storage devices*). Setiap *shift register* mempunyai sebuah *input line* yang berfungsi sebagai jalan masuknya data dan sebuah *output line* yang mengindikasikan nilai yang tersimpan dalam masing-masing tempat penyimpanan itu. Kemudian pada suatu kurun waktu tertentu secara kontinyu nilai yang ada pada masing-masing tempat penyimpanan ini akan bergeser, digantikan oleh nilai-nilai baru dari *input line* (Stalling 2004).

***Weight* (W)**

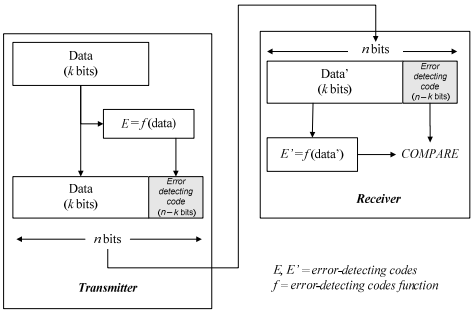
Weight adalah jumlah terjadinya kegagalan deteksi *error* untuk suatu jumlah *error* tertentu (Koopman 2002).

***Message Transmission Unit* (MTU)**

Message transmission unit (MTU) adalah banyaknya jumlah *bit* data yang dapat dikirimkan dalam satu kali transmisi. Panjang MTU berbeda-beda bergantung dari protokol yang digunakan.

DESKRIPSI ALGORITME CRC

Cyclic redundancy codes atau yang banyak juga disebut *cyclic redundancy checks* bekerja layaknya prinsip algoritme-algoritme lain yang ada pada *error detection codes* seperti yang dapat dilihat pada gambar berikut ini:



Gambar 1 Proses pendeteksian e*rror* (Stalling 2004).

Dapat dilihat pada Gambar 1, misalkan *transmitter* hendak mengirimkan sebuah data sepanjang *k bit*, maka sebelum dikirimkan, akhir dari data tersebut ditambahkan *bit-bit* sebanyak ($n - k$) yang akan berfungsi sebagai *check bit*, *bit-bit* ini didapatkan melalui perhitungan dari suatu algoritme tertentu yang telah disepakati bersama oleh *transmitter* dan *receiver.* Setelah melalui semua prosedur tersebut, maka terbentuklah sebuah *codeword* sepanjang *n bit* yang siap dikirimkan. Setelah *codeword* diterima, *receiver* akan melakukan perhitungan yang sama terhadap data yang diterima. Kemudian, *receiver* akan membandingkan hasil perhitungannya dengan *check bit* yang dikirimkan. Apabila terjadi ketidakcocokan maka dapat dipastikan bahwa telah terjadi *error* selama proses pengiriman data.

Algoritme yang paling sederhana untuk mendapatkan *check bit* ini adalah dengan menambahkan satu *bit parity* di belakang data yang berguna untuk menggenapkan atau mengganjilkan jumlah *bit* tak nol dari data yang hendak dikirimkan. Kemudian karena penambahan *parity bit* dianggap tidak memadai untuk mendeteksi sekian banyak kombinasi *error*, maka berkembanglah algoritme *error detection* yang lain, yaitu c*hecksum*. *Checksum* merupakan hasil penjumlahan sederhana dari *bit-bit* data yang hendak dikirimkan. Namun ternyata, seperti yang telah dapat diperkirakan, *checksum* pun tidak cukup memberikan perlindungan yang baik terhadap integritas data. Oleh karena itu dibutuhkan suatu algoritme lain yang lebih canggih untuk melakukan hal ini, dan CRC adalah solusi yang tepat (William 1993).

Ide Dasar CRC

Ide dasar CRC adalah memperlakukan *string* data sebagai *bit-bit* biner dan kemudian melakukan pembagian antara data tersebut dengan suatu *polynomial* GF (2) yang memiliki koefisien tertinggi dan terendah sama dengan 1. Penggunaan *polynomial* ini sebelumnya telah disepakati bersama oleh *transmitter* dan *receiver*. Sisa bagi dari proses inilah yang digunakan sebagai *check bit* dan kemudian akan ditambahkan pada bagian akhir dari data sehingga membentuk *codeword*. Oleh karena itu setiap c*odeword* yang dihasilkan pasti akan dapat habis dibagi oleh *polynomial* yang telah disepakati tersebut. Sehingga jika nantinya *receiver* menemukan hasil tak nol dari pembagian antara data yang diterima dengan *polynomial*, maka dapat dipastikan telah terjadi *error*.

Namun sebelum proses itu dilakukan ada beberapa hal yang perlu diperhatikan, antara lain penggunaan *polynomial* GF (2), atau yang sering disebut dengan *field biner*. *Field biner* memiliki operasi XOR untuk penjumlahan dan operasi AND untuk perkalian. Aritmetika semacam ini dikenal sebagai aritmetika modulo 2.

Selain itu sebelum dilakukan perhitungan pembagian modulo 2, data yang hendak dikirimkan dikalikan dengan 2^{n-k} , dalam aritmetika modulo 2 mengkalikan suatu bilangan dengan 2^i akan memberikan *effect* penambahan angka 0 sebanyak *i* dibelakang bilangan tersebut. Hal ini penting dilakukan agar *check bit* dapat digabungkan pada akhir data.

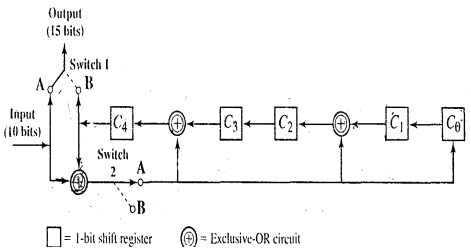
Untuk pembuktian secara matematis dapat dilihat pada Lampiran 1. Dan untuk mendapatkan gambaran yang lebih jelas dari cara kerja CRC, pada Lampiran 2 disertakan contoh perhitungan CRC secara detail.

Walaupun sepintas perhitungan CRC ini tampak rumit, namun dalam implementasinya telah dibuktikan bahwa sebuah sirkuit *shift register* yang sederhana mampu untuk melakukan perhitungan tersebut.

Sirkuit *shift register* itu dikonstruksi sebagai berikut:

1. Ada sebanyak *n – k bit register*, dimana *n* adalah total *bit codeword* dan *k* adalah total *bit* data.
2. Ada paling banyak *n – k gate* XOR, yang penentuan letaknya ditentukan berdasarkan koefisien *polynomial* yang digunakan.

Dari keterangan di atas, dapat dicontohkan bahwa sirkuit untuk *polynomial* $x^5 + x^4 + x^2 + 1$ adalah sebagai berikut:



Gambar 2 Implementasi *Shift Register* (Stalling 2004).

Contoh proses penggunaan *shift register* yang ditunjukkan secara langkah per langkah dapat dilihat pada Lampiran 3.
(Stalling 2004)

Pemilihan *Polynomial*

Jika berbicara mengenai kinerja dari sebuah CRC, dan ingin tahu mengapa suatu CRC dapat menghasilkan kinerja yang lebih baik dari CRC yang lainnya, maka jawabannya terletak pada usaha pemilihan *polynomial* pembagi yang tepat bagi CRC tersebut.

Untuk menjelaskan hal itu, perhatikan penjelasan berikut ini, misalkan T(x) adalah sebuah *codeword*, G(x) adalah *polynomial*, dan E(x) adalah *error* yang terjadi. Sekarang bayangkan jika T(x) akan dikirim dan selama berlangsungnya pengiriman terjadi *error* yang menyebabkan perubahan *bit-bit* pada T(x) sehingga *codeword* yang akan diterima oleh *receiver* adalah T(x) + E(x). *Receiver* kemudian akan mengecek sisa bagi antara (T(x) + E(x)) dengan G(x), jika (T(x) + E(x)) mod G(x) = 0, maka *receiver* akan menganggap bahwa data yang diterima adalah valid, dengan demikian berarti telah terjadi kegagalan deteksi *error*. Untuk menghindari hal semacam itu, perhatikan bahwa (T(x) + E(x)) mod G(x) = T(x) mod G(x) + E(x) mod G(x), karena T(x) mod G(x) = 0 (*lihat* Ide Dasar CRC) maka (T(x) + E(x)) mod G(x) = E(x) mod G(x). Oleh karena itu untuk menghindari terjadinya *error* yang tidak terdeteksi maka nilai E(x) mod G(x) ≠ 0. Dan jelas terlihat bahwa untuk menghindari kondisi dimana E(x) mod G(x) = 0, maka satu-satu nya hal yang perlu dilakukan adalah dengan memilih *polynomial* yang tepat, yaitu mencari nilai G(x) yang tidak akan pernah membagi habis semua kemungkinan E(x). Ada beberapa petunjuk yang dapat digunakan untuk mencari *polynomial* terbaik, beberapa diantaranya akan dijelaskan pada paragraf di bawah ini.

Pertama, untuk melindungi *error* sebanyak 1 *bit*, ditulis E(x) = x^i, dimana i menunjukkan posisi *bit error*. Maka jika G(x) adalah *polynomial* yang memiliki minimal dua atau lebih *bit*, dapat dipastikan G(x) tidak akan mungkin membagi habis E(x). Oleh karena itu, semua 1 *bit error* pasti akan terdeteksi oleh G(x).

Kedua, untuk melindungi dua buah *error* masing-masing sebesar 1 *bit*, ditulis E(x) = $x^i + x^j$, dimana $i > j$. Maka E(x) dapat dituliskan sebagai berikut E(x) = x^j (x^{i-j} + 1). Dan kemudian jika ingin G(x) tidak membagi habis E(x), maka harus diyakinkan bahwa G(x) bukanlah merupakan faktor dari x^j ataupun (x^{i-j} + 1). Agar G(x) tidak menjadi faktor dari x^j, maka G(x) haruslah terdiri dari minimal dua atau lebih *bit* (*lihat* penjelasan pertama). Selanjutnya agar G(x) tidak menjadi faktor dari (x^k + 1) untuk semua nilai k, dimana $k = i - j$, maka pilih G(x) yang tidak hanya terdiri dari (x + 1), (x^2 + 1), (x^3 + 1), (x^4 + 1), dst. Sebagai contoh untuk G(x) = $x^{15} + x^{14} + 1$, maka G(x) tidak akan pernah habis membagi x^k + 1, untuk nilai k dibawah 32.768.

Ketiga, untuk melindungi dari semua *error* yang berjumlah ganjil, maka G(x) perlu diatur agar memiliki (x + 1) sebagai salah satu faktornya, hal ini dikarenakan dalam aritmetika modulo 2 dapat dipastikan tidak akan ada bilangan dengan jumlah *bit* 1 ganjil yang akan habis dibagi dengan (x + 1) . Untuk membuktikan hal ini, asumsikan E(x) adalah *error* yang berjumlah ganjil dan E(x) habis dibagi dengan (x + 1), maka E(x) dapat difaktorkan menjadi (x + 1) Q(x). Sekarang periksa jika x = 1, maka E(1) = (1 + 1) Q (1), E(1) = 0 Q (1) = 0. Karena E(1) akan selalu sama dengan 0, tanpa dipengaruhi oleh Q(1), maka dapat terlihat bahwa tidak mungkin ada sebuah bilangan yang memiliki *bit* 1 ganjil yang dapat dibagi habis oleh (x + 1).

Terakhir, yang perlu diketahui adalah *polynomial* dengan panjang r akan mampu mendeteksi semua *burst error* yang panjangnya tidak melebihi r. Perhatikan, misal *burst error* dengan panjang k direpresentasikan E(x) = x^i (x^{k-1} + ... + 1), dengan i menunjukkan jarak antara posisi *bit* pertama pada data awal dengan posisi terjadinya *burst error*. Pemakaian *polynomial* G(x) = $x^r + g_{r-1}x^{r-1}$ +…+ g_1x + 1, dimana g_1, g_2, .., g_{n-k-1} sama dengan 0 atau 1 maka dapat dipastikan bahwa G(x) bukanlah faktor dari x^i (*lihat* penjelasan pertama) dan selama derajat dari (x^{k-1} + ... + 1) tidak melebihi derajat G(x) maka juga dapat dipastikan bahwa G(x) bukanlah faktor dari (x^{k-1} + ... + 1). Sehingga terbukti bahwa *polynomial* G(x) = $x^r + g_{r-1}x^{r-1}$ +…+ g_1x + 1 akan selalu mampu mendeteksi semua *burst error* yang panjangnya tidak melebihi r.

Jika panjang dari *burst error* sama dengan r + 1, maka sisa bagi dari G(x) akan sama dengan 0 jika dan hanya jika *burst error* tersebut identik dengan G(x). Dan berdasarkan definisi dari *burst error* dapat dipastikan

bahwa *bit* pertama dan *bit* terakhir dari *error burst* selalu sama dengan 1, sehingga faktor penentu E(x) dapat identik dengan G(x) terletak pada r - 1 *bit* antaranya. Oleh karena itu, kemungkin terjadinya hal semacam ini adalah sebesar $1/2^{r-1}$.

Sedangkan untuk *burst error* dengan panjang lebih dari r+1, maka kemungkinannya adalah $1/2^r$ (dengan mengasumsikan bahwa semua *bit* akan tepat sama).
(Tanenbaum 1981)

Pengukuran Kinerja CRC

Kinerja CRC dapat diukur dari seberapa banyak *error* yang mampu terdeteksi oleh CRC. Secara umum hal ini dapat diekspresikan dengan menggunakan *hamming distance* (HD). Nilai HD ini sendiri ditentukan dengan melihat elemen dari *weight polynomial* yang memiliki nilai tak nol pertama. Elemen *weight* ini merepresentasikan banyaknya *bit error* yang terjadi. Contoh, misalkan dari hasil penelitian ditemukan bahwa *polynomial* IEEE Std. pada panjang data 12112 *bit* memiliki *weight* untuk masing-masing panjang *error* adalah sebagai berikut; $W_2 = 0$, $W_3 = 0$, dan $W_4 = 223059$. Maka nilai HD untuk CRC sama dengan empat. Ini artinya CRC tersebut berhasil mendeteksi semua kemungkinan 2 *bit error*, 3 *bit error* namun gagal mendeteksi 223059 kemungkinan 4 *bit error*. Nilai weight itu sendiri dicari dengan mendaftar semua kemungkinan kombinasi E(x) dari ruang kombinasi $C_4^{(12112+32)} = \frac{12144!}{4!.12140!} = 906.10^{12}$ yang memenuhi persamaan E(x) mod G(x) = 0 (ingat bahwa *error* akan tidak terdeteksi jika dan hanya jika memenuhi persamaan di atas, *lihat* Pemilihan *Polynomial*).

Pencarian nilai *weight* untuk elemen *weight* yang di atas nilai HD tidak perlu dilakukan karena peluang terjadinya *error* untuk jumlah-jumlah *error* tersebut lebih kecil dari peluang terjadinya *error* pada nilai HD, dan akan terus relatif konstan untuk jumlah-jumlah *error* seterusnya.

Oleh karena itu, semakin tinggi nilai HD dari suatu *polynomial* maka semakin baik pulalah kinerja CRC tersebut.
(Koopman 2002)

Pencarian *Weight* CRC

Seperti yang telah disebutkan pada bagian Pengukuran Kinerja CRC, nilai *weight* didapatkan dengan cara menghitung jumlah banyaknya *error* yang tidak terdeteksi. *Error* akan tidak terdeteksi jika dan hanya jika E(x) mod G(x) = 0, oleh karena itu setiap E(x) yang merupakan kelipatan dari G(x) pasti akan menghasilkan sisa bagi sama dengan 0. Namun perlu diingat bahwa pembagian pada CRC, bukanlah pembagian antar bilangan biasa melainkan pembagian dengan suatu bilangan GF(2), yang artinya untuk mencari kelipatan G(x) tidak hanya semata mengkalikan G(x) dengan suatu bilangan seperti layaknya pencarian kelipatan pada perhitungan bilangan desimal.

Pada perhitungan pembagian dengan bilangan GF(2), berlaku sifat-sifat yang ada pada GF(2). Salah satu sifat GF(2) yang dapat dimanfaatkan kali ini adalah *linearity* dari perhitungan XOR, yaitu:

CRC (A xor B) = CRC (A) xor CRC (B)

Sekarang jika dimisalkan sebuah *codeword* memiliki P *bit* data dan E *bit check bit* dan kemudian dibuat sebuah vektor *error* yang menyerang V *bit* pada data dan W *bit* pada *check bit.* Maka *codeword* yang akan diterima adalah X *bit* data dan Y *bit check bit*, dimana X = (P xor V) dan Y = (E xor W).

Berdasarkan definisi diketahui :

E = CRC (P) .....(1)

Error akan tidak terdeteksi jika dan hanya jika:

CRC (X) = Y .....(2)

Dari keterangan di atas, nilai X dan Y pada Persamaan 2 dapat disubsitusikan menjadi:

CRC (P xor V) = E xor W.....(3)

Dengan menggunakan prinsip liner dari GF(2), Persamaan 3 dapat diubah menjadi:

CRC (P) xor CRC (V) = E xor W.....(4)

Berdasarkan Persamaan 1, nilai E pada Persamaan 4 dapat disubsitusi dengan CRC (P), sehingga:

CRC(P) xor CRC (V) = CRC (P) xor W..(5)

Persamaan 5 dapat disederhanakan menjadi :

CRC (V) = W.....(6)

Dengan demikian, berarti dapat dipastikan bahwa setiap vektor *error* yang membentuk *codeword* pasti tidak akan dapat terdeteksi oleh CRC.

Berdasarkan hasil perhitungan tersebut, maka didapat cara untuk melakukan perhitungan *weight*, yaitu dengan cara

memeriksa seluruh kemungkinan *k bit error* pada *n bit codeword* apakah ada yang memenuhi persamaan CRC (V) = W, jika tidak ada yang memenuhi persamaan, berarti W untuk *k* tersebut = 0. Untuk mendapatkan nilai HD nya, iterasi akan terus di ulang dari 1 sampai dengan k_{max}, dimana k_{max} adalah nilai *k* dengan $W_k \neq 0$ yang pertama.
(Chakravarty 2001)

PENELITIAN SEBELUMNYA

Penelitian dalam meningkatkan kinerja CRC telah dilakukan oleh banyak peneliti selama bertahun-tahun. Berbagai cara telah ditawarkan, salah satu nya dengan memperluas panjang *polynomial* yang digunakan.

Untuk CRC-32 sendiri, ada begitu banyak *polynomial* yang ditawarkan, dan salah satunya adalah $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$ yang telah dijadikan standar oleh IEEE pada tahun 1985 untuk pemakaian di Ethernet. *Polynomial* ini memiliki HD = 4 untuk panjang data 1 MTU pada Ethernet. Namun ternyata penemuan CRC ini belum mampu menutup rasa keinginan para peneliti untuk terus mencari CRC-CRC lain yang lebih baik, hal ini juga didorong oleh adanya kebutuhan industri yang menghendaki diciptakannya standar CRC yang lebih baik dari sebelumnya.

Oleh karena itu, setelah melalui proses panjang para ahli matematika yang diwakili oleh penelitian Castagnoli *et al.* pada tahun 1993, kemudian menjawab tantangan ini. Castagnoli *et al.* menawarkan *polynomial* $x^{32} + x^{28} + x^{27} + x^{26} + x^{25} + x^{23} + x^{22} + x^{20} + x^{19} + x^{18} + x^{14} + x^{13} + x^{11} + x^{10} + x^9 + x^8 + x^6 + 1$ yang dinilai lebih unggul dari *polynomial* yang digunakan pada IEEE Std. *Polynomial* ini memiliki HD = 4 untuk panjang data 1 MTU pada Ethernet, namun untuk data yang lebih pendek *polynomial* ini mampu memberikan mencapai HD = 6. Sheinwald pada *draft* i-SCSI (*Internet Protocol Small Computer System Interface*) tahun 2001 menuliskan bahwa ini merupakan kandidat *polynomial* CRC-32 terbaik yang dapat digunakan pada i-SCSI.

Tidak puas dengan hasil yang diberikan oleh Castagnoli *et al*., Philip Koopman pada tahun 2002 kemudian melakukan penelitian serupa. Namun untuk penelitiannya, Koopman melakukannya dengan cara yang berbeda. Daripada berkutat dengan penurunan rumus-rumus matematika, Koopman memilih untuk melakukan pencarian *polynomial* terbaik secara *brute force*. Menurut Koopman walaupun penelitian ini sulit untuk diterapkan (mengingat begitu besar ruang kunci yang harus dicobakan), namun pencarian *brute force* tetap penting untuk dilakukan karena metode ini akan menutup semua kemungkinan akan adanya *polynomial* lain yang lebih baik. Akhirnya dengan menggunakan metode *filtering* yang telah ia rancang sedemikan rupa dan setelah melalui percobaan selama tiga bulan penuh, Koopman berhasil menemukan *polynomial* lain yang lebih baik, yaitu $x^{32} + x^{30} + x^{29} + x^{28} + x^{26} + x^{20} + x^{19} + x^{17} + x^{16} + x^{15} + x^{11} + x^{10} + x^7 + x^6 + x^4 + x^2 + x + 1$, *polynomial* ini mencapai HD = 6 untuk panjang data 1 MTU pada Ethernet dan HD = 4 untuk data yang sangat panjang (mencapai 64 *Kbit*). Untuk lebih jelasnya, hasil dari penelitian Koopman ini dapat dilihat pada Lampiran 4.

METODE PENELITIAN

Pada penelitian kali ini akan dibuat suatu implementasi yang mampu mensimulasikan kerja dari CRC-32 dan juga mampu untuk melakukan perhitungan kinerja CRC-32. Oleh karena itu, percobaan yang perlu dilakukan dalam penelitian kali ini dibagi menjadi dua bagian utama, yang pertama yaitu pembuatan simulasi CRC-32 dan yang ke dua adalah pembuatan perhitungan kinerja CRC-32.

Untuk bagian pertama, yaitu simulasi CRC-32, yang akan dilakukan dalam percobaan kali ini adalah murni penerapan algoritme *simple shift register* yang diambil dari buku *Data and Computer Communications* (Stalling 2004). Setiap percobaan yang dilakukan akan dihitung total waktu pemrosesannya yang ketelitiannya akan diukur sampai dengan satuan milidetik. Pengukuran waktu hanya akan meliputi lamanya waktu pemrosesan CRC saja, tidak termasuk proses pengetikan hasil dan proses-proses lainnya.

Sedangkan untuk bagian kedua, yaitu perhitungan kinerja CRC-32 akan mengimplementasikan algoritme pencarian *weight* seperti yang telah dijelaskan pada Pencarian *Weight* CRC. Pada bagian ini akan dicobakan tiga *polynomial* yang didapat dari penelitian sebelumnya (IEEE Std, Castagnoli *et al.* dan Koopman) untuk dianalisis kinerjanya pada 12 panjang data yang berbeda, yaitu 64 *bit*, 128 *bit*, 256 *bit*, 512 *bit*, 1 *Kbit*, 2 *KBit*, 4 *KBit*, 8 *KBit*, 16 *KBit*, 32

KBit, 64 *KBit* dan 128 *KBit*. Dari masing-masing panjang data kemudian akan dicari *Hamming Distance* (HD) nya. *Polynomial* yang memiliki HD paling tinggi untuk data yang paling panjang adalah *polynomial* yang terbaik.

Sebagai tambahan, selain mengukur nilai HD, perbandingan juga akan dilakukan terhadap total waktu pemrosesan CRC yang dibutuhkan suatu *polynomial* pada tiap-tiap panjang data yang berbeda. Untuk melakukan hal ini, masing-masing *polynomial* dan panjang data terpilih akan dicari CRC nya dengan menggunakan implementasi pada bagian pertama yaitu simulasi CRC-32. Kemudian agar tidak terjadi galat error, percobaan akan diulang sebanyak sepuluh kali.

IMPLEMENTASI

Deskripsi Umum Implementasi

Implementasi ini dirancang dengan tiga tujuan utama:

1. Mensimulasikan kerja CRC-32 dengan pemakaian sembarang *polynomial* dan kemudian menghitung total waktu pemrosesannya.
2. Melakukan perhitungan terhadap kinerja CRC-32 versi IEEE Std., Castagnoli *et al.*, dan Koopman.
3. Melakukan perhitungan waktu rata-rata terhadap proses kerja CRC-32 versi IEEE Std., Castagnoli *et al.*, dan Koopman.

Perancangan Implementasi

Implementasi ini dibuat secara bertahap dimulai dari analisis kebutuhan, kemudian analisis *design* dan terakhir pengkodean.

Pada tahap analisis kebutuhan, didefinisikan semua permasalahan yang sekiranya ingin diselesaikan dengan menggunakan implementasi ini. Hasil dari tahap ini telah dicantumkan pada bagian deskripsi umum implementasi.

Setelah mendapatkan gambaran secara garis besar mengenai implementasi yang akan dibuat beserta *features* yang diinginkan, maka tahap selanjutnya yang harus dikerjakan adalah menganalisis *design*. Pada tahap ini spesifikasi kebutuhan akan diterjemahkan ke dalam sebuah representasi sistem sebelum proses pengkodean dapat dilakukan. Tahap ini terdiri dari empat proses, yaitu:

1. Perancangan *Input* dan *Output*

   Untuk simulasi CRC-32, implementasi ini hanya akan membutuhkan dua *input* dari *user*, yaitu data awal yang hendak dilindungi dengan CRC-32 dan *polynomial* CRC-32 yang *user* inginkan. Untuk mempermudah *user*, maka *input* data akan dimasukkan dalam bentuk karakter ASCII sedangkan untuk *input polynomial* dimasukkan dalam bentuk bilangan heksadesimal, hal ini dikarenakan sebagian besar *polynomial* biasanya direpresentasikan dalam bentuk bilangan heksadesimal. Misal, untuk $G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$ maka *user* harus memasukkan bilangan heksedesimal 82608EDB, dengan angka "8" pada bilangan heksadesimal pertama melambangkan empat *bit* teratas dari *polynomial* (x^{32} sampai x^{22}) dan demikian seterusnya sampai pada representasi heksadesimal terakhir yaitu "B" yang merepresentasikan ($x^4 + x^2 + x$), sedangkan "+1" tidak direpresentasikan secara eksplisit (hal ini sudah menjadi kewajaran dalam banyak implementasi CRC). Untuk *output*, implementasi ini akan menghasilkan tiga *output*. *Output* pertama yaitu hasil perhitungan CRC yang juga akan ditampilkan dalam bentuk bilangan heksadesimal, o*utput* kedua adalah hasil pengecekan CRC yang akan memberi informasi apakah selama pengiriman telah terjadi *error* atau tidak, dan o*utput* ketiga berupa total waktu yang dibutuhkan dalam proses perhitungan CRC.

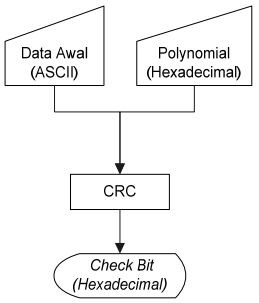
   Untuk perhitungan kinerja CRC-32, walaupun secara prinsip memiliki kinerja yang sama namun ada sedikit perbedaan pada tata cara pemasukan *input* dan *output* yang dihasilkan. *Input* yang dimasukkan pada bagian ini disesuaikan dengan faktor percobaan yang akan dianalisis. Oleh karena itu, pertama *user* diharuskan untuk memilih satu dari tiga pilihan *polynomial* yang akan digunakan (*polynomial* versi IEEE 802.3 (0x82608EDB), *polynomial* versi Castagnoli *et al.* (0x8F6E37A0), atau *polynomial* versi Koopman (0xBA0DC66B), setelah itu *user* kembali diharuskan untuk memilih, namun kali ini memilih panjang data yang ingin diujicobakan (ada 12 tingkatan panjang

data). Setelah dilakukan pemilihan, maka *user* dapat menekan suatu tombol untuk memulai proses percobaan yang diinginkan. *Output* yang dihasilkan adalah informasi nilai HD untuk percobaan tersebut.

Sedangkan untuk percobaan pengukuran waktu rata-rata, karena faktor yang akan dianalisis sama dengan faktor yang dianalisis pada perhitungan kinerja CRC-32, maka percobaan ini akan memiliki *input* yang sama dengan yang ada pada perhitungan kinerja CRC-32, yang berbeda hanya pada *output* yang berupa waktu rata-rata percobaan (dalam satuan milidetik).

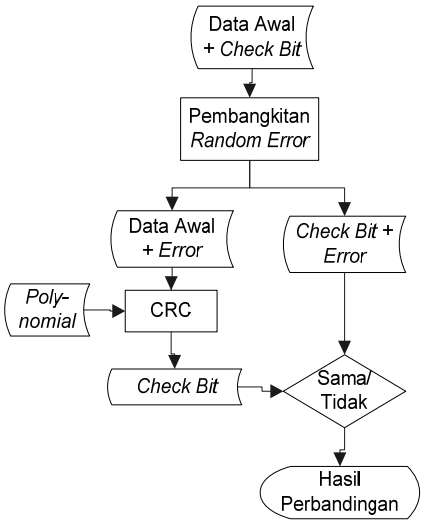
2. Perancangan Proses

Untuk simulasi CRC-32, *Input* yang telah dimasukkan oleh *user* kemudian akan diolah oleh algoritme CRC. Algoritme ini memproses *input* per *byte*, setelah semua *byte* selesai dikalkulasi maka akan ditampilkan hasil perhitungan CRC yang berfungsi sebagai *check bit*. Diagram alirnya dapat digambarkan sebagai berikut:



Gambar 3 Diagram alir proses perhitungan CRC-32.

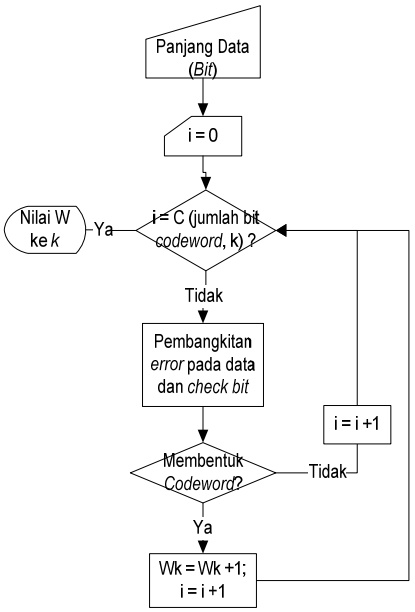
Setelah itu data awal beserta *check bit* akan diberikan *error* secara *random*, proses penempatan *error* ini dibuat sedemikian rupa sehingga dapat memproses *error* per *bit* dan dapat diyakinkan bahwa tidak akan ada *error* yang dapat menempati posisi yang sama. Kemudian data yang telah rusak tersebut akan dihitung ulang dengan menggunakan algoritme CRC beserta *polynomial* yang sama, lalu diperiksa apakah menghasilkan *check bit* yang sama atau tidak. Diagram alir digambarkan sebagai berikut:



Gambar 4 Diagram alir proses pengecekan CRC-32.

Yang tidak kalah penting dalam proses ini adalah pencatatan total waktu pengerjaan untuk satu proses CRC.

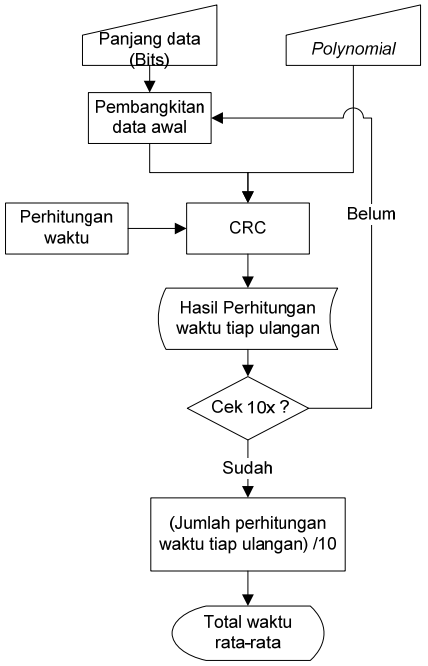
Untuk perhitungan kinerja CRC-32, proses dibuat persis seperti algoritme yang telah dijelaskan pada bagian Pencarian *Weight* CRC. Dimana proses untuk setiap k (1, ..., k_{max}) memiliki diagram alir seperti yang dapat terlihat pada gambar berikut:



Gambar 5 Diagram alir proses perhitungan kinerja CRC-32.

Proses ini dimulai dengan membangkitkan *error* sebanyak *k* pada *bit-bit codeword.* Kemudian untuk setiap kombinasi yang mungkin pada *codeword* akan diperiksa apakah ada yang membentuk *codeword*, jika ada maka nilai W untuk *k* tersebut akan bertambah satu, demikian seterusnya sampai seluruh kombinasi dari (jumlah *bit codeword*, *k)* telah diperiksa. Nilai HD nya adalah nilai k_{max} untuk *polynomial* tersebut.

Untuk perhitungan waktu pemrosesan rata-rata CRC-32, prosesnya cukup sederhana karena tidak banyak berbeda dari proses simulasi CRC-32 (tanpa termasuk proses pengecekannya). Yang berbeda pada proses ini adalah perlunya ditambahkan proses pembangkitan data awal dan juga penambahan modul yang dapat melakukan pencatatan waktu sampai pada satuan milidetik. Diagram alirnya digambarkan sebagai berikut:



Gambar 6 Diagram alir proses perhitungan waktu rata-rata CRC-32.

3. Algoritme Proses

Berdasarkan analisis pada perancangan proses, maka dapat disimpulkan bahwa terdapat dua proses utama yang dibutuhkan untuk membangun implementasi seperti yang diinginkan, yaitu proses CRC itu sendiri dan proses pembangkitan *error* acak. Algoritme untuk masing-masing proses tersebut akan disertakan dalam Lampiran 5 dan Lampiran 6.

Semua algoritme yang digunakan akan diujucobakan dahulu pada data-data yang pendek, kemudian akan diperiksa apakah telah menghasilkan hasil sesuai yang diinginkan atau tidak. Hal ini dimaksudkan agar hasil perhitungan dapat terjamin kevalidannya.

4. Perancangan Antarmuka Pengguna

Karena melibatkan tiga proses yang berbeda maka antarmukanya pun akan dibuat dengan menggunakan 3 *form* yang berbeda pula; yang pertama untuk simulasi CRC-32, yang kedua untuk pengukuran kinerja CRC-32, dan yang ketiga untuk pengukuran waktu rata-ratanya. Hal ini dimaksudkan agar implementasi lebih mudah dimengerti oleh *user.*

Antarmuka pada bagian simulasi CRC-32 akan dibagi menjadi dua bagian secara vertikal, untuk mensimulasikan adanya *transmitter* dan *receiver*.

Sedangkan antarmuka untuk bagian proses perhitungan kinerja CRC-32 dan perhitungan waktu rata-rata, disediakan pilihan-pilihan input yang dibutuhkan untuk memulai masing-masing percobaan, yaitu panjang data dan jenis *polynomial*.

Spesifikasi Implementasi

Implementasi dirancang dan dibangun dengan menggunakan perangkat keras dan lunak sebagai berikut:

Perangkat keras:

1. Prosesor dengan *clock speed* 940 MHz
2. Memori RAM 128 MB
3. Kapasitas *hard disk* 40 GB

Perangkat lunak:

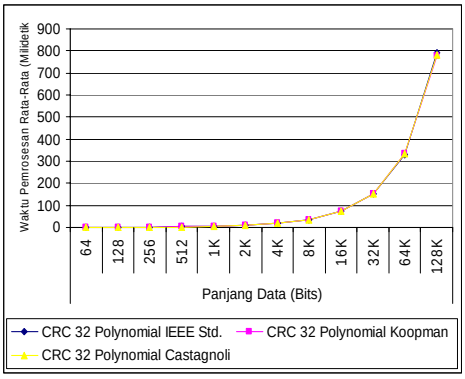
1. Sistem operasi Windows XP pada resolusi 1024 x 768 Pixel.
2. Visual Basic 6.0

HASIL DAN PEMBAHASAN

Setelah melalui analisis dan perancangan, maka proses selanjutnya yang harus dijalankan adalah proses pengkodean. Dan seperti yang telah dicantumkan pada bagian Deskripsi Umum Implementasi, implementasi yang akan dibuat kali ini akan dibagi menjadi tiga bagian yang berbeda, yaitu; implementasi yang mampu mensimulasikan kerja CRC-32 dengan pemakaian sembarang *polynomial* dan kemudian menghitung total waktu pemrosesannya, implementasi yang mampu melakukan perhitungan terhadap kinerja CRC-32 versi IEEE Std., Castagnoli, dan Koopman, serta implementasi yang mampu melakukan perhitungan waktu rata-rata secara otomatis terhadap proses kerja CRC-32 versi IEEE Std., Castagnoli *et al.*, dan Koopman.

Untuk implementasi yang pertama dan ketiga berhasil dibuat sesuai dengan rancangan yang ada dan pengujian kesesuaian antara *input* - *output* pun telah dilakukan.

Hasil yang didapat dari perhitungan total waktu pada implementasi ketiga dapat digambarkan melalui grafik di bawah ini:



Gambar 7 Grafik perbandingan waktu pemrosesan rata-rata CRC-32.

Seperti yang dapat diperkirakan waktu rata-rata perhitungan CRC akan terus meningkat jika data yang diujicobakan semakin panjang. Namun hasil perhitungan waktu rata-rata ini sedikit di luar perkiraan karena seperti yang terlihat pada gambar 7, waktu rata-rata perhitungan CRC dari ketiga *polynomial* yang diujicobakan menunjukkan hasil yang hampir sama (untuk lebih jelasnya lihat Lampiran 7). Hal ini tidak sesuai dengan teori yang ada, karena jika dibaca dari teorinya dikatakan bahwa lamanya waktu pemrosesan yang dibutuhkan dalam perhitungan CRC bergantung dari panjangnya *shift register* yang harus dilalui. Dan seperti yang telah dijelaskan pada Ide Dasar CRC (implementasi *shift register*), panjang *shift register* bergantung pada panjang *polynomial* yang digunakan. Oleh karena itu, seharusnya *polynomial* IEEE Std yang memiliki representasi biner $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$ seharusnya memiliki waktu rata-rata pemrosesan yang lebih cepat dibandingkan dengan kedua *polynomial* lainnya yaitu *polynomial* Castagnoli *et al.* $x^{32} + x^{28} + x^{27} + x^{26} + x^{25} + x^{23} + x^{22} + x^{20} + x^{19} + x^{18} + x^{14} + x^{13} + x^{11} + x^{10} + x^9 + x^8 + x^6 + 1$ dan *polynomial* Koopman $x^{32} + x^{30} + x^{29} + x^{28} + x^{26} + x^{20} + x^{19} + x^{17} + x^{16} + x^{15} + x^{11} + x^{10} + x^7 + x^6 + x^4 + x^2 + x + 1$ yang memiliki *bit polynomial* yang lebih panjang. Tidak tampaknya perbedaan hasil ini mungkin disebabkan karena sedikitnya perbedaan jumlah *bit* antara *polynomial* satu dengan yang lainnya.

Setelah berhasil dengan pembuatan implementasi untuk bagian satu dan tiga, masalah datang pada pembuatan implementasi bagian kedua. Idealnya, implementasi harus dibuat berdasarkan pada perancangan yang telah dijelaskan, namun ternyata dikarenakan adanya keterbatasan teknis dan waktu pengerjaan maka hal ini tidak dapat dilakukan.

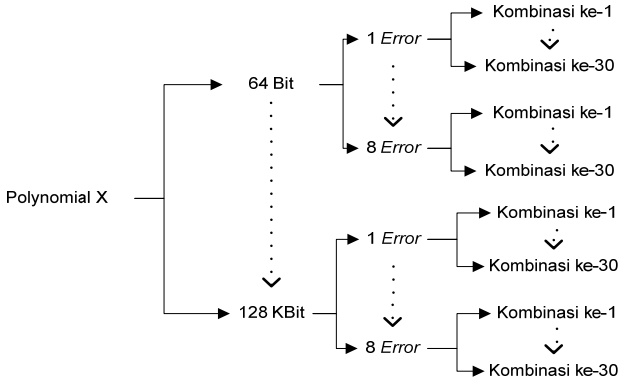
Keterbatasan teknis yang dimaksudkan adalah kemampuan komputer yang digunakan untuk melakukan perhitungan. Mengingat bahwa *range* panjang data yang akan dicobakan berkisar dari 64 *bit* sampai dengan 128 *Kbit* maka berarti komputasi yang terjadi sangatlah besar (pada *worst case* akan terjadi komputasi sebanyak $2^{(64+32)}$ sampai dengan $2^{(128000+32)}$ atau dengan kata lain dari 79228162514264337593543950336 sampai dengan 2,96759650437673496508858331229 77e+38541). Perhitungan sebanyak ini tidak mungkin untuk dilakukan pada komputer-komputer biasa. Selain itu sekalipun dijalankan pada komputer yang super canggih, proses perhitungannya pun akan memakan waktu yang lama.

Oleh karena itu, implementasi kemudian dibatasi dengan cara menetapkan jumlah k_{max} = 8 untuk tiap-tiap panjang data. Angka delapan diambil mengingat bahwa pada penelitian-penelitian sebelumnya belum diketemukan suatu *polynomial* yang mampu mencapai HD lebih dari delapan. Namun seperti yang dapat dilihat pada Lampiran 8, walaupun hal ini telah dilakukan namun tetap

saja ruang kombinasi yang dibutuhkan masih sangat besar.

Dalam perkembangannya kemudian semakin disadari bahwa penghitungan semua ruang kombinasi untuk setiap *k* pun tidaklah memungkinkan untuk dilakukan, sehingga kemudian diputuskan untuk menggunakan pengambilan contoh. Hal yang perlu diperhatikan kali ini adalah pemilihan algoritme penarikan contoh yang digunakan, hal ini dimaksudkan agar kesimpulan yang dihasilkan nantinya dapat merepresentasikan keadaan yang sebenarnya.

Metode penarikan contoh pertama yang diterapkan adalah metode penarikan contoh secara acak sebanyak jumlah minimum yang disarankan, yaitu 30 (Walpole 1982). Jika digambarkan perancangan percobaannya akan tampak seperti gambar di bawah ini:



Gambar 8 Perancangan percobaan 1.

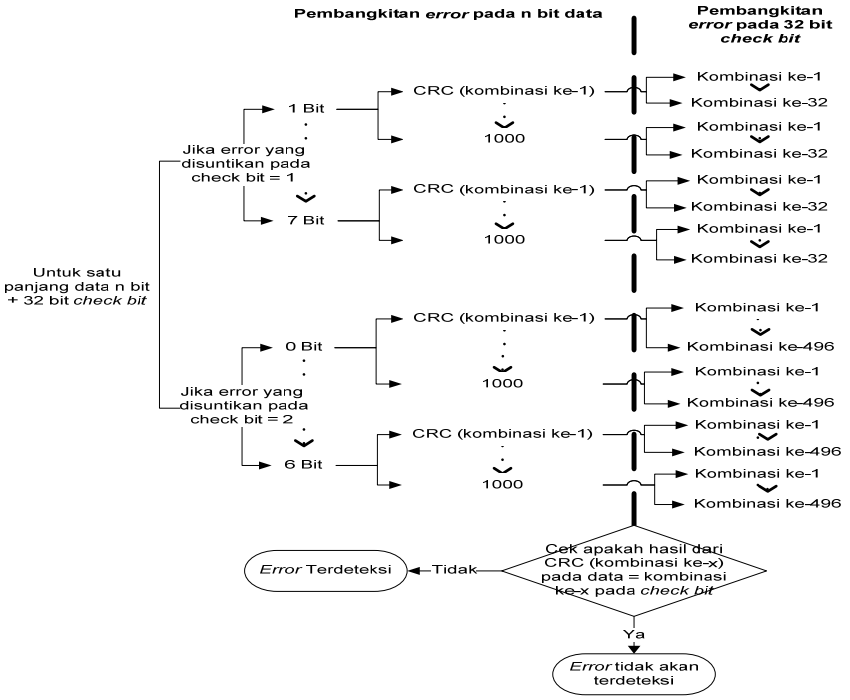
Hasil dari penerapan metode ini menunjukkan bahwa untuk semua *polynomial* di semua panjang data menghasilkan HD yang lebih besar dari 8. Dengan kata lain masing-masing *polynomial* pada percobaan ini memiliki $W_1 = 0$, $W_2 = 0$, …, $W_8 = 0$ (untuk 30 kombinasi).

Hasil ini jauh berbeda dengan hasil penelitian-penelitian sebelumnya (*lihat* Lampiran 4). Oleh karena itu dapat dikatakan bahwa percobaan kali ini gagal karena tidak mampu menghasilkan suatu kesimpulan yang dapat merepresentasikan hasil yang sebenarnya.

Untuk memperbaiki hasil, ruang pengambilan contoh pun kemudian diperbesar tetapi ternyata hasilnya tetap saja tidak memuaskan. Sebenarnya hal ini sudah dapat dipredikisi mengingat sangat kecilnya peluang kemungkinan terjadinya kegagalan deteksi *error*, sebagai contoh perhatikan pada bagian Perhitungan Kinerja CRC telah dinyatakan bahwa *polynomial* CRC-32 versi IEEE Std.memiliki $W_4 = 223059$ sedangkan ruang kombinasinya = $906.\ 10^{12}$, artinya peluang terjadinya kegagalan deteksi *error* = $223059 / 906.\ 10^{12} = 2{,}4.10^{-10}$, peluang ini sangat kecil sehingga jika dipaksakan terus mencari dari ruang contoh yang kecil pula maka pencarian akan bagaikan mencari jarum dalam tumpukan jerami. Di lain pihak, usaha untuk terus memperbesar pengambilan ruang contoh pun tidak mungkin untuk dilakukan karena dibatasi oleh kemampuan komputer dan waktu.

Oleh karena itu, algoritme penarikan contoh kemudian diubah, tidak lagi menggunakan metode penarikan contoh secara acak seperti yang telah dijelaskan, melainkan dengan menampatkan error pada posisi "rawan" terjadinya gagal deteksi *error*.

Walaupun sebenarnya kombinasi dari *error* yang dapat mengalami kegagalan deteksi tidak memiliki suatu pola tertentu, namun berdasarkan penelitian Koopman 2002 disebutkan bahwa peluang terbesar terjadinya kegagalan deteksi *error* adalah ketika satu atau dua *bit error* sengaja diletakkan pada *check bit.* Oleh karena itu dalam metode kali ini, dari delapan *error* yang akan diujicobakan satu atau dua diantaranya diletakkan pada posisi *check bit*. Sehingga jika digambarkan prosesnya akan terlihat seperti gambar berikut ini:



Gambar 9 Perancangan Percobaan 2.

Pada gambar 9 terlihat bahwa dari *k bit error* yang akan disuntikkan ke *codeword*, maka satu dan dua *error* terlebih dahulu disuntikkan ke *check bit*. Sehingga total banyaknya *error* untuk data adalah sisa dari *k* dikurangi dengan satu atau dua. Kemudian masing-masing percobaan diulang sesuai dengan banyaknya kombinasi dari tiap panjang *error*. Namun untuk hasil kombinasi yang lebih dari 1000, ditetapkan hanya akan diulang sebanyak 1000 kali, hal ini disesuaikan dengan kemampuan komputer yang digunakan. Setelah terbentuk data yang telah disusupi *error* maka masing-masing kombinasi dicari nilai CRC nya, hasilnya akan dibandingkan dengan seluruh kemungkinan ruang kombinasi yang mungkin dari *check bit* yang telah disusupi satu atau dua *error*, untuk jumlah *error* pada *check bit* sama dengan satu, maka ada 32 (dari C_1^{32}) kombinasi yang perlu diperiksa, sedangkan jika jumlah *error* pada *check bit* sama dengan dua berarti akan ada 496 (dari C_2^{32}) kombinasi yang perlu diperiksa. Berdasarkan teori yang telah dijelaskan jika hasil CRC *error* pada data sama dengan *error* pada *check bit*, maka akan terjadi *error* yang terdeteksi. Namun jika sebaliknya, maka *error* tersebut pasti akan terdeteksi.

Perancangan percobaan yang ada pada gambar 9 baru untuk satu jenis panjang data, berarti jika terdapat dua belas panjang data untuk satu *polynomial*, maka perhitungan seperti itu harus dilakukan sebanyak 12 kali. Belum lagi percobaan untuk dua *polynomial* yang lainnya.

Namun ternyata walaupun perancangan percobaan telah disusun sedemikian rupa, hasil yang didapat pada percobaan kali ini tetap sama dengan percobaan sebelumnya dimana nilai HD untuk *polynomial* pada semua panjang data adalah di atas delapan. Hal ini menunjukkan bahwa tetap saja hasil yang didapat tidak merepresentasikan hasil yang sesungguhnya. Proses komputasi untuk percobaan ini pun memakan waktu yang tidak sedikit untuk 1000 ulangan pada panjang data 128*KBit* kira-kira diperlukan waktu selama 2 jam lebih. Sehingga dengan demikian jika hendak meningkatkan keakuratan hasil dengan menambah jumlah pengambilan ruang contoh kombinasinya (lebih dari 1000 kombinasi) rasanya hal ini tidak mungkin untuk dilakukan karena walaupun ruang contoh kombinasinya diperbesar tetapi tampaknya mengingat waktu komputasi yang lama maka jumlah perbesaran yang mampu untuk dilakukan pun sangat terbatas, oleh karena itu hasil yang akan didapat nantinya pun tidak akan banyak berbeda.

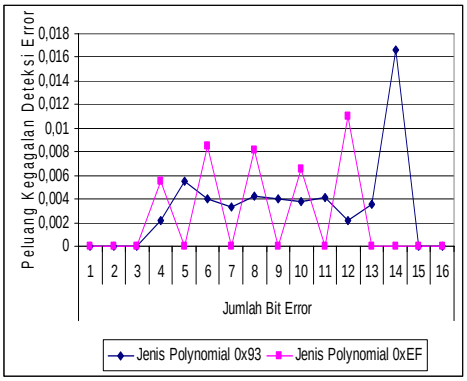
Oleh karena itu, berdasarkan hasil percobaan-percobaan yang telah dilakukan walaupun belum mencoba seluruh kemungkinan metode penarikan contoh yang ada, namun dapat disimpulkan bahwa pengukuran kinerja CRC-32 dengan menggunakan teknik pengambilan contoh kecil sekali kemungkinannya untuk dapat merepresentasikan hasil yang sebenarnya. Hal ini disebabkan karena kecilnya peluang kemungkinan terjadinya kegagalan deteksi *error* serta tidak adanya suatu pola keteraturan yang dimiliki oleh *error-error* yang gagal terdeteksi sehingga sangat susah untuk dianalisis jika hanya dengan menggunakan teknik menduga-duga secara acak.

Dengan demikian pengukuran kinerja CRC-32 gagal diimplentasikan, namun sebagai gantinya agar tetap memperoleh gambaran yang jelas mengenai proses ini, maka pengukuran kinerja CRC akan diimplementasikan pada lingkup yang lebih kecil yaitu CRC-8 dengan *polynomial* terpilih yaitu 0x93 dan 0xEF (diambil dari salah satu *polynomial* percobaan yang dilakukan oleh Chakravarty) dengan panjang data dibatasi hanya 8 *bit* dan 16 *bit* saja.

Dalam percobaan CRC-8 kali ini, implementasi dibuat sesuai dengan konsep perancangan awal, namun agar lebih sempurna iterasi *k* tidak hanya akan berhenti pada W tak nol pertama, atau dengan kata lain nilai k_{max} di tetapkan = panjang data + *check bit*. Untuk mempercepat proses komputasi pada kasus pencarian seluruh nilai *k*, maka algoritmenya sedikit mengalami perubahan seperti yang dapat dilihat pada Lampiran 9.

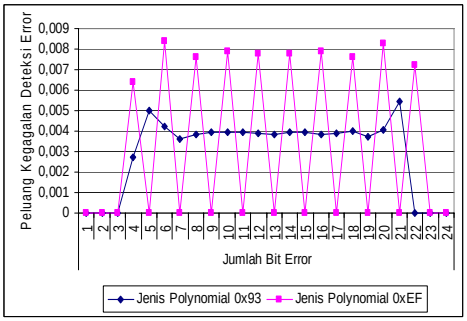
Hasil dari percobaan pengukuran kinerja CRC-8 ini berupa tabel *weight distribution* untuk setiap panjang *error*, tabel ini dapat dilihat pada Lampiran 10 dan Lampiran 11. Dari tabel *weight distribution* inilah kemudian dapat dicari peluang terjadinya kegagalan deteksi *error* untuk masing-masing jumlah bit *error* pada suatu panjang data, caranya yaitu dengan melakukan operasi pembagian antara setiap *weight* dengan jumlah keseluruhan kombinasi *error* yang mungkin. Informasi ini akan berguna untuk melihat titik-titik kuat maupun lemah dari suatu *polynomial* pada suatu panjang data.

Grafik peluang terjadinya kegagalan deteksi *error* pada CRC-8 untuk panjang data sama dengan 8 bit dapat dilihat pada gambar di bawah ini:



Gambar 10 Hasil percobaan CRC-8 untuk panjang data = 8 *bit*.

Sedangkan peluang terjadinya kegagalan deteksi *error* pada CRC-8 untuk panjang data sama dengan 16 bit dapat dilihat pada gambar di bawah ini:



Gambar 11 Hasil percobaan CRC-8 untuk panjang data = 16 *bit*.

Pada Gambar 10 dan 11, sumbu *x* menyatakan banyaknya jumlah *bit error* yang dicobakan sedangkan sumbu *y* menyatakan besarnya peluang kemungkinan terjadinya kegagalan deteksi *error* untuk suatu jumlah *bit error*.

Dari gambar dapat dilihat bahwa nilai HD kedua *polynomial* untuk panjang data sama dengan 8 dan 16 *bit* adalah 4. Angka ini didapat dari peluang tak nol pertama yang terjadi pada masing-masing *polynomial* di masing-masing panjang data. Sebenarnya pada percobaan kali ini pun telah dilakukan untuk panjang data 32 *bit*, namun setelah hampir delapan jam proses komputasi, jumlah kombinasi yang telah dicari CRC nya baru sekitar 18 juta padahal ruang seluruh kombinasi yang harus dicari yaitu 2^{32} = 4.294.967.296. Oleh karena itu kemudian komputasi pun dihentikan.

Jika ada dua atau lebih *polynomial* yang memiliki nilai HD yang sama, seperti hal nya

dalam contoh kasus ini, maka analisis yang perlu dilakukan untuk mengukur kinerja CRC selanjutnya adalah melihat besarnya peluang terjadinya kegagalan deteksi *error* (yang biasanya akan konstan pada tiap panjang data). Pada percobaan kali ini, *polynomial* 0x93 cenderung memiliki peluang terjadinya kegagalan deteksi *error* yang lebih kecil dibandingkan dengan *polynomial* 0xEF. Oleh karena itu, dapat disimpulkan untuk kasus ini *polynomial* yang memiliki kinerja terbaik adalah *polynomial* 0x93.

Selain menganalisis kinerja CRC, melalui hasil percobaan analisis juga dapat dikembangkan pada analisis penerapan teori-teori yang telah dijelaskan pada bagian Pemilihan *Polynomial*.

Teori pertama mengatakan bahwa semua *polynomial* yang memiliki minimal dua atau lebih *bit*, dapat dipastikan akan mendeteksi semua satu *bit error.* Pada percobaan ini, baik *polynomial* 0x93 maupun 0xEF tersusun lebih dari dua *bit* dan hasil percobaan menunjukkan bahwa kedua *polynomial* tersebut memang benar mampu mendeteksi semua satu *bit error*.

Teori kedua mengatakan bahwa *polynomial* yang memenuhi teori pertama dan tidak habis membagi $x^k + 1$, dimana $k = 1, ...,$ (panjang data + *check bit*), maka *polynomial* tersebut akan mendeteksi semua dua *bit error*. Pada percobaan ini pun kedua *polynomial* memenuhi persyaratan tersebut dan hasil percobaan menunjukkan bahwa kedua *polynomial* tersebut memang benar mampu mendeteksi semua dua *bit error*.

Teori ketiga mengatakan bahwa *polynomial* yang memiliki $(x + 1)$ sebagai salah satu faktornya akan dapat mendeteksi semua *bit error* yang berjumlah ganjil. Pada percobaan kali ini *polynomial* 0x93 adalah *polynomial* yang tidak dapat habis dibagi oleh $(x + 1)$ sedangkan *polynomial* 0xEF dapat dibagi habis oleh $(x + 1)$, sehingga dari hasil percobaan dapat dilihat bahwa *polynomial* 0xEF memiliki peluang terjadinya kegagalan deteksi sama dengan 0 untuk semua jumlah *error* ganjil namun tidak demikian dengan *polynomial* 0x93. Tetapi perhatikanlah bahwa di lain pihak peluang terjadi kegagalan deteksi pada *error* yang berjumlah genap dalam *polynomial* 0xEF akan menjadi dua kali lipat dibandingkan *polynomial* 0x93, hal ini sekaligus membuktikan salah satu hasil penelitian Koopman 2002 yang mengatakan bahwa peluang terjadinya kegagalan deteksi pada *error* berjumlah genap di *polynomial* yang dapat habis dibagi oleh $(x + 1)$ akan menjadi dua kali lipat jika dibandingkan dengan *polynomial* lainnya. Hasil dari penelitian ini menjawab mengapa ada sebagian CRC yang telah dijadikan standar namun tidak memenuhi syarat seperti yang telah disarankan dalam teorema-teorema yang telah dijelaskan pada bagian Pemilihan *Polynomial* (contoh: CRC-32 versi IEEE std).

Sedangkan untuk teori keempat yang mengatakan bahwa semua *polynomial* dengan panjang *r* akan mampu mendeteksi semua *burst error* yang panjangnya tidak melebihi *r* tidak dapat ditunjukkan pada percobaan kali ini karena pada percobaan kali ini kombinasi *error* tidak dianalisis satu persatu sehingga tidak akan pernah diketahui apakah *error* tersebut membentuk *burst error* atau tidak.

Demikian analisis yang dapat dilakukan dari teori-teori yang ada pada Pemilihan *Polynomial*.

Untuk selanjutnya, jika perhitungan kinerja dilakukan untuk setiap kemungkinan kombinasi *polynomial,* maka akan dapat diketahui CRC mana yang dapat menghasilkan kinerja terbaik (walaupun baru untuk satu panjang data saja). Sebagai contoh, pada penelitian kali ini dilakukan suatu percobaan pemilihan *polynomial* terbaik untuk CRC-8 pada panjang data 8 *bit*, hasil yang didapat yaitu *polynomial* 0xEB berhasil keluar sebagai *polynomial* terbaik karena merupakan *polynomial* satu-satunya yang memiliki nilai HD = 5 (untuk hasil yang lebih jelas dapat dilihat pada Lampiran 12 dan Lampiran 13). Namun sangat disayangkan pencarian *polynomial* terbaik tidak dapat dilanjutkan untuk CRC-8 pada panjang data = 16 *bit,* lagi-lagi hal ini dikarenakan keterbatasan waktu yang diperlukan untuk melakukan komputasi karena berdasarkan percobaan yang dilakukan pada dua *polynomial* sebelumnya (0x93 dan 0xEF), perhitungan untuk satu *polynomial* CRC-8 pada panjang data = 16 *bit* akan memakan waktu sekitar 45 detik, sedangkan banyaknya kombinasi *polynomial* yang harus diperiksa mencapai angka di atas 16 ribu, sehingga jika tetap dilakukan maka proses komputasi kira-kira akan memakan waktu sekitar 11 hari.

Hal terpenting yang perlu diingat ketika hendak memilih *polynomial* terbaik adalah kenali terlebih dahulu protokol dimana CRC ini akan diimplementasikan sehingga nantinya dapat ditentukan *features* apa yang harus

diperkuat untuk implementasi tersebut. Hal ini penting untuk dapat menyusun urutan prioritas dari *feature- feature* tersebut, karena tidak akan mungkin untuk bisa mendapatkan semua kualitas yang terbaik dalam satu *polynomial* saja, akan selalu ada kompensasi yang menyertainya sehingga segala sesuatunya harus dipertimbangkan berulang kali.

Kesimpulan dan Saran

Kesimpulan

Beberapa kesimpulan yang berhasil dihimpun pada penelitian kali ini antara lain:

1. Algoritme CRC dengan pemilihan *polynomial* yang tepat telah terbukti mampu memberikan suatu perlindungan yang baik terhadap integritas data.
2. Walaupun pengukuran kinerja CRC dengan memeriksa semua kemungkinan kombinasi *error* susah untuk dilakukan pada data yang besar namun hal ini tetap penting untuk dilakukan agar nilai HD yang didapat dari *polynomial* tersebut dijamin ketapatannya. Oleh karena itu, walaupun pada penelitian ini gagal mengimplementasikan pengukuran kinerja CRC pada CRC-32 namun agar tetap memperoleh gambaran yang jelas mengenai bagaimana mekanisme pengukuran kinerja CRC tersebut sebenarnya, maka implementasi kemudian disesuaikan dengan lingkup yang lebih kecil yaitu pengukuran kinerja CRC untuk CRC-8.
3. Definisi *polynomial* terbaik tidak bisa digeneralisasi untuk semua permasalahan, untuk mendapatkan hasil yang maksimal maka masing-masing permasalahan harus diteliti secara detail terlebih dahulu.

Saran

Saran yang dapat dilakukan untuk penelitian-penelitian selanjutnya adalah antara lain:

1. Melakukan pengukuran kinerja CRC-32 dengan menggunakan seluruh ruang kombinasi *error* yang mungkin, tanpa mengabaikan besarnya total waktu pemrosesan dan sumber daya yang diperlukan, seperti yang telah dilakukan oleh Koopman dalam penelitiannya tahun 2002. Hal ini sangat penting dilakukan karena dengan memeriksa seluruh ruang kombinasi akan didapat *weight* yang akurat sehingga nilai HD dapat ditetapkan secara pasti.
2. Pengembangan lingkup analisa, yaitu tidak hanya sebatas pada analisa teori seperti yang dilakukan pada penelitian ini namun juga bisa menjangkau analisa implementasi, seperti analisa penggunaan *memory* dan uji perbandingan kecepatan antara suatu algoritme implementasi dengan algoritme implementasi yang lainnya, karena pada kenyataannya ada begitu banyak algoritme implementasi yang telah berkembang.
3. Mencoba mengembangkan penelitian untuk mengukur kinerja CRC dengan menggunakan *heuristic search.*
4. Mengembangkan penelitian dengan memasukkan unsur *bit error rate* (BER) pada suatu media transmisi tertentu.

Daftar Pustaka

Bose R. 2003. Information Theory, Coding and Cryptography. Singapura: McGraw Hill.

Chakravarty T. 2001. M.S. Project Report: Performance of Cyclic Redundancy Codes for Embedded Networks. http://www.ece.cmu.edu/~koopman/thesis/chakravarty.pdf. [17 Maret 2005].

Koopman P. 2002. 32-Bit Cyclic Redundancy Codes for Internet Application. http://www.ece.cmu.edu/~koopman/networks/dsn02/dsn02_koopman.pdf. [27 November 2004].

Menezes A, Van Oorschot P, & Vanstone S. 1997. Handbook of Applied Cryptography. NewYork: CRC press.

Stallings W. 2004. Data and Computer Communications. Upper Saddle River: Pearson Education.

Tanenbaum, AS. 2003. *Computer Networks 4th ed.* San Fransisco: Pearson Education International.

Walpole RE. 1982. Pengantar S*tatistika Edisi ke-3*. Bambang Sumantri, penterjemah. Jakarta: PT Gramedia Pustaka Utama. Terjemahan dari: *Introduction to statistic 3rd edition.*

Williams RN. 1993. A Painless Guide to CRC Error Detection Algorithms. http://www.ross.net/crc/crcpaper.html. [10 Desember 2004].

LAMPIRAN

Lampiran 1. Pembuktian matematika algoritme CRC (Stalling, 2004)

Kita definisikan:

T = *n-bit* frame yang akan ditransmisikan (Codeword)

D = *k-bit* blok data, atau pesan, merupakan *k bit* pertama dari T

F = (*n-k*) *bit* sisa pembagian, merupakan (*n - k*) *bit* terakhir dari T

P = *Polynomial* pembagi, terdiri dari $n - k + 1$ *bit*, polynomial ini sudah ditentukan dan diketahui bersama.

Kita hendak melakukan pembagian T dengan P sehingga sedemikian rupa pembagian tersebut menghasilkan sisa bagi sama dengan 0.

Diketahui:

$$T = 2^{n-k} D + F \qquad \ldots.(1)$$

Proses perkalian D dengan 2^{n-k} akan memperbesar nilai *D* dengan cara menambahkan angka 0 sebanyak *n - k bit*. Sedangkan proses penambahan *F* artinya melakukan penggabungan antara *D* dengan *F*, yang kemudian menghasilkan *T*. Sekarang yang kita inginkan adalah membuat *T* dapat tepat habis dibagi oleh *P*. Misalkan kita membagi $2^{n-k} D$ dengan *P*, maka:

$$\frac{2^{n-k} D}{P} = Q + \frac{R}{P} \qquad \ldots.(2)$$

Dapat dilihat bahwa terdapat hasil pembagian atau *quation* (*Q*) dan sisa pembagian atau *reminder* (*R*). Karena ini merupakan pembagian modulo 2 maka *reminder* yang dihasilkan selalu 1 *bit* lebih pendek dari pembaginya. *Reminder* inilah yang akan berfungsi sebagai *check bit* (F). Maka:

$$T = 2^{n-k} D + R$$

Apakah *R* ini akan memenuhi persyaratan kita dimana *T/P* akan memiliki *R*=0. Untuk membuktikannya perhatikan persamaan berikut:

$$\frac{T}{P} = \frac{2^{n-k} D + R}{P} = \frac{2^{n-k} D}{P} + \frac{R}{P}$$

Dari persamaan (1) didapat:

$$\frac{T}{P} = Q + \frac{R}{P} + \frac{R}{P}$$

Dan karena semua bilangan biner dalam modulo 2 yang ditambahkan dengan dirinya sendiri akan menghasilkan 0, maka:

$$\frac{T}{P} = Q + \frac{R + R}{P} = Q$$

Dapat dibuktikan bahwa tidak ada *reminder*, dan oleh karena itu terbukti bahwa *T* dapat tepat habis dibagi oleh *P*.

Lampiran 2. Contoh kasus perhitungan algoritme CRC (Stalling, 2004)

Misal hendak dikirim sebuah pesan dengan panjang 10 *bit*, D = 1010001101. Dan disepakati *polynomial* yang digunakan adalah P = 110101 (6 *bit*)

Maka Check *Bit* atau sisa hasil bagi (R) dapat dicari sebagai berikut:

1. Cari banyaknya *bit* R
   Jika P = R + 1, berarti dapat dihitung R = P - 1 ; R = 6 - 1 = 5
2. Kalikan 2^R dengan D , berarti 2^5 x 1010001101 = 101000110100000
3. Hasil dari langkah 2) kemudian dibagi dengan P

```
                      1 1 0 1 0 1 0 1 1 0 ← Q
P → 1 1 0 1 0 1 ) 1 0 1 0 0 0 1 1 0 1 0 0 0 0 0 ← 2^R x D
                  1 1 0 1 0 1
                    1 1 1 0 1 1
                    1 1 0 1 0 1
                      1 1 1 0 1 0
                      1 1 0 1 0 1
                        1 1 1 1 1 0
                        1 1 0 1 0 1
                          1 0 1 1 0 0
                          1 1 0 1 0 1
                            1 1 0 0 1 0
                            1 1 0 1 0 1
                              0 1 1 1 0 ← R
```

4. Setelah didapat nilai dari R, maka tambahkan nilai R dengan 2^R x D,

```
  1 0 1 0 0 0 1 1 0 1 0 0 0 0 0
+ (0 1 1 1 0)
```

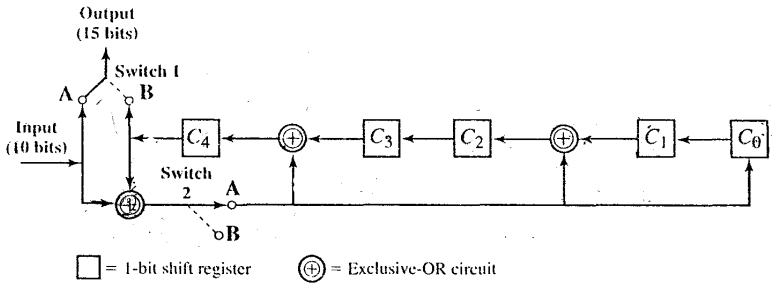
1 0 1 0 0 0 1 1 0 1 0 1 1 1 0, Data ini lah yang akan kita transmisikan **(T)**

Sedangkan proses pemeriksaan yang dilakukan oleh penerima adalah sebagai berikut:

```
                      1 1 0 1 0 1 0 1 1 0 ← Q
P → 1 1 0 1 0 1 ) 1 0 1 0 0 0 1 1 0 1 1 1 1 0 ← 2^R x D
                  1 1 0 1 0 1
                    1 1 1 0 1 1
                    1 1 0 1 0 1
                      1 1 1 0 1 0
                      1 1 0 1 0 1
                        1 1 1 1 0 1
                        1 1 0 1 0 1
                          1 0 1 1 1 1
                          1 1 0 1 0 1
                            1 1 0 1 0 1
                            1 1 0 1 0 1
                                  0 ← R, artinya tidak terjadi error
```

Lampiran 3. Contoh kasus perhitungan CRC menggunakan S*hift Register* (Stalling, 2004)

Misal untuk kasus yang sama dengan Lampiran 2, maka akan diimplementasikan dalam shift register seperti di bawah ini:



(a) Shift-register implementation

Sedangkan langkah per langkah pengerjaannya terekam dalam tabel berikut ini:

	C_4	C_3	C_2	C_1	C_0	$C_4 \oplus C_3 \oplus I$	$C_3 \oplus C_1 \oplus I$	$C_4 \oplus I$	I = input	
Initial	0	0	0	0	0	1	1	1	1	
Step 1	1	0	1	0	1	1	1	1	0	
Step 2	1	1	1	1	1	1	1	0	1	
Step 3	1	1	1	1	0	0	0	1	0	
Step 4	0	1	0	0	1	1	0	0	0	Message to be sent
Step 5	1	0	0	1	0	1	0	1	0	
Step 6	1	0	0	0	1	0	0	0	1	
Step 7	0	0	0	1	0	1	0	1	1	
Step 8	1	0	0	0	1	1	1	1	0	
Step 9	1	0	1	1	1	0	1	0	1	
Step 10	0	1	1	1	0					

(b) Example with input of 1010001101

Figure 6.5 Circuit with Shift Registers for Dividing by the Polynomial $X^5 + X^4 + X^2 + 1$

Lampiran 4. Hasil percobaan Koopman 2002

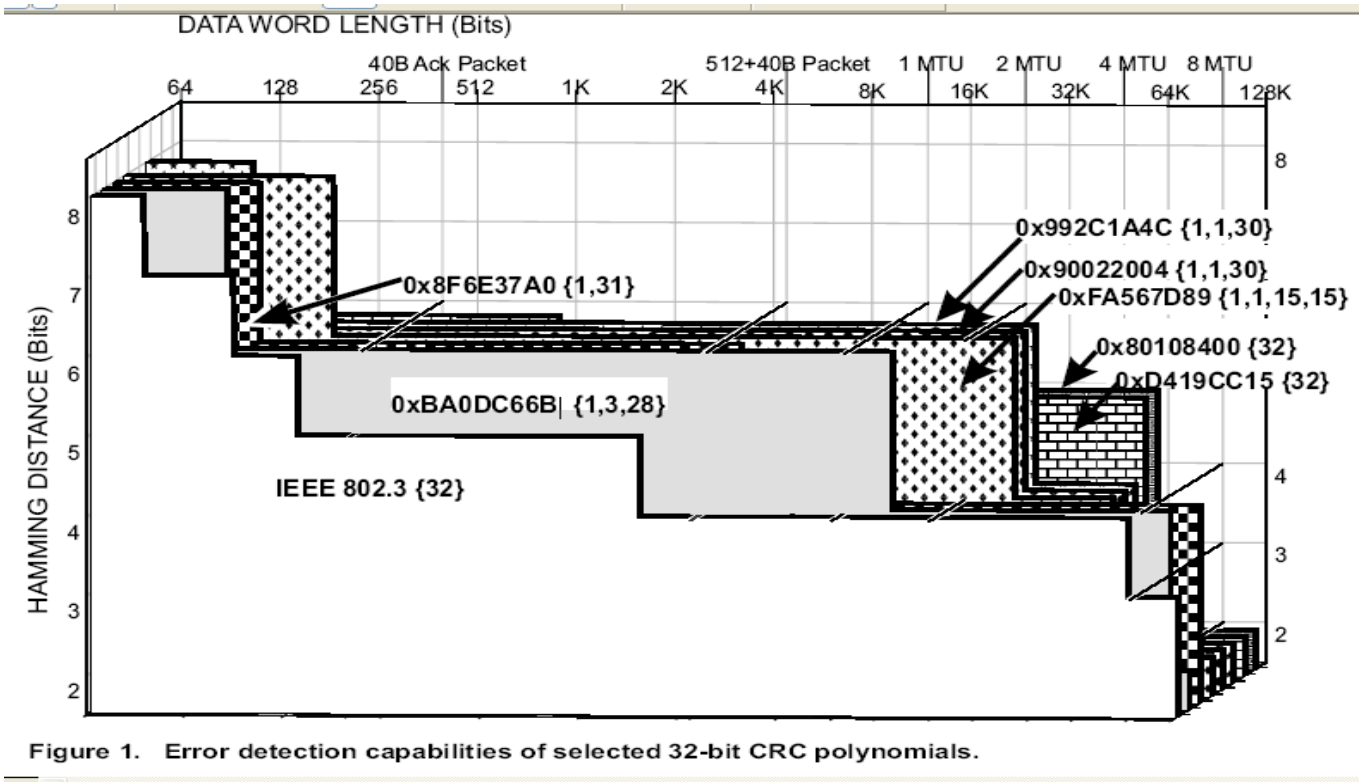


Figure 1. Error detection capabilities of selected 32-bit CRC polynomials.

Lampiran 5. Algoritme CRC

```
Function CRC (pesan, polynomial)
    Aktifkan perhitungan waktu pemrosesan CRC
    Inisialisasi shift register dengan 0
    For i = 1 to banyaknya karakter pada pesan
        CRCUpdate (karakter ke-i, polynomial)
    Next i
    Ubah nilai shift register terakhir ke dalam bentuk bilangan heksadesimal
    Hentikan perhitungan waktu dan cari antaranya
End Function

Sub CRCUpdate(shift register, karakter ke-i, polynomial)
 For Bit = 1 To 8
     Ambil MSB dari shift register
     Ambil MSB dari karakter ke-i
     Cek (MSB shift register XOR MSB karakter)
        Jika hasilnya = 0, maka
           shift register = leftShift (shift register)
            Jika hasilnya = 1, maka
           shift register = (leftShift (shift register Xor polynomial)) Or (MSB karakter ke-i)
 Next Bits
End Sub

Function leftShift (shift register)
  "Nol" kan nilai MSB dari shift register  (Left Shift = (shift register) And (&H7FFFFFFF))
  Untuk menghindari overflow maka periksa:
      If left Shift > &H3FFFFFFF Then
         Left Shift = (left Shift) Or (&H80000000)
      End If
  Gesar shift register ke kiri sebanyak satu bit (left Shift = left Shift * 2)
End Function
```

Lampiran 6. Algoritme pembangkitan *error* acak

```
Function bangkit error (jumlahError, pesan)
  Mencari posisi error acak
    For i = 1 To jumlahError
    randomPos:
        Cari posisi karakter error
        Mencari posisi bit error dari karakter yang telah terpilih
        Mendapatkan posisi untuk error acak (posisi = (posisi karakter * 8) + posisi bit)
        Memeriksa apakah posisi tersebut sudah pernah dipakai untuk error
          For j = 1 To i
            If posisi = error (j) Then GoTo randomPos
          Next j
        Simpan posisi error agar tidak terpakai pada ulangan berikutnya (error (i) = posisi)
        Ubah bit data dengan cara meng-XOR kan bit pada posisi tersebut dengan 2^posisi
        Simpan hasil perubahan pada pesan
    Next i
End Function
```

Lampiran 7. Tabel waktu rata-rata pemrosesan CRC-32

		Panjang Data (*Bit*s)											
		64	128	256	512	1K	2K	4K	8K	16K	32K	64K	128K
CRC-32 Polynomial	IEEE Std.	2	0	1	3	4	9	18	36	73	152	331	791
	Koopman	0	1	1	3	4	9	19	34	72	152	333	778
	Castagnoli	0	0	2	2	4	9	18	35	72	151	334	782

Catatan: Hasil dalam Milidetik

Lampiran 8. Hasil kombinasi

96	1	96
	2	4560
	3	142880
	4	3321960
	5	61124064
	6	927048304
	7	11919192480
	8	1,32601E+11
Jumlah		**1,45512E+11**

544	1	544
	2	147696
	3	26683744
	4	3608976376
	5	3,89769E+11
	6	3,50143E+13
	7	2,69E+15
	8	1,81E+17
Jumlah		**1,83366E+17**

4128	1	4128
	2	8518128
	3	11715265376
	4	1,20814E+13
	5	9,96E+15
	6	6,85E+18
	7	4,03E+21
	8	2,08E+24
Jumlah		**2,0811E+24**

32800	1	32800
	2	537903600
	3	5,88072E+12
	4	4,82E+16
	5	3,16E+20
	6	1,73E+24
	7	8,10E+27
	8	3,32E+31
Jumlah		**3,32051E+31**

160	1	160
	2	12720
	3	669920
	4	26294360
	5	820384032
	6	21193254160
	7	4,66252E+11
	8	8,91706E+12
Jumlah		**9,40535E+12**

1056	1	1056
	2	557040
	3	195706720
	4	51519794040
	5	1,08398E+13
	6	1,90E+15
	7	2,85E+17
	8	3,73E+19
Jumlah		**3,76331E+19**

8224	1	8224
	2	33812976
	3	92670096224
	4	1,9046E+14
	5	3,13E+17
	6	4,29E+20
	7	5,04E+23
	8	5,17E+26
Jumlah		**5,17712E+26**

65568	1	65568
	2	2149548528
	3	4,69791E+13
	4	7,70E+17
	5	1,01E+22
	6	1,10E+26
	7	1,03E+30
	8	8,47E+33
Jumlah		**8,46998E+33**

288	1	288
	2	41328
	3	3939936
	4	280720440
	5	15944920992
	6	7,52069E+11
	7	3,02976E+13
	8	1,06E+15
Jumlah		**1,09527E+15**

2080	1	2080
	2	2162160
	3	1497656160
	4	7,77658E+11
	5	3,22884E+14
	6	1,12E+17
	7	3,31E+19
	8	8,57E+21
Jumlah		**8,6062E+21**

16416	1	16416
	2	134734320
	3	7,37176E+11
	4	3,02E+15
	5	9,93E+18
	6	2,72E+22
	7	6,37E+25
	8	1,31E+29
Jumlah		**1,30644E+29**

131104	1	131104
	2	8594063856
	3	3,75566E+14
	4	1,23E+19
	5	3,23E+23
	6	7,05E+27
	7	1,32E+32
	8	2,16E+36
Jumlah		**2,16442E+36**

Keterangan: Kolom paling kiri dari setiap tabel adalah panjang data + *check bit*, kolom tengah adalah banyaknya bit *error*, & kolom paling kanan adalah jumlah kombinasi yang mungkin.

Lampiran 9. Algoritme pengukur kinerja CRC

Sub kinerjaCRC (panjang pesan, *polynomial*)

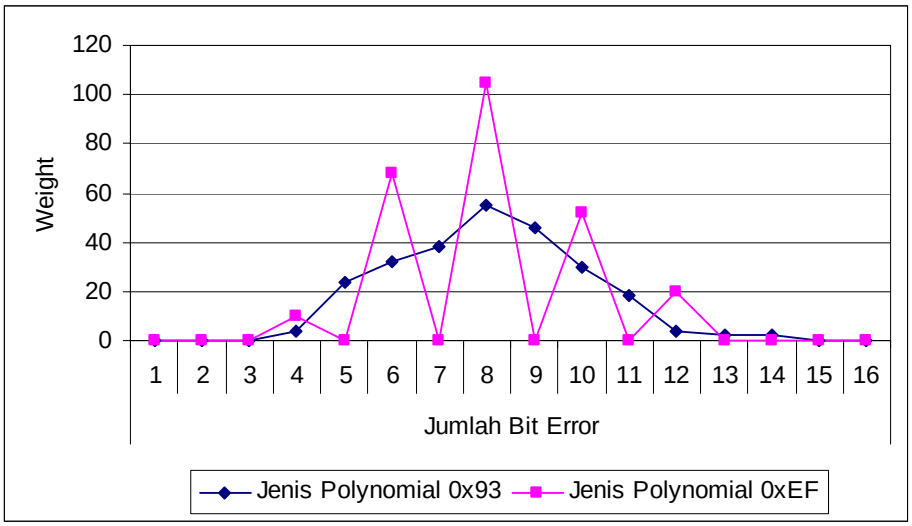
For i = 1 To ($2^{\text{panjang pesan}} - 1$)
Cek apakah i <= 255
Jika Ya, berarti karakter ASCII dari pesan dapat langsung dicari
Jika tidak, berarti pesan harus dipecah menjadi bagian-bagian kecil sepanjang 8 *bit* sehingga dapat dicari karakter ASCII nya
Cari *check bit* dari karakter-karakter pesan tersebut dengan memanggil fungsi CRC
Hitung jumlah *bit* "1" pada pesan dan *check bit,* jumlah ini menunjukkan banyaknya *error* yang terjadi
Untuk setiap jumlah *error*, simpan dalam suatu array tersendiri yang nilainya akan bertambah jika ada jumlah error yang sama dengannya
Next i

End Sub

Lampiran 10. *Weight distribution* untuk percobaan CRC-8 pada panjang data = 8 *bit*

		Jenis *Polynomial*	
		0x93	0xEF
Jumlah *Bit Error*	1	0	0
	2	0	0
	3	0	0
	4	4	10
	5	24	0
	6	32	68
	7	38	0
	8	55	105
	9	46	0
	10	30	52
	11	18	0
	12	4	20
	13	2	0
	14	2	0
	15	0	0
	16	0	0

Tabel di atas jika digambarkan dalam bentuk grafik maka akan tampak seperti gambar di bawah:

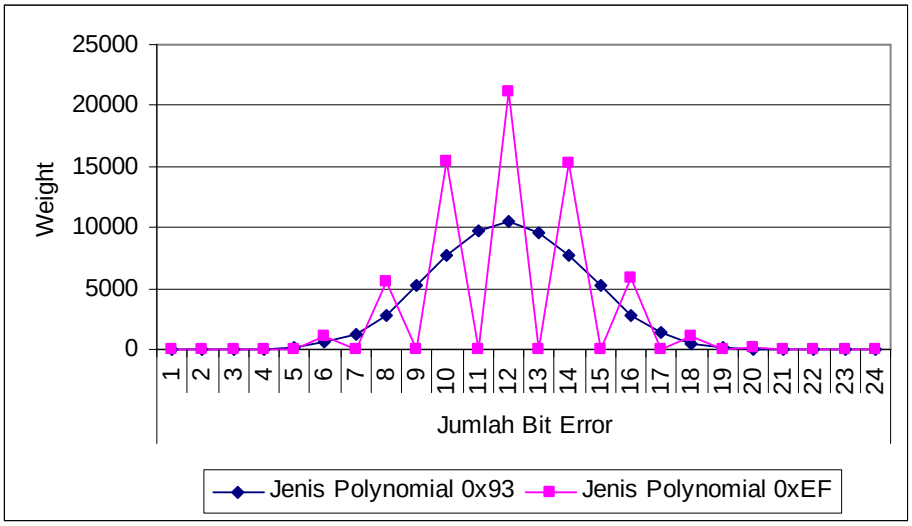


Lampiran 11. *Weight distribution* untuk percobaan CRC-8 pada panjang data = 16 *bit*

		Jenis *Polynomial*	
		0x93	0xEF
Jumlah *Bit* Error	1	0	0
	2	0	0
	3	0	0
	4	29	68
	5	213	0
	6	566	1130
	7	1250	0
	8	2824	5590
	9	5190	0
	10	7694	15472
	11	9786	0
	12	10528	21084
	13	9632	0
	14	7714	15268
	15	5190	0
	16	2831	5809
	17	1338	0
	18	538	102

	8		4
	9	158	0
	10	43	88
	11	11	0
	12	0	2
	13	0	0
	14	0	0

Tabel di atas jika digambarkan dalam bentuk grafik maka akan tampak seperti gambar di bawah:



Lampiran 12. Tabel perhitungan *polynomial* terbaik untuk CRC-8 pada panjang data = 8 bit

Jumlah Bit *Error*	Jenis *Polynomial*																			
	129	131	133	135	137	139	141	143	145	147	149	151	153	155	157	159	161	163	165	167
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	8	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	2	0
4	8	16	9	12	18	0	5	11	10	4	2	11	9	8	8	7	16	3	5	9
5	8	0	0	11	0	24	22	0	0	24	17	0	17	0	0	19	0	23	20	0
6	28	51	73	22	52	42	30	68	68	32	41	66	32	76	76	32	56	36	30	73
7	40	0	0	62	0	34	42	0	0	38	42	0	44	0	0	44	0	40	38	0
8	45	117	95	55	105	41	57	99	105	55	41	103	43	93	93	47	105	51	57	95
9	56	0	0	36	0	56	46	0	0	46	46	0	54	0	0	50	0	46	50	0
10	44	58	62	26	68	37	26	60	52	30	38	60	32	60	60	32	64	28	26	62
11	16	0	0	18	0	12	14	0	0	18	19	0	12	0	0	12	0	16	16	0
12	2	10	15	12	12	6	9	17	20	4	4	13	11	18	18	9	14	9	9	15
13	0	0	0	1	0	0	4	0	0	2	1	0	1	0	0	3	0	3	2	0
14	0	3	1	0	0	0	0	0	0	2	1	2	0	0	0	0	0	0	0	1
15	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

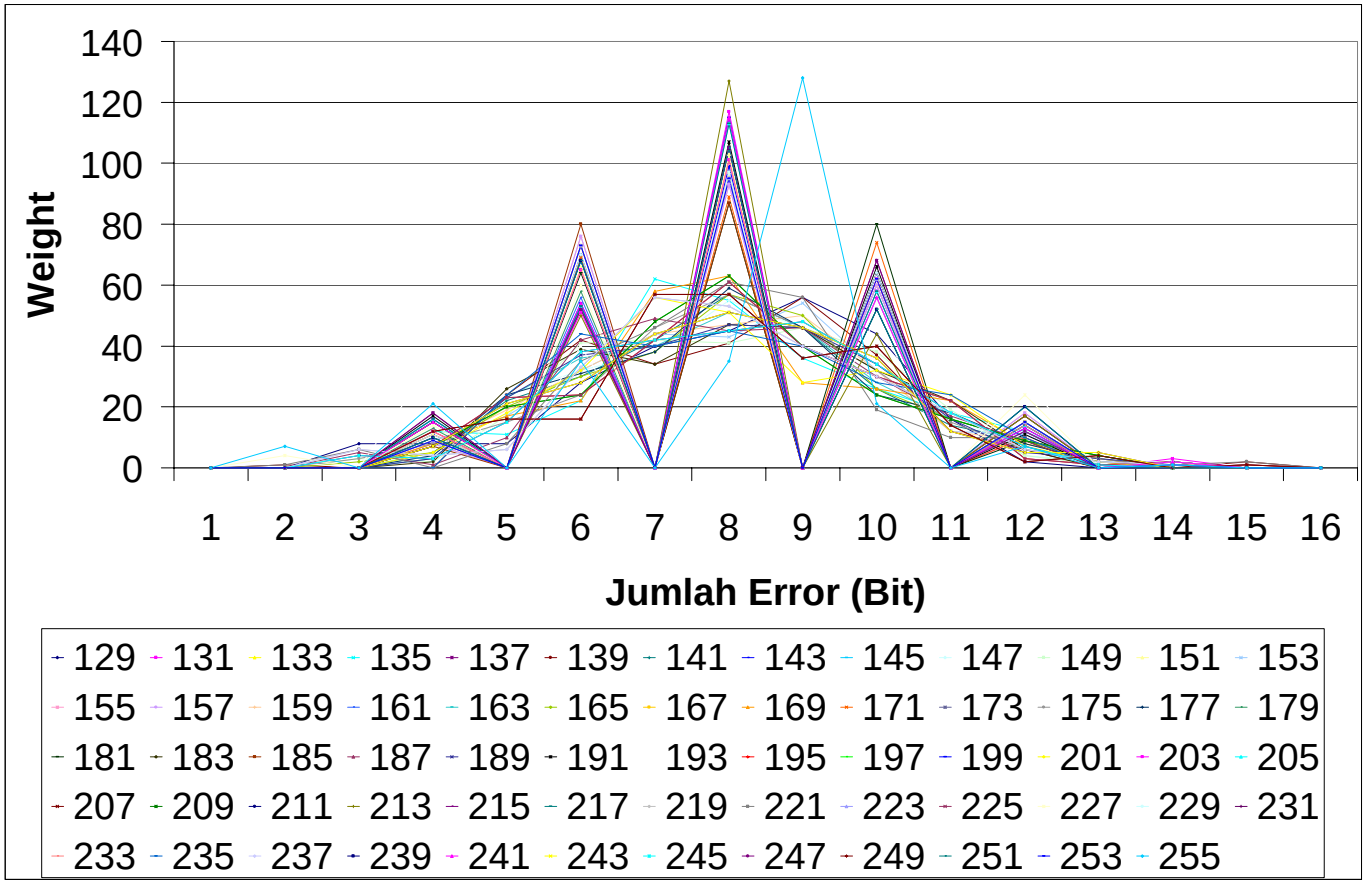
Jenis *Polynomial*																						
169	171	173	175	177	179	181	183	185	187	189	191	193	195	197	199	201	203	205	207	209	211
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0
0	0	0	3	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0
8	12	9	8	4	13	15	2	7	1	3	17	35	7	7	15	5	12	16	12	7	10
17	0	0	15	24	0	0	26	0	10	23	0	0	23	20	0	18	0	0	16	20	0
22	69	73	24	31	58	64	39	80	42	37	53	40	24	24	54	32	65	53	16	24	70

58	0	0	46	38	0	0	34	0	49	40	0	0	42	48	0	56	0	0	57	48	0
63	89	95	57	59	115	87	47	87	45	47	107	95	61	63	115	51	101	113	57	63	97
28	0	0	46	46	0	0	46	0	46	46	0	0	40	40	0	28	0	0	36	40	0
26	74	62	32	24	52	80	32	64	30	34	66	50	30	24	56	32	62	58	40	24	64
22	0	0	15	18	0	0	22	0	17	16	0	0	22	16	0	24	0	0	14	16	0
8	10	15	6	8	15	9	6	17	9	5	11	29	3	9	13	7	14	14	2	9	12
3	0	0	3	2	0	0	0	0	0	3	0	0	1	4	0	2	0	0	4	4	0
0	1	1	0	1	2	0	1	0	0	1	1	4	2	0	2	0	1	1	0	0	2
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

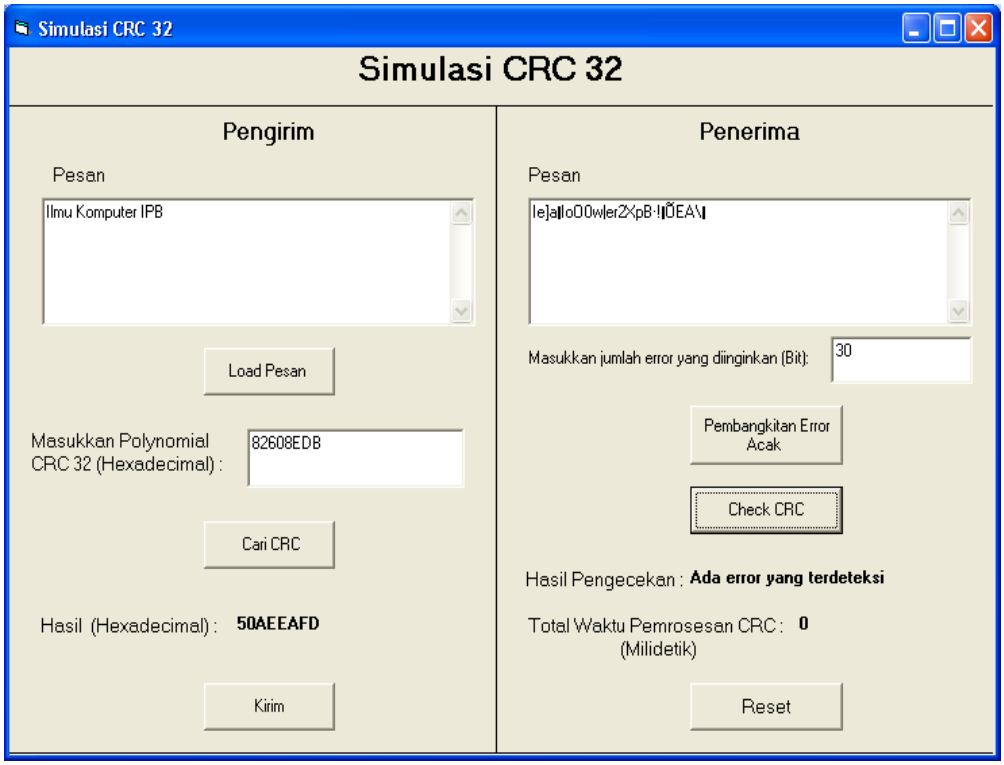
Jenis *Polynomial*

213	215	217	219	221	223	225	227	229	231	233	235	237	239	241	243	245	247	249	251	253	255
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	7
0	4	0	6	6	0	0	0	0	0	0	0	6	0	0	0	4	0	0	0	0	0
15	3	16	4	0	9	7	14	10	7	12	0	4	10	15	7	3	18	12	10	9	21
0	15	0	6	8	0	23	0	0	21	0	24	6	0	0	21	15	0	16	0	0	0
50	38	53	33	35	73	24	60	70	28	65	44	33	68	54	28	38	52	16	68	73	35
0	42	0	56	46	0	42	0	0	44	0	40	56	0	0	44	42	0	57	0	0	0
127	45	113	53	61	95	61	97	97	51	101	45	53	105	115	51	45	105	57	105	95	35
0	48	0	40	56	0	40	0	0	46	0	40	40	0	0	46	48	0	36	0	0	128
44	34	58	30	19	62	30	56	64	36	62	28	30	52	56	36	34	68	40	52	62	21
0	18	0	18	10	0	22	0	0	12	0	24	18	0	0	12	18	0	14	0	0	0
17	7	14	6	10	15	3	24	12	5	14	10	6	20	13	5	7	12	2	20	15	7
0	1	0	2	0	0	1	0	0	5	0	0	2	0	0	5	1	0	4	0	0	0
2	0	1	1	1	1	2	0	2	0	1	0	1	0	2	0	0	0	0	0	1	1
0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

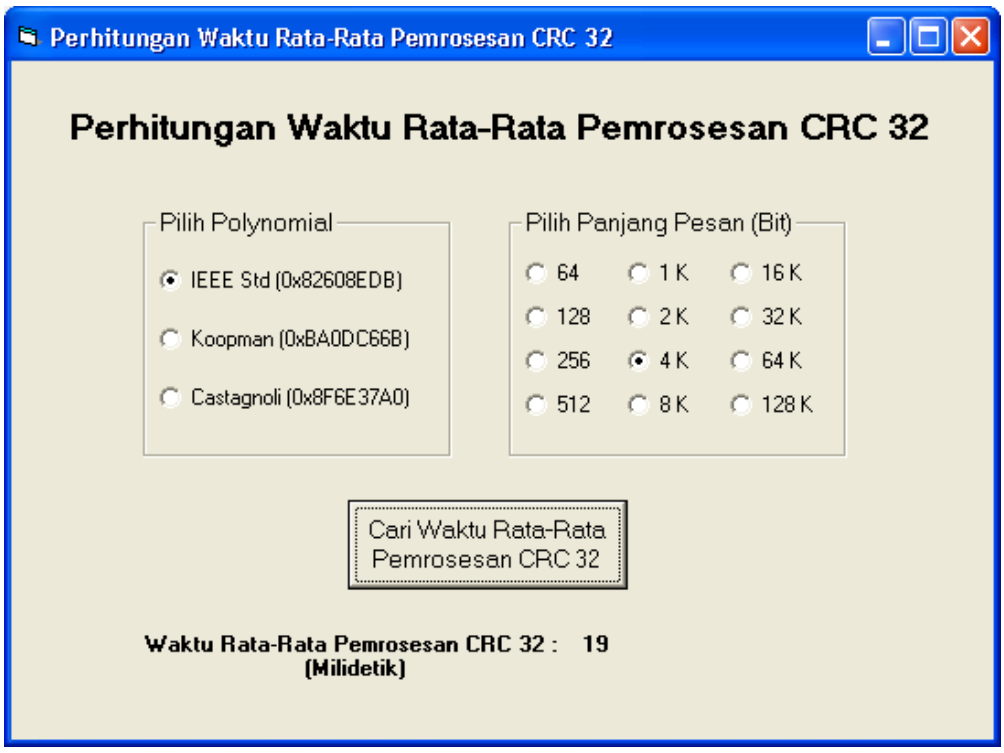
Lampiran 13. Grafik perhitungan *polynomial* terbaik untuk CRC-8 pada panjang data = 8 bit



Lampiran 14. Antarmuka implementasi simulasi CRC-32



Lampiran 15. Antarmuka implementasi perhitungan waktu rata-rata CRC-32



Lampiran 16. Antarmuka implementasi perhitungan kinerja CRC-8