Sistem Verifikasi BitTorrent

Irvan Aditya (13510016)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13510016@std.stei.itb.ac.id

***Abstrak*—Sistem BitTorrent sudah tidak asing lagi bagi kita. Sistem ini memungkinkan para penggunanya untuk berbagi *file* secara cepat dengan memanfaatkan jaringan *peer-to-peer*. Dalam sistem ini, cara pendistribusian data yang biasanya dalam jumlah besar memanfaatkan fungsi *hash*. Dengan sistem seperti itu, BitTorrent memiliki keuntungan dan kerugiannya sendiri. Dalam makalah ini, secara spesifik yang akan dibahas adalah tentang pengaplikasian salah satu materi struktur diskrit yaitu *hash* dalam sistem BitTorrent.**

***Kata Kunci*—*hash*, BitTorrent**

I. PENDAHULUAN

Ketika mengunduh *file*, apalagi dalam jumlah yang banyak atau ukuran yang besar, kita pasti ingin mendapatkannya dengan cepat namun aman dan andal. Kita seringkali kesal menunggu komputer selesai mengunduhnya, apalagi jika unduhan tiba-tiba putus di tengah jalan dan tidak bisa dilanjutkan. Kita harus mengulang dari awal. Ada juga kasus ketika unduhan dinyatakan selesai oleh komputer namun ternyata *file* hasil unduhan tersebut rusak atau tidak lengkap.

Selain mengunduh, kita juga pasti pernah memiliki kepentingan untuk berbagi *file* dengan orang lain atau mengunggahnya untuk bisa digunakan oleh orang banyak. Namun, untuk bertemu secara fisik dengan orang tersebut akan sulit. Jika kita ingin mengunggahnya pun, kita harus menggunakan *file-hosting*, dan kadang cara itu terlalu lama dan merepotkan.

BitTorrent adalah protokol untuk berbagi *file* yang dirancang dan diperkenalkan oleh Bram Cohen pada tahun 2001[1]. Protokol berbagi *file* ini memanfaatkan jaringan *peer-to-peer*. Sistem BitTorrent ini sudah terbukti menjadi cara berbagi *file* yang sangat andal. Dengan BitTorrent, semua urusan transfer data dalam jumlah besar menjadi sangat mudah dan cepat. Kita hanya perlu mengunduh program klien dan menggunakannya untuk mendapatkan *file* yang kita inginkan.

Efisiensi protokol BitTorrent ini tidak terlepas dari sistem pendistribusian datanya yang berbasis pengembangan *hash table* melalui jaringan *peer-to-peer*.

II. DASAR TEORI STRUKTUR DISKRIT

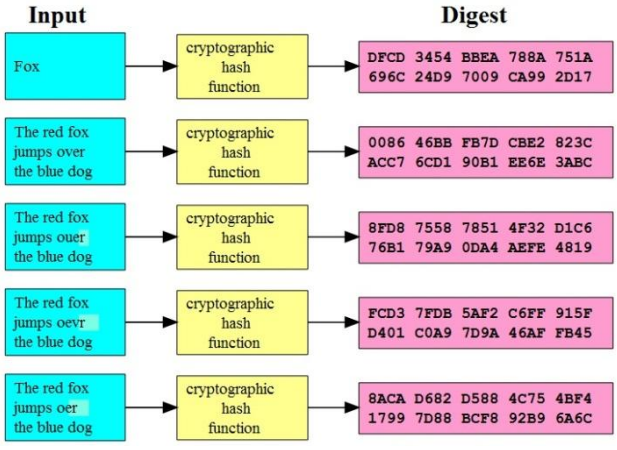
A. *Fungsi* Hash

Fungsi *hash* adalah algoritma atau subprogram yang memetakan set data yang besar menjadi set data yang lebih kecil[2]. Secara umum, sebuah fungsi *hash* memiliki masukan dengan panjang bebas dan memiliki keluaran dengan panjang tertentu[3].

Fungsi *hash* yang sederhana memanfaatkan teori kekongruenan. Bentuk sederhana fungsi *hash* tersebut adalah $h(K) = K \bmod m$. Fungsi *hash* h memetakan kunci K yang berupa bilangan bulat ke dalam alokasi memori yang ada sebanyak m tempat. Fungsi *hash* ini bertujuan untuk pengalamatan secara cepat ke slot-slot yang tersedia.

Fungsi *hash* kriptografi adalah fungsi *hash* yang lebih rumit lagi. Dalam fungsi hash sederhana, kemungkinan untuk terjadi *collision* atau tabrakan sangat besar. Fungsi *hash* kriptografi memperkecil terjadinya tabrakan ini. Selain itu, fungsi *hash* ini dibuat sedemikian rupa sehingga perubahan kecil dalam set data dapat membuat perubahan yang jelas dalam nilai hasil enkripsinya.

Fungsi hash kriptografi mengenkripsi sebuah set data menjadi pesan yang dienkripsi dalam bentuk *string*. Fungsi ini memungkinkan pengecekan terhadap set data, apakah set data tersebut masih sesuai dengan aslinya atau sudah tidak lagi relevan. Contoh fungsi *hash* kriptografi dapat dilihat pada Gambar 1 di bawah ini.



Gambar 1 Contoh fungsi *hash* kriptografi

B. Hash Table

Hash table adalah struktur data yang menggunakan

fungi *hash* untuk memetakan nilai identifikasi, disebut juga kunci, ke dalam indeks nilai tertentu[4]. Idealnya, setiap kunci dipetakan ke slot indeks yang unik, berbeda-beda dengan indeks dari setiap kunci lainnya. Namun pada kenyataannya sangat sering terjadi *collision* atau tabrakan. Hal ini sangat mungkin terjadi seiring dengan banyaknya jumlah set data yang diproses.

Untuk menjamin seminimal mungkin tabrakan, fungsi *hash* yang digunakan harus memiliki jenis distribusi seragam. Keseragaman sulit diuji dari rancangan fungsinya. Biasanya keseragaman diketahui melalui uji statistik. Keseragaman ini hanya menjadi syarat untuk selang tertentu dalam tabel yaitu pada slot-slot yang disediakan dalam *hash table*.

C. Distributed Hash Table

DHT adalah jenis sistem distribusi yang terdesentralisasi yang menggunakan *lookup* seperti dalam *hash table*[5]. Kunci dan nilai indeks tersimpan dalam DHT, dan setiap simpul dalam jaringan dapat memperoleh nilai indeks yang tepat dengan menggunakan kunci tertentu. Pemetaan dibuat sedemikian rupa agar terjaga secara terdistribusi oleh setiap simpul sehingga meminimalkan gangguan ketika terjadi perubahan set peserta. Ini memungkinkan DHT menampung jumlah simpul dengan banyak jumlah kedatangan, kepergian, dan kegagalan.

Pada dasarnya DHT memiliki 3 karakteristik yang harus dipenuhi, yaitu: tidak diperlukan adanya koordinasi terpusat; sistem tetap terjaga keandalannya walaupun banyak simpul yang datang, pergi, dan gagal secara kontinu; dan sistem tetap efisien walaupun jumlah simpul dapat mencapai jutaan. Selain itu, DHT juga harus memenuhi syarat-syarat dasar sistem terdistribusi yaitu penyeimbangan beban, integritas data, dan performansi atau kinerja.

Sistem *distributed hash table* pada dasarnya menggunakan prinsip pemetaan *hash table*. Sistem ini memungkinkan setiap *peer* untuk menjadi "*tracker*". Sistem ini juga menggunakan prinsip *hash*, sama seperti kriptografi. Karena itu, sistem ini juga termasuk menggunakan teori kekongruenan. Namun, sistem ini tidak akan dibahas dalam makalah ini.

III. DASAR SISTEM BITTORRENT

A. Cara Kerja Dasar BitTorrent

Pada dasarnya, sistem BitTorrent memerlukan paling sedikit sebuah *tracker* untuk beroperasi. *Tracker* adalah *server* yang membantu komunikasi antar *peer* yang menggunakan protokol BitTorrent. *Tracker* bertugas mneghubungkan sebuah *peer* yang meminta untuk dihubungkan dengan *peer* lain yang tersedia untuk dihubungkan.

Sebuah sistem BitTorrent yang menangani sebuah *file* pada awalnya bekerja dengan cara membentuk sebuah *swarm*. *Swarm* dibentuk oleh *tracker* dengan sebuah peer yang memiliki *file* autentik yang lengkap, atau biasa disebut *seed*. *Seed* pertama ini membuat sebuah *descriptor* untuk *swarm* tersebut.

Sistem BitTorrent memperbolehkan semua yang memiliki *descriptor* tersebut untuk memasuki *swarm* sebagai *peer*. Masing-masing *peer* kemudian akan menjadi sebuah simpul dalam jaringan *swarm* tersebut. Fungsi descriptor ini sebenarnya adalah memuat informasi dasar tentang sebuah *torrent*. *Descriptor* pada dasarnya memuat nilai dari fungsi *hash* kriptografi, atau sering disebut *checksum*. Umumnya fungsi *hash* kriptografi yang digunakan untuk BitTorrent adalah fungsi SHA-1.

File yang akan didistribusikan akan dibagi-bagi menjadi segmen-segmen, atau biasanya disebut *piece*. Setiap *piece* akan diterjemahkan menjadi suatu nilai lain oleh fungsi *hash* kriptografi ini. Hal ini memungkinkan setiap *piece* yang rusak karena aktivitas tertentu dengan mudah terdeteksi. Setiap *peer* yang selesai mendapatkan sebuah *piece* akan menjadi sumber untuk *piece* tersebut bagi *peer* lainnya yang belum memiliki *piece* tersebut.

Peer yang selesai mengunduh semua *piece*, artinya telah mendapatkan *file* secara utuh, akan berubah statusnya menjadi *seed*. Dia akan menjadi sumber *file* tersebut untuk peer lainnya. Begitulah cara kerja *swarm* dalam sebuah sistem BitTorrent.

B. Keunggulan BitTorrent

Pengaplikasian cara kerja tersebut membuat sistem BitTorrent memiliki beberapa keuntungan. Di antara keuntungannya adalah sistem BitTorrent tidak membebani server mana pun walau pengguna yang berpartisipasi dalam sebuah *swarm* mencapai jutaan, karena jaringan diciptakan dari simpul-simpul jaringan yang berbagi *bandwidth*. Selain itu, karena fungsi *hash* yang digunakannya, sistem BitTorrent dapat juga digunakan untuk mengecek dan jika perlu memperbaiki integritas data.

C. Kelemahan Sistem BitTorrent dengan Tracker

Cara kerja dasar BitTorrent ternyata memiliki kelemahan. Ketika *tracker* tidak dapat beroperasi, seluruh *swarm* akan lumpuh. Semua simpul tidak akan dapat berkomunikasi satu sama lain. Karena alasan itu, sistem BitTorrent kini telah disempurnakan lagi. Beberapa pengembangannya adalah teknologi *multi-tracker* dan *distributed hash table* atau DHT.

Sistem *multi-tracker* adalah sistem yang menggunakan beberapa *tracker* untuk menjalin komunikasi antara *peer*. Sistem ini memungkinkan pergantian *tracker* ketika salah satu *tracker* tidak dapat beroperasi.

Sistem *distributed hash table* adalah sistem yang paling unggul. Sistem ini pada dasarnya mendistribusikan fungsi *tracker* kepada semua simpul dalam *swarm*. Hal ini memungkinkan pembentukan *swarm* tanpa *tracker*. Karenanya sistem ini disebut sistem desentralisasi, di mana setiap *peer* dapat langsung menghubungkan diri dengan *peer* lain tanpa harus dibantu server terpusat yaitu *tracker*.

IV. *HASH* DALAM BITTORRENT

A. Pengenalan SHA-1

Seperti telah dijelaskan dalam bab sebelumnya, setiap *piece* dalam sebuah *torrent* diterjemahkan dengan sebuah fungsi *hash* kriptografi. Fungsi *hash* kriptografi yang umumnya digunakan adalah algoritma SHA-1. SHA adalah singkatan dari Secure Hash Algorithm. Algoritma ini dikembangkan oleh National Security Agency, Amerika Serikat.

Algoritma *hash* kriptografi SHA-1 adalah algoritma fungsi *hash* kriptografi yang paling banyak digunakan di dunia. Keluaran dari algoritma fungsi SHA-1 berupa nilai 160 bit[6]. Algoritma ini disebut-sebut mirip dengan algoritma kriptografi MD5.

Contoh penggunaan fungsi SHA-1, misalnya ketika masukan fungsi adalah *string* "anu", maka keluaran fungsi adalah "e067b048 6639afb9 97c04c24 de3cb95b b74f6b23". Ketika masukan *string* kosong, keluaran fungsi SHA-1 adalah "da39a3ee 5e6b4b0d 3255bfef 95601890 afd80709".

Algoritma *hash* kriptografi SHA-1 memiliki *avalanche effect*, yaitu ketika masukan diganti sedikit, keluaran fungsi akan berbeda sangat jauh. Contoh yang paling populer adalah ketika kita memberi masukan "The quick brown fox jumps over the lazy dog", keluaran fungsi tersebut adalah "2fd4e1c6 7a2d28fc ed849ee1 bb76e739 1b93eb12". Sedangkan, ketika kita memberi masukan fungsi berupa "The quick brown fox jumps over the lazy cog", keluaran fungsi itu berubah menjadi "de9f2c7f d25e1b3a fad3e85a 0bd17d9b 100db4b3". Bisa terlihat bahwa perbedaan keluarannya sangat signifikan.

Pada dasarnya, setiap fungsi *hash* memiliki kemungkinan untuk terjadi tabrakan. Sampai saat ini, tabrakan yang telah diteliti untuk SHA-1 masih merupakan sebuah pernyataan teoretis. Karena itu, algoritma *hash* kriptografi SHA-1 masih dianggap aman dan andal dalam penggunaannya.

Penggunaan SHA-1 sangat banyak di sekitar pengolahan data. Algoritma SHA-1 sering digunakan untuk mengecek integritas data. Selain itu SHA-1 juga dapat digunakan sebagai kode identifikasi data atau *file*. Secara khusus, algoritma SHA-1 digunakan sebagai pengamanan aplikasi dan protokol. Salah satu contohnya adalah penggunaan algoritma ini untuk verifikasi *signature* pada konsol permainan Nintendo Wii ketika *booting*. Algoritma SHA-1 juga digunakan sebagai algoritma *hash* kriptografi resmi pemerintah Amerika Serikat untuk penanganan informasi yang bersifat sensitif dan rahasia.

B. Algoritma SHA-1

Sebelum membahas fungsi *hash* SHA-1, ada baiknya kita mencermati dulu parameter-parameter dan operasi dasar yang digunakan. Ada beberapa yang perlu diperhatikan, yaitu:

Parameter	Spesifikasi
a, b, c, .., h	Peubah yang digunakan dalam perhitungan
$H^{(i)}$	Nilai *hash* yang ke-*i*. $H^{(0)}$ adalah nilai *hash* awal, $H^{(N)}$ adalah nilai *hash* akhir.
$H_j^{(i)}$	Kata ke-*j* dari nilai *hash* yang ke-*i*.
K_t	Nilai konstanta yang digunakan dalam perhitungan *hash*.
k	Jumlah nol yang ditambahkan dalam tahap *padding*.
l	Panjang pesan *M* dalam bit.
m	Jumlah bit dalam blok pesan $M^{(i)}$.
M	Pesan yang akan dihitung.
$M^{(i)}$	Blok pesan yang ke-*i*, dengan panjang *m* bit.
$M_j^{(i)}$	Kata ke-*j* dari blok pesan yang ke-*i*.
n	Jumlah bit yang dirotasi atau digeser ketika kata dioperasikan
N	Jumlah blok dari pesan yang telah melalui *padding*.
T	Kata sementara sepanjang *w*-bit yang digunakan dalam perhitungan *hash*.
w	Jumlah bit dalam kata
W_t	Kata sepanjang *w*-bit yang ke-t dalam daftar pesan.

Perlu diperhatikan pula beberapa operasi dasar yang digunakan dalam perhitungan.

Operasi	Spesifikasi
$\wedge$	Operasi AND.
$\vee$	Operasi OR.
$\oplus$	Operasi XOR.
$\neg$	Menyatakan komplemen.
$+$	Penambahan modulo 2^w. Misal $x + y$.Peubah x dan y merepresentasikan *integer* X dan Y, $0 \le X \le 2^w$dan $0 \le Y \le 2^w$. Untuk *integer* positif U dan V, U mod V adalah sisa pembagian U terhadap V. Hitung $Z = (X + Y) \bmod 2^w$. Kemudian $0 \le Z \le 2^w$. Konversikan *integer* Z ke kata z, dan definisikan $z = x + y$.
$\ll$	Operasi penggeseran ke kiri. Jika $x \ll n$ maka n-bit di paling kiri x dihilangkan dan dilakukan *padding* dengan n-bit nol di paling kanan x.
$\gg$	Operasi penggeseran ke kanan. Jika $x \gg n$ maka n-bit di paling kanan x dihilangkan dan dilakukan *padding* dengan n-bit nol di paling kiri x.
$ROTL^n(x)$	Rotasi kiri, yaitu $ROTL^n(x) = (x \ll n) \vee (x \gg w - n)$.

Adapun fungsi *hash* kriptografi dari algoritma SHA-1, yaitu $f_t(x, y, z)$ adalah sebagai berikut:

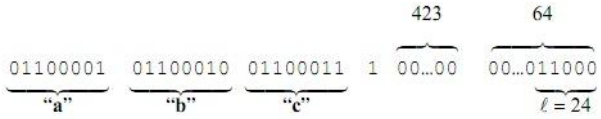
$$f_t(x,y,z) = \begin{cases} Ch(x,y,z)=(x \wedge y) \oplus (\neg x \wedge z) & 0 \le t \le 19 \\ Parity(x,y,z)=x \oplus y \oplus z & 20 \le t \le 39 \\ Maj(x,y,z)=(x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) & 40 \le t \le 59 \\ Parity(x,y,z)=x \oplus y \oplus z & 60 \le t \le 79. \end{cases}$$

Fungsi tersebut menggunakan sekuens dari fungsi logika $f_0, f_1, .., f_{79}$. Setiap fungsi f_t selama $0 \le t \le 79$ mengoperasikan tiga buah kata sepanjang 32-bit *x*, *y*, dan *z*.

Sementara itu, algoritma SHA-1 menggunakan sekuens dari delapan puluh konstanta kata 32-bit yaitu $K_0, K_1, .., K_{79}$, sebagai berikut:

$$K_t = \begin{cases} \texttt{5a827999} & 0 \le t \le 19 \\ \texttt{6ed9eba1} & 20 \le t \le 39 \\ \texttt{8f1bbcdc} & 40 \le t \le 59 \\ \texttt{ca62c1d6} & 60 \le t \le 79 \end{cases}$$

Sebelum perhitungan dimulai, pesan masukan harus melalui proses *padding* terlebih dahulu. Tujuan dari proses *padding* adalah untuk menjamin semua pesan masukan tepat 512-bit. Misal ada sebuah pesan *M* dengan panjang *l*-bit. Tambahkan bit "1" di akhir dari pesan, diikuti *k*-bit nol, dan *k* adalah solusi tidak negatif dari persamaan $l + 1 + k \equiv 448 \bmod 512$. Kemudian, tambahkan 64-bit blok yang diambil dari representasi biner dari angka *l*. Contohnya, pesan "abc" memiliki panjang $8 \times 3 = 24$-bit. Jadi pesan tersebut ditambahkan dengan bit "1", kemudian dengan bit "0" sebanyak $448 - (24 + 1) = 423$-bit, kemudian dengan blok sepanjang 64-bit yang merupakan representasi biner dari angka 24. Setelah proses ini, pesan masukan kemudian akan menjadi pesan sepanjang 512-bit. Pesan 512-bit dapat dilihat pada Gambar 2.



Gambar 2 Contoh pesan yang telah melalui *padding*

Setelah melalui proses *padding*, pesan harus melalui *parsing* terlebih dahulu menjadi blok-blok *N* sepanjang 512-bit, yaitu $M^{(1)}, M^{(2)}, .., M^{(N)}$. Karena blok 512-bit dapat dibuat sebagai 16 buah kata sepanjang 32-bit, maka 32-bit pertama dari blok pesan *i* tersebut dinotasikan M_0^i, 32-bit selanjutnya dinotasikan M_1^i, dan seterusnya hingga 32-bit terakhir dari pesan menjadi M_{15}^i.

Sebelum memulai perhitungan, kita harus mengatur nilai *hash* awal $H^{(0)}$. Nilai *hash* awal diatur sebagai berikut:

$$\begin{aligned} H_0^{(0)} &= \texttt{67452301} \\ H_1^{(0)} &= \texttt{efcdab89} \\ H_2^{(0)} &= \texttt{98badcfe} \\ H_3^{(0)} &= \texttt{10325476} \\ H_4^{(0)} &= \texttt{c3d2e1f0} \end{aligned}$$

Nilai *hash* awal tersebut terdiri dari lima buah kata sepanjang 32-bit dalam representasi *hexa-decimal*.

Langkah selanjutnya adalah melakukan perhitungan *hash*. Ada empat langkah yang harus diulang sebanyak *N* kali. Empat langkah tersebut adalah menyiapkan daftar pesan, menyiapkan lima peubah yaitu *a*, *b*, *c*, *d*, dan *e* dengan nilai hash (*i*–1), pengerjaan, dan perhitungan nilai hash menengah $H^{(i)}$ yang ke-*i*.

Pertama, kita menyiapkan daftar pesan yang dinotasikan dengan W_t. Daftar pesan didapat sebagai berikut:

$$W_t = \begin{cases} M_t^{(i)} & 0 \le t \le 15 \\ ROTL^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) & 16 \le t \le 79 \end{cases}$$

Setelah itu, kita menyiapkan lima buah peubah yang akan digunakan. Pertama $a = H_0^{(i-1)}$, kedua $b = H_1^{(i-1)}$, ketiga $c = H_2^{(i-1)}$, keempat d= $H_3^{(i-1)}$, dan kelima $e = H_4^{(i-1)}$.

Setelah selesai menyiapkan peubah, kita harus mengoperasikannya satu per satu sebanyak delapan puluh kali, yaitu ketika $0 \le t \le 79$. Yang dikerjakan adalah sebagai berikut:

$$\begin{aligned} T &= ROTL^5(a) + f_t(b,c,d) + e + K_t + W_t \\ e &= d \\ d &= c \\ c &= ROTL^{30}(b) \\ b &= a \\ a &= T \end{aligned}$$

Setelah melakukannya sebanyak delapan puluh kali, kita masuk tahap selanjutnya, yaitu menghitung nilai hash menengah $H^{(i)}$ yang ke-*i*. Ada beberapa hal yang harus dilakukan, yaitu:

$$\begin{aligned} H_0^{(i)} &= a + H_0^{(i-1)} \\ H_1^{(i)} &= b + H_1^{(i-1)} \\ H_2^{(i)} &= c + H_2^{(i-1)} \\ H_3^{(i)} &= d + H_3^{(i-1)} \\ H_4^{(i)} &= e + H_4^{(i-1)} \end{aligned}$$

Keempat langkah di atas harus diulangi sebanyak *N* kali. Setelah selesai, hasil keluaran fungsi *hash* kriptografi SHA-1 sepanjang 160-bit dari proses yang ke-*N* (setelah memproses sebanyak *N* kali) dengan masukan pesan *M* dapat didefinisikan sebagai $H_0^{(N)} \| H_1^{(N)} \| H_2^{(N)} \| H_3^{(N)} \|$

$H_4^{(N)}$.

Pada akhirnya, representasi yang sangat sering dipakai untuk menunjukkan nilai *hash* keluaran fungsi ini bukanlah nilai bit biner, melainkan representasi *hexa-decimal*. Hal ini disebabkan tidak efektifnya representasi bit biner, karena pesan akan sangat panjang (160-bit). Sementara itu, representasi *hexa-decimal* dapat memetakan 4-bit biner menjadi 1-bit *hexa-decimal*, yang artinya hanya dibutuhkan seperempat dari panjang pesan jika direpresentasikan dalam bentuk bit biner.

C. Penggunaan SHA-1 dalam BitTorrent

Dalam BitTorrent, fungsi *hash* digunakan untuk verifikasi set data yang disebut *piece*. Masing-masing *piece* memiliki nilai *hash*. Karena itulah setiap *piece* dapat dikenali secara unik. Selain itu, ketika terdapat masalah seperti kerusakan *piece*, hal ini dapat segera dikenali karena sifat *avalanche effect* yang dimiliki oleh fungsi *hash*.

Dengan adanya pemetaan menggunakan algoritma *hash* seperti ini, ada beberapa keuntungan yang diperoleh. Salah satu keuntungannya adalah kemampuan memperbaiki data berdasarkan pengecekan dari nilai *hash* setiap *piece*-nya.

Karena setiap *piece* diterjemahkan oleh fungsi *hash*, dan karena fungsi *hash* yang digunakan memiliki sifat *avalanche effect*, kerusakan sekecil apapun jika terdapat dalam sebuah *piece* akan terdeteksi. Dengan prinsip itu, sebuah aplikasi klien BitTorrent dapat melakukan pengecekan *piece* mana saja yang sudah dimiliki oleh *peer* tersebut dan *piece* mana saja yang sudah dimiliki *peer* tersebut yang sesuai dengan *piece* aslinya. Jika ada piece yang belum dimiliki oleh *peer* tersebut, maka *peer* ini akan mengunduh *piece* itu dari *peer* lainnya yang mempunyai *piece* yang *valid*. Jika ada *piece* yang nilai *hash*-nya tidak sesuai dengan nilai *hash* yang *valid*, *peer* tersebut harus mengganti *piece* yang tidak *valid* dengan *piece* yang sama namun *valid*. Dengan cara ini, seluruh data yang didistribusikan akan seluruhnya *valid*.

V. KESIMPULAN

Sesungguhnya sistem BitTorrent adalah sistem yang efektif. Sistem ini tidak hanya mampu mendistribusikan data secara mandiri, tetapi juga dapat memastikan bahwa seluruh data yang terdistribusi adalah data yang *valid*. Penggunaan algoritma *hash* kriptografi SHA-1 dalam sistem BitTorrent dapat memastikan keamanan dan validitas data.

REFERENSI

[1] http://finance.groups.yahoo.com/group/decentralization/message/3160 diakses tanggal 15 Desember 2012.

[2] http://burtleburtle.net/bob/hash/index.html diakses tanggal 15 Desember 2012.

[3] http://x5.net/faqs/crypto/q94.html diakses tanggal 15 Desember 2012.

[4] http://crpit.com/confpapers/CRPITV91Askitis.pdf diakses tanggal 15 Desember 2012.

[5] http://www.cs.berkeley.edu/~istoica/papers/2003/cacm03.pdf diakses tanggal 17 Desember 2012.

[6] http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf diakses tanggal 17 Desember 2012.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 18 Desember 2012

ttd

Irvan Aditya
13510016