BAB II
LANDASAN TEORI

2.1 Proxy Server

Dalam jaringan komputer, sebuah *proxy server* adalah sebuah *server* yang menjembatani komunikasi antara *client* yang mencari sumber daya dan *server* yang menyediakan sumber daya. Sesuai dengan tujuan diciptakannya, *proxy* memberi struktur dan enkapsulasi pada sistem terdistribusi (Shapiro, 1986). *Client*, yang meminta layanan berupa dokumen, laman, atau layanan lainnya, tidak terhubung langsung ke *server* penyedia layanan, melainkan terhubung kepada *proxy server*. Permintaan *client* yang diterima oleh *proxy server* akan diproses dan dievaluasi sehingga *proxy server* dapat menetukan penyederhanaan permintaan dan meneruskannya kemudian.

Jenis *proxy* yang paling banyak digunakan saat ini adalah *web proxy*, yang menjembatani akses ke dalam World Wide Web dan menyediakan akses anonim. Fungsi-fungsi yang umum ditemukan dalam *web proxy*, antara lain: pengawasan, penyaringan konten, *logging*, *caching*, terjemahan bahasa, anonimitas, pengamanan jaringan lokal (TLDP, 2015), peningkatan performa jaringan lokal (Thomas, 2006).

Proxy server dapat dijalankan dalam komputer lokal ataupun pada berbagai titik dalam jaringan di antara komputer lokal dan Internet. Jenis-jenis *proxy server* antara lain sebagai berikut.

a. *Proxy server* yang meneruskan akses sumber daya tanpa modifikasi disebut *tunneling proxy*.
b. *Forward proxy* adalah *proxy* yang digunakan untuk mengakses beragam layanan dari dalam Internet.
c. *Open proxy* adalah *forward proxy* yang bisa diakses oleh siapa aja yang terhubung dalam Internet.
d. *Reverse proxy* adalah *proxy* yang digunakan untuk meregulasi akses ke dalam server atau jaringan lokal.

Reverse proxy adalah *proxy server* yang terlihat sebagai *client* oleh *server* umum. Satu permintaan *client* dapat saja diproses oleh 2 atau lebih *proxy server* secara berkesinambungan. Balasan dari permintaan juga disampaikan kepada *client* dengan seakan-akan berasal langsung dari server, sehingga *client* tidak perlu mengetahui rincian dari *server* yang memberi layanan (Apache, 2015). *Proxy server* ditempatkan di dekat satu atau lebih *web server*. Semua lintasdata yang datang dari Internet dengan tujuan salah satu dari *web server* tersebut akan ditangani oleh *proxy server*. *Reverse proxy* ditempatkan dekat dengan *web server* untuk mengoptimalkan fungsi enkripsi, penyerataan beban, *caching*, kompresi, mendukung pengamanan jaringan lokal, dan layanan publikasi ke jaringan luar.

2.1.1 Squid

Squid adalah sebuah *proxy server* dan *web cache daemon* bebas biaya yang di bawah lisensi GNU General Public License. Squid dapat menjalankan fungsi *cache*, DNS, penyaringan lintasdata, dan berbagai fungsi lainnya. Selain menangani protokol HTTP dan FTP, Squid juga menyediakan layanan terbatas untuk protokol-protokol lainnya seperti TLS, SSL, Internet Gopher, dan HTTPS (Squid, 2015).

Salah satu fungsi yang disediakan Squid adalah *cache*. Squid dapat digunakan pada konfigurasi normal, yaitu Squid tidak membatasi jumlah web server yang dapat terdaftar dalam *cache* untuk melayani jumlah *client* yang terbatas. Squid juga dapat digunakan dapat digunakan dalam konfigurasi *reverse proxy*, dimana Squid membatasi jumlah web server yang dapat terdaftar dalam *cache* untuk jumlah *client* yang tidak terbatas.

Squid juga memiliki fitur anonimitas yang diterapkan dengan mengubah bagian-bagian spesifik dari header dalam HTTP. Pengguna bisa saja tidak mengetahui bahwa informasi-informasi tersebut dicatat oleh Squid. Konfigurasi ini terdapat dalam dokumentasi `header_access` dan `header_replace` dalam Squid.

2.1.2 Squidguard

Squidguard adalah perangkat lunak pengalih URL yang dapat digunakan untuk mengontrol konten laman yang dapat diakses pengguna. Squidguard

dikembangkan sebagai pengaya untuk Squid; menggunakan satu atau lebih *blacklist* untuk mengatur pengalihan akses laman. Squidguard perlu dijalankan dalam *server* dengan sistem operasi Unix atau GNU/Linux, tetapi setelah terpasang, semua komputer dalam jaringan, tidak terbatas pada sistem operasinya, dapat menggunakan layanan Squidguard.

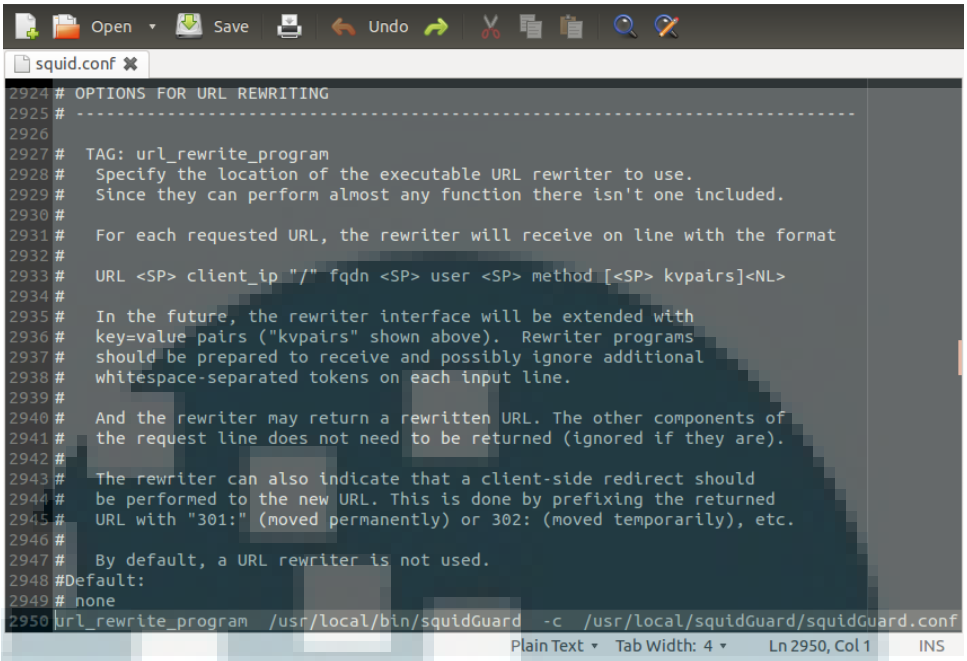
Squidguard dikembangkan oleh Pål Baltzersen dan Lars Erik Håland. Squidguard diimplementasikan dan diperkaya oleh Lars Erik Håland selama tahun 1990-an di Tele Danmark InterNordia (Squidguard, 2015). Versi 1.5 diluncurkan pada 2010. Squidguard adalah perangkat lunak bebas biaya di bawah lisensi GNU General Public License (GPL) versi 2.

Kemampuan Squidguard untuk menyaring lintasdata sangat tergantung pada kualitas *blacklist* yang digunakan. Beberapa *blacklist* yang cukup baik yang dapat ditemukan daring antara lain dapat ditemukan pada situs shallistlist.de, squidblacklist.org, dan urlblacklist.com.

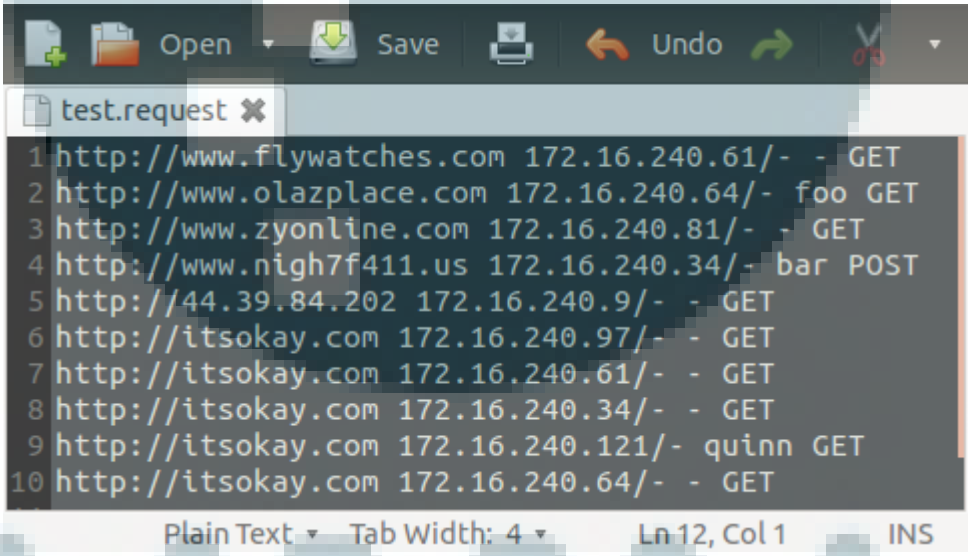
Squidguard menerima masukan baris URL dari Squid. Hal ini dilakukan dengan mengubah/menambahkan konfigurasi `url_rewrite_program` pada konfigurasi Squid, seperti yang terlihat pada Gambar 1. Squidguard menerima masukan berupa string dengan format sebagai berikut.

```
[URL] [IP ADDRESS]/[CLIENT_NAME] [CLIENT_ID] [METHOD]
```

Contoh *request* yang dapat diproses oleh Squidguard ditunjukkan oleh Gambar 2.



Gambar 1. Konfigurasi `url_rewrite_program` pada Squid



Gambar 2. Contoh request yang dapat diproses oleh Squidguard

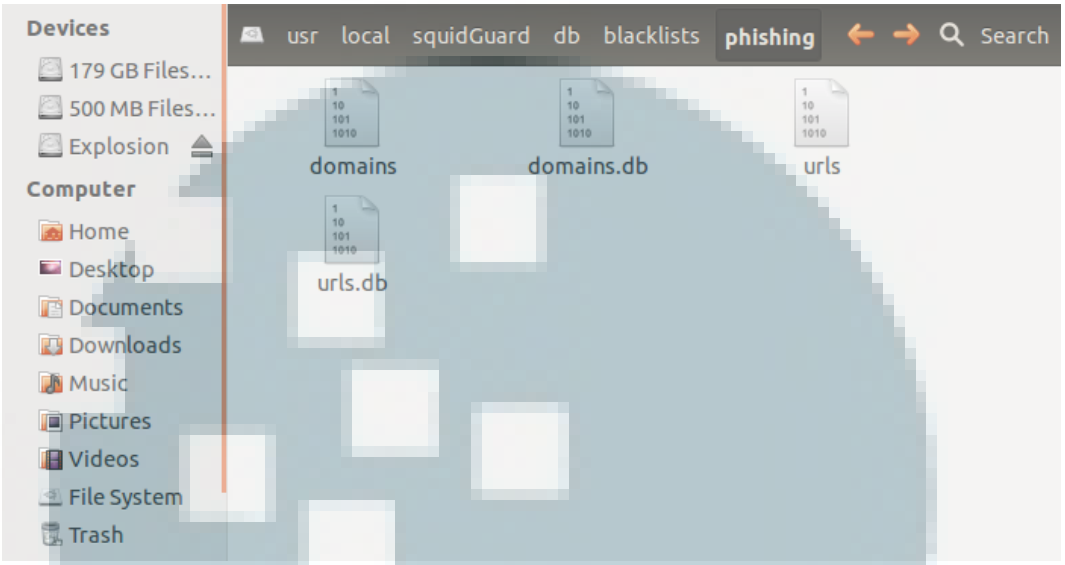
Squidguard mengalihkan akses laman berdasarkan dokumen konfigurasi Squidguard yang digunakan. Squidguard dapat melakukan pengalihan berdasarkan alamat IP (baik asal maupun tujuan), *blacklist* DNS, kelompok pengguna, dan waktu akses. Contoh sederhana konfigurasi Squidguard dapat dilihat pada Gambar 3. Pada konfigurasi Gambar 3, `dbhome` menunjukkan letak *blacklist* yang digunakan oleh Squidguard, dan `logdir` menunjukkan letak dokumen log Squidguard.

```
dbhome /usr/local/squidGuard/db/blacklists
logdir /usr/local/squidGuard/log

src admin {
     ip         1.2.3.4 1.2.3.5
     user       root foo bar
}

src foo-clients {
     ip 172.16.2.32-172.16.2.101
     ip 192.168.240.0/24
}

dest phishing {
     domainlist     dest/phishing/domains
     urllist        dest/phishing/urls
     redirect
          http://admin.loc/blocked?clientaddr=%a+cli
          entname=%n+clientuser=%i+clientgroup=%s+ta
          rgetgroup=%t+url=%u
}


acl {
     admin {
          pass any
     }

     foo-clients {
          pass !phishing any
     }

     default {
          pass none
          redirect
               http://admin.loc/blocked?clientaddr=%
               a+clientname=%n+clientuser=%i+clientg
               roup=%s+targetgroup=%t+url=%u
     }
}
```

Gambar 3. Contoh sederhana konfigurasi Squidguard

Pada konfigurasi Gambar 3, *blacklist* phishing yang digunakan terletak pada `/usr/local/squidGuard/db/blacklists/phishing/`. *Blacklist* tersebut terdiri dari dua buah dokumen: `domains` dan `urls`. Keduanya terlihat pada Gambar 4. Dokumen `domains` berisi daftar nama domain dan `urls` berisi

daftar URL. Keduanya dapat digunakan sebagai acuan pengalihan URL oleh Squidguard. Dokumen dengan ekstensi `.db` ditempatkan oleh Squidguard secara otomatis.



Gambar 4. Dokumen yang digunakan Squidguard sebagai blacklist

Pada Gambar 3, terlihat perintah `redirect` sebagai berikut.

```
redirect
      http://admin.loc/blocked?clientaddr=%a+clientname=
      %n+clientuser=%i+clientgroup=%s+targetgroup=%t+url
      =%u
```

Baris perintah ini berisi URL alihan yang akan digunakan oleh Squidguard jika Squidguard memutuskan akan mengalihkan akses laman tertentu. Squidguard juga memiliki enam variabel bawaan yang memudahkan kontrol pengalihan akses laman: `%a`, `%n`, `%i`, `%s`, `%t`, `%u`. Penjelasan variabel-variabel tersebut sebagai berikut.

a. `%a` memuat alamat IP asal dari permintaan,
b. `%n` memuat nama dari *client* pada akses bersangkutan (jika ada),
c. `%i` berisi login id Squid pada akses bersangkutan (jika ada),
d. `%s` memuat nama kelompok asal permintaan (contohnya `foo-clients`; `none` jika tidak ditemukan kecocokan),
e. `%t` menunjukkan kelompok tujuan akses (contohnya `phishing`; `default` jika tidak ada kecocokan), dan
f. `%u` menunjukkan URL dalam permintaan akses.

Squidguard dapat mengembalikan baris kosong (*empty line*) kepada Squid yang mengartikan penulisan ulang URL (`url_rewrite`) tidak perlu dilakukan atau mengembalikan string lain dengan format yang sama seperti string masukan untuk menggantikan URL awal.

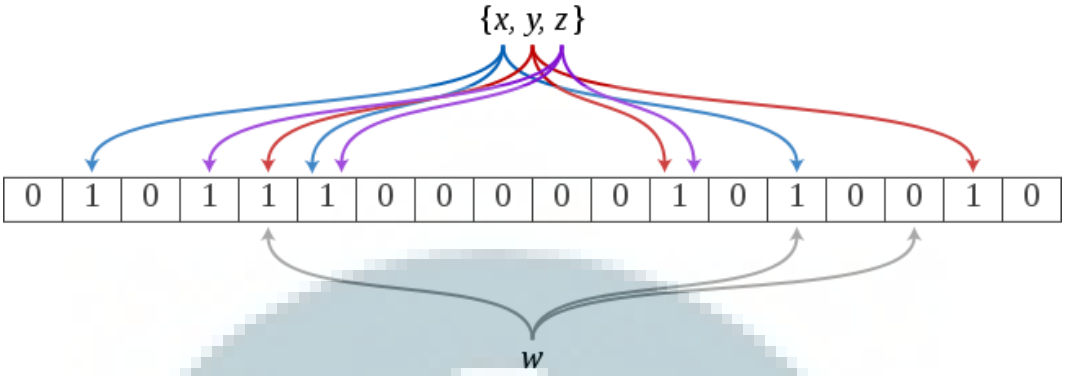
2.2 Bloom Filter

Bloom filter adalah sebuah struktur data probabilistik yang digunakan untuk menguji keberadaan sebuah elemen dalam himpunan. Bloom Filter dikemukakan pertama kali oleh Burton Howard Bloom pada tahun 1970 (Bloom, 1970) dan diusulkan oleh Marais dan Bharat untuk digunakan dalam jaringan (Marais & Bharat, 1997).

Bloom mengusulkan sebuah teknik yang menggunakan lebih sedikit memori dibandingkan dengan aplikasi yang menggunakan teknik hash bebas galat. Teknik bloom mengizinkan galat isbat (*false positive*) tapi tidak mengizinkan galat ingkar (*false negative*). Dengan kata lain, teknik Bloom dapat mengembalikan hasil *mungkin terdapat dalam himpunan* atau *pasti tidak terdapat dalam himpunan*. Penerapan klasik Bloom filter memungkinkan penambahan elemen tetapi tidak memungkinkan penghapusan elemen dari dalam filter. Semakin banyak elemen yang terdapat dalam filter, semakin tinggi kemungkinan Bloom filter mengembalikan hasil galat isbat. Bloom memberi contoh bahwa dengan menggunakan hanya 15% memori dari jumlah memori yang diperlukan teknik hash bebas galat, 85% akses memori dapat dihilangkan (Bloom, 1970).

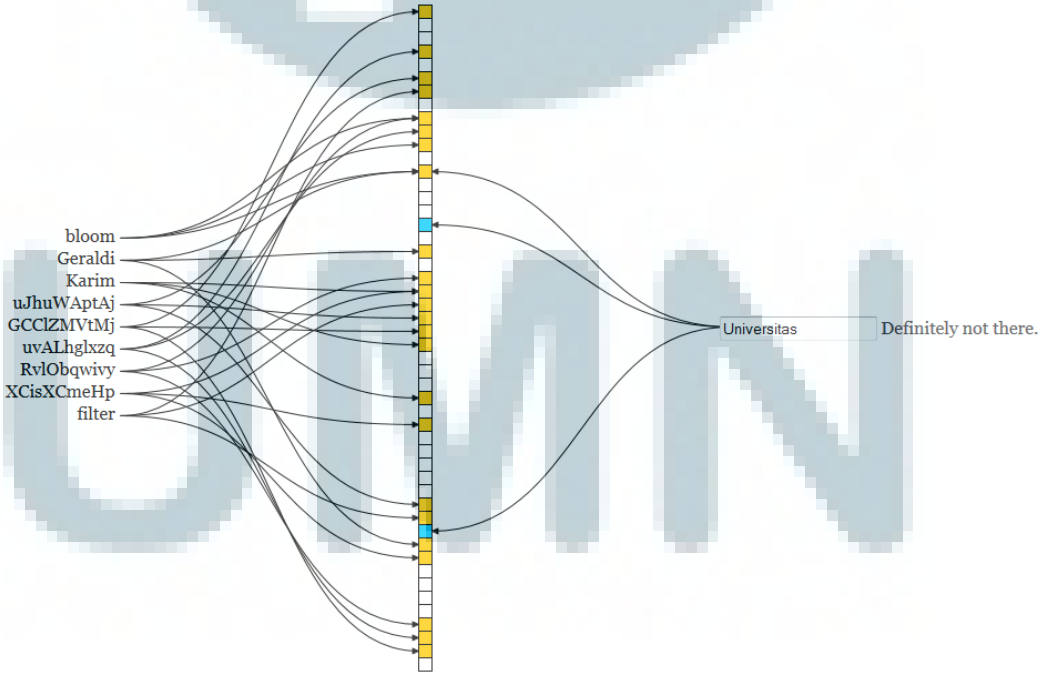
Bonomi menunjukkan bahwa dengan kemungkinan galat isbat sebesar 1%, kurang dari 10 bit memori diperlukan per elemen, terlepas dari jumlah atau besar elemen yang telah berada di dalam himpunan (Bonomi et al., 2006).

Dalam penerapannya, bloom filter menggunakan k buah fungsi hash dan memetakan hasilnya ke dalam larik dengan besar m, yang bernilai awal nol. Semua fungsi hash yang digunakan akan melakukan pemetaan pada rentang m terdistribusi random uniform. Sebuah Bloom Filter dengan nilai $m = 18$ dan $k = 3$ ditunjukkan oleh Gambar 5.

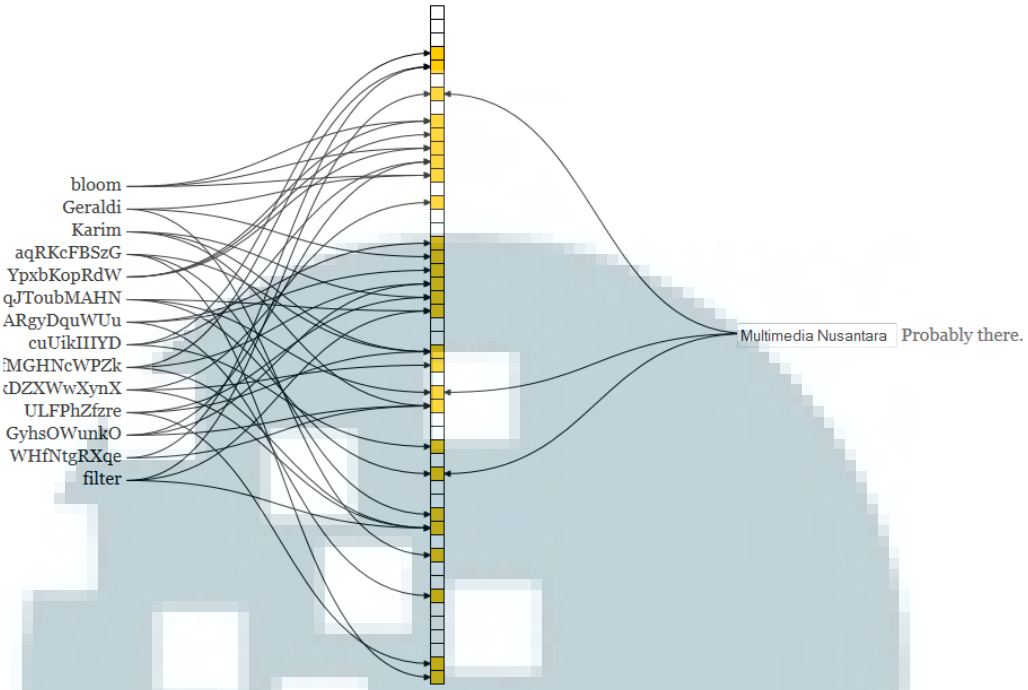


Gambar 5. Contoh Bloom filter dengan nilai m=18 dan k=3 (sumber: en.wikipedia.org)

Saat menambahkan elemen ke dalam Bloom filter, elemen tersebut akan menjadi masukan *k* buah fungsi hash untuk mendapatkan *k* buah posisi larik. Lalu semua posisi larik bersangkutan diberi nilai 1. Ketika pengujian elemen dilakukan, *k* buah posisi larik ditentukan berdasarkan elemen yang dicari. Jika salah satu dari bit pada posisi tersebut bernilai nol, elemen tersebut pasti tidak ada di dalam larik, seperti ditunjukkan oleh Gambar 6. Jika semua bit pada posisi tersebut bernilai 1, elemen tersebut berada di dalam larik atau ada elemen/elemen-elemen lain yang telah menyebabkan bit pada posisi tersebut bernilai 1; dalam kasus ini, pengujian memberi hasil galat isbat, seperti ditunjukkan oleh Gambar 7.



Gambar 6. *Query* Bloom filter dengan hasil ingkar (sumber: jasondavies.com)



Gambar 7. *Query* Bloom filter dengan hasil galat isbat (sumber: jasondavies.com)

Meski dapat menghasilkan galat isbat, Bloom Filter dapat mencatat sebuah elemen tanpa mempertimbangkan ukuran elemen tersebut. Dibandingkan dengan struktur data seperti larik, *binary search tree*, atau larik bertaut yang harus menyimpan elemen tersebut secara utuh, Bloom Filter menggunakan jauh lebih sedikit memori. Batas galat isbat dapat ditekan hingga 0.1% hanya dengan menggunakan 14.4 bit per elemen (Bonomi et al., 2006). Dibandingkan dengan struktur data yang telah disebutkan juga, waktu akses dan komputasi Bloom Filter tidak bertambah seiring bertambahnya elemen dalam filter; Bloom Filter tepat melakukan k komputasi, baik saat menambahkan maupun melakukan pengujian, menjadikan kompleksitasnya $O(k)$.

Besar kemungkinan galat isbat saat *query* Bloom Filter dapat dihitung menggunakan persamaan berikut.

$$p \approx \left(1 - e^{-\frac{kn}{m}}\right)^k \quad (1)$$

Dengan n menunjukkan jumlah elemen dalam filter, k menunjukkan jumlah fungsi hash independen, dan m menunjukkan jumlah bit dalam larik. Melalui persamaan di atas, kita dapat menyimpulan bahwa besar kemungkinan galat isbat akan

berkurang dengan menambahkan bit pada larik dan semakin tinggi saat lebih banyak elemen dimasukkan ke dalam filter.

Untuk nilai m dan n yang diketahui, nilai k yang membuat nilai p menjadi minimum ditunjukkan oleh persamaan berikut.

$$k = \left\lceil \frac{m}{n} \ln 2 \right\rceil \quad (2)$$

Persamaan (1) dan (2) menunjukkan bahwa untuk nilai p tertentu, jumlah bit m dalam larik dapat dihitung dengan persamaan sebagai berikut.

$$m = -\frac{n \ln p}{(\ln 2)^2} \quad (3)$$

Menggunakan persamaan (3), persamaan (2) dapat ditulis ulang sebagai berikut.

$$k = \left\lceil -\frac{\ln p}{\ln 2} \right\rceil \quad (4)$$

Pada aplikasinya, nilai n tidak bisa ditentukan pada awal penerapan. Yang dapat ditentukan adalah jumlah maksimum elemen N yang dapat ditampung filter dengan batas maksimum galat isbat p. Pemilihan k dan m dapat dilakukan dengan dua persamaan di atas dengan mengganti n dengan N. Dengan demikian, Bloom filter yang dibentuk mampu menampung hingga N elemen dengan kemungkinan galat isbat terbatas pada p.

Sebagai kesimpulan, untuk membentuk sebuah Bloom filter dengan kapasitas N dan batas atas galat isbat p, kita memilih nilai m yang bersesuaian dengan persamaan berikut.

$$m = -\frac{N \ln p}{(\ln 2)^2} \quad (5)$$

Dengan jumlah bit per elemen bpe didefinisikan dengan persamaan $bpe = m/N$, persamaan di atas dapat ditulis ulang sebagai berikut.

$$bpe = -\frac{\ln p}{(\ln 2)^2} \quad (6)$$

Kemudian kita memilih k buah fungsi hash $h_0, h_1, \dots, h_{k-1}$. Setiap fungsi hash dapat menghasilkan nilai $0, 1, \dots, m-1$, yang menunjukkan posisi bit dalam larik filter berukuran m. Untuk menambahkan elemen x ke dalam filter, bit pada larik pada posisi $h_0(x), h_1(x), \dots, h_{k-1}(x)$ diubah menjadi 1. Sebuah elemen y mungkin terdapat dalam filter jika seluruh bit pada posisi $h_0(y), h_1(y), \dots, h_{k-1}(y)$ bernilai 1.

2.3 Metode Hash

Sebuah metode hash memetakan data digital dengan ukuran sembarang ke dalam data digital dengan ukuran yang telah ditentukan sebelumnya, dengan perbedaan kecil pada data masukan membuat perbedaan besar dalam data keluaran (Carroll & Krotoski, 2014). Pemilihan metode hash yang baik juga menjadi salah satu penentu kualitas pembangunan Bloom filter.

Beberapa kriteria sebuah metode hash yang baik dijelaskan sebagai berikut.

a. Deterministik. Sebuah metode hash harus selalu menghasilkan hash yang sama ketika diberi masukan yang sama. Hal ini berarti sebuah metode hash tidak dapat bergantung pada pembangkit data acak, penunjuk waktu, ataupun pembacaan lokasi memori.
b. Uniform. Setiap nilai yang mungkin menjadi keluaran dari metode hash harus keluar dengan probabilitas yang sama; tidak ada keluaran yang lebih menonjol daripada yang lainnya. Dua buah nilai masukan berbeda yang menghasilkan keluaran hash yang sama dikenal dengan istilah tumbukan (*collision*). Untuk aplikasi tertentu, kemerataan dapat dikorbankan jika metode hash terlalu lamban.
c. *Range* yang terbatas. Keluaran sebuah metode hash harus dibatasi. Hal tersebut bisa dilakukan, salah satunya dengan jumlah bit. Misalnya, metode hash dengan keluaran 32-bit.
d. *Non-invertible*. Hal ini memiliki pengertian, jika diberikan sebuah keluaran metode hash $h(x)$, x tidak dapat dengan mudah ditemukan tanpa menghabiskan banyak waktu komputasi.

Salah satu metode hash yang paling dikenal adalah penggunaan *hash table* yang mempercepat perhitungan hash. Metode hash ini memercepat pencarian dalam

tabel atau basisdata dengan menghapus duplikasi dalam kelompok data yang besar. Metode-metode hash tertentu juga digunakan dalam kriptografi, terutama karena sulitnya merekontruksi kembali data jika hanya diketahui hasil hash dari data tersebut.

Secara umum, metode hash dapat dikelompokkan ke dalam 4 jenis: *cyclic redundancy check* (CRC), *checksum*, non-kriptografis, dan kriptografis.

Penyisipan metode hash ke dalam *proxy server*, salah satunya, mengutamakan kecepatan layanan. Pengamanan membuat metode hash kriptografis tidak dapat digunakan oleh *proxy server* dengan tujuan mencatat informasi akses laman karena berpotensi lebih tinggi mengakibatkan *bottleneck* pada server. Metode *checksum*, pada sisi lainnya, dibuat untuk mendeteksi galat yang pada transmisi atau penyimpanan data; sebagai tambahan kecil pada data besar.

CRC digunakan pada umumnya sebagai penanda galat yang ditambahkan mengikuti data yang dikirimkan lewat jaringan. Penanda galat dihitung menggunakan pembagian polinomial. Pada sisi penerima, penghitungan CRC akan kembali dilakukan untuk menguji integritas data. Perbaikan dapat dilakukan berdasarkan hasil pengujian pada sisi penerima. CRC dikembangkan pertama kali oleh W. Wesley Peterseon (Peterson & Brown, 1961).

Metode hash non-kriptografis adalah metode hash yang tidak termasuk dalam ketiga kategori lainnya. Metode hash non-kriptografis dapat dikembangkan menggunakan tabel hash, operasi xor, operasi tambah, operasi kali, operasi putar bit, dan sebagainya. Hal ini membuat kumpulan metode hash non-kriptografis yang paling beragam. Beberapa metode hash non-kriptografis yang dikenal luas antara lain Pearson (Pearson, 1990), Fowler–Noll–Vo (dikembangkan 1991), Jenkins (Jenkins, 1997), Murmurhash (Appleby, 2008), dan Cityhash (dikembangkan pada tahun 2010 oleh Google).