ネットワークトラフィック分析のための Iterative Bloom Filter の提案

松本義秀[†] 櫨山寛章[†] 門林雄基[†]

ネットワークトラフィック分析の利用例の１つである Hash-Based IP トレースバックにおいて，Bloom Filter (BF) を用いることで，入力パケットの Hash 値の重畳により記憶空間の節約と処理の高速化が可能である．しかし，BF に対し同一 Hash 値を複数登録した場合は重畳されるため，登録回数による分析は不可能になる．そこで，Hash 値の登録回数を記録できる Counting Bloom Filter (CBF)[2] の利用が考えられるが，登録回数の記憶領域を事前に確保する必要があり，偏りのあるトラフィックに対しては記憶効率が悪く，結果的に false positive の確率が上がることになる．

本論文は，BF のアルゴリズムを拡張した，Iterative Bloom Filter (IBF) を提案する．IBF の基本的な考え方は，入力 Hash 値が重複した場合には Hash 関数の個数を増加させることで，同じ BF の配列内に重複回数の情報を追加して埋め込むことにある．また，参照処理においては登録されている Hash 関数の個数から，重複数の推定値を算出する．IBF は，CBF に比べ正確な登録回数の算出できないが，登録回数の記憶領域の見積もりが不要であり，基本的にはカウンタのオーバフローが発生しないという特徴を持つ．本論文では，CBF，IBF の特性を比較し，IP トレースバックだけではなく，偏りのある入力値を持つアプリケーションにおいても適用が可能であることを示す．

Iterative Bloom Filter for Network Traffic Analysis

Yoshihide Matsumoto,[†] Hiroaki Hazeyama[†]
and Youki Kadobayashi[†]

Bloom Filter (BF) is used as one of the fundamental modules in several network processing algorithms and application such as route lookup, packet classification, per-flow state management and network monitoring. BF is a simple space-efficient randomized data structure for representing a set in order to support membership queries. However, BF generates false positives, and cannot count number of element. A counting Bloom filter (CBF) can count number of element and be changing dynamically via insertions and deletions, but CBF needs more space than BF.

We propose an alternative data structure of BF based on iterative hashing and overwriting a bit array. We called it Iterative Bloom Filter (IBF). The number of Hash function used IBF is changed dynamically, and set bit of hash to space as the number of collision about an element. The multiplicity of an element represented by IBF can be estimated probabilistically. IBF can realize the same functionality as a CBF, but uses less space and with no overflow. We describe the construction of IBFs, provide an mathematical analysis.

1. はじめに

ネットワーク分析の１つとして，DoS/DDoS 攻撃の発信元の探索方式の１つとして Hash-Based IP トレースバックがある．IP トレースバックは，ルータを経由する IP パケットの Hash 値を計算し一定期間蓄積する．被害ノードは，蓄積された Hash 値から，攻撃パケットの Hash 値を検索することで，攻撃経路を調べることが可能である．IP トレースバックの Hash 値の蓄積方法の１つとして，記憶空間の節約と処理の高速化が可能な Bloom Filter (BF) が用いられる．BF を用いることで，入力パケットのシグネチャである Hash 値を同じ記憶空間に重畳するため，記憶空間の効率化と処理の高速化が可能である．その反面，BF に対し同一 Hash 値を複数登録した場合は重畳されるため，Hash 値の登録回数の分析は不可能になる．DoS/DDoS 攻撃におけるトレースバックの場合，同一 Hash 値の登録回数は攻撃シグネチャのトラフィック量であり，インシデントかどうか重要な判断材料となる．

そこで，Hash 値の登録回数をカウント可能な

† 奈良先端科学技術大学院大学
Nara Institute of Science and Technology

Counting Bloom Filter (CBF)[2] の利用が考えられるが，登録回数を記録するためのメモリ領域を事前に見積もる必要があり，偏りのあるトラフィック程多くの記憶領域を必要とする．結果的に IP トレースバックの false positive の確率が上がることにつながる．

本論文は，BF のアルゴリズムを拡張した，Iterative Bloom Filter (IBF) を提案する．IBF の基本的な考え方は，入力 Hash 値が重複した場合には 新しい Hash 関数を繰り返し追加することで，同じ BF の配列内に重複回数の情報を追加して埋め込み，その登録回数の値を推定し算出する．

2 節では BF,CBF および IP トレースバックの問題点を示す．3 節では提案方式である IBF について示す．4 節では評価を行い，5 節で考察を行う．最後に 6 節でまとめを行う．

2. Bloom Filter

BF (Bloom Filter) とは，効率的な記憶空間と高い処理速度を持つデータ構造であり，Hash-Based IP Traceback[3] のパケットのシグネチャの格納方式としてだけではなく，TCP のコネクションの管理，ネットワークトラフィックの分析，単語のスペルチェック等に用いられている．BF は，記憶効率が良く，単純なビット演算のみで判定ができる反面，false positive と判定する可能性がある．また，削除処理を行わない限り，false negative は発生しない．

図 1 は，Hash 関数の個数 k が 4 の場合の Bit Vector と呼ばれるビット配列を表している．BF に登録する場合は，入力値 a_n から k 個の Hash 値を計算し，ビット配列に k ビットセットする．ビット配列のサイズは m であり，n 回の登録をすることを表している．BF から参照する場合は，入力 Hash 値のビットすべてが 1 の場合，登録済みと判断する．そうでない場合は未登録と判断する．よって，ビットが衝突した場合 false positive が発生する可能性があるが，m, n の値から記憶効率が最大になる k を算出することができる．(図 2)

2.1 false positive の確率

BF の配列のサイズを m とした場合，あるビットが 1 にセットされない確率は以下となる．

$$1 - \frac{1}{m}. \tag{1}$$

Hash 関数の数を k，入力値の数を n とした場合，あるビットが 0 である確率は以下の式に近似できる．

$$\left(1 - \frac{1}{m}\right)^{kn} \approx e^{-\frac{kn}{m}} \tag{2}$$

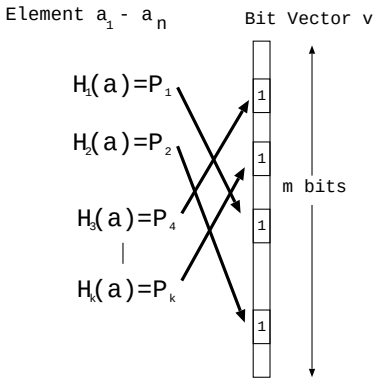

図 1 Bloom Filter with four hash function.

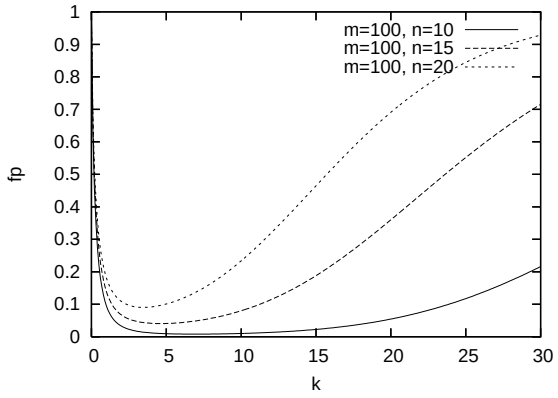

図 2 false positive probability

BF のあるビットが 1 である確率 p は以下で示される．

$$p = 1 - e^{-\frac{kn}{m}} \tag{3}$$

上記を用いて, Hash 関数で得られる k 個のビットすべてが 1 となる確率，つまり，false positive となる確率 f_{pBF} は以下の式で表される．

$$f_{pBF} \approx \left(1 - e^{-\frac{kn}{m}}\right)^k \tag{4}$$

記憶効率が最も良い条件は, $p = \frac{1}{2}$ の場合である．よって，式 3, 式 4 から以下が導かれる．

$$f_{pBF} \approx \left(\frac{1}{2}\right)^k \approx 0.6185^{m/n} \tag{5}$$

2.2 Counting Bloom Filter

BF は登録したビット列を重畳するため，入力値の削除や，入力値の登録回数のカウントができない．これを可能にするために，Counting Bloom Filter (CBF)[2] が提案されている．CBF は，BF の各ビットにカウン

1.	登録する Hash 値に対応する，更新前の BF のビットが 1 かどうか判定する．
2.	ビットが 1 の場合は，k (Hash 関数) の数を+1 し，0 のビットが見つかるまで繰り返す．
3.	ビットが 0 の場合は，1 をセットし終了する．

表 1 提案する IBF の insert 処理

1.	参照する Hash 値に対応する BF の Hash 値のビットが 1 かどうか判定する．
2.	ビットが 1 の場合，k (Hash 関数の個数) の数を +1 し，0 のビットが見つかるまで繰り返す．
3.	繰り返し回数を j と定義し，j を元に登録回数を推定する．

表 2 提案する IBF の query 処理

ト用のビットを追加し，登録時にインクリメント，削除時にデクリメントする．ただし，配列ビットごとにカウント用の領域が必要であり，上限を超えた場合はオーバーフローが発生する．入力値の削除操作が可能なため，蓄積型のアプリケーションに利用される．また，入力値の偏りが少ない程，カウンタがオーバーフローする可能性は少なくなる．

2.3 IP トレースバックにおける トレードオフ

BF は，通常削除処理を行わないため false negative (偽陰性) が発生することはない．しかし，IP トレースバックのようなリアルタイムのトラフィックの保存や分析を行う場合には，複数の bloom filter を用いて，一定期間ごとにローテーションと削除を行う．そのため，保存期間を超えたデータに関しては false negative が発生するといえる．メモリサイズ一定の条件で保存期間を長くするには，登録回数 n の値を増やす必要があり，結果的に false positive の確率は増加する．

IP トレースバックでは，メモリサイズと保存期間を一定にした場合，単位時間あたりの入力パケット数が変化することで false positive の確率が変化する．

2.4 IP トレースバックにおける Bloom Filter の問題点

AS 間を連携したインタードメイントレースバック[5] において，攻撃パケットの通過有無の誤判定は，他組織を被疑してしまうことになるため，信頼関係の問題や対策の後戻りが発生することになる．

そのため，インシデントかどうかの判断材料として，攻撃パケットの通過の有無だけでなく，攻撃パケットのトラフィック量の情報を加えることで，IP トレースバック全体の精度が向上する．攻撃パケットのトラフィック量，つまり，Hash 値の登録回数の記録が可能な CBF の適用が考えられるが，カウント専用の記憶領域が必要であり，DoS/DDoS 攻撃によりトラフィックの変化が激しい場合オーバフローする可能性がある．

3. 提案方式:Iterative Bloom Filter (IBF)

表 1,2 は, IBF の登録処理，参照処理を表し, 図 3 は，BF, CBF, IBF の登録処理を表している．CBF では，入力 Hash 値が重複する場合，カウンタをインクリメントし，登録回数を記録する．IBF では，入力

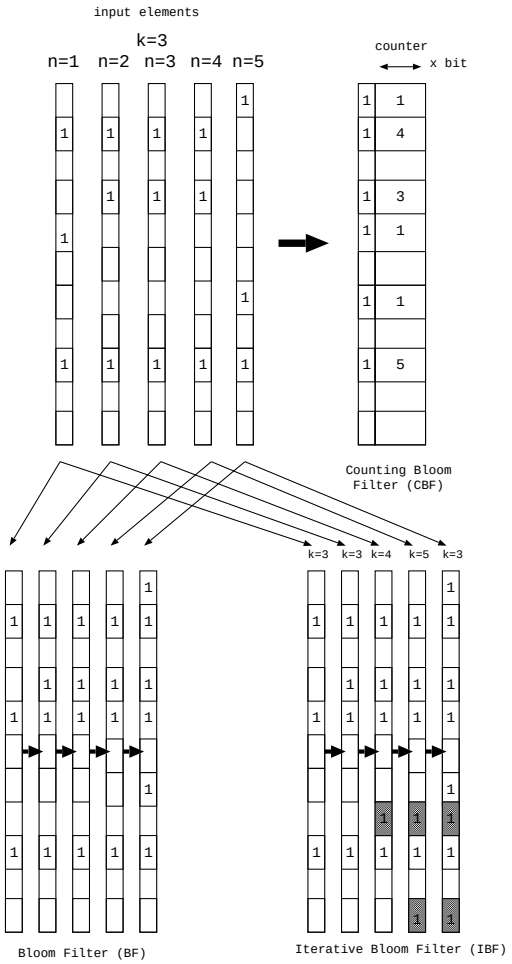

図 3 BF, CBF, IBF の insert 処理

Hash 値が重複する場合，Hash 関数の数 k を増やしビットの追加を行う．これにより，配列内に重複回数の情報を 1 ビット紛れ込ませ，特別な領域を使用せずに空間の利用効率を高めている．登録数を調べるには，登録方法と同じように Hash 関数の数 k から登録回数を推定する．

3.1 異なり数, 異なり率の定義

IBF は，入力 Hash 値が重複するごとに 1 ビットセットされるため，空間の利用率は重複数により上昇する．そのため，入力 Hash 値の重複の度合いを表す値として，異なり数を定義する．異なり数は入力値の

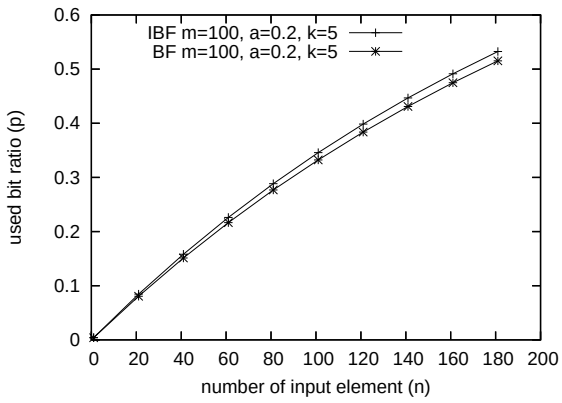

図 4 入力数 n に対する ビット配列の使用率 p

異なっている値の個数つまりユニークな個数で表される．例えば，入力値が {1,1,2,2,3,3} の場合は，ユニークな値は {1,2,3} 3 値となるため異なり数は 3 となる．また，異なり率 = 異なり数/総数 と定義する．上記場合，異なり率 $\alpha = 3/6 = 0.5$ となる．異なり率が大きい程，重複が少ないということになる．

3.2 入力値の登録回数の推定

IBF では入力 Hash 値の登録時，k 個の Hash 値が 1 の場合は Hash 関数 k を増やし，0 ビットが出現するまでスキップし，見つかった 0 のビットを 1 に変更する．入力 Hash 値の登録回数を算出するには，初めに ビット値 0 が出現するまでの繰り返し回数 (ビット値 1 の連続数) j を計算する．実際の登録回数 i は，登録時に衝突した可能性があるため j より少なくなると考えられる．そのため，以下の様に登録回数を補正する必要がある．

図 4 は，BF のあるビットが 1 である確率 p を入力数ごとに表示したものである．BF において空間効率が最大となるのは $p = 0.5$ の場合である．$p = 0.5$ になるまで単調増加であり，入力数が少ない初期であるほど衝突が少ない．

そこで，入力 Hash 値の異なり率を α と定義すると，BF におけるビットが衝突しない確率の繰り返し回数は kn は $\alpha kn+(1-\alpha)n$ と表される．よって，BF のあるビットが 1 である確率 p (式 3) より， IBF のカウント用ビットの衝突確率は以下となる．

$$p_{IBF} = \left(1 - e^{-\frac{\alpha kn+(1-\alpha)n}{m}}\right) \tag{6}$$

1 回の登録において，使用するハッシュ関数の個数 k に対応するビット値 1 の連続数はスキップするビット数 + 1 増加する．衝突確率を p_{IBF} とした場合，単位ビットあたりのスキップするビット数は，衝突率／未衝突率であり，$\frac{p_{IBF}}{1-p_{IBF}}$ となる．入力値の重複した値の集合を N，要素 n の衝突率を p_n とすると，BF へすべてのデータ登録後，ハッシュ関数 k に対応するビット値 1 の連続数 j は以下で表される．

$$j = \sum_{n \in N}^{i} \frac{p_n}{1-p_n} + 1 \tag{7}$$

重複した入力値が均等に到着する場合，IBF のカウント用ビットの衝突確率は，平均値 $p_n/2$ を用いて近似すると以下となる．

$$j' = \frac{p_n/2}{1-p_n/2} \cdot i + i \tag{8}$$

i を元に展開し，登録数 i の推定値 E(i) は以下の式で表される．

$$\begin{aligned} E(i) &= \left(1 - \frac{p_n}{2}\right) \cdot j' \\ &= \frac{1}{2}\left(1 + e^{-\frac{\alpha kn+(1-\alpha)n}{m}}\right) \cdot j' \end{aligned} \tag{9}$$

ただし，上記平均値を用いた登録数の推定方法の場合，重複した入力値の到着に偏りがある場合，誤差が大きくなる可能性がある．

4. 評　　価

4.1 空間量, 計算量の比較

入力値 I のカウント数を C_I と定義し，表 3 に，BF,CBF,IBF の空間量と計算量の比較を示す．

空間量については，CBF はカウント用のビットサイズ (カウント数 C_I の最大値の対数) と BF のサイズ m の乗算であるのに対し，IBF はカウント数 C_I の総和と BF のサイズ m の加算で表される．

計算量については，IBF は BF と同様に ビット演算 AND OR の繰り返しであり，特殊な演算は発生しない．しかし，IBF は，CBF に比べ登録処理，参照処理でカウント数 C_I に比例した計算が発生する．

	Space	Insert
BF	m	$O(k)$
CBF	$m \cdot \log(max(C_I))$	$O(k)$
IBF	$m + sum(C_I)$	$O(k + C_I)$
	Query	Query count
BF	$O(k)$	-
CBF	$O(k)$	$O(1)$
IBF	$O(k)$	$O(k + C_I)$

表 3 BF, CBF, IBF の空間量と計算量

4.2 入力値の偏りパターンの定義

CBF は，入力値のカウント数の最大値によりカウン

ト用の記憶領域が決定される．そのため，重複した入力値の偏りが大きい程，記憶領域を必要とする．例えば，入力値が {1,2,2,2,2,3} {1,1,2,2,3,3} の場合，異なり率 0.5 であるが，個数の最大値は 4 と 2 で確保すべきカウント用記憶領域は異なる．

入力値の集合を I, 入力値それぞれのカウント数を C_I の場合，$\alpha = sum(C_I)$ で表わすことができる．また，評価に使用する入力値の偏りのパターンとして表 4,5 を定義する．

Case 1	偏り 小	重複した入力値はすべて均等に分散.
Case 2	偏り 中	重複した入力値の半数を同一値, 残り半数を均等に分散.
Case 3	偏り 大	重複した入力値すべて，同一値.

表 4 偏りパターン

	カウンタ最大値 $max(C_I)$	カウンタビット数 $c = log_2(max(C_I))$
Case 1	$\frac{(1-\alpha)n}{\alpha n}$	$\log_2 \frac{(1-\alpha)}{\alpha}$
Case 2	$\frac{(1-\alpha)n}{2}$	$\log_2 \frac{(1-\alpha)n}{2}$
Case 3	$(1-\alpha)n$	$\log_2 (1-\alpha)n$

表 5 偏りパターンごとの CBF カウンタ用記憶領域

4.3 false positive の比較

4.3.1 IBF の false positive

入力値の異なり率 α，Hash 関数の個数 k における IBF がセットするビット数は，重複しない入力個数 × Hash 個数 + 重複する入力個数 ×1[bit] つまり $\alpha nk + (1-\alpha)n$ と表される．

よって，式 4 を元に IBF の false positive の確率 f_{pIBF} は以下となる．

$$f_{pIBF} \approx \left(1 - e^{-\frac{\alpha nk + (1-\alpha)n}{m}}\right)^k. \qquad (10)$$

4.3.2 CBF の false positive

CBF の入力値のカウント数の記憶に必要なビット数が c の場合，メモリサイズを m とすると，CBF が実際にカウント使用できるメモリサイズは m/c となる．よって，CBF の false positive の確率 f_{pCBF} は以下となる．

$$f_{pCBF} \approx \left(1 - e^{-\frac{knc}{m}}\right)^k. \qquad (11)$$

4.3.3 異なり率，偏りの影響

図 5 は，BF, IBF, CBF の異なり率 α に対する false positive の確率を表している．

CBF は，異なり率と偏りの影響により，カウンタのオーバフローが発生する．オーバーフローを発生させないためにカウント用のメモリを多く確保した場合，

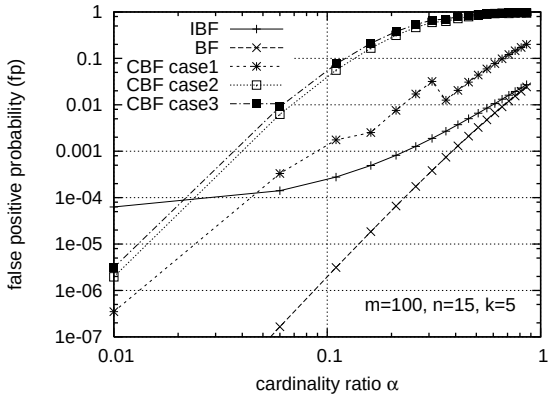

図 5 異なり率 (α) に対する false positive の確率 f_p

結果的に false positive 確率が上昇することにつながる．偏りパターンによって false positive の確率に約 10 倍前後の差があり，偏りの予測を間違うと大きな影響を受けると考えられる．

IBF は，入力値の偏りによる影響を受けないが，異なり率の影響を受ける．IBF は，異なり率が大きい場合 CBF よりも false positive は低い確率となり，異なり率が小さい場合 CBF よりも高い確率となる．その理由は以下である．

異なり率 α 入力数 n の場合，重複数の合計は $(1-\alpha)n$ と表される．異なり率が少ない場合，入力値の重複合計数のビットが 1 に更新されるため，カウント数の情報量は重複合計数に依存する．IBF では，カウント数の情報量は $n$bit であるのに対し，CBF ではカウンタ用のビットが独立しているためカウント数の情報量は $log(n)$bit で表されるためである．

図 6 は，異なり率 $\alpha = 0.5$ における入力数 n が増加した場合の BF, IBF, CBF の false positive を示している．k= 0.031 の場合に空間の利用効率が最大となるが，トラフィックの増加により，BF の能力を超えた入力を行う場合，case 2, case 3 が最も傾きが急であり急激に false positive の確率は増加する．

5. 考　　察

5.1 IBF の偏りのある入力データに対する優位性

CBF は事前にカウント用の記憶領域の確保が必要である．そのため，偏りのある入力値は容易にカウンタのオーバーフローが発生する．IBF は，カウンタ専用の記憶領域は持たず，BF の隙間にカウント情報を追加するため，オーバーフローは発生せず，入力値の偏りによる影響を受けない．

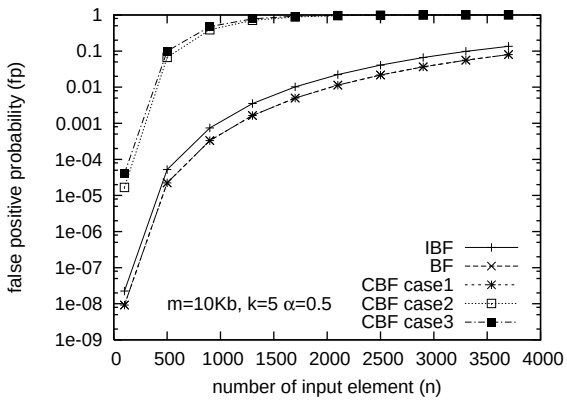

図 6 入力数に対する false positive の確率 f_p

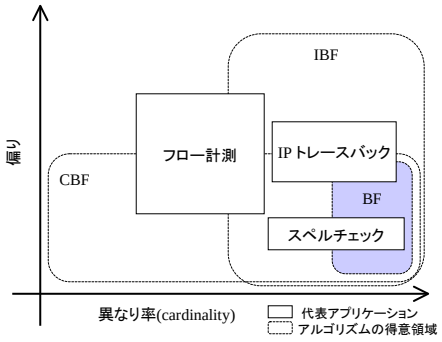

図 7 IBF のアプリケーションに対する適用領域

しかし，重複した入力値に対しても必ず 1 ビットセットするため，false positive が上昇する．そのため，カウンタ合計数の上限を設け，空間の利用効率が最大となった時点でカウンと機能を停止させる必要がある．

5.2 IBF の予測困難な入力データに対する優位性

上記で述べたように，CBF は事前にカウント用の記憶領域の確保が必要である．そのため，入力データ数，入力データの偏りの推定値から，適切なカウント用の領域を事前に見積もる必要がある．IBF は特別なカウント用の記憶領域を持たないため，予測の必要がない．

5.3 IBF の欠点

IBF はカウンタ用ビットと Hash 値のビットを区別していないため，強制的にビットを削除した場合，false negative が発生する可能性ある．そのため，IBF において削除処理が現実的ではない．

図 5 より，異なり率が小さい場合において，IBF は CBF に比べ false positive が高く，つまり，空間効率が悪くなる．

5.4 dlCBF との比較

dlCBF[9] は CBF を改良し CBF よりも空間効率を高めた方式である．dlCBF の基本的な考え方は，BF を d 個のテーブルに分割し，テーブルのビット使用率が均等になるように insert と query を行う．そのため，毎回 d 回の参照処理が必要とされる．dlCBF は CBF と同様に事前にカウンタ用のビットを見積もる必要があるため，入力データの予測が必要である．そのため，dlCBF はオーバーフローする可能性があるが，分割数 d に依存し発生確率が変化すると考えられる．

5.5 IBF のアプリケーションへの適用領域

以上より，IBF は異なり率が大きく，偏りのあるデータに対し適用の優位性があると考えられる．(図 7)

特に，フロー計測等のアプリケーションでは，フローごとの通過数を計測するため，異なり率が小さい場合が多いと考えられる．IP トレースバック等のアプリケーションでは，IP パケットをマスク処理後 Hash 値を計算するため，異なり率は高い．また，DoS 攻撃等予測できないトラフィックにより偏りが大きくなると考えられる．

よって，重複度の大きいフローの計測等においては CBF の方が適しているといえる．また，それ以外の重複度の少ない IP トレースバックようなアプリケーションにおいては IBF が適しているといえる．

5.6 IP トレースバックへ IBF の適用

SPIE[3] の研究では，28Byte の MASK 値によるコリジョンは WAN で 0.00092%, LAN で 0.139%とされ，異なり率が高い．また，DoS/DDoS が発生した場合，単位時間中のパケット重複率や特定 Hash 値の偏りの発生が考えられる．よって，IP トレースバックにおいては，異なり率が高くトラフィックが偏る可能性があるため CBF に比べ IBF が適していると考えられる．

5.6.1 登録値の削除処理

IP トレースバックの実装では複数の bloom filter を用意し，一定期間ごとにローテーションと BF 単位で削除を行う実装が考えられる．そのため，CBR でのみ実現可能な登録値の削除処理は必須でないと考えられる．

5.6.2 カウント数の推定誤差

図 4 より，入力数の増加に伴い，ビット使用率が増加する．IP トレースバックにおいて複数の BF を一

定期間ごとに切り替える場合，DoS 攻撃等によるのトラフィックの変動時により，BF に対する入力値の到着の最初と最後で重複率が変化する可能性がある．その場合，式 7 では平均値を元にカウント数を計算しているため，誤差が大きくなる可能性がある．

5.6.3 合計カウント数の上限

BF は重複した入力値の場合には，重畳されるため false positive ratio は変化しない．しかし，IBF は重複した入力値に対し必ず 1 ビットセットするため，false positive が上昇する．そのため，IP トレースバックへ適用する際には，総カウント数の上限を設け，空間の利用効率が最大となった時点でカウント処理を停止させる必要がある．

6. ま と め

本論文では，トラフィック分析のための IBF を提案した．IBF は，CBF と同様に BF では実現することのできない登録回数のカウントが可能である．IBF は BF と同じ記憶領域にビットを追加していくため，CBF のようにカウント用の記憶領域を事前に見積もる必要がなく，偏りのある入力値に対してもオーバーフローが発生しないという特徴を持つ．

しかしその反面，入力データによっては登録回数の推定により誤差が生じる場合がある．また，IBF はカウンタのオーバーフローは発生しないが，入力値の重複数の増加に伴い false positive が増加するため，実際の利用時においては登録するカウンタ総数に上限を設ける等の注意が必要である．

IBF は, IP トレースバックだけでなく入力値の偏りや異なり率が大きいアプリケーションへも適用が可能であると考えられる．

7. 今後の課題

実環境における IBF の精度を明らかにするため，実際のネットワークトラフィックを用いたシミュレーションと実装による評価を行う予定である．

本研究は情報通信研究機構の委託研究「インターネットにおけるトレースバック技術に関する研究開発」により，実施した．

参 考 文 献

1) F. Bonomi, M. Mitzenmacher, R. Panigraphy, S. Singh, and G. Varghese, “ Beyond Bloom filters: From approximate membership checks to approximate state machines, ” in Proc. ACM SIGCOMM, Sept. 2006.
2) L. Fan, P. Cao, J. Almeida, and A. Z. Broder, “ Summary cache: a scalable wide-area Web cache sharing protocol, ” IEEE/ACM Trans. on Networking,vol. 8, no. 3, pp. 281-293, 2000.
3) A. Snoeren, C. Partridge, L. Sanchez, W. Strayer, C. Jones and F. Tchakountio, ”Hash-Based IP Traceback,” Proc. ACM/SIGCOMM, to appear, August 2001.
4) L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: a scalable wide-area Web cache sharing protocol. IEEE/ACM Transactions on Networking, 8(3):281-293, 2000.
5) Hiroaki Hazeyama and Youki Kadobayashi and Daisuke Miyamoto and Masafumi Oe, “ An Autonomous Architecture for Inter-Domain Traceback across the Borders of Network Operation ”Proceedings of 11th IEEE Symposium on Computers and Communications (ISCC ’06), pp. 378–385,June,2006.
6) B. Chazelle, J. Kilian, R. Rubinfeld, and A. Tal. The Bloomierfilter: an efficient data structure for static support lookup tables. In Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp 30-39, 2004.
7) B. Donnet, B. Baynat, and T. Friedman, “ Retouched Bloom filters: Allowing networked applications to trade off selected false positives against false negatives, ” arXiv, cs.NI 0607038, Jul. 2006.
8) R. P. Laufer, P. B. Velloso, and O. C. M. B. Duarte,“ Generalized Bloom filters, ”Electrical Engineering Program, COPPE/UFRJ, Tech. Rep. GTA-05-43, Sept. 2005.
9) F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh,and G. Varghese, “ An improved construction for counting Bloom filters, ” in Proc. ESA, Sept. 2006.