IDSでの利用に適したパターンマッチアルゴリズム
A Pattern Matching Algoritym Suitable for IDSs

柳瀬 葵, 今泉 貴史†

Aoi YANASE, Takashi IMAIZUMI†

imaizumi_takashi@faculty.chiba-u.jp

千葉大学 統合情報センター†
Institute of Management and Information Technologies, Chiba University†

概要

ネットワークが広く使われるようになるとともに、ネットワークを介した脅威も増加し、対策としてIDSやIPSが広く用いられている。しかし、トラフィックが増えるにつれ、IDSなどで行うパターンマッチの負荷が問題となってきている。

本稿では、パターンマッチに用いるパターンに制約を加え、さらに、パターンマッチの際に少量の誤りを許容することにより、必要とするメモリ空間を抑え、高速にパターンマッチを実行するアルゴリズムについて述べる。このアルゴリズムを用いることで、誤りが許容範囲に収まれば、IDSなどにおいてもより広帯域のネットワークへ適用することが可能となる。

キーワード

侵入検知システム, パターンマッチアルゴリズム

1 はじめに

近年、インターネットを筆頭としてネットワークの利用は拡大の一途をたどっている。ネットワークを利用する際に用いる装置も、ほぼ計算機のみであった時代から、PDA、携帯電話、タブレットなど、多様な端末が用いられるようになり、さらには一般の家電製品の中にもネットワーク利用を前提としたものが出現するようになった。ネットワークが広く使われるようになるにつれて、悪意を持つユーザも増加してきた。さらに、マルウェアを簡単に作成できるツールキットなども配布されており、脅威の増加はとどまるところを知らない。ネットワーク上に存在する様々な脅威から組織のネットワーク全体を守る方法として、ファイアウォールに代表されるセキュリティ機器を組織ネットワークの入り口に配置し、組織に流入する（もしくは組織から流出する）トラフィックを監視することで、セキュリティを維持しようとするものがある。この際に用いるセキュリティ機器の中でも、トラフィックのコンテンツにまで注目したIDSやIPSでは、あらかじめ準備されているシグネチャとのマッチングを取ることでトラフィックの安全性を判断することが多い。しかし、ネットワーク利用の増大にともなうトラフィックの増加につれ、パターンマッチの負荷が問題となってきている。

IDSにおけるパターンマッチの負荷を下げる試みとしては、単に機器を構成する素子等の高性能化を図るのみならず、複数のIDSを並列に配置することで対応しようとするもの [1, 2]、GPUを利用して高速化を図るもの [3]、さらには、FPGAを用いることで処理を高速化するもの [4, 5] など、数多くの研究が行われている。アルゴリズム自体を研究するものは、TCAMを用いるものなどいくつかが報告されてはいるものの、基本的には複数文字列マッチングアルゴリズムに関するものであり、パターンマッチアルゴリズムに関するものではない [6, 7]。汎用のパターンマッチアルゴリズムとしては、正規表現からDFAを作成し、1文字ずつ入力を読みながら状態遷移機械で処理をする方法が基本となる。

この手法はかなり高速に動作する。文字列マッチングの場合には、単一の文字列に対する処理と複数の文字列に対する処理が基本的に異なるため、この部分を用いて高速化を図ることが可能であった。しかしパターンマッチの場合には、正規表現の 1 つに「選択」演算があり、基本的に単一パターンの検索と複数パターンの検索に違いはない。そのため、並列化による恩恵も簡単には受けられず、アルゴリズム自体の研究は進んでいない。

複数文字列のマッチングアルゴリズムである SOG-like 法 [8] は、単一文字列マッチングアルゴリズムを並列化することで複数文字列マッチングの性能を上げようとするものである。これは、アルゴリズムの適用範囲を IDS に限定することで単に並列化する以上の効果を得るものである。戦略としては、文字列マッチングを検索する際に若干誤りが生じることを許し、複数の文字列の検査を同時に行えるようにする。これにより、厳密な一致を調べることなく、マッチする可能性の高いものを検出することで処理を高速化している。しかし、このアルゴリズムもそのままパターンマッチに応用できるものではない。

本研究では、パターンマッチの際に誤りを許すことによってパターンマッチアルゴリズムの効率化を実現する multiple-ESA 法を提案する。ここで目指す効率化は、パターンマッチを複数組み合わせることで実現するのは難しい。そこで、パターンマッチに利用できるパターンに制限を設けることで、効率化を実現する。パターンを制約することで、パターンマッチの際に使用するメモリ空間を抑える。さらにそれを誤りを許容しながら複数のパターンに同時に適用できるようにすることで、目的を実現する。本論文で提案するアルゴリズムを用いることで、誤りが許容されれば、メモリ空間に制限のある IDS アプライアンスなどにおいても、より広帯域のネットワークへと適用することが可能となる。

2 関連研究

高速な文字列マッチングアルゴリズムとして Shift-And 法が提案されている [9]。この方法は、文字列パターンの各文字に関して現在の入力でどこまで一致しているのかを示す NFA をビットパターンで表現することで高速化を実現している。本論文では、このアルゴリズムを拡張して高速なパターンマッチアルゴリズムを構築するが、Shift-And 法を拡張する様々なアルゴリズムがすでに提案されている。それらの関係を、図 1 に示す。

Shift-Or 法 [10] は、Shift-And 法のビット表現を変更することで、処理を簡便にしてさらに高速化したものである。SOG 法は、Shift-Or 法の入力アルファベットを n-Gram を用いることで拡張し、処理の高速化とパターン長の拡大を実現している。これらのアルゴリズムは、対象とするのは単一の文字列となっている。照合誤りを許すことで、SOG 法を複数文字列に対応させた SOG-like 法 [8] もあるが、対象とするのはやはり文字列であり、正規表現には対応できない。

Shift-And 法を正規表現に対応させる試みも行われている。Extended Shift-And 法 [11] は、正規表現の一部の機能を持つ拡張文字列のマッチング処理を Shift-And 法を拡張して実現したアルゴリズムである。さらに、Extended Shift-And 法を拡張する探索アルゴリズム（図 1 では FBPM と表現）も提案されている [12]。この手法では、拡張文字列に対してビット並列オペレーションとして Scatter, Gather, Propagate を追加している。また、NFA を階層的にモジュール分割することでより長い拡張文字列を扱うことが可能な RunNFA も提案されている [13]。しかしこれらの拡張手法も、拡張文字列の能力を上げてはいるものの複数の拡張文字列をまとめて記述できるほどにはなっておらず、単一の拡張文字列が対象となる。

これらに対して、本論文で提案する multiple-ESA 法は、基本的な拡張文字列を対象としているが、複数の拡張文字列をまとめて照合することが可能となっている点が異なる。拡張文字列には選択演算が存在しないため、単一の拡張文字列の処理と複数の拡張文字列の処理は本質的に異なる。

3 multiple-ESA 法

本論文で提案するパターンマッチアルゴリズム multiple-ESA 法は、使用可能なパターンを正規表現よりも制限した拡張文字列とする。拡張文字列には正規表現にある「選択」機能は存在しないため、これを誤りを許しながら並列化することで複数の拡張文字列をまとめて検査できるようにする。

3.1 拡張文字列

拡張文字列は、Extended Shift-And 法が対象とする正規表現よりも制限された言語クラスである [9]。これを使うと、正規表現よりは制限されるものの、単なる文字列よりも多彩なパターン表現が可能になる。

拡張文字列の定義を次に示す。1.〜3. は文字種の定義、4.〜6. は文字種の量化についての演算の定義である。また、L は拡張文字列があらわす言語の集合を示す関数である。

1. 定数文字
   $a \in \Sigma.\ L(a) = \{a\}$

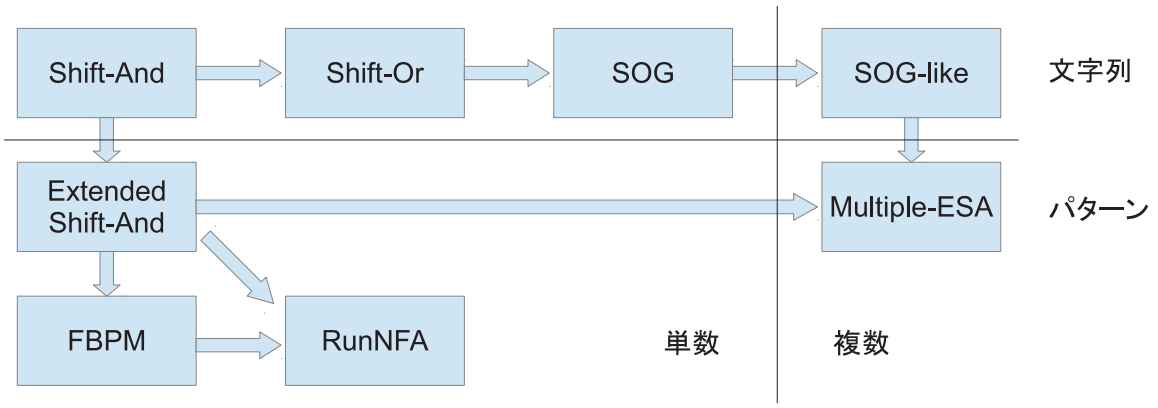

図- 1: 各探索アルゴリズムの関連

2. ワイルドカード文字
   “.”. $L(.) = \Sigma$

3. 文字クラス
   $\beta = [abc...] \subseteq \Sigma$. $L(\beta) = \{a\} \cup \{b\} \cup \{c\} \cup \cdots$

4. オプション文字
   $\beta? = (\beta|\epsilon)$. $L(\beta?) = L(\beta) \cup \{\epsilon\}$.

5. 0 個以上の非限定繰り返し
   β^*. $L(\beta^*) = L(\beta)^*$

6. 1 個以上の非限定繰り返し
   β^+. $L(\beta^+) = L(\beta)L(\beta)^*$

正規表現と異なる点は、繰り返し (閉包)、選言 (集合和) の演算の扱いである。正規表現では、3 つの基底演算 (選言・連接・繰り返し) が、文字種・文字列の両方に適用できる。本論文で扱う拡張文字列では、文字種に対しては 3 つの基底演算全てを適用できるものの、文字列に対しては連接演算のみが許される。

3.2 拡張文字列のマッチングアルゴリズム

Extended Shift-And 法 [11] について簡単に述べる。このアルゴリズムは、前準備と照合処理の 2 段階に分かれる。

3.2.1 前処理

Extended Shift-And 法では、パターン P に対して次の 2 種類の長さを定義している。

- 虚パターン長: 各演算記号 (?+*) を含み、文字クラスを 1 文字とカウントしない場合のパターン長
- 実パターン長: 演算記号を含まず、文字クラス全体を 1 文字とカウントした場合のパターン長

以降の説明で用いる L は、実パターン長を表している。照合の前処理として、2 個の幅 L のビットテーブル、3 個の L ビットのビットマスクを計算する。

計算するビットテーブルは次の 2 つである。

- 文字テーブル B: アルファベット Σ に含まれる全ての文字について、どの文字が、どの位置に現れるかを表す。$0 \leq i \leq L-1$ において、$B[c]$ & $(1 \ll i) \neq 0$ なら、文字 $c \in \Sigma$ がパターンの i 番目に現れることを示す。
- 繰り返し文字テーブル S: パターン中に現れる繰り返し文字について、どの文字が、どの位置に現れるかを表す。$0 \leq i \leq L-1$ において、$S[c]$ & $(1 \ll i) \neq 0$ なら、文字 $c \in \Sigma$ がパターンの i 番目で繰り返されることを示す。

一方、ビットマスクは、オプション文字の連続する範囲 (ブロック) を表し、次の 3 つがある。

- $Emask$: $0 \leq i \leq L-1$ において、$Emask$ & $(1 \ll i) \neq 0$ なら、i 番目の文字がオプション文字であることを示す。
- $Ebeg$: $0 \leq i \leq L-1$ において、$Ebeg$ & $(1 \ll i) \neq 0$ なら、$i+1$ 番目の文字において 1 つ以上連続するオプション文字の列が始まることを示す。
- $Eend$: $0 \leq i \leq L-1$ において、$Eend$ & $(1 \ll i) \neq 0$ なら、i 番目の文字において 1 つ以上連続するオプション文字の列が途切れることを示す。

3.2.2 照合処理

照合処理では、実行時の状態保存用として、L ビットの状態マスク D、Df を使用する。照合の手順は次の通りである。ここで、pos は現在照合している入力テキストの文字の位置、$T[pos]$ は現在照合している入力テキストの文字を表す。

1. 実行時状態マスク D (L ビット) を、0 で初期化
2. $0 \leq pos \leq n$ (テキスト長) -1 の間、以下を繰り返す
   (a) $D1 \leftarrow \{(D \ll 1) \mid 1\}$ & $B[T[pos]]$
   (b) $D2 \leftarrow D$ & $S[T[pos]]$
   (c) $D \leftarrow D1 \mid D2$
   (d) $Df \leftarrow D \mid Eend$
   (e) $D \leftarrow D \mid \{Emask$ & $(\sim (Df - Ebeg) \bigoplus Df)\}$
   (f) もし、D & $2^{pos} = 1$ ならマッチング成功
3. これまでに成功していなければマッチング失敗

ここで、$\bigoplus$ は排他的論理和、$-$ は算術減算を示す。

3.3 複数の拡張文字列への対応

Extended Shift-And 法は 1 度の走査で単一の拡張文字列のマッチングしか検査できない。multiple-ESA 法では、複数の拡張文字列をまとめてマッチング処理できるように、Extended Shift-And 法に対して以下の変更を加えた。

- 各パターンごとにパターン長に差があるため、どこで、どの文字が末尾に来るかという情報を保持するビットテーブル End を準備する。
- オプション文字の位置情報をアルファベットごとに記録するために、ビットテーブル E を準備する。このテーブルを、照合処理の際に $Emask$ の代わりに使用する。

miltiple-ESA 法による処理は Extended Shift-And 法と同様に、前処理と照合処理の 2 段階に分かれている。multiple-ESA 法は、パターン先頭から L_{max} 文字までの照合の可能性を確かめる。

3.3.1 前処理

拡張文字列パターンの集合を $P = \{P_1, P_2, \cdots\}$ とする。パターン P_i について、m_i は P_i の虚パターン長、L_i は P_i の実パターン長である。

ビットテーブルにおいて、パターン P_i の pos ($0 \leq pos \leq L_i - 1$) 番目の文字照合中に、文字 c を読んだ状況は次のようになる (例として、ビットテーブル B の場合を示す)。

- $B[c]$ & $(1 \ll pos) \neq 0$
  パターン P_i について、pos 番目の文字が文字 c である可能性がある
- $B[c]$ & $(1 \ll pos) = 0$
  パターン P_i について、pos 番目の文字が文字 c とはなりえない

マスクにおいて、パターン P_i の pos 番目の文字照合中の状況は次のようになる (例として、マスク $Emask$ の場合を示す)。

- $Emask$ & $(1 \ll pos) \neq 0$
  パターン P_i について、pos 番目の文字はオプション文字である可能性がある
- $Emask$ & $(1 \ll pos) = 0$
  パターン P_i について、pos 番目の文字はオプション文字とはなりえない

以下に、前処理の概要を示す

1. $Emask$ の初期化 ($Emask := 0$)
2. 全ての拡張文字列パターン $P_i \in P$ について、パターン文字列 P_i を走査し、以下の情報を取得
   - L_i: メタ文字を含めず、文字種を 1 文字とカウントした場合のパターン長 ($1 \leq L_i \leq L_{max}$ (L_{max} の場合、長さが L_{max} 以上であることを示す))
   - B: パターン中の文字について、出現する可能性のある位置を記憶
   - End: パターン中の末尾文字について、出現する可能性のある位置を記憶 (パターンが L_{max} 文字以上なら、L_{max} ビット目にフラグとして 1 をセット)
   - S: パターン中の繰り返し文字について、出現する可能性のある位置を記憶
   - E: パターン中の文字について、一つ先以降の文字 (遷移先) にオプション文字が連続して出現する可能性のある場合、その範囲 (ϵ 遷移で移動できる状態の集合) を記憶
   - $Emask$: パターンにおいて、オプション文字 (ϵ 遷移) が出現する可能性のある位置を記憶
3. マスク $Emask$ を走査して、以下の情報 (全て、L_{max} ビットのマスク) を取得
   - $Ebeg$: オプション文字かもしれない文字について、ブロックが始まる直前の位置を記憶
   - $Eend$: オプション文字かもしれない文字について、ブロックが終わる位置を記憶

表- 1: ビットテーブル

文字	B	End	S	E
A	2			
I	1, 4, 5	4	1	2, 3
N	2, 3, 4			3, 5
P	1, 6	6		
Y	1, 3		1, 3	2, 3

3.4 照合処理

本処理の工程を次に示す。pos は現在照合中の入力テキストの位置を表し、$T[pos]$ は入力テキストの現在照合している文字を表す。Extended Shift-And 法における照合処理と異なるのは、オプション文字の位置判定に E を利用することおよび終了判定である。muiltiple-ESA 法では、状態マスク D と末尾文字用ビットテーブル End との論理積の値によって、照合の完了を確認する。

1. 実行時状態マスク D (L ビット) を、0 で初期化
2. $0 \leq pos \leq n$ (テキスト長) -1 の間、以下を繰り返す
   (a) $D_1 \leftarrow \{(D \ll 1) \mid 1\}$ & $B[T[pos]]$
   (b) $D_2 \leftarrow D$ & $S[T[pos]]$
   (c) $D \leftarrow D_1 \mid D_2$
   (d) $Df \leftarrow D \mid Eend$
   (e) $D \leftarrow D \mid \{E[T[pos]]$ & $(\sim (Df - Ebeg) \bigoplus Df)\}$
   (f) もし、D & $End[T[pos]] \neq 0$ ならマッチング成功
3. これまでに成功していなければマッチング失敗

3.5 アルゴリズムの実行例

multiple-ESA 法の実行例を示す。パターンを

$P_a =$ “PANI”
$P_b =$ “[IY]N?Y*NI?P”

とし、テキストを

$T =$ “PANYAAYNIPIN”

とする。

まずパターン P_a と P_b を入力として、前処理を行なう。そのとき作成されるビットテーブルビットマスクの様子を、それぞれ表 1 と図 2 に示す。

次に照合処理の過程を見る。ビットマスクの値は、図 2 のように最上位ビットから $a_L, \cdots, a_2, a_1$ となっており、文字列中の位置とマスクの各桁が対応する。

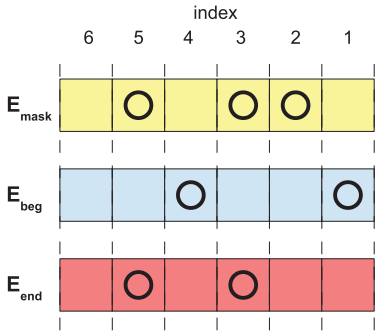

図- 2: ビットマスク

表- 2: 状態マスク D と $End[T[pos]]$ の変化

pos	$T[pos]$	D	$End[T[pos]]$	照合
-	-	0 0 0 0 0 0	- - - - - -	
1	P	0 0 0 0 0 1	1 0 0 0 0 0	
2	A	0 0 0 0 1 0	0 0 0 0 0 0	
3	N	0 0 0 1 0 0	0 0 0 0 0 0	
4	Y	0 0 0 1 1 1	1 0 0 0 0 0	
5	A	0 0 0 0 1 0	0 0 0 0 0 0	
6	A	0 0 0 0 0 0	0 0 0 0 0 0	
7	Y	0 0 0 1 1 1	1 0 0 0 0 0	
8	N	0 1 1 1 1 0	0 0 0 0 0 0	
9	I	0 1 1 1 1 1	0 0 1 0 0 0	×
10	P	1 0 0 1 1 1	1 0 0 0 0 0	○
11	I	0 0 0 1 1 1	0 0 1 0 0 0	
12	N	0 1 1 1 1 0	0 0 0 0 0 0	

注目する文字が進むにつれ、実行時状態マスク D と入力文字に対応する $End[T[pos]]$ の値がどう変化していくのかをビット列として表 2 に示す。照合欄に記号があるのは、アルゴリズムで照合を検知する場合である。このうち、○は正しい照合を示すが、×は正しくない照合を示す。ここで、正しくない照合とは、アルゴリズムとしては照合を検知してはいるが、実際にはパターンにマッチしていないものを指す。

3.6 照合精度の向上

multiple-ESA 法を単純に実行するだけでは、実用上許容出来るエラーの発生率が得られない場合がある。そこで、照合誤りの発生率を抑える手法を追加する。単純な multiple-ESA 法を「順行 multiple-ESA 法」とするなら、ここで追加するのは「逆行 multiple-ESA 法」と表現できる。

逆行 multiple-ESA 法では、テキスト上においてパターン末尾が発見された位置 pos、パターン末尾に一致した文字 $T[pos]$ から、逆向きに miltiple-ESA 法を実行する。逆行 multiple-ESA 法では、ビットマスクやビッ

トテーブルなどは順行 multiple-ESA 法で用いるものとは別に作成する。パターンには演算子が含まれ、さらにパターンごとにその長さが異なるため、順行処理用のテーブルやマスクと逆行処理用のテーブルやマスクは異なるものとなり、照合誤りが発生する場所も異なる。そのため、逆行アルゴリズムを適用することで、エラーを軽減できる。

また、逆行 multiple-ESA 法を実行する際には、末尾文字およびその出現位置ごとに、パターン集合に関する情報を求めておく。この処理により、求める情報は増えてしまうが、重ね合わせるパターン数を削減できるため、単に逆向きにアルゴリズムを適用する場合に比べて、さらにエラーの軽減率を上げることができる。

4 考察

4.1 アルゴリズムの処理性能

本研究で提案する multiple-ESA 法について、アルゴリズムの処理性能を計算量で示す。使用する変数については、パターン数を P_n 全パターンにおける実パターン長の平均を $\bar{m}$、虚パターン長の平均を $\bar{L}$、パターン表現に使用するアルファベットの総数を $|\Sigma|$ とする。

まず順行処理における前処理について示す。時間計算量は $P_n \times (\bar{m} + L_{max} \times \bar{L}) \times |\Sigma|$ となる。空間計算量は $|\Sigma|$ 行 L_{max} 列のビットテーブルが 5 つと L_{max} 次元のビットマスク 4 つ、よって $L_{max} \times (5|\Sigma| + 4)$ となる。つまり、最大パターン長に依存し、パターン数 P_n に依存しない。

次に順行処理における照合処理について示す。時間計算量は Extended Shift-And 法同様 ω をレジスタ長としたとき $O(n\lceil L_{max}/\omega \rceil)$ である。Aho-Corasick 法 Commentz-Walter 法などの手法も $O(n)$ であり、同等の処理速度といえる。空間計算量は、$|\Sigma|$ 行 L_{max} 列のビットテーブル 4 つと L_{max} 次元のビットマスクと状態マスクが合計 4 つ、よって $L_{max} \times (4|\Sigma| + 4)$ となる。時間計算量、空間計算量ともに、パターン数 P_n に依存しない。

逆行処理における照合処理における時間計算量においては、順行処理における前処理に nar(順行処理におけるパターン候補の絞り込み率) をかけたものと、順行処理における照合処理に nar_i(パターン位置の候補の絞り込み率) をかけたものとの和になる。

4.2 照合誤りの性質

提案手法では、Extended Shift-And 法のテーブルを重ね合わせてしまったため、以下の状況で照合誤りが発生する。

1. 文字遷移の誤り
   各パターンに出現するアルファベットに関して、パターン上での出現位置が重なっている場合。
2. 繰り返し文字遷移の誤り
   あるパターンの末尾であるアルファベットに関して、2 つ以上のパターンにおいて末尾となる位置が重なっている。
3. オプション文字判定の誤り
   各パターンに出現するアルファベットに関して、オプションとなっているものとオプションでないものが同じ位置に重なっている。

いずれの場合も、照合見逃し (false negative) は発生しないが、照合できていないのに照合ができたと誤検知する事例 (false positive) だけが起こり得る。IDS への応用を考えた場合、照合誤りの方向が片方向であることは重要である。IDS での検知にそのまま用いれば、False Positive は発生するが、False Negative は発生することがない。

1.〜3. の誤りが発生する原因は、ビットテーブル内で全てのパターンの情報をまとめて扱っているためである。例えば、パターン中での文字の出現位置を記録するビットテーブル B なら「どれかのパターンにおいて、アルファベット c がパターンがどこで出現するか」という情報を持つだけで、パターン照合時に確かめるのは正しい可能性である。

任意のパターン $P_i \in P$ の k 番目の文字 c を読んだときに、照合誤りが発生する条件を下に示す。ただし、B_i、S_i は、それぞれ、パターン P_i に対応する文字テーブル、繰り返し文字テーブルである。$Emask_i$ は、パターン P_i に対応するオプション文字マスクである。

1. $B[c]$ & $(1 \ll k) \neq 0$ かつ $B_i[c]$ & $(1 \ll k) = 0$
2. $S[c]$ & $(1 \ll k) \neq 0$ かつ $S_i[c]$ & $(1 \ll k) = 0$
3. $E[P_i[k]]$ & $(1 \ll k) \neq 0$ かつ $Emask_i$ & $(1 \ll k) = 0$

4.3 照合誤りの頻度

multiple-ESA 法における照合誤りの発生頻度は、使用するパターンや入力テキストに依存する。そのため、ここでは次のような仮定の下で照合誤りが発生する確率を求める。

- アルファベットの数: 128
- テキストの長さ: 1024
- 各パターンおよびテキストにはランダムな文字列を利用する。

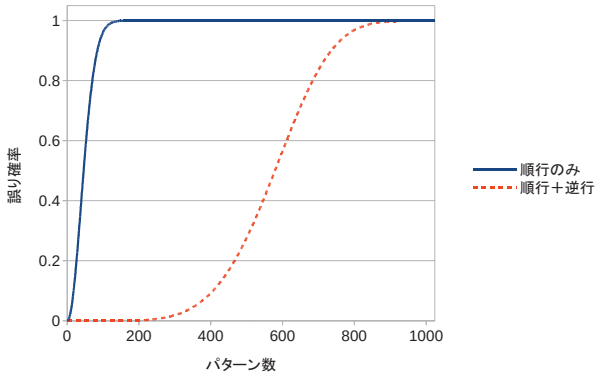

図- 3: パターン数による誤り確率の変化

- パターンの最小長: 2
- パターンの最大長: 8
- 各パターンの長さは 2〜16 で一様に分布する
- 繰り返し文字の頻度: 20%
  ただし、パターン P_i の先頭と末尾の文字は繰り返し文字ではない
- オプション文字の頻度: 20%
  ただし、パターン P_i の先頭と末尾の文字はオプション文字ではない
- 文字クラス・ワイルドカード文字の出現については考慮しない

これらの仮定に基づき計算した結果、パターン数が 256 個の場合、順行 muiltiple-ESA 法でテキストを処理し照合が報告された際、それが正しい照合である確率は 2.5×10^{-9} となる。順行に加えて逆行処理も加えた場合には、照合が報告された際にそれが誤りである確率は 7.0×10^{-3} である。横軸にパターン数をとった場合の誤り確率の変化を図 3 に示す。順行処理のみでは誤り率が高く実用に耐える精度は得られないが、逆行処理を組み合わせることにより、比較的良い結果が得られていることがわかる。

5　おわりに

本論文では、複数の拡張文字列をまとめて探索する multiple-ESA 法を提案した。このアルゴリズムは、照合の際に誤りを許すことで、処理の速度を落とさずに、照合に必要なメモリ空間を抑えることができる。しかし、照合誤りは IDS としての使用に許容できない程度に発生してしまうため、順行と逆行の 2 度の処理を行うことで、エラーの発生率を抑える工夫も行った。エラーの発生率は抑えられたものの、まだ IDS としての利用が可能になる程度には低くなっておらず、今以上にエラーの発生率を抑える手法を考える必要がある。

本論文では、基本的な拡張文字列を対象としたが、FBPM で用いる拡張文字列に拡張したり、RunNFA での階層化の概念を導入することも可能である。階層化は、導入の仕方によっては照合の対象とするパターンの表現能力を上げることに使える可能性もあり、エラーの発生率を抑える工夫とともに検討してゆきたい。また、提案したアルゴリズムを実際に Snort などに実装し、現実的なシグネチャを与えた場合の性能についても検討してゆきたい。

本研究は JSPS 科研費 24500074 の助成を受けたものです。

参考文献

[1] Patrick Wheeler and Errin Fulp. A taxonomy of parallel techniques for intrusion detection. In *Proceedings of the 45th annual southeast regional conference*, ACM-SE 45, pp. 278–282, New York, NY, USA, 2007. ACM.

[2] T. Limmer and F. Dressler. Adaptive load balancing for parallel IDS on multi-core systems using prioritized flows. In *Computer Communications and Networks (ICCCN), 2011 Proceedings of 20th International Conference on*, pp. 1–8, 2011.

[3] Nen-Fu Huang, Hsien-Wei Hung, Sheng-Hung Lai, Yen-Ming Chu, and Wen-Yen Tsai. A GPU-based multiple-pattern matching algorithm for network intrusion detection systems. In *Advanced Information Networking and Applications - Workshops, 2008. AINAW 2008. 22nd International Conference on*, pp. 62–67, 2008.

[4] Abhishek Mitra, Walid Najjar, and Laxmi Bhuyan. Compiling pcre to fpga for accelerating snort ids. In *In ACM/IEEE Symp. on Architecture for Networking and Communication Systems (ANCS)*, 2007.

[5] Toshihiro Katashita, Yoshinori Yamaguchi, Atusi Maeda, and Kenji Toda. FPGA-based intrusion detection system for 10 gigabit ethernet. *IEICE - Trans. Inf. Syst.*, Vol. E90-D, No. 12, pp. 1923–1931, December 2007.

[6] C.J. Coit, S. Staniford, and J. McAlerney. Towards faster string matching for intrusion detection or exceeding the speed of snort. In *DARPA*

Information Survivability Conference amp; Exposition II, 2001. DISCEX '01. Proceedings, Vol. 1, pp. 367–373 vol.1, 2001.

[7] M. Alicherry, M. Muthuprasanna, and V. Kumar. High speed pattern matching for network IDS/IPS. In *Network Protocols, 2006. ICNP '06. Proceedings of the 2006 14th IEEE International Conference on*, pp. 187–196, 2006.

[8] 今泉貴史, 水野恵祐. IDS に特化した文字列探索アルゴリズム. 情報処理学会研究報告. IOT, [インターネットと運用技術], Vol. 2009, No. 21, pp. 107–112, February 2009.

[9] Karl Abrahamson. Generalized string matching. *SIAM J. Comput.*, Vol. 16, No. 6, pp. 1039–1051, December 1987.

[10] Ricardo Baeza-Yates and Gaston H. Gonnet. A new approach to text searching. *Commun. ACM*, Vol. 35, No. 10, pp. 74–82, October 1992.

[11] Gonzalo Navarro and Mathieu Raffinot. *Flexible Pattern Matching in Strings: Practical On-Line Search Algorithms for Texts and Biological Sequences*. Cambridge University Press, New York, NY, USA, 1st edition, 2007.

[12] Yusaku Kaneta, Shin-Ichi Minato, and Hiroki Arimura. Fast bit-parallel matching for network and regular expressions. In *Proceedings of the 17th international conference on String processing and information retrieval*, SPIRE'10, pp. 372–384, Berlin, Heidelberg, 2010. Springer-Verlag.

[13] 笹川裕人, 金田悠作, 有村博紀. 長大な拡張文字列パターンに対する大規模文字列照合の高速化. 日本データベース学会論文誌, Vol. 11, No. 1, pp. 55–60, June 2012.