Vol.2009-ARC-184 No.21
2009/8/5

パケットストリームの正規表現処理を可能とするネットワークプロセッサ

永冨泰次[†]　石田慎一[†]　原島真悟[†]　鯉渕道紘[††]
川島英之[†††]　西宏章[†]

正規表現は，ネットワーク侵入検知システム（NIDS）などをはじめ様々なネットワークアプリケーションで広く使われている．しかしこれらの多くはソフトウェアベースのため処理速度が遅くネットワークのサービス品質に影響を及ぼしている．高速な処理を妨げる要因の一つは処理時間の大半を占めるパターンマッチングである．NIDS のパターンマッチングに用いられるパターン数は膨大であり，かつ複雑な正規表現も含まれている．このような膨大かつ複雑な正規表現を高速で処理できるエンジンのニーズは高い．本論文では膨大なパターン処理を可能とするパターン絞り込み機構を提案する．

Character-string Processing Architecture for High-throughput Complex Regular-Expression Matching

Yasutsugu Nagatomi[†]　Shinichi Ishida[†] Shingo Harashima[†]
Michihiro Koibuchi[††]　Hideyuki Kawashima[†††]
and Hiroaki Nishi[†]

Regular-expression is widely used in various network applications, such as a network intrusion detection system (NIDS). However, high-throughput regular-expression execution couldn't achieve enough processing throughput because most of NIDSs have been implemented by software. Pattern matching function, which is a part of regular-expression processing, requires comparatively longer processing time. A huge amount of patterns are used for this pattern matching in NIDS. High-throughput processing engine are strongly required in order to execute a large number of complicated regular expression. In this paper, a pattern filtering architecture is discussed that enables to achieve a large number of patterns matching.

1. イントロダクション

正規表現は，ネットワーク侵入検知システム（NIDS）などをはじめ様々なネットワークアプリケーションで広く使われている．代表的な NIDS である Snort[1]や Bro[2]もパターンファイルに正規表現を用いている．しかしこれらはソフトウェアベースのため処理速度が遅くネットワークのサービス品質に影響を及ぼしている．高速な処理を妨げる要因の一つは処理時間の大半を占めるパターンマッチングである．NIDS のパターンマッチングに用いられるパターン数は膨大であり，かつ複雑な正規表現も含まれている．このような膨大かつ複雑な正規表現を高速で処理できるエンジンのニーズは高い．

これまで決定性有限オートマトン(DFA)や非決定性有限オートマトン(NFA)ベースのリコンフィギュラブルなデバイスを使うことにより正規表現処理を加速する多くの研究が広く行われてきた[3][4][5][6]．しかし DFA や NFA ベースの再構築可能なシステムは，正規表現処理のパターンを更新するごとに回路を書き変えるため通常大きなオーバーヘッドを生じてしまう．このためパターンを頻繁に更新する必要がある場合には不適切といえる．

本論文では迅速なパターン更新を可能とするプロセッサタイプの正規表現エンジンと膨大なパターン処理を可能とするパターン絞り込み機構を提案する．

本論文の構成は以下の通りである．第 2 節に関連研究を示し，第 3 節では本アーキテクチャのアプリケーション例の一つである snort のルールパターンについて説明する．第 4 節では提案する正規表現プロセッサについて説明し，第 5 節ではその評価を示す．最後に第 6 節で結論を述べる．

2. 関連研究

インターネットのトラフィック増加にともなって，NIDS においてスループットを左右する正規表現処理は重要な課題となっている．このニーズに応じるように正規表現処理に特化したハードウエアエンジンに関する研究が存在する．FPGA を用いた NFA ベースのアプローチでは NFA の並列性を生かすことで多くのパターンを同時処理できる．しかしその一方で，IDS ルールを更新するために回路の再構成が必要となる. このオーバーヘッドの大きさは無視できない. 近年では Titanic systems 社が NFA[7]に基づく正規表現プロセッサを製品化している．

[†]慶應義塾大学大学院理工学研究科
Graduate School of Science and Technology, Keio University
[††]国立情報学研究所/総合研究大学院大学
National Institute of Informatics/The Graduate University for Advanced Studies
[†††]筑波大学大学院システム情報工学研究科 Graduate School of Systems and Information Engineering, University of Tsukuba

前述したパターンファイル更新に伴うオーバーヘッドを減らすため，Baker らはメモリを更新するだけでパターンファイルの更新ができる DFA ベースのアプローチを提案した[3]．Baker らの採用した DFA は状態遷移先が必ず一つに決まる特徴をもつ．しかしこの特性により複雑な正規表現では指数関数的な状態数の増加が起きる．このアプローチは状態遷移がメモリサイズに制限を受けるため複雑な正規表現パターンを持つ膨大なパターンの処理には向かない．

ネットワーク向けに研究されているこれらのエンジンのほかに，文字列処理を効率的に処理する命令を備えたパソコン向け商用プロセッサも商品化されている．Intel 社の Core i7 である．SSE 4.2 の拡張命令として XML の字句解析高速化を意図した String & Text New Instructions (STTNI)と呼ばれる命令セットを持ち，正規表現も効率的に処理できる．

3. Snort

3.1 アラートの概要

この章では本提案のアプリケーション例として代表的な侵入検知システムの一つである snort について触れ，パターンファイルの特徴について説明する．

Snrot は 5700 ものアラートから構成される．図 1 は実際に使われているアラートの一例である．ペイロード情報やポート番号，パケットペイロードサイズなどを手掛かりに不正アクセスを検出する．

```
alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS (msg:"POLICY Google Desktop initial install - installer request"; flow:to_server , established; uricontent:"/installer?"; uricontent:"action=install"; uricontent:"version="; uricontent:"id="; uricontent:"brand=GGLD"; uricontent:"hl="; content:"User-Agent|3A|"; nocase; content:"Google"; distance:0; nocase; content:"Desktop"; distance:0; nocase; pcre:"/User-Agent¥x3A[^¥n¥r]+Google[^¥n¥r]+Desktop/smi"; classtype:policy-violation; sid:7859; rev:1;)
```

図 1 snort に用いられるアラートの一例

図 1 のように各アラートは複数の要素からなる．これらの要素がすべて真となるとき警告や指定された処理が実行される．アラートの構成要素は大きく分けてルールヘッダとルールオプションの 2 つである．ルールヘッダは送信元，宛先のアドレスおよびポート番号やプロトコルの指定が行われる．ルールオプションは主に 2 種類からなる．ペイロード情報に含まれるキーワードを指定するものとノンペイロード情報から得られる内容を指定するものである．ペイロード情報を指定するオプションは主に 3 種類ある．ペイロード情報に含まれる文字列を指定する content オプション，文字列の指定に正規表現を用いる PCRE オプション，URI 部分に含まれる文字列を指定する uricontent オプションである．次節にルールヘッダとルールオプションを分析した結果を示す．

3.2 ルールヘッダの分析

ルールヘッダは送信元，宛先のアドレスおよびポート番号やプロトコルの指定が行われる．送信元，宛先のアドレス，ポート番号指定が any(特定ポートでない)である割合を表 1 に示す．(2009 年 6 月現在 snortrules-snapshot-2.8 の rules フォルダ内におけるコメントアウトされていないアラートを対象とした．また，$EXTERNAL_NET の設定は any を用いることが一般的なため any として集計した．以下同様)

表 1 ルールヘッダに用いられる any の割合

条件	any の割合
送信元アドレス	73%
宛先アドレス	27%
送信元ポート	80%
宛先ポート	29%

表 1 より any の割合が低い宛先アドレス，宛先ポート指定は，ヒットする可能性のあるアラートを絞り込むのに適しているといえる．このため，特にこの 2 つのオプションに焦点を当てる．snort アラートに用いられる宛先アドレスの内訳について表 2 に示す．

表 2 アラートに用いられる宛先アドレスの内訳

条件	割合
Any	27%
内部ネットワーク	73%
その他	0.2%

表 2 より宛先アドレスの 7 割が内部ネットワークであることがわかる．これより内部ネットワークとその他のルールを分けることにより，宛先アドレスが内部向けのケースでは評価するアラートを 3 割程度削減でき，宛先アドレスが内部以外の場合では

評価するアラートを 7 割程度削減できる.

次にアラートに指定される宛先ポート番号の割合を表 3 に示す. ($HTTP_PORTS は 80 番ポート用いるのが一般的なため 80 として集計した.)

表 3 アラートに用いられる宛先ポート番号の割合

ポート番号	割合
80	33%
Any	29%
80 と any を除くその他のポート	39%

表 3 より，80 番ポートとそれ以外を区別することにより，評価するアラートを 3 割~4 割削減できるため，マッチングにかかる負荷を削減できる.

3.3 ルールオプションの分析

ルールオプションにおいてペイロード情報を指定するオプションは主に 3 種類ある. ペイロード情報に含まれる文字列を指定する content オプション, 文字列の指定に正規表現を用いる PCRE オプション, URI 部分に含まれる文字列を指定する. uricontent オプションである. アラートがペイロード情報による識別オプション(content, uricontent, pcre)を含む割合は，98%である. ペイロード情報による各識別オプションが用いられる割合を表 4 に示す.

表 4 ペイロードによる各識別オプションが用いられる割合

条件	割合
PCRE のみ	0.1%
PCRE と content, uricontent	45%
content, uricontent のみ	55%

表 4 より pcre のみが用いられるパターンは極少ないことがわかる. このため，後述するインスペクションモジュールでは content と uricontent のみを用いて hash テーブルをつくり精査するアラートの絞り込みを行う. content と uricontent の割合を表 5 に示す.

表 5content と uricontent の用いられ方

条件	割合
uricontent および content	11%
uricontent のみ	19%
content のみ	70%

表 5 より uricontent もしくは content を含むアラートのうち 7 割が content のみを含むことがわかる. uricontent は検索領域が URI 部のみに限られているため，hash 化した際に衝突の起きる確率が相対的に低い. 一方 content は検索領域が限定されていないため hash の衝突が起きる確率が相対的に高くなる. hash が衝突することによる問題を抑制するため，宛先ポート番号，宛先アドレス指定と検索領域を限定するオプションを利用することを考える. 宛先ポート番号と宛先アドレスは，前述したルールヘッダで指定されるものである. 検索領域を指定するオプションはルールオプションで指定されるもので主に 2 つある. ストリーム先頭からバイト単位で検索開始位置をオフセットさせる offset オプションと検索する深さを指定する depth オプションである. ここでは offset は限定力が弱いため depth のみ考える. Content のみを含むアラートのうち宛先ポート番号指定，もしくは検索領域を限定するオプションを含む割合を表 6 に示す.

表 6 Content のみを含むアラートにおける
宛先ポート番号，検索範囲限定オプションの割合

条件	割合
宛先ポート指定と depth	22%
宛先ポート指定のみ	20%
depth のみ	15%
どちらも指定なし	42%(30%)

()内の数は hash 化対象の全アラートにおける割合

表 6 より content のみを含むアラートにおいて 6 割，全アラートにおいては 7 割のアラートが 2 つのオプションを用いることで絞り込み精度を向上させることができる.

文字列を Hash 化する際, 文字列長毎に hash テーブルを用意するのが一般的である. Content オプションで用いられるパターンの文字列長分布を示す.

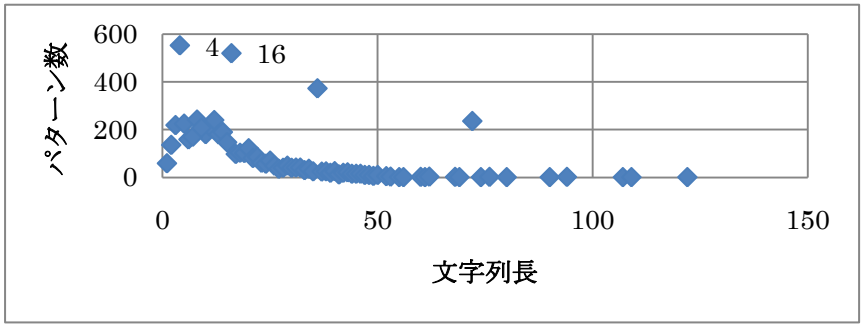

図 2 snort の全アラートにおけるパターン長の分布(uricontent，content オプションで指定されるアラート内の最長パターンを抽出)

本報告では，図 2 より出現頻度の高い文字列長が 4 と 16 のパターンを対象に評価を行う．また，文字列長 4 以上のパターンが 9 割以上を占めることから 4 文字以上のパターンについても評価を行う．なお，文字列長 4 以下の残り 1 割についてはパケットサイズ指定など絞り込みに有用な他のオプションを含んでおり，また，検索範囲がストリームの先頭からのみなど極めて限定的であるものが多い．これらのことから，本報告で評価対象とならない，文字列長 4 以下のパターンがスループットを低下させるドミナントな要因とならないと推察できる．

表 7 に全アラートにおいて大文字と小文字をケアしない nocase オプションを含む割合を示す．

表 7　全アラートにおいて nocase オプションを含む割合

条件	割合
nocase を含む	63%
nocase を含まない	37%

表 7 より全体の 6 割が大文字と小文字をケアする必要のないことがわかる．一律に大文字と小文字をケアしなければ大文字小文字を区別するパターンと区別しないパターンの両方をサポートするケースと比べ半分のハードウェア資源ですむ．これは，hash テーブルと hash 生成用のモジュールの数が半分しか必要ないためである．しかし，処理を一律にすることで，絞り込みの精度が落ちると考えられる．第 5 章では大文字小文字を一律でケアしないことによるマッチ回数の増加を評価する．

4. 正規表現プロセッサ

4.1 正規表現プロセッサ概要

正規表現プロセッサの全体図を図 3 に示す．提案するアーキテクチャはソフトウェアとハードウェアからなる．ソフトウェアは主にハッシュテーブルジェネレータ，ビットマップテーブルジェネレータ，インストラクションジェネレータからなる．ハードウェアは，主にリスクコア，インスペクションモジュール，マッチングモジュールからなる．ハッシュテーブルは，インスペクションモジュールで使われる．インスペクションモジュールは，被比較対象として与えられる膨大な正規表現パターンを絞り込むものである．ビットマップテーブルは，マッチングモジュールで使われる．マッチングモジュールは，絞り込まれたパターンを精査するものである．インストラクションは，リスクコアで使われる．リスクコアは，各モジュールをコントロールするものである．インスペクションモジュールとマッチングモジュールについては次節で詳しく説明する．

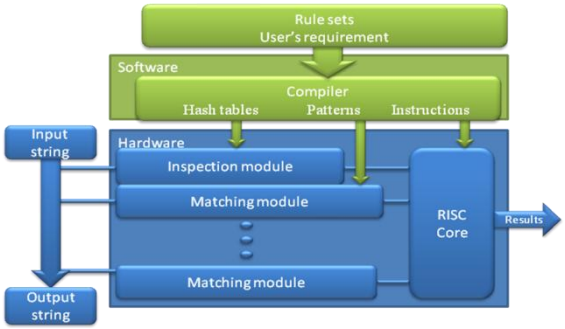

図 3 正規表現プロセッサ全体図

4.2 インスペクションモジュール

インスペクションモジュールは，被比較対象として与えられる膨大な正規表現パターンを絞り込むものである．第 3 章で示したように，snort のアラートの 99.9%は uricontent や content オプションで指定されるスタティックな文字列を含む．こうしたスタティックなパターンを捉えることによってストリームごとに評価するアラートを絞り込み，後続のマッチングモジュールへ処理を引き継ぐモジュールである．このモジュールは，CRC ジェネレータ，ハッシュメモリおよび，バイトカウンタ，ポートディテクタからなる．CRC ジェネレータは， ハードウェアコストが低い CRC を用いて入力文字列をハッシュ化するブロックである．このハッシュ値をアドレスとしてハッシュメモリを参照し，マッチした場合はマッチングモジュールに引き継ぐ．バイトカ

ウンタはストリームの先頭から何文字目を hash 化しているかを保持するカウンタである. ポートディテクタはストリームの宛先ポート番号を検出するモジュールである. ハッシュテーブルのキーは，パターンの先頭数文字分をハッシュ化して作られる．ハッシュテーブルの値は真偽値のほかに真偽判定の確度を高めるための付加的な情報が格納される．この付加的な情報とは，宛先ポート番号情報，宛先アドレス情報，検索指定範囲情報である．これらすべての情報を一律に持つのではなく，ユーザのトラフィックパターンに応じて必要なものを選択する．なお，この付加情報は数ビットに圧縮されて保持される．たとえば，ポート番号のみを付加情報とする場合を考える．前章で示したパターン中に含まれる宛先ポート番号とトラフィック中を流れる宛先ポート番号の割合を鑑みて，パターンを次の 3 つに分類できる．80 番ポートとそれ以外のポート，そしてどちらのポートでも良いパターンである．これら 3 状態を真とし，ほかを偽とする 4 状態を 2 ビットの情報として保持する．ヒットしたハッシュ値は，付加情報を処理するためにバイトカウンタとポートディテクタへ送られる．付加情報を処理する各モジュールからの最終判断とヒットしたハッシュキーは RISC Core へ渡される．これらの情報をもとに RISC Core は，空いているマッチングモジュールにマッチさせるべきパターンと入力文字列中の比較位置を渡し，マッチングモジュールで精査される．

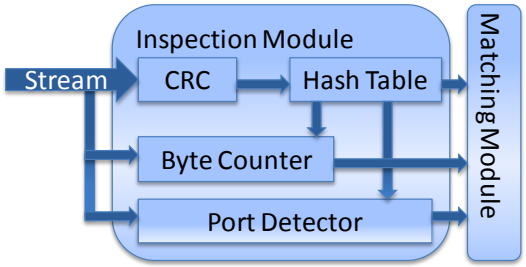

図 4　インスペクションモジュールの構造

4.3 マッチングモジュール

マッチングモジュールは，絞り込まれたパターンを精査するものである．マッチングモジュールの個数を増やすことで，同じサイクルで処理できるパターン数をスケーラブルに増やすことができる．このモジュールに関する詳細は参考文献[8]を参照されたい.

5. 評価

この章では下記 4 点を評価する.

- Hash を用いたパターン絞り込み手法の優位性評価
- 絞り込み精度を向上させるために有効なオプションの評価
- 大文字小文字をケアしないことによるハードウェア資源削減可能性の評価
- ハッシュテーブルに用いるメモリサイズの見積もり

まずマッチングのみによる手法に対する Hash による絞りこみをした後マッチングさせる手法の優位性を確認するために，処理ステップ数による評価をおこなう．snort では大文字小文字をケアしないパターンが過半を占めるが，すべてのパターンにおいて大文字小文字をケアせず評価すれば hash テーブルを別に用意する必要がなく，また hash 生成用のモジュールも削減できる．そこで，すべてのパターンにおいて大文字小文字をケアせずに絞り込みを行ってもマッチ回数の増加が許容できる範囲であるかを評価する．加えて，snort の各オプションが hash による絞り込みの精度を向上させるためにどの程度有効であるか，また，どのオプションの組み合わせが精度向上に寄与するか評価する. 最後にオプションによる絞り込みに必要な hash テーブルを格納するメモリのサイズについて見積もりを行う.

5.1 評価環境

当研究室の外部ネットワークに通ずるゲートウェイにおいて，外部と接続される NIC を通過する TCP ストリームを再構築しながらダンプするシミュレータを設置し評価データを取得した．メモリ資源に限りがあるため 1 ストリーム 10kbyte のサイズ制限をつけダンプを行い, これを評価用ストリームとした. 評価ストリームの特徴を　に示す．評価に用いた snort のバージョンは snortrules-snapshot-2.8 であり，rules フォルダ内において初期状態でコメントアウトされていないアラートを対象とした．また，アラートで指定されるペイロード情報には主に uricontent と content の 2 種があるが，全体の 7 割を占める content のみを含むアラートを特に対象として評価を行った.

表 8　評価用ストリームの特徴

	5.3，5.4 節	5.2，5.5 節
ストリーム数	47514	500
宛先ポートが 80 番ポートである割合	90%	95%
宛先アドレスが内部ネットワークである割合	17%	43%

5.2 Hash とマッチングを併用した手法と hash を併用しない手法における処理ステップ数の評価

Hash による絞りこみをした後マッチングさせる手法とマッチングのみによる手法の処理ステップ数評価を行う．マッチングにはストリームを一文字ずらして比較パターン分のマッチングを行う方式を考える．マッチング回数は 1 パターンにつき，ストリームの文字列長－パターン長 +1 で求めることができる．また，hash を引く回数も同式で示すことができる．一方，マッチングはこの式にパターン数を積算する必要がある．アラートを構成するある content オプションがストリーム中に確実に含まれることを断定できるまでのステップ数について，hash を用いる手法と hash を用いない手法で比較する．Hash を用いない場合には，マッチング回数×パターン数となる．Hash を用いる手法では，マッチング回数+hash ヒット回数×ヒットした hash の key に対応するパターン数となる．その hash に対応するパターンは複数存在することが考えられるため，hash の key に対応するパターン数を積算した．これらを指標として，処理ステップ数を見積もった結果を表 9 に示す．見積もりには，当研究室の実トラフィック 500 ストリーム分のデータと snort のパターンを用いた．Snort のパターンは content オプションで使用される文字列長が 4 と 16 のパターンを各アラートから 1 つずつ抽出し用いた．抽出した文字列長 4 のパターン数は 268，文字列長 16 のパターン数は 98 だった．特に文字列長 4 と 16 を用いて評価するのは，3 章に示したように snort の content オプションで利用されるパターン長の分布を調べた結果最も使用頻度が高いためである．また，hash 関数には CRC16 を用いた．

表 9 処理ステップ数の見積もり

	4 文字パターン	16 文字パターン
マッチング	12,777,060,800	4,671,619,624
CRC16	47,706,384	47,675,429

表 9 より 16 文字パターンのケースではマッチングのみの手法が hash とマッチングを併用した手法の 100 倍ステップ数が必要なことがわかる．4 文字パターンのケースでは hash を併用したときの 270 倍のステップ数を要する．これよりハードウェア資源を要しても hash を併用した方が効率的なマッチングが行えると推察できる．

5.3 大文字と小文字の違いをケアしないことによるマッチ回数の増加

第 3 章に示したように，snort では，大文字小文字をケアしない nocase オプションを含むアラートの割合が 6 割である．Hash による絞り込みを考えた場合，パターン長毎に，大文字小文字をケアするものとケアしないものの 2 つの hash テーブルが必要となる．この両方をサポートするには hash テーブル用のメモリ資源，hash 生成モジュールがそれぞれ 2 つずつ必要となる．しかし，hash の性質上一定の割合で hash による衝突が起きるため，これらを厳格に区別することは劇的なマッチ回数増加がない限り必要ないと考えられる． 図 5 は，nocase オプションを併用しないパターンも含むすべてのパターンに対して大文字小文字をケアしないマッチングを行った結果が大文字小文字をケアする必要のあるパターンに対しては大文字小文字を区別してマッチングした結果に対してマッチ回数の増加率を示したものである．図中の depth-port は検索範囲指定 と 宛先ポート番号指定 ，ad は宛先アドレス，depth-ad は検索範囲指定と宛先アドレス指定，port-ad は宛先ポート番号指定と宛先アドレス指定，depth-port-ad は，検索範囲指定と宛先ポート番号指定と宛先アドレス指定の組み合わせを意味する．また凡例の 16，4，4 以上はそれぞれ，文字列長 16，4，4 以上のパターンによる評価であることを示す．これらは次節以降も同様である．

大文字小文字をケアしないマッチングは，概ね 1.1～1.2 倍のマッチ回数増加となっていることがわかる．この程度の増加であればハッシュメモリのハードウェアコストを半分にできるメリットの方が上回ると考えられる．しかし，組み合わせによってはマッチ回数が 3 倍程度となるものもある．こうした場合，後続のマッチングモジュールの並列度を上げる必要があるため，パターン長や組み合わせオプションによっては hash テーブルを 2 つ用いた方が全体のハードウェア効率は高くなると思われる．

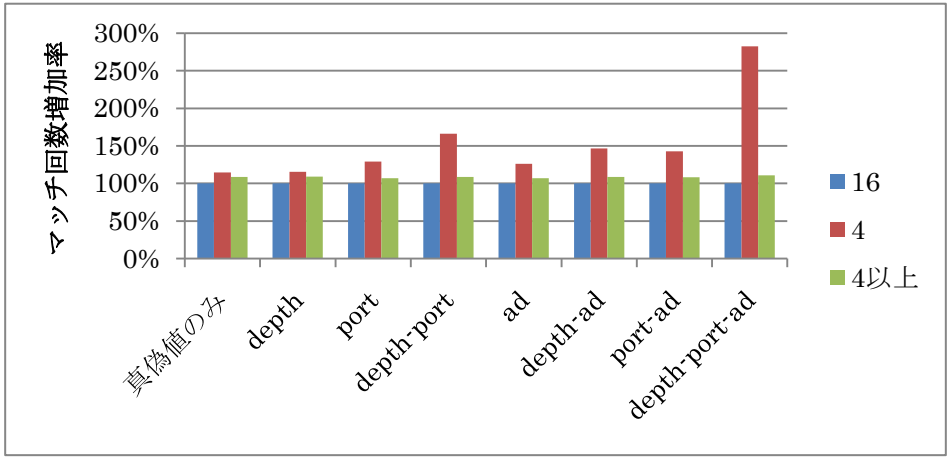

図 5 一律で大文字小文字をケアしないマッチングを行った際のマッチ回数増加率

5.4 宛先ポート，宛先アドレス，検索範囲指定による絞り込みへの寄与

絞り込みに有効なオプションの組み合わせを評価するために実トラフィックを用いたマッチングによる評価を行った. Any(指定なし)の割合が少なく絞り込みに有効と思われる宛先ポート，アドレスと検索範囲指定を行うオプションを評価対象とした．これらのオプションの組み合わせと条件を変えながらマッチングによりマッチ回数を評価した．評価対象のパターンは，3 章で示したように最も使用頻度の高い文字列長が 4 と 16 のパターンおよび全アラートの 9 割以上をカバーできる文字列長が 4 以上のものを評価に用いた．

5.4.1 文字列長 4 以上の content オプションを用いた評価

3 章で示したように, snort のアラートの 9 割以上はパターン長 4 以上の content オプションを含んでいる. ペイロード情報として content オプションのみを含むアラートから content オプションで指定される文字列を一つずつ抽出しマッチングさせる. アラートに複数の content オプションが含まれる場合には，depth オプションが併用されているものを代表して抽出する. こうして得られた 2508 パターンの文字列群を評価用ストリームとマッチングさせる．宛先ポート，アドレスと検索範囲指定を行うオプションを変えながら，マッチングモジュールへの処理引き渡し回数の変化を調べた結果を図 6 に示す. 図は真偽値のみでかつ大文字小文字をケアするケースを 100%としたときの変化を示している．なお絞り込みを行う条件は，1.マッチングのみ，2.宛先ポート指定のみ，3.宛先アドレス指定のみ，4.検索範囲指定のみとこれらの組み合わせ 4 種の計 8 種を設定し，それぞれ大文字小文字を一律でケアしないマッチングと一律ではなくパターンの設定を尊重するマッチングを行った．宛先ポート指定では，宛先ポートが 80 番ポートのパターンとそれ以外を分け, 該当するストリームの時に対応するパターンがマッチングするようにした．また，宛先アドレス指定では，宛先が内部ネットワークであるか否かでパターンをわけた. depth と offset オプションは本来パターン毎に任意の値をとることができるが, Depth と offset で指定される値がストリームの先頭から 0-50 文字の間となるオプションのみ有効とし，このほかの場合は検索範囲を限定せずマッチングさせることとした．なお，先頭から 0-50 文字の間に含まれるオプションについても限定範囲をそのまま反映させるのではなく, 一律で先頭から 0-50 文字の間に指定して評価した．一律の設定としたのは，ハッシュメモリのエントリサイズを圧縮するためである．このケースでは範囲指定の有無を保持する 1 ビットの情報がパターン毎にあれば済む. 図 6 よりオプションをいずれかひとつ選ぶとすれば宛先ポートを選ぶことが最も絞り込み効果が高いことが読み取れる．また，いずれか 2 つを選ぶという条件であれば，検索範囲指定と宛先アドレスの指定が最も絞り込み効果が高い．3 オプションすべてを用いると宛先アドレス指定のみの場合の半分程度まで絞り込むことができる．

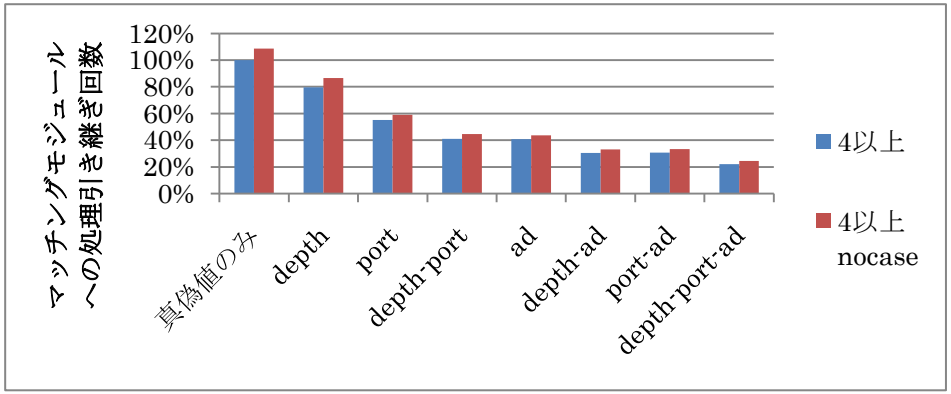

図 6　抽出した文字列長 4 以上のパターンのマッチ回数

5.4.2 文字列長 4，16 の content オプションを用いた評価

文字列長以外の条件は前節と条件を揃えて評価をおこなった．文字列長 4 のパターン数は 268，文字列長 16 のパターン数は 98 である．この結果を図 7 と図 8 に示す. 特に snort の content オプションで利用されるパターン長の分布を調べた結果最も使用頻度が高い文字列長 4 と 16 について評価する．

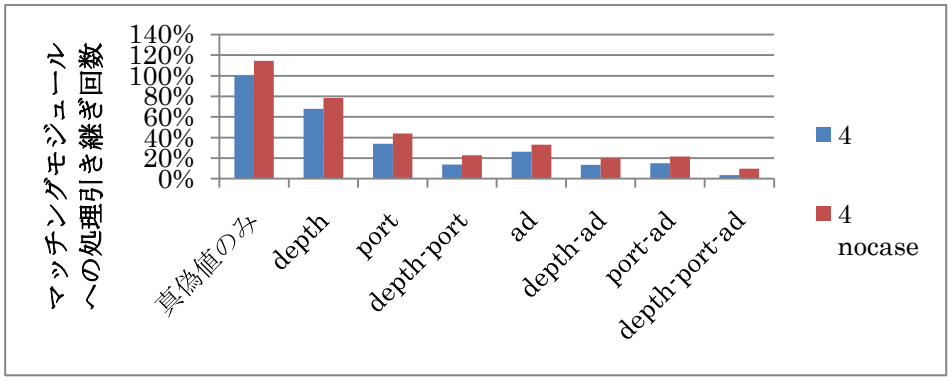

図 7　抽出した文字列長 4 のパターンのマッチ回数

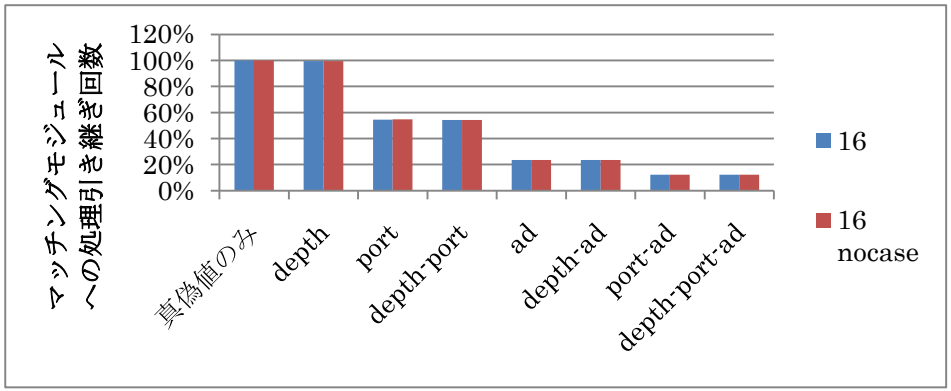

図 8 抽出した文字列長 16 のパターンのマッチ回数

図 7 および図 8 より，文字列長により適するオプションの組み合わせが異なることがわかる．どれかひとつのオプションを選ぶ場合，文字列長 4 のパターンでは宛先ポート指定, 文字列長 16 のパターンでは宛先アドレス指定が効果的であることがわかる．いずれか二つのオプションを選ぶ場合文字列長 4 のパターンでは宛先アドレス指定と検索範囲指定, 文字列長 16 のパターンでは宛先アドレス指定と宛先ポート指定の組み合わせが効果的であることがわかる．3 つのオプションすべてを用いる場合，宛先アドレスのみを用いるケースと比べ文字列長 16 の時では半分, 文字列長 4 の時では 1/10 程度までマッチ回数を削減することができる．

5.5 ハッシュテーブルに用いるメモリサイズの見積もり

前節と同様にして抽出した文字列長 4, 16 のパターンからビット幅の異なる hash テーブルをつくり, ストリームを 1 文字ずつずらしながら hash キーを引いていくことを繰り返す評価を行った．hash 関数には CRC を用いた．図 9 はその時の結果を示したものである．縦軸はマッチングモジュールへ処理が引き継がれる回数である．Hash の性質上, 複数のパターンが同じ hash キーを持つことがあるがこれを考慮してキーのヒット回数にそのキーに対応するパターンの数を積算することによってマッチングモジュールへ処理が引き継がれる回数を求めた．

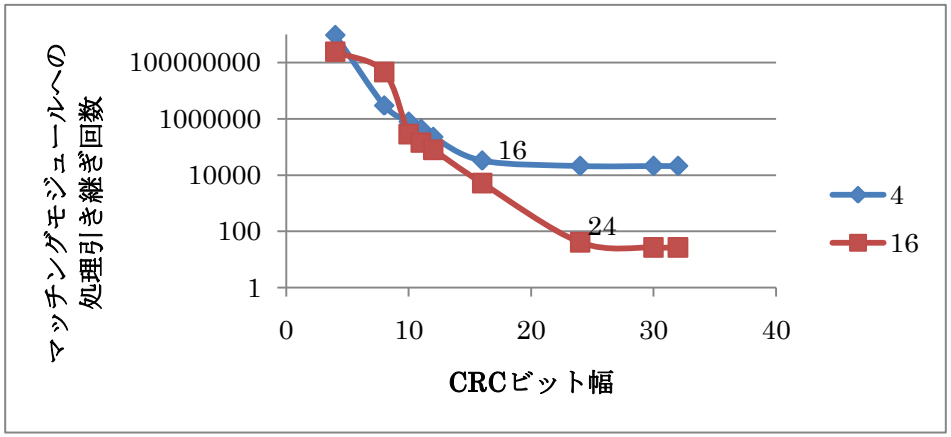

図 9　文字列長 4 および 16 のパターンにおけるマッチングモジュールへの処理引き継ぎ回数の評価

図 9 より文字列長 4 においては，傾きが緩やかとなるビット幅 16 以下が適しているといえる．また，文字列長 16 のパターンにおいては，同様にビット幅 24 以下が適しているといえる．より詳細に検討するにはマッチングモジュールの削減コストと CRC ビット幅増加によるコスト増を勘案すべきだが本報告により効果を持ち得る最大のビット幅が求められた．以上より文字列長 4 のパターンに対しては 16 ビット，文字列長 16 のパターンに対しては 20 ビットのビット幅を持つ hash テーブルを考え，必要なメモリサイズを見積もると表 10 のようになった．メモリサイズは hash 値のビット幅に2^{16}，2^{20}を積算することで求めた．hash 値のビット幅は表 11 に示す各状態数を保持できる最低限のビット幅とした．

表 10hash テーブルに用いるメモリサイズの見積もり

オプション	ビット幅	文字列長 4 におけるメモリサイズ(Kbyte)	文字列長 16 におけるメモリサイズ(Kbyte)
真偽値のみ	1	8.192	131.072
宛先アドレス	2	16.384	262.144
宛先アドレスとポート	4	32.768	524.288

表 11　各オプションの状態数

オプション	状態数	状態の内訳
検索範囲指定	2	有，無
宛先アドレス	3	any，内部ネットワーク，内部ネットワーク以外
宛先ポート	3	any，80 番，80 番以外

表 10 より文字列長 4 ではオプションを 2 つ併用する場合で，33Kbyte 程度，文字列長 16 でオプションを 2 つ併用する場合で 524Kbyte ということがわかる．これは，昨今のプロセッサの 2 次キャッシュサイズが 1Mbyte~6Mbyte であることを考えると，実装可能な大きさであるといえる．

6. 結論

正規表現パターンの更新を迅速に行うことのできるプロセッサタイプの正規表現エンジンを提案した．また，膨大なパターン処理を可能とするパターン絞り込み機構を提案し，その有効性と絞り込みに有効なオプションの予備評価を示した．今後の課題として，ハッシュエントリの有効利用法，絞り込むことによって削減できるマッチングモジュールのハードウェアコストと機能増加により必要となるハードウェアコストの比較が挙げられる．

謝辞　This work is partially supported by VLSI Design and Education Center(VDEC)， the University of Tokyo， Japan， in collaboration with Synopsys， Inc.， and National Institute of Information and Communications Technology (NICT).

参考文献

1)　SNORT，http://www.snort.org/
2)　BRO，http://www.bro-ids.org/
3)　Z. Baker，H.-J. Jung，and V. Prasanna. Regular Expression Software Deceleration for Intrusion Detection Systems. In Proceedings of the International Conference on Field-Programmable Logic and its Applications (FPL '06)，pages 418-425，2006.
4)　B. C. Brodie，D. E. Taylor，and R. K. Cytron. A scalable architecture for high-throughput regular-expression pattern matching. In Proceedings of the International Symposium on Computer Architecture(ISCA)，pages 191-202，2006.
5)　S. V. Ajnirban Majumder，Rajeev Rastogi. Scalable regular expression matching on data streams. In Proceedings of SIGMOD Conference，pages 161-172，2008.
6)　I. Sourdis，J. Bispo，J. M. Cardoso，and S. Vassiliadis. Regular expression matching in reconfigurable hard-ware. Journal of Signal Processing Systems for Signal，2007.
7) RegularExpressionProcessor，
http://www.titanicsystems.com/products/item/1/regular-expression-processor-rxp/
8) 永冨泰次，石田慎一，三野峻徳，川島英之，鯉渕道紘，西宏章: リッチなユーザサービスを提供するセマンティックルータにおける正規表現プロセッサの提案，電子情報通信学会，ネットワークシステム研究会(NS) (2008).