社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2009-SLDM-138 (33)
2009 / 1 / 30

NIDS専用正規表現マッチングマシンの構成とそのFPGA実装

川中 洋祐† 若林 真一† 永山 忍†

† 広島市立大学大学院 情報科学研究科 〒731-3194 広島市安佐南区大塚東3-4-1

あらまし 本稿では，ネットワーク侵入検知システム（NIDS）におけるパケット検査に対するパターンマッチングのための専用ハードウェアの構成を提案する．提案する専用マッチングマシンは，NIDSソフトウェアSnortで採用されている正規表現に基づくウィルスパターン記述を入力とし，高速にマッチングを実現する．提案専用マッチングマシンは，ウィルスパターンの更新に瞬時に対応するため，回路構成がパターンに依存する従来方式のマッチングマシンと，我々が提案しているパターンを実行時に設定できるマッチングマシンで構成されている．

キーワード ストリングマッチング，NIDS，正規表現，Snort

An Architecture of Regular Expression Matching Machine for NIDS and Its FPGA Implementation

Yosuke KAWANAKA†, Shin'ichi WAKABAYASHI†, and Shinobu NAGAYAMA†

† Graduate School of Information Sciences, Hiroshima City University
3-4-1, Ozuka-higashi, Asaminami-ku, Hiroshima 731-3194, Japan

Abstract This paper proposes an organization of special-purpose hardware dedicated to pattern matching for packet inspection in network intrusion detection systems (NIDSs). The proposed matching machine receives virus patterns as input, which are described as regular expressions defined in Snort NIDS software, and performs efficient matching. For instant updating of pattern sets, the proposed matching machine consists of an existing matching machine, in which patterns are implemented as hardware, and a new matching machine, in which patterns can be set during the execution time of matching.

Key words String Matching, NIDS, Regular Expression, Snort

1. はじめに

ネットワーク侵入検知システム (Network Intrusion Detection System, NIDS) [2] はネットワーク上で伝送されるウィルス等をリアルタイムに検出することでネットワークシステムのセキュリティの保持を実現するシステムである．NIDSではウィルス検出はストリングマッチング [1] として定式化されるため，NIDSではストリングマッチングを高速に実行することが要求される．そのため，ストリングマッチングアルゴリズムをハードウェアとして実現し，高速にストリングマッチングを実行することでウィルス等を検出するNIDSの研究が盛んに行われている．一方，NIDSにおいてはウィルスパターンが頻繁に更新されるため，検索パターンの更新に対して瞬時に対応可能なストリングマッチングアルゴリズムが望まれる．

NIDSにおけるパターンは一般に正規表現で記述されるため，正規表現に対する高速ストリングマッチングを実現する必要がある．本稿では，パターンを回路に組み込む従来のパターン依存ハードウェアと，我々が [4] で提案したパターンを実行時に設定可能なパターン非依存ハードウェアを組み合わせた新しいNIDSのシステム構成を提案する．提案システムでは，既知のパターンに対してはコンパクトな回路構成で高速にストリングマッチングを実行できるパターン依存ハードウェアでストリングマッチングを行い，更新されたパターンに対してはパターン非依存ハードウェアでストリングマッチングを行う．そのため，提案システムは，パターンの更新に対して瞬時に対応可能でありながら，かつ，実現するためのハードウェアコストが低い，という特徴がある．

2. Snort

ネットワーク侵入検知システム (NIDS) とは監視対象となるネットワークに設置された監視システムであり，当該ネットワークに流れる通信パケットとウィルスパターンのストリングマッチングを実行することによって不正侵入や攻撃がないかチェックを行うシステムである．

NIDS に対する代表的なオープンソフトウェアとして，Snort が知られている [6]．Snort においては，パターン（Snort ではルールとよぶ）は正規表現で記述され，パターンをより簡潔に記述するために多くの演算子が使用可能である．以下に，Snort のルールセットで使用される演算子の定義を示す [7]．今回使用したルールセットは Snort v2.4 である．

入力文字系列を構成する文字の集合を $\Sigma = \{a_0, a_1, a_2, \cdots, a_n\}$，$R$，$S$，$T$ をおのおの Σ 上の正規集合 R，S，T を意味する正規表現とする．

[定義 1] $R|S = \mathrm{R} \cup \mathrm{S}$

[定義 2] $R* = \{\varepsilon\} \cup \mathrm{R}^1 \cup \mathrm{R}^2 \cup \cdots$

[定義 3] $R? = \mathrm{R}|\varepsilon$

[定義 4] $R+ = \mathrm{R}^1 \cup \mathrm{R}^2 \cup \cdots$

[定義 5] $R\{\mathrm{n}\} = \mathrm{R}^n$

[定義 6] $R\{\mathrm{n,\ m}\} = \mathrm{R}^n \cup \mathrm{R}^{n+1} \cup \cdots \cup \mathrm{R}^m$

[定義 7] $R\{\mathrm{n,\ }\} = \mathrm{R}^n \cup \mathrm{R}^{n+1} \cup \cdots$

[定義 8] $. = a_0|a_1|a_2|\cdots|a_n$

[定義 9] $[a_0 a_i a_j] = a_0|a_i|a_j$

[定義 10] $[a_0 \text{ - } a_n] = a_0|a_1|a_2|\cdots|a_n$

[定義 11] $[\hat{}a_i] = a_0|a_1|a_2|\cdots|a_{i-1}|a_{i+1}|\cdots|a_n$

[定義 12] '^' は文字列の先頭，'$' は文字列の終端もしくは改行の前にそれぞれマッチする．

[定義 13] 定義 2 から 7 の各演算子の後ろに'?' を続けることによって，最低限の繰り返しでマッチングを実行する．

[定義 14] 以下の演算子は次のように定義されている．なお，各演算子の大文字はその演算の否定を意味する．

- \n = 改行のエスケープ文字
- \r = リターンのエスケープ文字
- \w = 英数字及び'_'
- \s = 空白文字
- \d = 数字
- \b = 単語境界 (片方を\w，もう片方を\W で挟まれた点)

[定義 15] '(' と')' で囲まれた部分正規表現をグループと呼ぶ．パターンの左から i 番目のグループにマッチした文字列を'$i' とする．'\$i' は文字列'$i' を意味する演算子である．'\$i' はパターンの後方参照として使われる．

[定義 16] $(?= R)S$ は，入力系列 T が R とマッチした場合のみ S と T のマッチングを開始する（先読み）．また，$(?! R)$ は $(?= R)$ の否定の意味となる．

[定義 17] 各修飾子は以下のマッチングを行う．

- /i = 大小文字の違いを無視
- /s = '.' が改行文字ともマッチ
- /m = '^' と'$' が各行の行頭・行末とマッチ
- /U = 定義 13 における繰り返し最大と最小の意味の入れ替え

3. ストリングマッチングハードウェア

著者らは [3], [4] において，正規表現のサブクラスをパターンとし，パターン非依存の回路構成を持つストリングマッチング専用ハードウェアを提案した．このハードウェアを拡張することで，NIDS において，Snort のルールに対するマッチングをハードウェアで高速に実行することが可能になる．以下に，マッチングハードウェアの概要を示す．

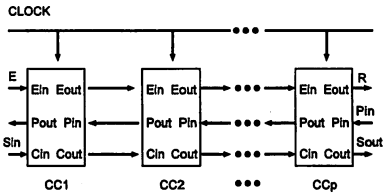

図 1 アルゴリズムのアーキテクチャ

3.1 パターン定義

ストリングマッチングにおいては，パターンとして単純な文字列だけでなく，形式言語に基づくパターンを用いることも多い．本稿では，前節で定義される正規表現で使用される演算子のうち，定義 1，2，3，8 で定義される演算子を用いて提案ハードウェアのパターンを以下のように定義する [3]．なお，演算子'-' の意味は定義 11 と同じである．

[定義 18] 正規表現において，各演算子 (*，|，•，-，?) のオペランドとなっている部分正規表現を**項**と呼ぶ．項が-以外の演算子を含まないとき，特にその項を**単一項**と呼ぶ．

[定義 19] 単一項と連接演算子・のみからなる項を *C*-**単純項**，単一項とユニオン演算子 | のみからなる項を *U*-**単純項**と呼ぶ．

[定義 20] s を単一項とする．$s*$，$s?$をそれぞれ *-**単純項**，?-**単純項**と呼ぶ．単一項および C-単純項，U-単純項，*-単純項，?-単純項をあわせて**単純項**と呼ぶ．

[定義 21] 項が単純項，U-単純項以外の単純項の連接，又は，それらのユニオンである場合，その項を *UC*-**項**と呼ぶ．例) $ab*c|d|.(\text{-}a?)$

[定義 22] 正規表現 P が C-単純項，もしくは C-単純項のユニオンをオペランドとするクリーネ閉包演算の項である場合，P を *UK*-**項**と呼ぶ．例) $(ab|c|def)*$

提案ハードウェアのパターンは UC-項，UK-項，あるいはそれらの連接とする．このパターンの定義では，演算子*，?の入れ子（ネスト）構造を禁止している．この制約により，上記の定義で表現されるパターンは通常の正規言語の部分集合となるが，正規表現をパターンとするストリングマッチングの多くのアプリケーションにおいては，このサブクラスに含まれるパターン記述でも十分な表現能力を持つ．

3.2 アーキテクチャ

提案アルゴリズムは 1 次元配列状に比較セル (Comparison Cell，CC) と呼ばれるプロセッシングユニットを相互接続することで実現される (図 1)．セルの左側から 1 クロックごとに入力系列 S が入力され，あらかじめ右端のセルから順番に入力されて各セルに記憶されたパターン P とマッチングを行う．提案ハードウェアのアーキテクチャは規則正しい構造を持ち，クロック信号に同期して動作する．

3.3 比較セル

比較セル CC は 1 文字単位のマッチングを行う回路である．CC の動作は複雑なので，まず CC の構造とパターンに UK-項

を許可しない場合のCCの動作について説明する．パターンにUK-項を許可する場合の動作については後述する．

図2にCCの構造を示す．CONはCCの全体の動作の制御を行う回路で，各CCはグローバルクロック信号に同期して動作する．E0，E1，E2はストリングマッチングの基本動作を実現するために使用されるフラグ，UC，EU，ECはUC-項がパターンとして与えられた場合のストリングマッチングを実現するために使用されるフラグである．OP0は'('と')'を，OP1は'-'，'ε'と'.'を，OP2は'*'と'?'を，OP3は'—'をそれぞれ記憶するレジスタである．LPはパターン中の1文字を，LCは入力系列中の1文字をそれぞれ記憶するレジスタで，CMPはレジスタLPとLCに記憶された文字を比較する比較器である．

3.4 比較セルの動作

CCの動作は2つのフェーズで構成される．1つはクロックt_0で動作する入力フェーズで，各セルのレジスタLPに記憶されたパターンと，そのセルでのマッチング結果を右隣のセルに出力する．もう1つはクロックt_1で動作するマッチングフェーズである．各CCは与えられた入力系列が右端のCCから出力されるまでこの動作を繰り返す．ここではパターンの入力動作については省略する．

3.4.1 基本動作

CCの基本動作について説明する．いま，クロック周期T_jにおいてi番目のCC(CC_i)が入力系列のk番目の文字（s_k）をレジスタLCに記憶していてマッチング動作を実際に行っているものとし，その動作を図3に示す．CC_iにおけるマッチングの結果に応じて，CC_iは他のCCにその結果を知らせ，次のマッチングの動作をそれらのCCに行わせる必要がある．CC_iの結果に応じて他のCCが行うべきことを図4に示す．この図において，△はCC_iにおけるマッチングの結果，マッチした場合に限りそのクロック周期でマッチングを行うことを表している．この動作は，クロック周期T_jでフラグE1を1にし，次のクロック周期T_{j+1}でE1の値をE2へ代入することで実現される．□はCC_iのマッチング結果に関わらず，そのクロック周期でマッチングを行うことを表している．この動作はフラグE2を1にすることで実現される．また◇はCC_iにおいて，p_iとs_kがマッチした場合，T_{j+1}以降の各クロック周期において，マッチングが失敗するまでマッチング動作を継続することを表している．これはフラグE0を1にすることで実現される．

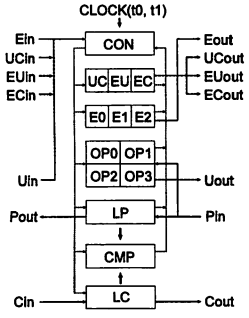

図2 Comparison cell (CC)
Fig. 2 Comparison cell(CC).

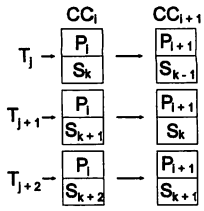

図3 基本的なストリングマッチング
Fig. 3 Basic string matching.

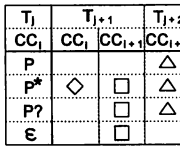

図4 p*，p?，εのストリングマッチング
Fig. 4 String matching of p*, $p^{\textcircled{a}}$ and ε.

フラグE2の値がそのCCでのマッチングの結果として回路の出力端子Rより出力される．右隣のCCではその値をenable信号E_{in}として受け取り，E_{in}が1ならばそのCCでマッチングを開始する．

3.4.2 UK-項のマッチング

UK-項のマッチングは前述の基本的なストリングマッチングよりもさらに複雑になる．紙面の都合上，ここではアルゴリズムの基本的なアイデアのみ説明し，アルゴリズムの形式的記述および正当性の証明は省略する．

パターンP=($abc|bcd|de$)*，入力系列S=$abcde$としたときのUK-項のマッチング動作を例として，以下ではUK-項のストリングマッチングのアルゴリズムについて説明する．

まず，UK-項はクリーネ演算の定義より入力系列の空語とマッチしなければならないため，入力系列が出力される前に"match"信号を出力しなければならない．提案ハードウェアでは，各CCは特別なフラグEKを持っており，UK-項内の最初のCCから最後のCCまでこのフラグを使用して"match"信号が伝達される．UK-項の左端のCCではE_{in}が1となったときにEKが1となり，任意のUK-項のCCでは，左隣のCCのEKが1のとき自身のEKも1となる．UK-項の右端のCCは，自身のEKが1となったとき"match"信号として1を出力する．UK-項が空語とマッチする場合の動作を図5に示す．図中の点線の円はEKが1であることを示している．

次に，パターンPの1回目の繰り返しで入力系列Sとマッチしたかを確認するため，UK-項はUK-項内の各項と入力系列のマッチングをそれぞれ行う．このマッチングは仮マッチングと呼ばれ，各項の先頭文字が入力系列とマッチした場合，その項と入力系列のマッチングが開始される．図6において仮マッチングが開始されるCCは，CC_1，CC_4，およびCC_7である．ここでCC_1およびCC_4で仮マッチングが開始されたときの動作を図6に示す．図6では，仮マッチングの成功は四角で示されており，CC_1はクロック周期T_1で，CC_4はT_5でそれぞれ仮マッチングが開始されている．

第i項で仮マッチングが開始されるとKS_i信号を，仮マッチングに成功するとKT_i信号を，失敗するとTF_i信号を決定セ

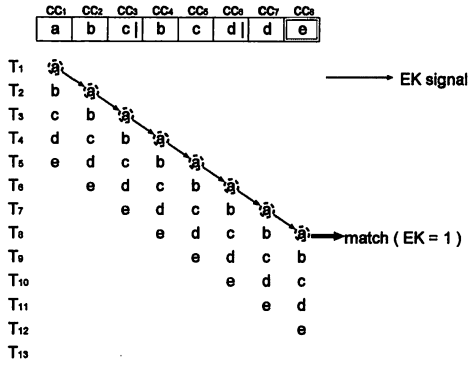

図 5 EK の動作
Fig. 5 Behavior of EK.

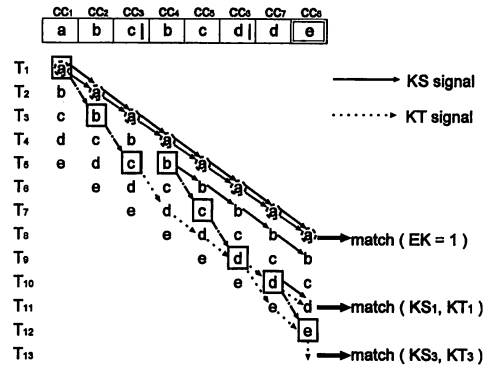

図 7 UK-項の 2 回目の繰り返し
Fig. 7 The second repetition of a UK-term.

ルと呼ばれる UK-項の右端の CC に出力する．そのため，決定セルを含む項では KT，KF は出力されず，そのマッチング結果は決定セルに記憶される．また，これらの信号が決定セル以外の CC に入力された場合，そのまま次の CC に出力する．

決定セルは，各項から送られた KT の中から"match"信号として出力される KT を選択する．このとき，各項から出力される KS が重要となる．UK-項のマッチングでは，UK-項の繰り返しが 1 回目の場合は入力系列の先頭文字からマッチングに成功しなければならないため，上記の例では入力系列の a からマッチングが開始されなければならない．ここで，UK-項の左端の CC では E_{in} が 1 となったときに EK が 1 になることから，各 CC は入力系列の先頭文字が入力されたときに EK が 1 になると言い換えられる（図 5）．また，KS 信号は仮マッチングを開始した文字と同時に右隣の CC に出力されることから，UK-項の繰り返し 1 回目では，決定セルの EK が 1 になるときに KS 信号が入力された場合のみ，入力系列の先頭文字から仮マッチングが開始されたことになる．図 6 では EK が 1 になったときに KS_1 が入力されていることから，第 1 項 abc が入力系列の先頭文字から仮マッチングを開始したことがわかる．さらに，第 1 項では仮マッチングに成功しているため，決定セルは KT_1 を"match"信号として出力する．ここで，第 2 項 bcd も仮マッチングに成功しているが，決定セルの EK が 1 になったときに KS_2 が入力されていない，つまり，入力系列の先頭文字から仮マッチングが開始されていないため，KT_2 は"match"信号として出力されない．以上より，パターン P の 1 回目の繰り返しでは入力系列の abc までマッチしたことになる．

繰り返しが n 回の場合，n-1 回目の繰り返しで UK-項の任意の項とマッチングに成功した入力系列の文字を s_{k-1} とすると，s_k からそれぞれマッチングに成功しなければならない．ここで，"match"信号として出力される KT 信号は仮マッチングに成功した次の文字と同時に右隣の CC に出力されるため，決定セルでは，"match"信号が出力されるときに入力された KS を選択候補とし，その項から KT が出力された場合のみそれを"match"信号として出力する．上記の例で $n = 2$ の場合の動作を図 7 に示す．図 6 より，$n = 1$ で"match"信号として出力された KT_1 と同時に KS_3 が決定セルに入力されていることから，KS_3 が選択候補となる．第 3 項 de が仮マッチングに成功していることから，決定セルは KT_3 を"match"信号として出力する．以上より，パターン P は 2 回の繰り返しで入力系列 S とマッチしたことがわかる．3 回目以降の繰り返しがある場合でも，同様の方法でマッチングを行う．

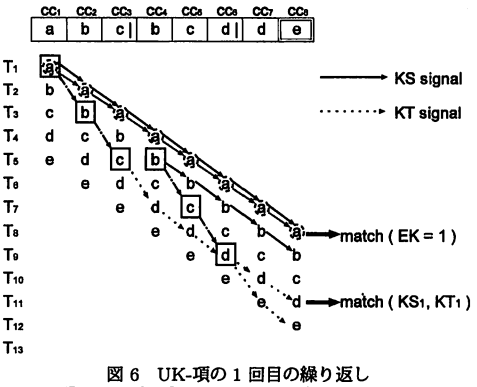

図 6 UK-項の 1 回目の繰り返し
Fig. 6 The first repetition of a UK-term.

3.5 提案アーキテクチャ

本稿では，さまざまなパターンに対してより柔軟にストリングマッチングを実行するために，3.2 節で示したアーキテクチャの拡張を行う．拡張アーキテクチャを図 8 に示す．このアーキテクチャでは，CC_{ip} (i = 1, 2,..., n + 1) の出力 E_{out} と C_{out} は次の CC の入力および回路の出力 R と S_{out} に接続される．CC_{jp+1} (j = 1, 2,..., n) の入力 E_{in} と C_{in} は前の CC の出力および回路の入力 E と S_{in} にマルチプレクサを通して接続される．この回路では，マルチプレクサの $select\_bit$ を 1 に設定することで，CC_{jp+1} (j = 1, 2,..., n) から複数の入力系列を同時に入力することが可能となる．また，ストリングマッチングは各 CC で並列に実行される．

この回路でストリングマッチング可能なパターンの最大文字数は (n + 1) p であり，パターンが最大文字数のストリングマッチングはすべてのマルチプレクサの $select\_bit$ を 0 に設定することで実行される．その一方で，パターンの文字数が p 以下ならば，すべてのマルチプレクサの $select\_bit$ を 1 に設定することで，回路は最大 n + 1 個の入力系列（パケット）を

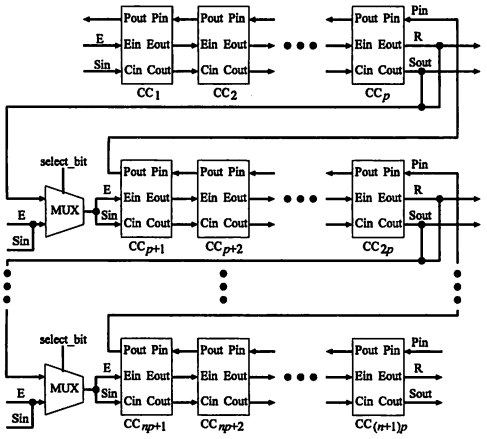

図 8 拡張アーキテクチャ
Fig. 8 Extended architecture.

同時に処理することが可能である．このアーキテクチャでは，*select_bit* の値を任意に設定することによって，さまざまなパターンに対して柔軟にストリングマッチングを実行することが可能となる．*select_bit* はシフトレジスタを使用することで実現され，それらの値もパターンと同様，ストリングマッチングを開始する前に設定される．

3.6 実験的評価

提案ストリングマッチングハードウェアを FPGA 上に実装し，実際にテキストといくつかのパターンを与えてストリングマッチングを行った．ハードウェア記述は Verilog-HDL を用い，FPGA 設計ツールには Xilinx ISE Version9.2i と Synplify Premier9.4，シミュレーションツールには ModelSim XEIII6.3c を使った．

今回は，図 8 におけるパラメータを $p = 16$，$n = 10$ と設定して実装を行った．提案ハードウェアを実現するために記述した Verilog-HDL の行数は 7812 行，論理合成の結果，回路で使用した LUT 数は 84360 である．回路の最大クロック周波数は 181.7MHz であった．提案ハードウェアは 1 クロックで最大 11 文字まで同時に処理を実行できるため，1 秒あたり最大で約 20 億 200 万文字の入力系列に対してストリングマッチングを実行することが可能である．1 文字 8bit であることから，この提案ハードウェアの最大スループットは 16Gbps(Gigabits per second) となる．よって，本アルゴリズムを NIDS のパターンマッチングに用いた場合，ギガビットネットワークにも対応可能である．一方，ソフトウェア（60Mbps）ではギガビットネットワークに対応できないことが知られている [5].

4. 提案 NIDS

前節で示したストリングマッチングハードウェアを用いた新たな NIDS について説明する．

4.1 NIDS のシステム構成

提案ハードウェアに基づく NIDS の新しいシステム構成を提案する．FPGA で実現することを前提とした従来の NIDS では，パターンも含めて回路として実現することで，パターンマッチングの高速実行とコンパクトな回路構成を実現していた [2]．しかし，このような実現方式においては，パターンが更新されるたびに論理合成，配置，配線，FPGA のコンフィギュレーションなどの一連の手順を実行する必要がある．一般にパターンマッチングハードウェアは大規模回路になるので，その論理合成，配置・配線には多大な時間を必要とすることが一般的である．一方，NIDS においては，パターンの更新はできるだけ早急に行われることが望ましい．そこで，従来の方式のパターンマッチングハードウェアと，[4] で提案したパターンマッチングハードウェアを組み合わせた新しい NIDS のシステム構成を提案する．図 9 に提案する NIDS のシステム構成を示す．

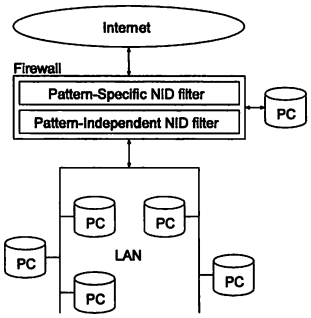

図 9 NIDS のシステム構成
Fig. 9 System architecture for NIDS

提案する NIDS においては，定常状態においては，ウィルス検出はパターン依存ハードウェアで行う．一般にパターンの更新は，新しいウィルスパターンの追加なので，新しいウィルスパターンが追加された場合には，パターン非依存ハードウェアに追加されたウィルスパターンを設定しウィルス検知を行い，更新以前のパターンについてはパターン依存ハードウェアでウィルス検出を行うことで，ウィルスパターン更新への即座の対処を可能とする．

パターン非依存ハードウェアは瞬時にパターンの更新が可能であるが，任意のパターンに対するマッチングを実現可能とするため，従来のパターン依存ハードウェアと比較して多大なハードウェア資源を必要とする．このため，NIDS のすべてのパターンに対してマッチングを並列実行する専用回路をパターン非依存ハードウェアで構成することはハードウェアコストの観点から非実用的である．そこで，提案する NIDS はパターン更新時に次のように対処する．まず，更新されたパターン集合全体に対して，パターン依存ハードウェアの再構成データを準備する．新規に追加されたパターンについてのウィルス検知はすでに実現されているので，回路の再構成データの準備には時間をかけることができる．回路の再構成データが準備でき次第，更新パターンを回路に組み込んだパターンマッチングハードウェアに更新する．このようにすることで，パターン更新に対する瞬時の対応が可能になり，ネットワークのセキュリティが向上する．

4.2 Snort ルールセットのマッチング

ウィルスを記述した Snort のルールセットが与えられた場合，各ルールをストリングマッチングハードウェアに入力可能なパ

表 1　各定義のマッチング
Table 1　Matching of each define

定義 1	◎	定義 2	◎	定義 3	◎	定義 4	◎	定義 5	○
定義 6	○	定義 7	○	定義 8	◎	定義 9	○	定義 10	○
定義 11	◎	定義 12	◎	定義 13	△	定義 14	◎	定義 15	△
定義 16	△	定義 17	△						

ターンに変換することが可能であれば，Snort と同等の機能をハードウェアで高速に実現することが可能になる．3 章で説明したストリングマッチングハードウェアが 2 章で示した Snort のルールの各定義に対してどこまでストリングマッチングを実行可能かを表 1 に示す．◎は与えられたルールを変換することなく直接，提案ハードウェアでマッチング可能なもの，○はルールをあらかじめハードウェアが処理可能なパターンに変換することで提案ハードウェアでマッチング可能なもの，△は機能の拡張を行った提案ハードウェアでマッチング可能であると予想されるものを意味する．

紙面の都合上，以下では△の定義に対するハードウェアの拡張についてのみ説明する．定義 13 では，'?' を記憶しているセルを CC_i，CC_i の右隣のセルを CC_{i+1} としたとき，CC_i はマッチングに成功している間'match' 信号を CC_{i+1} に出力し続ける．CC_{i+1} は'match' 信号が入力されるとマッチングを開始するが，最初にマッチングに成功したときのみ CC_{i+1} から'match' 信号を出力することで最小回数の繰り返しを実現できる．定義 15 は，マッチした入力系列を記憶しておくレジスタが必要となる．定義 15 に関しては，どのように参照された入力系列とのマッチングを効率よく行うかが重要となり，そのためにはレジスタの効率的な使い方が課題として挙げられる．定義 16 は，入力系列に対して先読みのパターンと先読み以外のパターンとのマッチングを別々の列のセルで同時に実行し，各マッチング結果を参照することで実行可能である．定義 17 に対しては，パターン入力時に各識別子に対応した制御信号をパターンと一緒に全セルに出力することによって実行可能である．

また，これらの定義以外に，Snort のルールセットでは定義 2 から 7 のネスト構造が存在する．定義 3，5，6 のネストに関してはパターンの変換を行うことでストリングマッチング実行可能となるが，定義 2，4，7 のネストに関しては，繰り返しを行う部分正規表現に演算子'*'，'+'，'{n, }' が含まれた場合のみ変換不可となるため，ストリングマッチングを実行できない．しかし，Snort v2.4 のルールセットで使用されるネスト構造は (*aggregate*\s+)*のみであり，これは入力系列に前処理を行うことによって提案ハードウェアでストリングマッチングを実行することが可能となる．ここで行われる前処理は，空白文字が 1 文字以上連続した場合，2 文字目以降の空白文字を提案ハードウェアに入力させずに破棄するものである．そのため，空白文字が 1 文字以上連続した場合でも提案ハードウェアには空白文字が 1 文字だけ入力されることになる．ここで\s+は空白文字の 1 回以上の繰り返しを意味するが，前処理によって空白文字が 1 文字以上連続した場合でも空白文字 1 文字のみしか提案ハードウェアには入力されないため，このパターンは (*aggregate*\s)*と変換することができる．変換されたパターンはネスト構造が存在しないため，提案ハードウェアでストリングマッチングを実行することが可能となる．

以上より，パターンの変換および拡張したハードウェアを提案することによって，提案ハードウェアは全ての Snort ルールに対してストリングマッチングの実行が可能であると考えられる．

4.3　パターン変換ソフトウェア

前節で述べたように，Snort ルールセットの中には，ルールを変換することで，提案ハードウェアでストリングマッチングを実行できるものが存在する．例えば，パターン *a*{3, } は *aaa*+ と変換することによって提案ハードウェアでストリングマッチングが実行可能となる．我々は与えられた Snort ルールを，提案ハードウェアで実行可能なルールに変換するパターン変換ソフトウェアを Perl を用いて開発した．前節で説明した提案ハードウェアの拡張を行い，パターン変換ソフトウェアにより与えられた Snort ルールを提案ハードウェアの入力パターンに変換することで，提案ハードウェアは現在公開されているすべての Snort ルールに対してストリングマッチングが実行可能となる．

5. おわりに

本稿では，我々が提案したストリングマッチングハードウェアに基づく高速ネットワークに対するネットワーク侵入検知システム（NIDS）の新しいシステム構成を提案した．また，NIDS に対する代表的なオープンソフトウェアである Snort のルールセットを解析し，パターンの変換およびハードウェアの拡張を行うことで全てのルールに対して，ストリングマッチングが実行可能であることを示した．今後の課題としては，Snort のすべてのルールをパターンとして設定可能なハードウェアの拡張と FPGA 上での実現，および NIDS の実装などがある．

パターン変換ソフトウェアの開発にあたっては，本学情報工学部情報工学科 4 年若葉陽一君の協力を得た．本研究の成果の一部は平成 20 年度科学研究費補助金基盤研究 (C)(課題番号 20500054) による．

文　献

[1] Aoe, J. (ed.): *Computer Algorithms: String Pattern Matching Strategies,* IEEE Computer Science Press, 1994.
[2] Bispo, J., Sourdis, I., Cordoso, J. M. P., and Vassiliadis, S.: Regular expression matching for reconfigurable packet inspection, *Proc. 2006 IEEE International Conference on Field Programmable Technology,* pp.119–126, 2006.
[3] 若林真一：制限された正規集合を受理する VLSI 向きパターン・マッチング・マシン，信学技報，AL85-33，1985.
[4] 川中洋祐，若林真一，永山忍：パターンを実行時に設定可能な正規表現ストリングマッチングマシンと FPGA による実現，電子情報通信学会リコンフィギャラブルシステム研究会技術研究報告，RECONF2007-61，2008.
[5] H. C. Roan, W. J. Hwang, and C. T. Dan Lo.: Shift-Or Circuit for Efficient Network Intrusion Detection Pattern Matching, *Proc. International Conference on Field Programmable Logic and Applications,* pp.785–790, 2006.
[6] http://www.snort.org/
[7] http://perldoc.perl.org/perlre.html