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

2006－DPS－126（8）
2006－CSEC－32（8）
2006／3／16

マトリックス分解によるパケットフィルタリングルールの圧縮

松田 勝志
NEC インターネットシステム研究所

概要

企業や組織のネットワークを外部からの不正なアクセス等から守る方法の一つにパケットフィルタリングがある．パケットフィルタリングを適切に運用管理するには，複雑で多数のルールを正確に把握しなければならない．しかしながら，ルールは日々の運用で単調に増加する傾向があるため，徐々に運用管理のコストが上がってきてしまう．

本稿では，パケットフィルタリングのルール集合を詳細に分析することができるマトリックス分解とそれを用いたルール数の削減手法について述べる．不要ルール削除，冗長条件ルール修正，ルール統合の3種類の手法を用い，ルール集合の圧縮を行う．実際に用いられていたルール集合を用いてルール集合の圧縮実験を行ったところ，525 ルールを 348 ルールに削減することができた(圧縮率 66.3%)．

A Packet Filtering Rules Compression by Decomposing into Matrixes

Katsushi MATSUDA
Internet Systems Research Laboratories, NEC Corp.

Abstract

Packet filters are essential for organizations that are connected to the Internet. Network administrators have to understand precisely complicated rules to manage the packet filter. Management cost, however, will rise gradually as the number of the rules increase at daily operation.

In this paper, we propose a novel model called “matrix decomposition” which enables to analyze rules of filtering, and a rule set compression method using this model. We formulated three techniques, removable rules detection, revisable rules detection and rules combination, and implemented a prototype system. The experiment using an actual rule set showed that our system could reduce the number of rules from 525 to 348, namely the compression ratio was 66.3%.

1 はじめに

企業や組織のネットワークを外部からの不正なアクセス等から守る方法の一つにパケットフィルタリングがある．パケットフィルタリングは，複数のネットワークを接続するゲートウェイやルータに設置されるネットワーク機器またはソフトウェアであり，ネットワークを流れるパケットの属性をルールと照合することで，そのパケットの通過の可否を決定し，内部ネットワークを守る．

パケットフィルタリングにより内部ネットワークを守ることは可能であるが，そのためにはパケットフィルタリングのルール集合を適切に設定・管理する必要がある．しかしながら，パケットフィルタリングのルール数は数 10～数 100，時には数 1000 に及ぶ．しかも各ルールの条件は複数の属性の組み合わせになるため，ネットワーク管理の専門家ですらルール集合全体としてどうなっているのかを理解することは難しい．

特に，ルール集合の管理においては，時間とともに単調増加するルール数を如何に減少させて管理を容易にするかが問題である．実際，長期間運用しているルール集合には，なくてもルール集合全体の意味が変らないルールが多数存在する．また,ルールを作成した状況や日時によって生まれる無用に複数のルールに分割されたルールな

ども多数存在する．これらのようなルールはルール数を増加させ，その結果ネットワーク管理者の管理コストを増大させている．

筆者らは，パケットフィルタリングのルール集合を詳細に分析することを可能にするマトリックス分解の手法と，それを用いて不要なルールを検出する方法について報告した[1]．本稿では，マトリックス分解を用いることで，複数のルールを統合し，ルール集合のルール数を削減する方法について述べる．

2. マトリックス分解

ルール集合のルール数を削減するためには，ルール集合中の削除対象となるルールを検出する適切なデータ表現形式が必要となる．筆者らはこのためにマトリックス空間データを用いている[1]．本節では，マトリックス空間データの概念とその生成方式であるマトリックス分解について述べる．

マトリックスとは，ルール集合で作る多次元空間を，各ルールの各条件属性の範囲指定された全ての境界点で区割りしてできる最小の多次元立方体である．このマトリックスにより，ルール集合全体を表現したものがマトリックス空間データである．そして，ルール集合からマトリックスを作成し，ルールとマトリックスの対応付けを行うことでマトリックス空間データを生成することをマトリックス分解と呼んでいる．以下，マトリックス分解について詳細に述べる．

2.1 マトリックス生成

ルール集合からマトリックスを作成するのがマトリックス生成フェーズである．一般的なパケットフィルタリングシステムでは，5 種類以上の条件属性（SrcIP，SrcPort，DestIP，DestPort，Protocol）が用いられているが，ここでは簡単のため，2 種類で説明する．また，それらの条件属性は 0 から始まり，比較的小さな値（例えば，13 や 15 等）で終わるものとする．

図 1 に 8 個のルールを持つルール集合の例とそれを 2 次元平面で表現する．優先順位は R1 が最も高く，R8 がデフォルトルールである．ルールの書式は（*第 1 属性の範囲, 第 2 属性の範囲, アクション*）とする．アクションは A が accept（通過を許可），D が deny（通過を禁止）を表し，2 次元平面では A が白，D が灰色となっている．

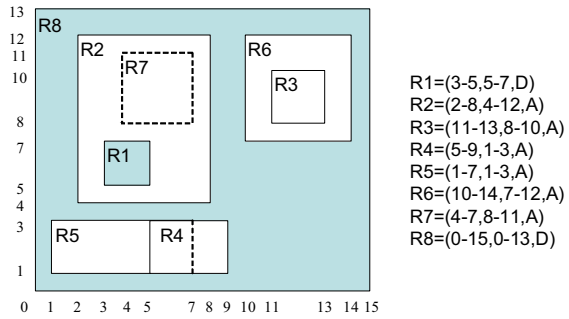

図 1．ルール集合の例

マトリックス生成は，まず条件属性毎にルール集合の各ルールの条件範囲の開始点と終了点（以降，境界点と呼ぶ）を集め，重複をなくした上でソートする．そして各属性を軸として，境界点の交点で極小の多次元立方体を作る．この多次元立方体がマトリックスである．図 1 のルール集合の例では，X 軸（第 1 属性）の境界点が 14 個，Y 軸（第 2 属性）の境界点が 11 個であるため，130 個（=(14-1)*(11-1)）のマトリックスが生成される．

マトリックスは，ルールと類似した書式を持つ．すなわち，（*第 1 属性の範囲, 第 2 属性の範囲, ルールリスト*）である．以降，この書式のデータをマトリックスデータと呼ぶ．

2.2 ルールとマトリックスの対応付け

マトリックスを生成した後，ルールとマトリックスの対応付けを行う．すなわち，ルールにはそのルールを構成するマトリックスのリストを，マトリックスにはそのマトリックスに関係するルールのリストを列挙することである．

この対応付けによって，ルールの書式は（*第 1 属性の範囲, 第 2 属性の範囲, アクション, マトリックスリスト*）となる．以降，この書式のデータをルールデータと呼ぶ．

ルールデータのマトリックスリストにはそのリストの順序には意味がないが，マトリックスデータのルールリストの順序はルール集合の優先順位に準拠している必要がある．例えば，図 1 の平面図の R1 の部分で説明すると，R1 の下に R2 があり，更にその下に R8 があるため，R1 を構成するマトリックスデータのルールリストは，（R1,R2,R8）となる．

2.3 マトリックス空間データ

ルールと対応付けたマトリックスデータ，マトリックスと対応付けたルールデータを総称したものがマトリックス空間データである．このマトリックス空間データを用いることで様々な分析が可能となる．

例えば，分析の例としてパケット通過テスト(packet classification)がある．

2.1 節では，マトリックスの開始点と終了点はそれぞれルール集合で作られる境界点としていたが，実際のマトリックス空間データでは開始点は-0.5，終了点は+0.5 の値をそれぞれ加えている．ルールに記載される条件範囲の両端の値もその条件に含まれるためであり，このように境界をずらすことで両端の値も正確に扱うことができる(詳細は[1]を参照)．

またマトリックスの特性上，ルール数が増加すると，生成されるマトリックス数も指数的に増加する．この増加を抑制するために，不要なマトリックスを削除している．すなわち，デフォルトルールにしか関係しないマトリックスをマトリックス空間データから削除する[1]．このマトリックス削除によって，生成されたマトリックスの 1/4～1/5 に削減できる．

3 ルール圧縮

本節では，時間の経過とともにルール数が増加したルール集合を分析し，ルール数を削減させるルール圧縮について述べる．ルール数を削減することは，ネットワーク管理者やシステム管理者のルール集合を把握するコストを削減する．これによって，メンテナンス性が向上し，結果的にセキュリティも向上する．

3.1 ルール圧縮手順

ルール集合の意味を変えずにルール数を削減するには，大きく分けて二種類の方法がある．一つは削除可能なルールを削除することであり，他方は統合可能なルールを統合することである．

ルール削除には，不要ルールの削除と冗長条件ルールの修正がある[1]．冗長条件ルールの修正は直接ルールを削減することはないが，この修正によって再度不要ルールを検出することができる．

ルール統合は，2 個のルールの統合を行う処理を繰り返し，統合可能なルールがなくなるまで行う．

ルール圧縮の効率向上のため，ルールを削除した後に統合を行う．具体的には以下の手順となる．

(i) 不要ルール削除
(ii) 冗長条件ルール修正
(iii) 再不要ルール削除
(iv) ルール統合

以下，それぞれについて詳細に述べる．

3.2 不要ルール削除

不要ルールとは，そのルールがルール集合から省かれた場合でも，ルール集合全体の動作に違いがないようなルールである．一般的には，このような不要ルールは 2 種類ある．1 つは，どのようなパケットが到達しても発火しないルールであり，他方は，そのルールがなくても優先順位の低い別のルールが同じアクションを行う場合である．ここでは前者を潜伏ルール，後者を冗長ルールと呼ぶ．

マトリックスによって，これらの潜伏ルールと冗長ルールを定義することが可能である．すなわち，あるルール R_i が潜伏ルールであることを Concealed(R_i)，冗長ルールであることを Verbose(R_i)とすると，以下のようになる．

$$\text{Concealed}(R_i) \text{ iff } \quad \forall m\left\{\left(m \in R_i\right) \subseteq \sum_{j=1}^{i-1} R_j\right\} \tag{1}$$

$$\text{Verbose}(R_i) \text{ iff}$$
$$\forall m\left(m \in R_i\right)\left\{act(R_i) = act(next(m, R_i))\right\} \tag{2}$$

ここでルール集合を $R=\{R_1,R_2,\cdots,R_n\}$，マトリックスを m，ルール R_iのアクションを act(R_i)，マトリックス mのルールリスト中のルール R_iの次のルールを next(m,R_i)とする．なお，式(1)は，$i>1 \wedge i \neq n$，式(2)は $i \neq n$ である．

式(1)の定義によると，R_i を構成するマトリックス全てについてそのルールリストに R_i より優先順位の高いルールが存在すれば，ルール R_iは潜伏ルールである．一方，式(2)の定義によると，R_iを構成するマトリックス全てについてそのルールリストの R_iの次のルールのアクションが R_iと同じアクションであれば，ルール R_iは冗長ルールである．

式(1)および式(2)はあるルールが不要ルールである十分条件であるが，必要条件ではない．すなわち，(1)と(2)を組み合わせた以下の式(3)が必要十分条件となる．

$$\text{Removable}(R_i) \text{ iff}$$
$$\forall m\left(m \in R_i\right)\left(m \subseteq \sum_{j=1}^{i-1} R_j\right) \vee$$
$$\left(act(R_i) = act(next(m, R_i))\right) \tag{3}$$

ルール集合から不要ルールを検出するための

アルゴリズムは式(3)から容易に作成できる．また，あるルール R_i が不要ルールかどうかの判定は，最大 $|R_j|$ 回のマトリックスデータのルールリスト調査で可能である．

3.3 冗長条件ルール修正

冗長条件ルールとは，そのルールの条件属性の一部または全てについて範囲を縮小した場合でも，ルール集合全体の動作に違いがないようなルールである．不要ルールがルールそのものを削除するのに対して，冗長条件ルールはルールの一部を削除することになる．

冗長条件ルールによって削除される条件部分についても前節の不要ルールと同じく Concealed と Verbose の状態がある．更に削除する条件部分を除いた残りの部分は，ルールとして記述可能でなければならないため，残りの部分は多次元立方体になっている必要がある．冗長な条件部分を削除する代わりにルール数が増えるのは本末転倒であるため，このような条件を設けている．以下，マトリックスを用いて冗長条件ルールを定義する．

$$\begin{aligned}&\text{Revisable}(R_i)\ \text{iff}\\&\quad \forall m(m\in M)\exists M(M\subset R_i)\exists k\forall l(k,l\in K)\{\\&\quad range(R_i,k)\neq range(M,k)\\&\quad \wedge range(R_i,l)=range(M,l)\\&\quad \wedge(R_i\neq top(m)\vee act(R_i)=act(next(m,R_i)))\}\end{aligned}\quad (4)$$

ここで M は多次元立方体のマトリックス集合，K は条件属性の全集合，M の属性条件 k の範囲を range(M,k)，マトリックス m のルールリストの先頭ルールを top(m)とする．

式(4)は 1 番目と 2 番目の論理項で，1 種類の条件属性を除く他の条件属性が R_i と同じであるマトリックス集合 M を規定している．そしてそのような M の構成マトリックス全てが Concealed か Verbose のいずれかの場合に R_i が冗長条件ルールとなる．

不要ルールと同様に，ルール集合から冗長条件ルールを検出するためのアルゴリズムは式(4)から容易に作成できる．また，あるルール R_i が冗長条件ルールかどうかの判定は，属性の種類が k 個の場合，高々 $2k$ 回のマトリックス集合の生成と調査で可能である．

3.4 再不要ルール削除

冗長条件ルールの修正によって，あるルールの条件の範囲が狭くなる．この条件範囲の縮小によって，不要ルールが新たに発生することがある．典型的な例を図 2 に示す． ルール R2 の条件範囲の一部がルール R1 より優先順位が低いところに存在しているため，冗長ルールとして R1 を不要ルール削除では削除できない(図 2 の(a))．しかし，冗長条件ルール修正によって，ルール R2 の R1 によって隠されている部分が縮小される(図 2 の(b))．ここで再度不要ルール削除を行うと，ルール R1 は冗長ルールとして削除される(図 2 の(c))．

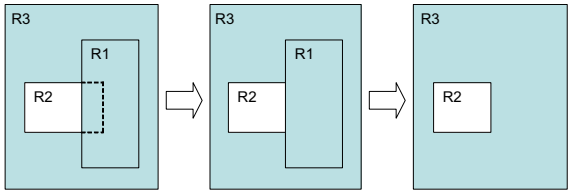

図 2．不要ルールの新規発生

不要ルール削除も冗長条件ルール修正も元のルール集合の意味を変えないため，何度でも適用可能である．すなわち，再不要ルール削除を行った後に再度冗長条件ルール修正と不要ルール削除を行っても構わない．

3.5 ルール統合

ルールの統合とは，ある 2 個のルールを統合して 1 個のルールにすることである．この処理によって，ルールが 1 個削除されることになる．ルール統合は，2 つのフェーズからなる．ある 2 個のルールが統合可能かどうかを判断するフェーズと，その統合されたルールをどの位置(優先順位)に挿入するかを判断するフェーズである．

ある 2 個のルールが統合可能かどうかの判断は，マトリックスを用いて以下のように定義する．

$$\begin{aligned}&\text{Unifiable}(R_i,R_j)\ \text{iff}\\&\quad \forall v(v\in(R_i+R_j))\{revised(R_i)\wedge revised(R_j)\\&\quad \wedge num(R_i+R_j)=num(R_i)+num(R_j)\\&\quad \wedge(in(R_i,v)\vee in(R_j,v))\}\end{aligned}\quad (5)$$

ここで v は多次元立方体の頂点，ルール R が冗長条件ルールではないことを $revised(R)$，ルール R の構成マトリックスの数を $num(R)$，ルール R の条件範囲の中に頂点 v が含まれていることを $in(R, v)$とする．

式(5)の第 1 項では，対象となるルールがそれぞれ冗長条件ルールではないことを前提条件としている．第 2 項では，それらのルールの構成マトリックスの和がそれらのルールでできる多次元立方体の構成マトリックスの数と等しいことを，第 3 項では，それらのルールでできる多次元立方体の頂点全

てがそれらのルールの中に含まれることをそれぞれ条件としている. すなわち, 2 個のルールが 1 個のルールになることを構成マトリックスの数で規定しており, 非常に高速に統合可能の判定が可能である.

統合されたルールをルール集合のどの位置に挿入するかは重要な問題である. 統合したルールの間に別のルールが存在しなければ, 元のルールの位置のいずれかに挿入すれば良い. しかしながら, 間にルールが存在した場合は状況に応じて挿入箇所を変えなければならない.

統合されたルール間に別のルールが存在するパターンは図 3 に示す 4 通りである. (a)と(b)はそれぞれ統合前のルールの一方と重なっている場合であり, (c)は 1 個のルールが統合前のルールの双方と重なっている場合であり, (d)は(a)と(b)の複合である.

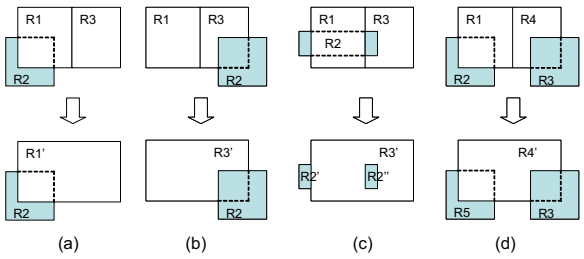

図 3. 統合ルールの挿入箇所

(a)および(b)の場合は, 統合前のルールの優先順位の高い方および低い方の位置に挿入すれば良い. (c)の場合は, 間にあるルールを分割する必要があるため, 統合そのものを中止する. (d)の場合は, 間にあるルールの優先順位を変えることで統合ルールを挿入することができるが, 元のルール集合における優先順位を変えることは避けたいため, この場合も統合を中止する.

4 フィルタリングルール分析システム

2 節で述べたマトリックス分解の仕組みを用い, 3 節で述べた不要ルールと冗長条件ルールの検出を行うルール分析システムを開発した. 分析システムは, Microsoft Windows Xp のアプリケーションで, Microsoft Visual C++ 6.0 で開発している. ルール集合の記述は, アクセス制御やファイアウォール設定や侵入検知システム設定を共通のフォーマットで記述できる汎用ポリシー記述言語 SCCML[2]を用いた.

図 4 に用いたルール集合の例を示す. このルール集合は Cisco IOS のパケットフィルタリングの例(一部省略している)であり, 時間とともにルールが増加し, 追加したルールはルール集合の上部に挿入されていったものとしている. 各ルールの詳細の説明は省くが, サーバが増減したり, サービスが開始終了したり(実際はサービスが交代している), と実際の運用によって生成される設定に近いルール集合となっている.

```
    ! policy
001 deny ip 10.33.109.0 0.0.0.255 10.26.192.0 0.0.0.255
002 permit tcp 10.33.138.0 0.0.0.255 host 10.26.195.172 range 20 23
003 permit tcp 10.33.138.0 0.0.0.255 host 10.26.195.172 eq www
004 permit tcp 10.99.3.0 0.0.0.255 host 10.26.192.3 range 20 21
005 permit ip 10.33.138.0 0.0.0.255 10.26.192.0 0.0.0.255
006 deny ip 10.33.109.0 0.0.0.255 host 10.26.192.10 range 0 19
007 deny ip 10.33.109.0 0.0.0.255 host 10.26.192.10 range 24 65535
008 permit ip 10.33.109.0 0.0.0.255 host 10.26.192.10
009 permit ip 10.33.109.0 0.0.0.255 10.26.192.0 0.0.0.255
010 deny ip 10.26.224.0 0.0.224.255 host 10.26.192.10
011 permit tcp 10.26.0.0 0.0.255.255 host 10.26.192.9 eq ssh
012 permit tcp 10.26.0.0 0.0.255.255 host 10.26.192.9 eq telnet
013 deny ip any host 10.26.192.5
014 permit tcp 10.26.0.0 0.0.255.255 host 10.26.192.10 range 20 21
015 deny tcp any gt 1023 host 10.26.192.192 eq www
016 permit ip 10.0.0.0 0.255.255.255 10.26.192.0 0.0.0.248 lt 1024
017 permit tcp any gt 1023 host 10.26.192.192 eq www
018 deny ip any any
```

図 4. ルール集合の例

SCCML への変換においては, 先頭から順にルール番号を割り当てている. このルール集合を SCCML に変換した後, 分析システムに読み込ませ, ルール圧縮を行った結果が図 5 である.

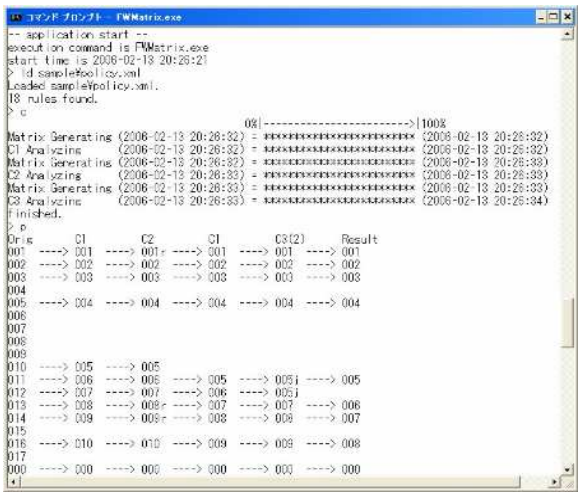

図 5. システムの実行例

c とは, ルール圧縮を行うコマンドであり, 実際には, 不要ルール削除を行う c1, 冗長条件ルール修正と再不要ルール削除を行う c2, そしてルール統合を行う c3 を連続して実行する. p とは, ルール集合がルール圧縮によってどのように変化したかを表示するコマンドである. c1 によってルール 004, 006～009, 015, 017 が削除され, c2 によ

ってルール 001, 013 と 014 が修正され, c1 によってルール 010 が削除され, c3 によってルール 011 と 012 が統合されたことを示している(元のルール番号で説明している). 図 4 のルール集合は 18 個のルールが 9 個のルールに圧縮されている(圧縮率 50%).

また, 筆者の所属している部門のルータのフィルタリングルールを用いて圧縮率の測定を行った. 実験に用いたルール集合は, Cisco IOS のフィルタリングルールであり, 525 個のルールである. IOS の記述から SCCML に変換し, システムでルール圧縮を行った. その結果を図 6 に示す.

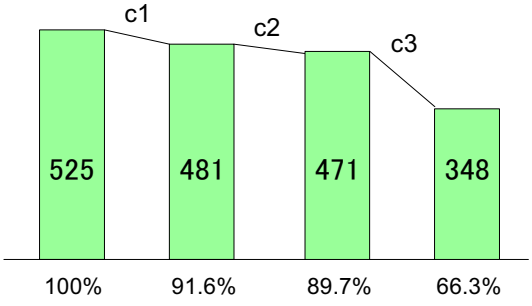

図 6. ルール圧縮の結果

[1]では, 不要ルール削除による 91.6%であったが, 今回冗長条件ルール修正と再不要ルール削除, ルール統合によって 525 個のルールを 348 個のルールに圧縮することができた. 圧縮率は 66.3%である. このように本システムを用いることでルール数を削減でき, ネットワーク管理のコストを低減させることができる.

5 関連研究

パケットフィルタリングルールの分析の研究で最も主流なのが, packet classification である. これは, あるパケットがどのルールと照合するかを速度とメモリ使用量の点から研究するというものである(例えば, [3]). 一方, ルール圧縮に関する研究は比較的最近始まった研究が多い. Eronen らのシステムでは, 制約論理プログラミングを用いて論理的に分析を行うことができ, 潜伏ルールを検出することが可能である[4]. また, Jalili らは高レベルな管理ポリシーであるが, 冗長ルールを検出することが可能であることを示している[5]. Al-Shaer らはルール間の関係を 5 種類に分類し, ルールのツリー(Policy Tree)を構築することで潜伏ルールと冗長ルールを検出している[6].

これらの研究はいずれもルール対ルールで分析しているため, 十分な圧縮ができるとは言えない. すなわち, 一部が冗長であり, 残りの部分が潜伏であるような状態のルールを検出することはできない. また, ルールを統合することもできていない.

6 おわりに

パケットフィルタリングのルール集合をマトリックス空間データによって表現することで, ルール数を削減することができることを示した. マトリックスは, ルール圧縮の方法である不要ルール削除, 冗長条件ルール修正, ルール統合を行う上で有効な表現形式であり, ルールをこれ以上細分化しても無意味という極小領域の集合として扱う手段を提供する. 本稿で提案したルール圧縮方法によって, 実際に運用されているルール集合の 525 個のルールを 348 個まで削減することができた.

本稿では, ルール圧縮について報告したが, マトリックスを用いることで更に様々な分析が可能である. 他の分析についても引き続き検討する予定である.

参考文献

[1] 松田,「マトリックス分解によるパケットフィルタリングルールの分析－不要ルールと冗長条件ルールの検出－, 情報処理学会研究報告 Vol.2005, No.122, pp.1-6, 2005.

[2] 岡城他,「セキュリティ運用管理のためのセキュリティポリシー言語 SCCML」, 情報処理学会研究報告 Vol.2004, No.129, pp.89-94, 2004.

[3] Gupta and McKeown, “Algorithms for Packet Classification”, IEEE Network, Vol.15, No.2, pp.24-32, 2001.

[4] Eronen and Zitting, “An Expert System for Analyzing Firewall Rules”, Proceedings of 6th Nordic Workshop on Secure IT-Systems (NordSec2001), pp.100-107, 2001.

[5] Jalili and Rezvani, “Specification and Verification of Security Policies in Firewalls”, EuroAsia-ICT 2002, LNCS 2510, pp.154-163, 2002.

[6] Al-Shaer and Hamed, “Modeling and Management of Firewall Policies”, IEEE Transaction on Network and Service Management, Vol.1-1, 2004.